POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

3D model-based tracking with one omnidirectional camera and particle filters.

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Matteo Matteucci
Correlatore Esterno: Prof. Alexandre Bernardino

Tesi di Laurea di:
Matteo Taiana, matricola 633410

Anno Accademico 2006-2007

Alla mia famiglia:
i miei nonni,
i miei genitori,
mia sorella
e il mio cagnolino.

Abstract

In order to carry out complex tasks robots need to extract information from the environment they operate in. Vision is probably the richest sensor employed in current robotic systems, due to the possibility of measuring a great diversity of features in the environment, e.g., colors, distances, textures and shapes. Omnidirectional vision cameras provide the additional benefit of an enlarged field of view.

In the context of semi-structured scenarios, as is the example of the RoboCup Middle Size League, the particular nature of the objects and their motions can be exploited to allow computer vision methods to work reliably and in real-time: objects are known to have specific colors and shapes, and motion follows the laws of physics. Color, shape and motion will be used in this thesis to develop a robust 3D object tracking system for autonomous robots equipped with one dioptric or catadioptric omnidirectional camera.

This thesis addresses three main problems in the context of 3D object tracking: (i) how to relate object's 3D shape and position information with the acquired 2D images; (ii) how to obtain measurements from the images taking into account the colors of objects and background; and (iii) how to integrate temporal information taking into account the laws of physics. We show that Particle Filtering methods constitute an adequate framework for addressing consistently all these problems.

The proposed architecture provides an efficient and reliable methodology to perform 3D target tracking for robots equipped with one omnidirectional camera. The accuracy and precision of the algorithm were tested in a RoboCup Middle Size League scenario, with static and moving targets, using either a catadioptric or a dioptric omnidirectional camera. The real-time capabilities of the system were also evaluated.

As a result of this work two papers were accepted in refereed international conferences: [38] and [39].

Acknowledgments

I would like to thank my main advisor, Prof. Alexandre Bernardino for his friendship and guidance, for making me feel at home form the very beginning in Portugal and at Instituto Superior Técnico.

I want to thank my other three advisors for the confidence they placed in me: Prof. Pedro Lima from Instituto Superior Técnico, Prof. Matteo Matteucci and Prof. Andrea Bonarini from Politecnico di Milano. I am grateful to Prof. José Gaspar, Dr Jacinto Nascimento and Abdolkarim Pahliani for the fruitful collaboration.

I would like to thank the friends who helped me throughout the work of this thesis, especially Dr Alessio Del Bue, Dr Luis Montesano and Hugo Costelha. I also wish to thank the lots of friends who supported me from the three different laboratories I belonged to: AIRLab, ISLab and VisLab.

Finally, I would like to thank all the friends who roamed with me the corridors of Politecnico, playing minella/studying. Your friendship is the most valuable outcome of those past years.

Riassunto

0.1 Introduzione

I robot sono sistemi controllati elettronicamente, in grado di interagire attraverso percezione e azione col mondo che li circonda; usano sensori per acquisire informazioni sull'ambiente e attuatori per compiere azioni. I robot sono definiti autonomi quando possono svolgere gli incarichi che sono loro assegnati senza la necessità di un intervento umano.

Una classe di sensori comunemente impiegata nella robotica, tipicamente con compiti di autolocalizzazione e navigazione, è quella delle telecamere omnidirezionali, cioè telecamere che hanno una ampiezza di campo visivo molto grande. Il vantaggio di questo tipo di telecamera, rispetto a quelle convenzionali, con funzione di proiezione prospettica, è quello di acquisire informazioni su una porzione maggiore dello spazio che circonda il robot. Le telecamere proiettano i raggi di luce provenienti dall'ambiente su un sensore di immagine, un piano di celle CCD o CMOS. Esso converte l'informazione luminosa in cariche elettriche, formando l'immagine digitale. A seconda della modalità con la quale la luce è proiettata sul sensore di immagine, le telecamere sono classificate come diottriche (quelle per cui la proiezione avviene solo tramite diffrazione) o catadiottriche (quelle per cui la proiezione avviene tramite riflessione e diffrazione). Uno svantaggio delle telecamere omnidirezionali rispetto alle convenzionali sta nel fatto che le immagini che producono sono soggette a forte distorsione, questo può costringere all'uso di algoritmi non standard per l'identificazione e l'inseguimento di oggetti. Un altro problema particolarmente evidente nel tracking con telecamere omnidirezionali è che l'oggetto inseguito, immerso in una scena dinamica, può presentare nell'immagine occlusioni, sovrapposizioni e, in generale, ambiguità.

Questa tesi presenta un sistema di inseguimento 3D basato su *particle filter*, per robot autonomi equipaggiati con telecamere omnidirezionali. Il sistema di inseguimento è progettato per funzionare in un ambiente parzial-

mente strutturato, in cui ogni oggetto abbia un proprio colore distintivo, ma nel quale l'illuminazione sia variabile, come quello della RoboCup Middle Size League (MSL). Le ipotesi del filtro rappresentano posizione e velocità tridimensionali dell'oggetto inseguito, la funzione di verosimiglianza si basa su colore e forma dell'oggetto.

0.2 Descrizione del sistema

Abbiamo progettato e implementato un sistema di inseguimento robusto per robot autonomi equipaggiati con una telecamera omnidirezionale, diottrica oppure catadiottrica.

Usiamo un *particle filter* per stimare a ogni istante la posizione e la velocità tridimensionali di un oggetto, date le osservazioni prese negli istanti di tempo precedenti e una funzione di densità di probabilità iniziale. In questa tesi consideriamo solo oggetti rotazionalmente simmetrici, ma il metodo può essere facilmente esteso perchè funzioni con oggetti rigidi di forma arbitraria, includendo la stima sull'orientazione e la velocità di rotazione.

I *particle filter*s usano una rappresentazione basata su campioni della funzione di densità di probabilità; ogni campione è chiamato particella oppure ipotesi. Per calcolare iterativamente la stima di posizione e velocità dell'oggetto inseguito, è necessario modellare probabilisticamente sia la sua dinamica di moto che la maniera in cui calcoliamo la verosimiglianza di una particella.

Per calcolare l'approssimazione della funzione di densità di probabilità a posteriori, gli algoritmi di inseguimento funzionano tipicamente in tre passi:

1. *Previsione* - calcola una approssimazione di $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1})$, muovendo ogni particella come indicato dal modello di movimento;

2. *Aggiornamento* - il peso di ogni particella i è aggiornato usando la verosimiglianza $p(\mathbf{y}_t \mid \mathbf{x}_t^{(i)})$:

$$w_t^{(i)} \propto w_{t-1}^{(i)} p(\mathbf{y}_t \mid \mathbf{x}_t^{(i)}) \tag{1}$$

3. *Resampling (ricampionamento)* - le particelle con un peso alto sono replicate, mentre quelle con un peso basso sono dimenticate.

Per caratterizzare la dinamica dell'oggetto usiamo un modello autoregressivo a velocità costante: a ogni istante di tempo la velocità delle particelle è aggiornata sommando a essa un rumore di accelerazione. La posizione delle particelle è aggiornata, in funzione della velocità che le caratterizzava nell'istante precedente e dell'accelerazione.

Abbiamo progettato un modello di osservazione per associare un valore di verosimiglianza a una posizione ipotetica, dato il modello dell'oggetto inseguito, una immagine acquisita dalla telecamera omnidirezionale e il corrispondente modello di proiezione della telecamera.

Nel corso del lavoro usiamo due telecamere omnidirezionali: una catadiottrica (si veda la Figura 1a) e una diottrica (si veda la Figura 2a). La telecamera catadiottrica è costituita da una telecamera prospettica di fronte alla quale è posto uno specchio, il cui profilo è stato progettato perchè il sistema finale produca una vista a risoluzione costante del pavimento. Per modellare esattamente questa telecamera sarebbe necessario usare un generico Catadioptric Projection Model (CPM) (Figura 1b), che tiene conto dello specifico profilo dello specchio. Dato che questo modello esatto è non lineare e non è esprimibile in forma chiusa, usiamo due modelli di proiezione approssimati: lo Unified Projection Model (UPM) (Figura 1c) e il Perspective Projection Model (PPM), che è un caso particolare dell'UPM. La telecamera omnidirezionale diottrica (fish-eye) è semplicemente una telecamera con una lente fish-eye, che le conferisce un'apertura di campo di 185°. Per modellarla usiamo l'Equidistance Projection Model (EPM) (Figura 2b).

Ognuno dei modelli di proiezione impiegato deve essere calibrato (si veda la Figura 3) prima di essere usato. I valori dei suoi parametri sono ottenuti minimizzando l'errore di *backprojection*. Vengono effettuate due proiezioni di un insieme di punti tridimensionali sull'immagine, una prioeizione è fatta fisicamente, usando la telecamera, mentre l'altra è effettuata tramite il modello. L'errore di *backprojection* è semplicemente la media quadratica delle distanze tra i punti proiettati con i due sistemi.

Gli oggetti inseguiti dal sistema sono caratterizzati dalla propria forma tridimensionale, dalla dimesione e dal colore. Modelliamo la forma tridimensionale degli oggetti con forme geometriche semplificate, mentre il loro colore è modellato con un istogramma di colore, il quale è costruito in una fase di impostazione del sistema, anteriore al tracking. L'istogramma di colore è costruito basandosi su alcune proiezioni dell'oggetto sull'immagine, acquisite in condizioni di illuminazione e posizioni diverse.

I passi per associare un valore di verosimiglianza a una ipotesi sono:

- Proiezione del modello di forma tridimensionale sul piano dell'immagine. Il risultato sono due insiemi di pixel, uno su ciascun lato del contorno che l'oggetto seguito proietterebbe se si trovasse esattamente nella posizione ipotetica.

  Per ottenere i due insiemi di pixel, il modello della forma è traslato e ruotato affinchè si trovi nella posizione ipotetica. La rotazione è resa

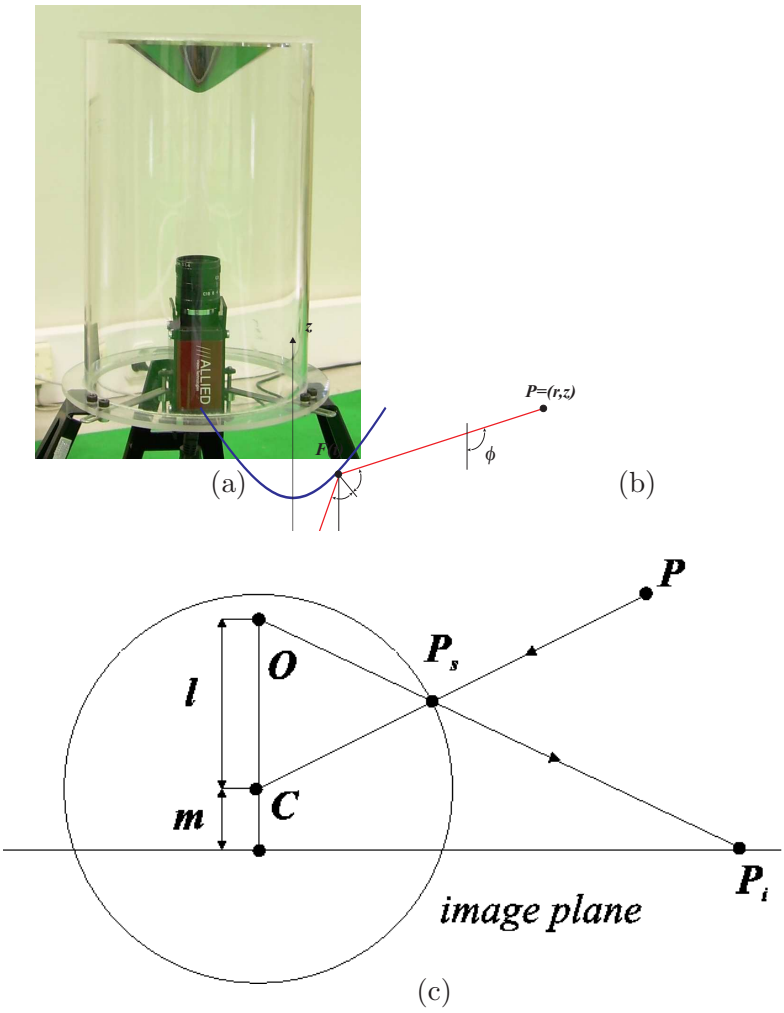
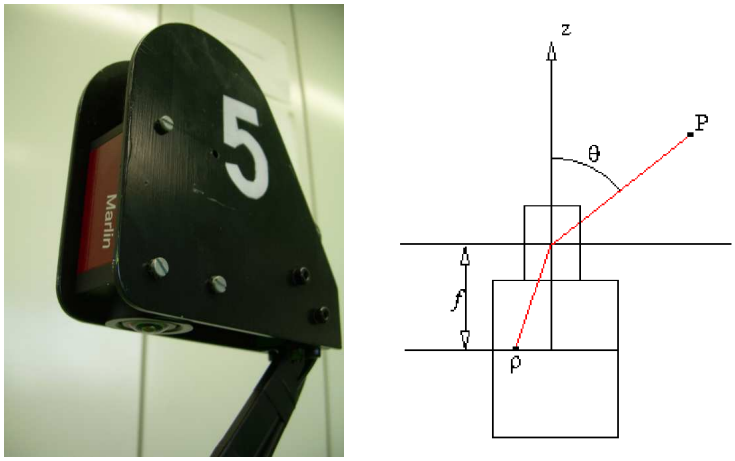
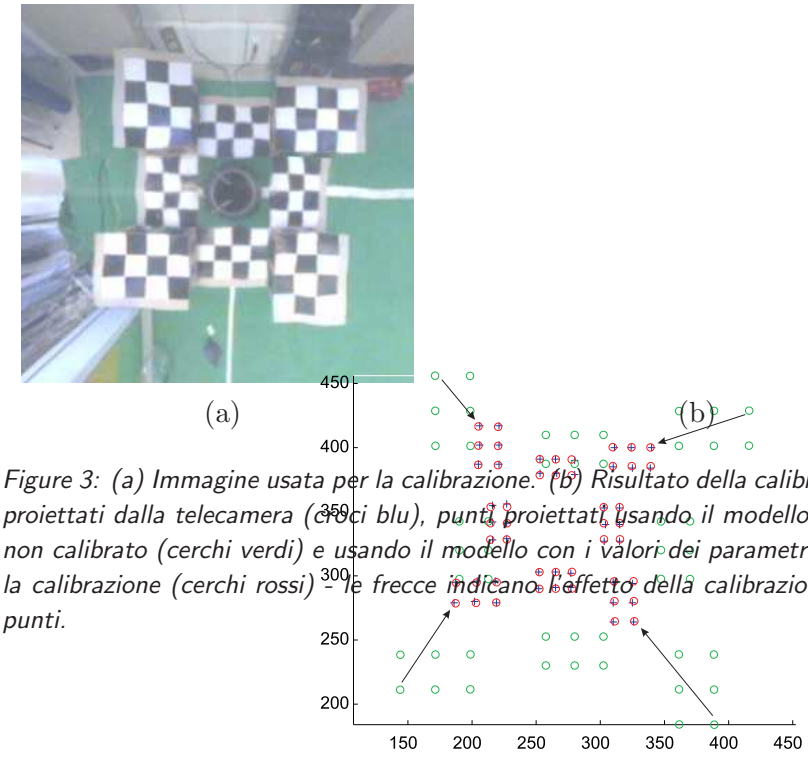


Figure 1: (a) Telecamera omnidirezionale catadiottrica. (b) Il generico modello di proiezione catadiottrico: Catadioptric Projection Model. (c) Il modello di proiezione per telecamere centrali: Unified Projection Model.



(a) (b)

Figure 2: (a) Telecamera omnidirezionale diottrica, si noti la lente fisheye. (b) Il modello di proiezione Equidistance Projection Model.



(a) (b)

Figure 3: (a) Immagine usata per la calibrazione. (b) Risultato della calibrazione: punti proiettati dalla telecamera (croci blu), punti proiettati usando il modello di proiezione non calibrato (cerchi verdi) e usando il modello con i valori dei parametri ottenuti con la calibrazione (cerchi rossi) - le frecce indicano l'effetto della calibrazione su quattro punti.

necessaria dal fatto che usiamo un modello semplificato della forma 3D, il quale ci permette di proiettare solo i punti 3D che andranno a formare il contorno della proiezione dell'oggetto.

Per determinare i due insiemi di pixel usiamo uno di due metodi: il primo metodo è basato sulle B-spline e su distanze misurate nell'immagine, mentre l'altro indivua in 3D i punti che verranno proiettati all'interno e all'esterno del contorno ipotetico (si veda il Capitolo 4 per i dettagli sui due metodi).

La proiezione dei punti tridimensionali verso i punti bidimensionali dell'immagine avviene attraverso l'applicazione del modello di proiezione scelto.

- Costruzione di due istogrammi di colore, che caratterizzano il colore dell'immagine nell'intorno interno ed esterno del contorno proiettato.

  Usiamo istogrammi nello spazio colore HSI (Hue, Saturation, Intensity) con $12 \times 12 \times 4$ bin nell'implementazione Matlab. Nell'implementazione C++, invece, usiamo lo spazio colore YUV e $4 \times 8 \times 8$ bin. In entrambi i casi usiamo un numero di bin per il canale che codifica l'informazione sull'intensità luminosa minore del numero dei bin usati per codificare l'informazione degli altri canali. Questa scelta serve a conferire robustezza al sistema rispetto ai cambiamenti di illuminazione. Trascurare completamente l'informazione sull'intensità luminosa ci avrebbe impedito di inseguire oggetti bianchi o neri.

- Calcolo della verosimiglianza in funzione della similarità tra coppie di istogrammi di colore.

  Calcoliamo la similarità tra due istogrammi con la metrica di Bhattacharyya. Maggiore la similarità tra l'istogramma che modella l'oggetto da inseguire e quello che descrive il colore dell'immagine all'interno del contorno proiettato, maggiore la verosimiglianza dell'ipotesi. Minore la similarità tra i due istogrammi che descrivono il colore dell'immagine ai due lati del contorno proiettato, maggiore la verosimiglianza. La prima di queste due regole in pratica significa che associamo una verosimiglianza alta a contorni che hanno al loro interno colori simili a quello dell'oggetto inseguito. La seconda significa che associamo una alta verosimiglianza a contorni nel cui intorno si trova una transizione cromatica (o di luminosità). Usiamo un coefficiente per pesare l'influenza delle due regole sulla funzione di verosimiglianza. Nella Figura 4 sono rappresentati i contorni proiettati da una palla, in tre diverse posizioni ipotetiche, su un'immagine acquisita

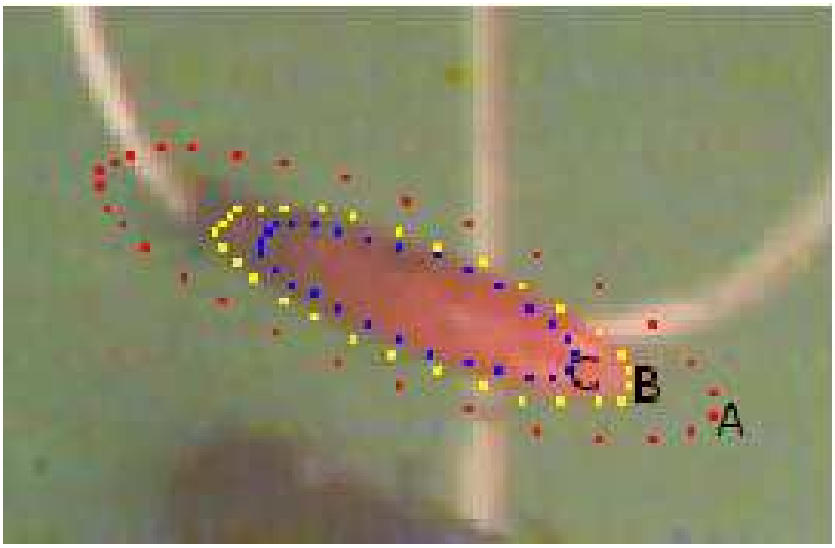


Figure 4: Tre contorni ipotetici proiettati sull'immagine. In questa discussione indichiamo per bervità con: "colore all'interno (o all'esterno) di un contorno" il colore all'interno (o all'esterno) del contorno, in prossimità dello stesso. Il contorno A ha una bassa verosimiglianza, a causa della grande differenza tra il modello di colore dell'oggetto e il colore all'interno del contorno (il colore all'interno del contorno non è arancione) e una grande similarità tra il colore ai due lati del del contorno. Il contorno C è associato con una verosimiglianza media, perchè il modello di colore dell'oggetto e il colore all'interno del contorno sono simili, ma i colori ai due lati del contorno sono simili. Il contorno B ha una alta verosimiglianza, in quanto il colore all'interno del contorno è simile al modello di colore della palla e i colori ai due lati del contorno sono diversi.

col sistema catadiottrico. Viene mostrato come la funzione di verosimiglianza privilegi l'ipotesi col contorno più simile a quello della palla presente nell'immagine. Questo metodo, comparato con altri che si basano sulla estrazione dei contorni, è più robousto dato che riesce a gestire anche contorni sfumati, che sono frequenti negli ambienti dinamici.

Per inizializzare il *particle filter* usiamo due metodi: (i) impostiamo la funzione di densità di probabilità iniziale secondo una Gaussiana con un'alta varianza, centrata su una stima fatta a mano della posizione e velocità iniziale dell'oggetto inseguito, oppure (ii) selezioniamo una posizione tridimensionale in base a una identificazione bidimensionale dell'oggetto e usiamo questa come come media della Gaussiana (si veda il Capitolo 5 per maggiori informazioni).

0.3 Risultati sperimentali e direzioni di ricerca

Abbiamo sviluppato sia un'implementazione del sistema in Matlab, che una in C++ basata sulla libreria Integrated Performance Primitives (IPP) della Intel. Abbiamo eseguito diversi esperimenti, nell'ambiente della RoboCup MSL, per verificare l'accuratezza e la precisione del sistema di inseguimento proposto.

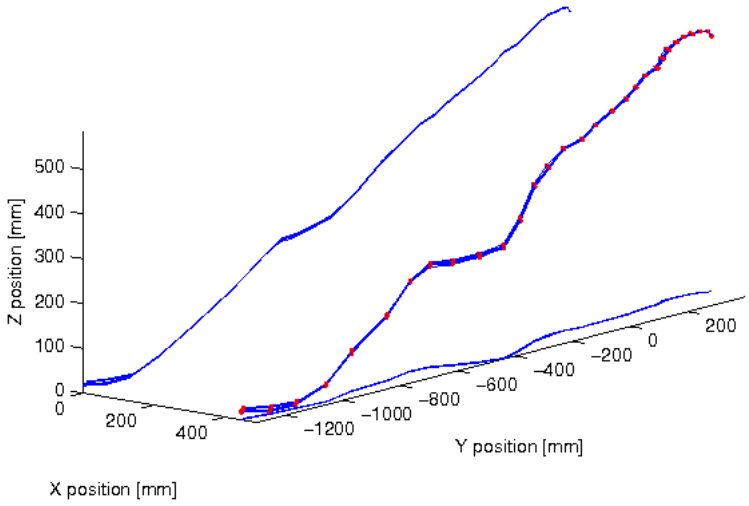
Abbiamo eseguito diversi inseguimenti su sequenze di immagini acquisite con la telecamera omnidirezionale catadiottrica. Usando il modello di proiezione UPM, abbiamo inseguito una palla in una sequenza di immagini in cui essa scendeva lungo una rampa, una palla lungo una traiettoria con cambiamenti repentini della direzione di moto (rimbalzi sul campo), si veda la Figura 5 e un robot lungo una traiettoria sul pavimento. Gli inseguimenti della palla mostrano la robustezza del metodo rispetto a motion blur e rumore del sensore di immagine (Figura 6a e 6b).

Sempre nella configurazione con telecamera catadiottrica, abbiamo eseguito un esperimento per misurare l'accuratezza del sistema proposto: abbiamo posto una palla in diverse posizioni attorno al robot e misurato l'errore nella localizzazione ottenuta col nostro metodo, rispetto alla posizione reale.

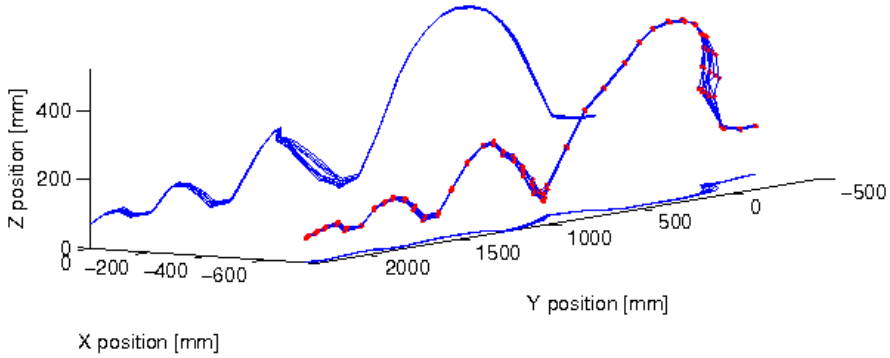
Abbiamo comparato i risultati di inseguimento e localizzazione ottenuti usando due diversi modelli del sensore catadiottrico, UPM e PPM, verificando che PPM fornisce prestazioni simili a UPM per accuratezza e precisione, ma offre il vantaggio di un costo computazionale meno elevato.

Abbiamo inoltre inseguito una palla in una sequenza di immagini acquisita con la configurazione diottrica del sensore omnidirezionale, verificando la robustezza del sistema contro le occlusioni (si veda la Figura 6c) e la sua capacità di funzionare in tempo reale (29fps su un processore Pentium 4 a 2.6GHz).

In futuro intendiamo estendere il sistema rendendolo in grado di inseguire più oggetti contemporaneamente. Vogliamo sperimentare l'uso di modelli di movimento multipli, per esempio uno che descriva il moto di una palla che rotola liberamente sul pavimento e un altro che descriva il comportamento della palla in presenza di un urto. Intendiamo inoltre sfruttare la caratteristica del sistema di avere un vettore di stato con le coordinate spaziali dell'oggetto inseguito, usando modelli di movimento basati sulle leggi della fisica (per esempio applicando l'accelerazione di gravità agli oggetti che non sono in contatto con il suolo). Esprimeremo le coordinate dell'oggetto in un sistema di riferimento inerziale. Per far questo useremo la *ego-motion compensation*: applicheremo il movimento effettuato dal robot, al contrario, sulle particelle del filtro.



(a)



(b)

Figure 5: (a) Palla che scende lungo una rampa: i dieci percorsi risultanti da dieci esecuzioni dell'algoritmo sulla stessa sequenza di immagini. Le dieci linee blu con punti rossi rappresentano le dieci traiettorie tridimensionali stimate, le linee blu sono le proiezioni di queste traiettorie sul piano di terra e su quello laterale. (b) Palla rimbalzante: stesso tipo di grafico per le traiettorie stimate nella sequenza di immagini nella quale la palla rimbalza sul pavimento.

(a) (b) (c)

Figure 6: Ingrandimenti di alcune immagini in cui il sistema di seguimento ha successo nonostante condizioni poco favorevoli. (a) Un'immagine che presenta del motion blur sulla proiezione della palla. (b) Un'immagine in cui è evidente il rumore del sensore di immagine. (c) Un'immagine in cui la palla è visibile solo in parte, a causa dell'occlusione da parte di un robot.

Contents

Abstract **I**

Acknowledgments **III**

Riassunto **V**

0.1 Introduzione V
0.2 Descrizione del sistema VI
0.3 Risultati sperimentali e direzioni di ricerca XII

1 Introduction **1**

1.1 Problem formulation 1
1.2 Description of the system 3
1.3 Structure of the thesis 5

2 State of the art **7**

2.1 Color based image measurements 7
2.2 Tracking 8
2.3 3D model based methods 9

3 Omnidirectional vision **11**

3.1 Catadioptric omnidirectional camera 11
3.1.1 Projection models 13
3.2 Dioptric omnidirectional camera 14

4 Model-based detection in omnidirectional images **19**

4.1 Detection algorithm 19
4.2 3D model 20
4.2.1 Selection of boundary pixels using sampling 23
4.2.2 Selection of boundary pixels using B-splines 23
4.3 Color model 26
4.4 Likelihood 30

5 Tracking with particle filters **33**
5.1 Particle filter tracking. 33
5.1.1 Motion model 35
5.1.2 Observation model 35
5.2 Initialization of the particle filter 35

6 Experimental results **37**
6.1 Catadioptric setup, UPM 37
6.1.1 Ball tracking 38
6.1.2 Robot tracking 38
6.1.3 Error evaluation 40
6.2 Catadioptric setup, UPM and PPM experimental comparison 43
6.2.1 Error evaluation 43
6.2.2 Ball tracking 46
6.3 Dioptric setup 47
6.3.1 Jumping ball 47
6.3.2 Occlusions 48
6.3.3 Multiple objects 48

7 Conclusions and future work **59**

Bibliography **61**

A Paper published in RoboCup Symposium 2007 **65**

B Paper to be published in IROS 2007 **79**

Chapter 1

Introduction

"Though the ether is filled with vibrations the world is in dark.
But one day man opens his seeing eye, and there is light."

Ludwig Wittgenstein

1.1 Problem formulation

Robots are computer-controlled devices capable of perceiving and manipulating the physical world [40]. They use sensors to acquire information on the environment and actuators to perform actions. Autonomous robots are machines which can accomplish tasks without direct human intervention [1].

One class of sensors which is commonly used in robotics, especially for tasks such as self localization and navigation [30],[16], is that of omnidirectional cameras i.e., cameras characterized by a very wide field of view [19]. Their main advantage over conventional, perspective cameras is that the first gather information from a lager portion of the space surrounding a robot. Cameras project light rays radiating from the environment onto the image sensor, a plane of CCD or CMOS cells, which in turn converts this information to electrical charges, forming the digital image. Depending on the way light is projected onto the image sensor, omnidirectional cameras are classified either as dioptric (those in which projection is performed solely by means of refraction) or catadioptric (those in which projection is performed by combined reflexion and refraction). One drawback which is common to omnidirectional cameras is that the images they produce are affected by strong distortion and perspective effects, which may lead to the use of non-standard algorithm for object detection and tracking. Another difficulty that affects tracking under these conditions is that the object of interest,

embedded in a dynamic scenario, may suffer from occlusion, overlap and ambiguities.

This thesis presents a 3D particle-filter [40, 13] (PF) tracker for autonomous robots equipped with omnidirectional cameras. The tracker is designed to work in a structured, color-coded environment such as the RoboCup Middle Size League (MSL). The particles of the filter represent 3D positions and velocities of the target, the likelihood function is based on the color and shape of the target. From one image frame to the next, the particles are moved according to an appropriate motion model. Then, for each particle, a likelihood is computed, in order to estimate the object state. To calculate the likelihood of a particle we first project the contour of the object it represents on the image plane (as a function of the object 3D shape and position) using a model for the omnidirectional vision system. The likelihood is then calculated as a function of three color histograms: one represents the object color model and is computed in a training phase with several examples taken from distinct locations and illumination conditions; the other two histograms represent the inner and outer boundaries of the projected contour, and are computed at every frame for all particles. The idea is to assign a high likelihood to the contours for which the inner pixels have a color similar to the object, and are sufficiently distinct from outside ones.

A work closely related to this is described in [32], although in that case the tracking is accomplished on the image plane. Tracking the position of an object in 3D space instead of on the image plane has two main advantages: (i) the motion model used by the tracker can be the actual motion model of the object, while in image tracking the motion model should describe movements of the projection of the object on the image plane and, because of the aforementioned distortion, a good model can be difficult to formulate and use; (ii) with 3D tracking the actual position of the tracked object is directly available, while a further non-trivial step is needed for a system based on an image tracker to provide it.

During the work for this thesis we developed both a Matlab and a C++ implementation of the tracker. The C++ implementation is based on vector and matrix operations provided by the IPP library by Intel. It was developed to assess the real-time capabilities of the method.

We ran several experiments, in a RoboCup MSL scenario, to assess the accuracy and precision of the proposed tracking method with color-coded objects. We tested the two implementations with either a dioptric or catadioptric omnidirectional camera and different projection models for the latter. We tracked a ball along movements in the 3D space surrounding the

robot, assessing the robustness of the method against occlusions and illumination changes. We tracked a robot maneuvering. We ran an experiment placing a still ball at different positions around the robot and measuring the error of the detected position with respect to the ground truth. We furthermore compared the results obtained in the tracking and detection, using two different models for the catadioptric sensor.

In future work we intend to extend the system with the ability to track multiple objects. We will introduce the use of multiple motion models, for example use a model describing the motion of a ball rolling freely and another describing the motion of a ball bouncing against an obstacle. We also intend to use physics-based motion models (e.g., applying gravitational acceleration to particles which model flying balls). We will express the target's coordinates in an inertial reference frame, instead of the robot-centered one we use now. This will be combined with ego-motion compensation: applying the inverse motion of the robot to the filter's particles. We also plan to test the tracker in a non-color-coded environment.

1.2 Description of the system

We designed and implemented a robust 3D tracking system for autonomous robots equipped with an omnidirectional, dioptric or catadioptric camera.

We use a particle filter to estimate at each time step the 3D position and velocity of a target, given the observations taken in the previous time steps and an initial probability density function (pdf). In this thesis we consider only targets which are rotationally symmetric, but the method can be easily extended to work with arbitrary rigid objects, including the estimate of the object's orientation.

PF's use sample-based representations of pdf's in which each sample is called a particle or an hypothesis. To iteratively compute the estimate on the target's position, one has to model probabilistically both its motion dynamics and the way we compute the likelihood of a particle.

For the motion dynamics we use a constant velocity autoregressive model: at each time step the velocity of the particles is updated by adding an acceleration disturbance to it. The position of the particles is updated, according to the velocity of the previous time step and the acceleration disturbance.

We designed an observation model to associate a likelihood to an hypothetical position, given the model of the tracked object, one image acquired with an omnidirectional camera and the corresponding, calibrated projection model for the camera.

The tracked objects are characterized by their 3D shape, size and color. We characterize the 3D shape of the objects with simplified geometric shapes and describe their color with a color histogram, which is built in a training phase with several examples taken from distinct locations and illumination conditions.

The steps to compute the likelihood of an hypothesis are:

- Projection of the target model onto the image plane, resulting in two pixel sets, one on each side of the contour projected by the hypothetical target.

  To obtain the two pixel sets, the object shape model is shifted and rotated to the hypothetical 3D position and projected onto the image (see Chapter 4 for details on the two alternative methods we use). The rotation of the model is needed because we use a simplified model for the shape, that allows us to projet only the 3D points which will result in the contour of the projection of the object.

  The projection of 3D points to 2D image points is done with a model for the omnidirectional camera in use. In this work we acquire images using a dioptric (fish-eye) or a catadioptric omnidirectional camera. We model the dioptric camera with the Equidistance Projection Model (EPM) and the catadioptric camera with either the Unified Projection Model (UPM) or the Perspective Projection Model (PPM). Each model has to be calibrated before being used.

- Building of two color histograms, characterizing the color in the image on the boundaries of the projected contour.

- Calculation of the likelihood as a function of the similarity between color histograms.

  We compute the similarity between couples of histograms, with the Bhattacharyya similarity metrics, as in [33]. The higher the similarity between the target model histogram and the histogram describing the color on the inside of the projected contour, the higher the likelihood. The lower the similarity between the two histogram describing the color on the two sides of the contour, the higher the likelihood.

To initialize the particles for the PF we either use a Gaussian distribution with large variance, centered at rough estimates of the object's initial position and velocity or select a position in 3D space, based on a 2D detection of the object, and use that as center for the Gaussian (see Chapter 5 for details).

1.3 Structure of the thesis

In Chapter 2 we describe the state of the art. In Chapter 3 we present our omnidirectional sensors and introduce the models we use for their projections. In Chapter 4 we delineate how the model-based detection of objects in omnidirectional images is performed. In Chapter 5 we detail our PF-based tracking algorithm. In Chapter 6 we present the experimental results obtained, while in Chapter 7 we draw conclusions and present future developments of this work.

Chapter 2

State of the art

"Our knowledge can be only finite,
while our ignorance must necessarily be infinite."

Karl Popper

This thesis addresses the object tracking problem in a semi-structured environment using an omnidirectional camera. The environment considered is the one of the RoboCup Middle Size League, where objects have particular shapes and colors but the illumination conditions can be diverse, and occlusions/ambiguities are frequent. In this chapter we will describe the current state of the art in the several topics this thesis addresses for achieving the proposed goals: (i) the image measurement methods; (ii) motion models and techniques for tracking; and (iii) 3D models based methods.

2.1 Color based image measurements

Since all elements in the RoboCup soccer field have distinct colors, most tracking algorithms in this domain use color based methods. Furthermore, since omnidirectional sensors highly deform the target image as a function of its position with respect to the robot, common approaches tend to privilege methods that relax rigidity constraints on the shape of the 2D targets. Thus, a large majority of methods consist in determining the similarity between the color histograms of the target and regions in the current image under analysis [17], [6] . The Camshift algorithm [4] was one of the first works demonstrating the use of color histograms for a practical real-time tracking system. The Camshift algorithm (Continuously Adaptive Mean Shift Algorithm) has 2 main steps: (i) assign to each pixel, in a search window around the current location, a likelihood of the pixel belonging to the target model

(histogram back-projection); (ii) then compute centroid of the likelihood image and update the search windows. The computation of the centroid and size of search window are done with the Mean-Shift algorithm [14], which is a non parametric approach to detect the mode of a probability distribution using a recursive procedure that converges to the closest stationary point.

Cheng [7] developed a more general version of mean shift for clustering and optimization. In [10], the mean-shift algorithm was used in conjunction with the Bhattacharyya coefficient measure to propose a real-time tracking system. The Bhattacharyya coefficient measure compares color histograms on the object model and image regions. Then this measure is minimized iteratively by using its spatial gradient, on a sequence of mean-shift iterations. Several extensions to these methods have been proposed recently, and try to either enrich the color models or introduce other features, e.g., edges and spatial relationships between object parts.

Kernel based image tracking [11] uses spatial masking to adapt the scale of the tracked region and include information from the background to improve tracking performance. In [9], different features from the color spaces of target and background are selected on-line in order to maximize the discrimination between target and background. Mixed approaches, incorporating color distribution and spatial information have been proposed in [3], [41]. Other approaches represent jointly color and edge information [8].

2.2 Tracking

The use of temporal information is of fundamental importance in the tracking problem. Objects move in space according to the laws of physics and motions cannot be arbitrary. By defining appropriate models for target motion it is possible to address the estimation problem under the occurrence of occlusions, manoeuvres and multiple-objects. Prior work concerning this issue is strongly related with the application of the Kalman filter [25], which is a linear method, assuming Gaussian noise in the motion and observation models. This method is based on two steps: i) a prediction step in which the state (target, position and velocity) is predicted based on the dynamics of the system (motion model); and ii) filtering step where the state is updated based on the observations collected from the image. However, the use of this technique is not robust in the presence of outliers, i.e., features that do not belong to the object of interest. An alternative solution is to use the Probabilistic Data Association - PDA [37], originally proposed in the context of target tracking from radar measurements. When several objects are supposed to be tracked alternative solutions should be conceived. In this

context, an extension of the Kalman filter is used to track distinct objects at the same time, this is known as Multiple-Hypothesis Tracking (MHT) [36]. In this method every single combination is considered among the observations. Unfortunately, the MHT algorithm is computationally exponential both in time and memory. An algorithm that alleviates this is presented in [12]. In [28] is presented a methodology which allows the association of one observation to a set of targets. The previous methods use linear dynamics to model the motion and Gaussian density functions for modelling either the noise in the dynamics and in the observations. However, these models cannot cope with situations where the Gaussian assumption is violated. To address this issue particle filtering has been widely use since the works presented in [22], [23]. This subject is still an active research topic; see for instance [24], [21], [31], [42], [27].

2.3 3D model based methods

Both histogram color models and motion tracking methods with omnidirectional sensors have been used previously in the context of RoboCup Middle Size League robots. A very interesting and recent method is presented in [32], which combines a metric based on the Bhattacharyya coefficient to compare color histograms on the target and background regions, together with a particle filtering method. However, not many works exploit properly the strong shape constraints existing in this scenario because tracking is usually performed in the image plane. For instance, balls and robots have very simple 3D shape models (spheres and cylinders) but when projected in the images these shapes are no longer constant.

We use 3D model-based tracking techniques (see [29] for a review) which improves the reliability of the detection step because the actual target shapes are enforced in the process. Additionally, the tracking motion models are also formulated in 3D, being more realistic with respect the physics laws and facilitating structure and parameter tuning. 3D model based tracking methods, as opposed to model free tracking, exploit knowledge on the object to be tracked, for example a 3D CAD model.

2D tracking follows the projection of a target in the image. Movements of the real object result in 2D transformations of its projection and are not easily modeled, especially in the case of omnidirectional images. The result of 2D tracking is an estimate of the object's position on the image. To estimate its 3D position a further, non-trivial step is needed. The objective of 3D tracking, instead, is to track the target along its 3D path. Classically, non-linear minimization approaches are used to register the 3D

object cues on the 2D images. In [34] a real-time system is shown, using the edges and texture of objects as measurements and as M-estimator in minimization process to improve the robustness of the method. In [35], image measurements are based on the edges and a particle filtering method is used instead of non-linear minimization. However, in some circumstances tracking edges and texture can be difficult e.g., because of motion blur, illumination changes, etc. In our case we will use the color of the target and its outside neighbourhood for the image measurements, in a way similar to [32], but using a 3D model based tracking paradigm.

Chapter 3

Omnidirectional vision

"All our knowledge has its origins in our perceptions."

Leonardo da Vinci

The shape of the projection of a 3D object on the image plane depends on (i) the projection function the vision sensor realizes and (ii) the relative position and orientation of the object w.r.t. the sensor. In this chapter we present the two vision sensors and the respective projection models which are used in this work. In particular we introduce our catadioptric omnidirectional camera and discuss three models for its projection function. Then we introduce our dioptric omnidirectional camera and its Equidistance Projection Model.

3.1 Catadioptric omnidirectional camera

Our catadioptric vision system, see Figure 3.1a, combines a camera looking upright to a convex mirror, having omnidirectional view in the azimuth direction [2]. The system is designed to have a wide-angle and a constant-resolution view of the ground plane [20, 15] and has only approximately constant-resolution at planes parallel to the reference one.

For objects off the ground floor, the constant resolution property does not hold anymore. If precise measurements are required, the generic Catadioptric Projection Model (CPM) should be employed. Because this exact model involves complex non-linear and non closed-form relationships between 3D points and their 2D projections, approximations are often used. A widely used approximation for catadioptric systems is the Unified Projection Model (UPM) pioneered by Geyer and Daniilidis [18]. The UPM models all omnidirectional cameras with a single center of projection and

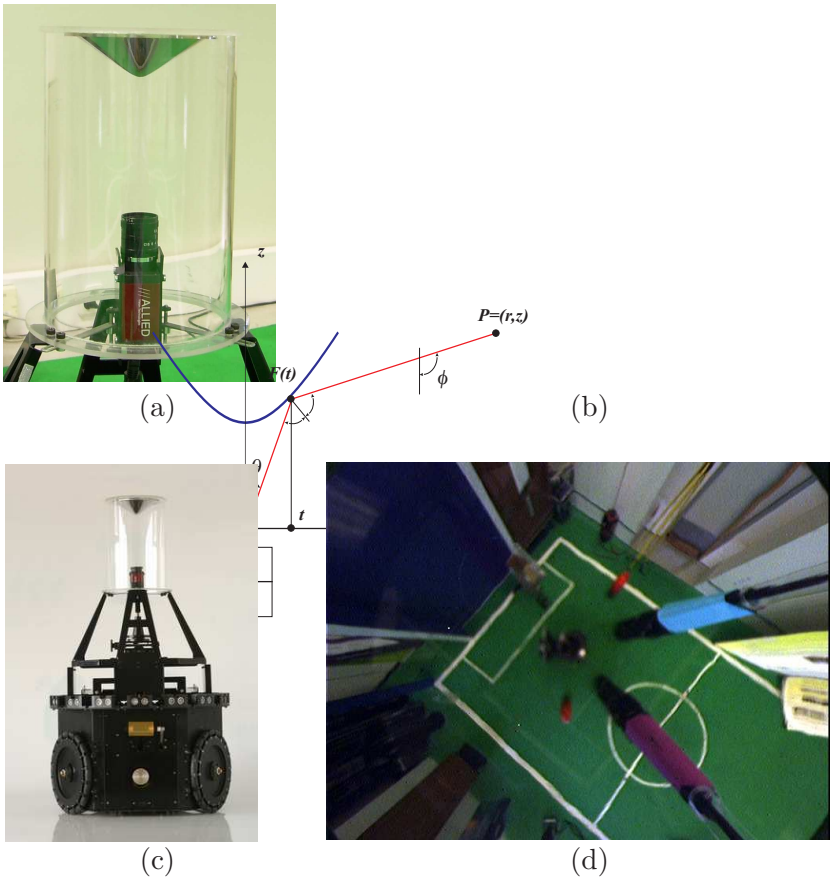


Figure 3.1: (a) Catadioptric omnidirectional camera. (b) The generic Catadioptric Projection Model. (c) The OmniISocRob robotic platform in the catadioptric configuration. (d) Sample image taken with the catadioptric omnidirectional camera in a RoboCup MSL scenario.

provides a good approximation to wide-angle constant resolution sensors. We show, however, that a simple Perspective Projection Model (PPM) is as good as the UPM, with additional advantages of simplicity and computational efficiency.

3.1.1 Projection models

Let $m = \mathcal{P}_0(M; \vartheta_0)$ represent the projection of a 3D point in cylindrical coordinates, $M = [r\ \varphi\ z]^T$ to 2D polar coordinates on the image plane, $m = [\rho\ \varphi_0]^T$, with ϑ_0 containing the system parameters. Considering cameras with axial symmetry and aligning the coordinate systems such that $\varphi \equiv \varphi_0$, one obtains the radial model (see Figure 3.1b):

$$\rho = \mathcal{P}\left([r\ z]^T; \vartheta\right). \tag{3.1}$$

In the case of the constant resolution design, $\mathcal{P}$ is trivial for the ground-plane, as it is just a scale factor between pixels and meters. Deriving $\mathcal{P}$ for the complete 3D field of view involves using the actual mirror shape, F which is a function of the radial coordinate t [15]. Based on first order optics, and in particular on the reflection law at the specular surface of revolution, (t, F), the following equation is obtained:

$$\text{atan}(\rho) + 2 \cdot \text{atan}(F') = -\frac{r-t}{z-F} \tag{3.2}$$

where $\phi = -(r-t)/(z-F)$ is the system's vertical view angle, $\theta = \text{atan}(\rho)$ is the camera's vertical view angle, and F' represents the slope of the mirror shape. When F denotes an arbitrary function and we replace $\rho = t/F$, Equation 3.2 becomes a differential equation, expressing the constant horizontal resolution property, $\rho = a \cdot r + b$, for one plane $z = z_0$. F is usually found as a numerical solution of the differential equation (see details and more designs in [20, 15]).

If F is a known shape then Equation 3.2 describes a generic CPM, as it forms an equation on ρ for a given 3D point (r, z). In general ρ has a non-closed-form solution.

Here we assume that the system approximates a single projection center system, considering that the mirror size is small when compared to the distances to the imaged-objects. Hence, we can use a standard model for catadioptric omnidirectional cameras, namely the Unified Projection Model (UPM) pioneered by Geyer and Daniilidis [18], that can represent all omnidirectional cameras with a single center of projection. It is simpler than the model which takes into account the actual shape of the mirror and gives good enough approximations for our purposes.

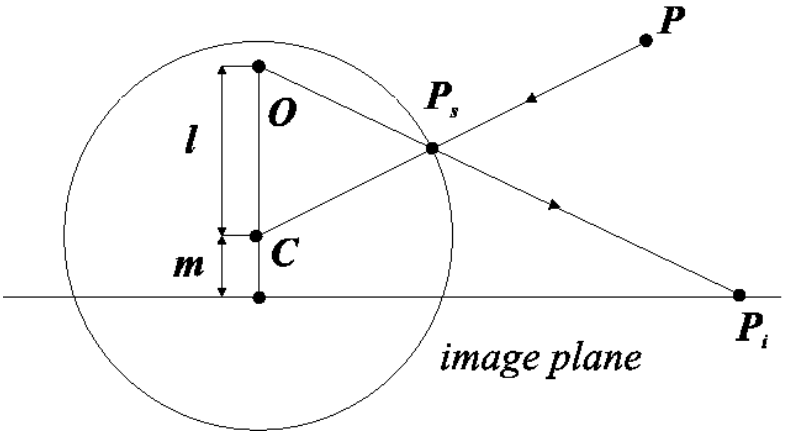


Figure 3.2: The Unified Projection Model.

The UPM consists of a two-step mapping via a unit-radius sphere [18]: (i) project a 3D world point, $P = [r\ \varphi\ z]^T$ to a point P_s on the sphere surface, such that the projection is normal to the sphere surface; (ii) project to a point on the image plane, $P_i = [\rho\ \varphi]^T$ from a point, O on the vertical axis of the sphere, through the point P_s. This mapping is graphically illustrated in Figure 3.2 and is mathematically defined by:

$$\rho = \frac{l+m}{l\sqrt{r^2+z^2}-z} \cdot r \tag{3.3}$$

where the (l, m) parameters describe the type of camera.

The PPM is a particular case of the UPM, obtained with $l = 0$ and by defining $k = -m$:

$$\rho = k \cdot r/z. \tag{3.4}$$

Equation 3.4 shows that the PPM has constant resolution, i.e., linear relationship between ρ and r, at all z-planes.

Both the PPM and the UPM model are parametric. To calibrate each one of them we use a set of known non-coplanar 3D points $[X_i\ Y_i\ Z_i]^T$ and measure their images $[u_i\ v_i]^T$. Then, we minimize the mean squared error between the measurements and the projection obtained with the parametric model, see Figure 3.3.

3.2 Dioptric omnidirectional camera

Our dioptric vision system, see Figure 3.4a, is a camera coupled with a fish-eye lens. It has an omnidirectional view in the azimuth direction, with

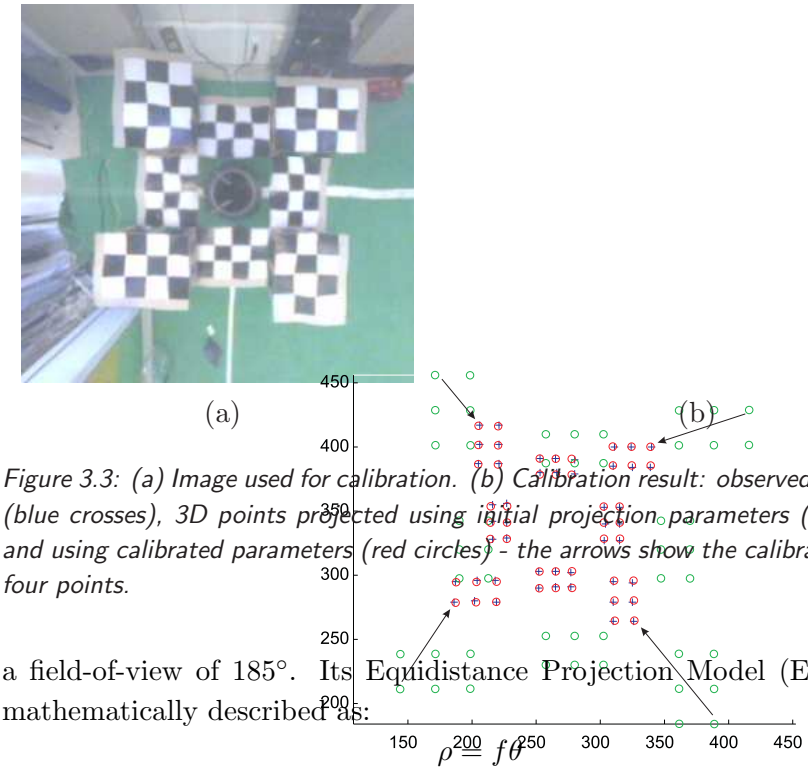
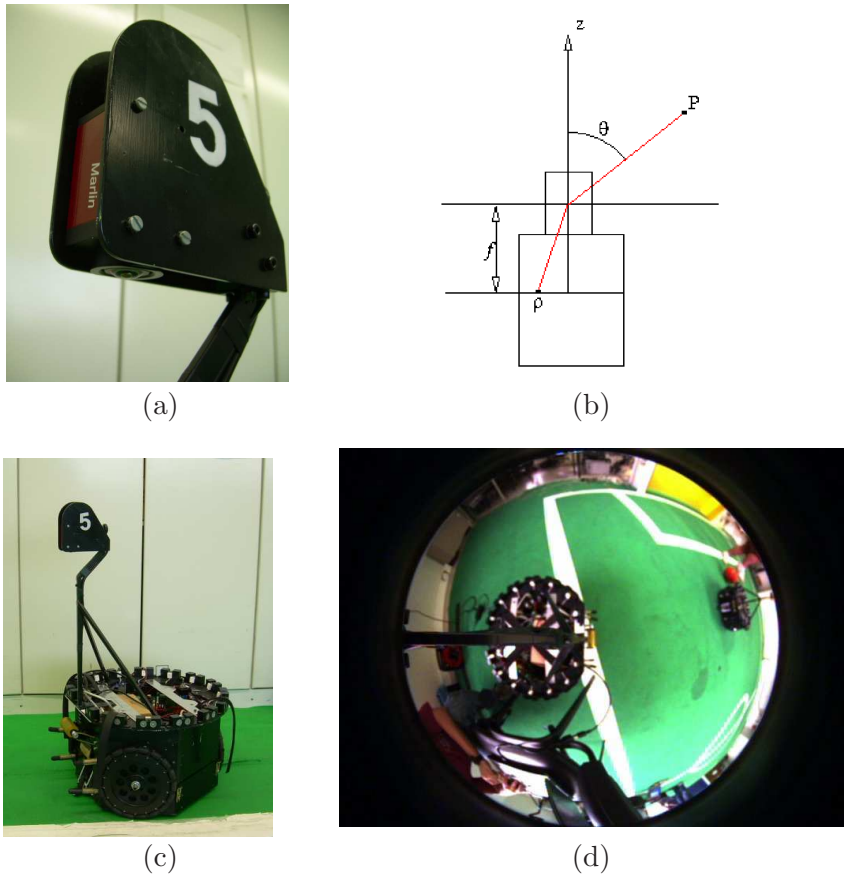


Figure 3.3: (a) Image used for calibration. (b) Calibration result: observed image points (blue crosses), 3D points projected using initial projection parameters (green circles) and using calibrated parameters (red circles) - the arrows show the calibration effect at four points.

a field-of-view of 185°. Its Equidistance Projection Model (EPM) [26] is mathematically described as:

$$\rho = f\theta \tag{3.5}$$

We calibrate the EPM minimizing the mean squared error between the the projection of 3D points performed by the actual vision system and by the model (see Figure 3.5), as we do for the other projection models.



(a) (b)

(c) (d)

Figure 3.4: (a) Dioptric omnidirectional camera. (b) The Equidistance Projection Model. (c) The OmniISocRob robotic platform in the dioptric configuration. (d) Sample image acquired with the dioptric omnidirectional camera.

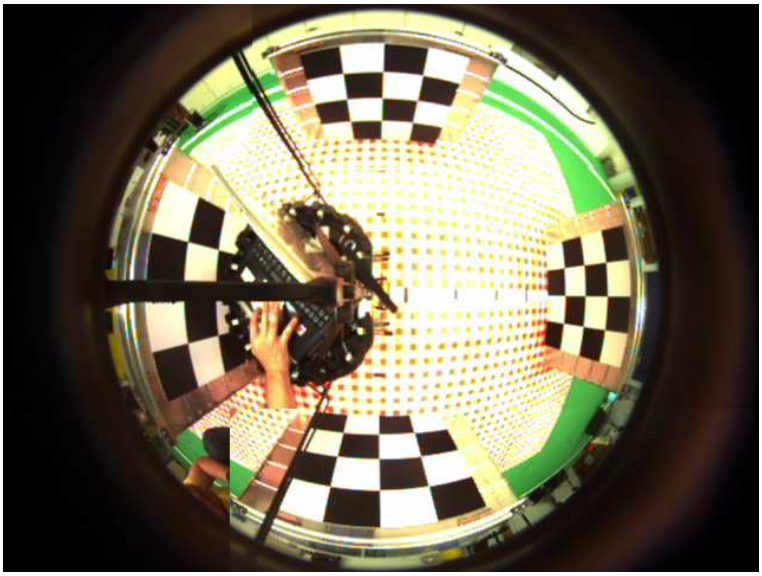


Figure 3.5: One of the images used for calibrating the EPM model. The checkers placed at known position identify known 3D points, which are projected by the camera onto the image.

Chapter 4

Model-based detection in omnidirectional images

"The usual approach of science of constructing a mathematical model cannot answer the questions of why there should be a universe for the model to describe. Why does the universe go to all the bother of existing?"

Stephen Hawking.

In this chapter we introduce our object model, describing the 3D shape model and the color model. We will complete the object model in Chapter 5 adding a motion model. Furthermore, we detail how we compute the likelihood of an object state, given the object model, the projection model for the omnidirectional camera and one omnidirectional image.

4.1 Detection algorithm

The detection algorithm is a fundamental part of the PF approach, it associates a likelihood value to the hypothetical 3D position of an object, given the model of such object, one image acquired with an omnidirectional camera and the corresponding, calibrated projection model for the camera, as described in Chapter 3.

The tracked objects are characterized by their 3D shape, size and color, thus we use these elements to calculate the likelihood of a hypothesis. We characterize the 3D shape of the objects with simplified geometric shapes (e.g. the 3D shape of a ball is well modeled by a sphere, that of a robot is approximated by a cylinder, etc.). We describe the color of an object with a color histogram. We simplify the problem by assuming that the objects to track are rotationally symmetric w.r.t. the axis around which they are able

to rotate. The RoboCup MSL robots satisfy this requirement as they can only rotate around their vertical axes and by means of this rotation their projections onto the image plane remain constant enough for our needs.

The detection algorithm receives as input the model for the object to track, one camera image, the appropriate model for the camera and one hypothetical position, relative to the omnidirectional camera projection center. It produces one likelihood value as output. The first step to calculate such a likelihood is to determine what shape an object of the specified class would project onto the image, when standing at the hypothesized position. The second step is to measure the color on the inside and outside boundaries of the projected, hypothetical contour. This is done by building two color histograms from two selected pixel sets. We measure the color on the inside and on the outside of the contour because we expect that, if the hypothetical position is correct, the color on the inner boundary will be similar to that of the model, while the color on the outer boundary will be different from that. The third step consists in calculating the likelihood of the hypothesis, based on similarities between color histograms. The higher the similarity between the model histogram and the histogram describing the color on the inside of the projected contour, the higher the likelihood. The lower the similarity between the two histogram describing the color on the two sides of the contour, the higher the likelihood.

The first of these two rules in practical terms means that we associate a high likelihood to a contour which has the color of the tracked object on its inside. The second one means that we associate a high likelihood to contours in the whereabouts of which a color transition occurs. This method, compared to other methods which rely on edge-detection, is more robust as it can deal with blurred edges, which are frequent in dynamic scenarios [32] (see Figure 4.1).

4.2 3D model

We use the 3D model of an object to calculate the 2D contour the object would project on an image, when at a hypothetical position, relative to the vision sensor. The computation of the 2D contour projected in an omnidirectional image by a generic 3D shape is not an easy task. First one has to determine the visibility boundary of the 3D shape. Then one has to project this boundary onto the image plane, going through the non linear projection model. For simple projection models and shapes it is sometimes possible to represent 2D contours by simple parametric curves (e.g. the projection of a straight line in a central camera is a conic), but in the general case there is

Figure 4.1: An example of motion blur: the contours of the orange ball are blurred due to its rapid motion.

no closed form solution. Another alternative is to approximate the 3D shape as a set of vertices and edges, project such elements onto the image plane using the appropriate projection model, and eventually use some method to determine the outer contour of the projected figure.

In this work we deal with balls and robots as objects to track, so we should use a polygonal model of a sphere (see Figure 4.2) and a polygonal model of a cylinder as object 3D models. Given the simple nature of the 3D shapes and taking advantage of their rotational symmetry, we are able to select a set of contour point directly in 3D, and then project them to form a sampled representation of the 2D hypothetical contour. For a ball at a certain relative position, the 3D contour points lie on the intersection between its spherical surface and the plane orthogonal to the line connecting the virtual projection center to the center of the sphere, see Figure 4.3. We obtain the set of 3D contour points by rotating and shifting (in accord to the position of the ball) a set of initial 3D points, equally distributed along a circle with the same radius as that of the ball. The same method, with a different initial distribution of the points, is used for the robot, see Figure 4.4.

The objective for the use of the 3D model is to define two sets of pixels for every hypothesis, one characterizing the color on the inner boundary of the projected contour and the other characterizing the color on the outer

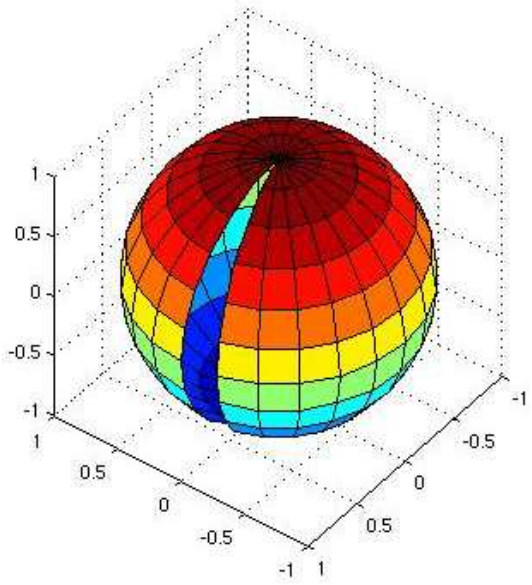


Figure 4.2: Polygonal model of a sphere (some of the faces have been removed for visualization).

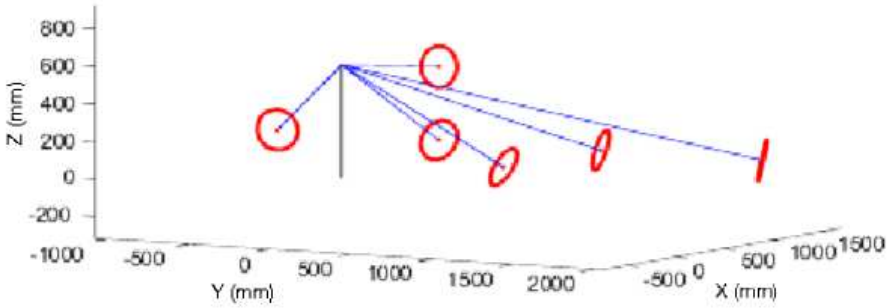


Figure 4.3: Plot of the 3D points projected to obtain the 2D contour points for balls at different positions. The black line connects the projection center to the ground plane.

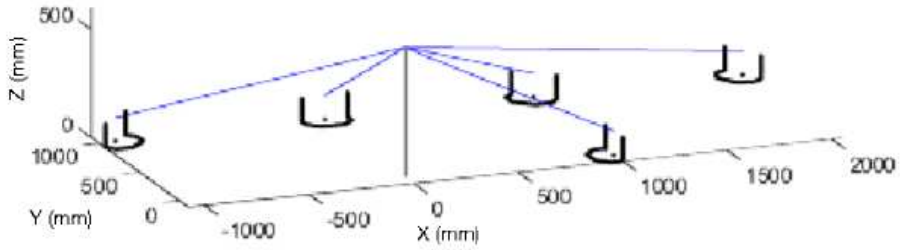


Figure 4.4: Plot of the 3D points projected to obtain the 2D contour points for robots at different positions. Again, the black line connects the projection center to the ground plane.

boundary. For this purpose, we use one of two methods: one based on simple sampling and the other one based on B-splines.

4.2.1 Selection of boundary pixels using sampling

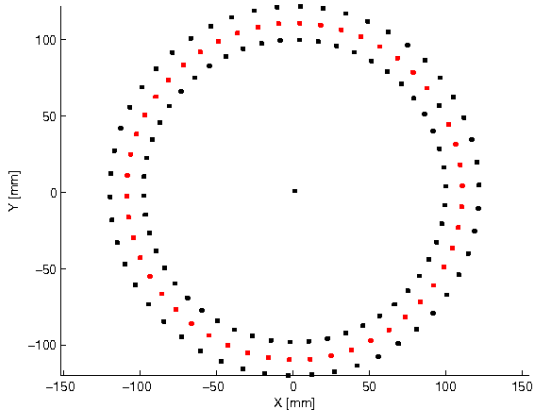
In this method we define the inner and outer boundary pixel sets by projecting two different 3D contours, one larger than the actual size of the tracked object, for the outside pixels, and one smaller than that for the inside pixels, see Figure 4.5a. Varying the size difference between tracked object and inner and outer 3D contours enables the user of the tracking system to set the balance between precision and processing time: the smaller the size difference, the higher the precision of the estimate, but the higher the number of particles needed and so the computation time. In our experiments of tracking the ball, which has a radius of 110mm, the 3D points were placed along circumferences of 99mm and 121mm respectively (a 10% difference from the actual value). This method is fast and has the advantage of being based on 3D distances, rather than on image distances. This means that the distance between projected inner and outer contour adapts to the size of the projected object, but stays the same in real world coordinates, see Figure 4.5b and 4.5c.

This method has one important drawback: whenever the shape projected by an object becomes too small, the inner and outer contours of an object superimpose and the tracking method fails. One solution to this would be to check the degree of superimposion after the projection of an hypothesis' contours. A good measure for this would be the relative number of inner boundary pixels which are also outer boundary pixels. When the superimposion would exceed a given threshold, new inner and outer pixels sets should be built, for example selecting pixels standing along normals to the hypothesized object contour, at a two pixels distance from it.

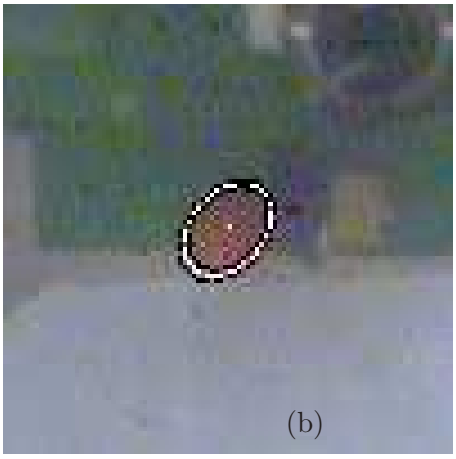
One interesting characteristic of this approach is that the number of points used to build the color histograms can be varied: increasing such number enhances the robustness of the tracker, while decreasing it makes the tracker faster, diminishing the number of projections to be made, together with the operations performed to build each color histogram.

4.2.2 Selection of boundary pixels using B-splines

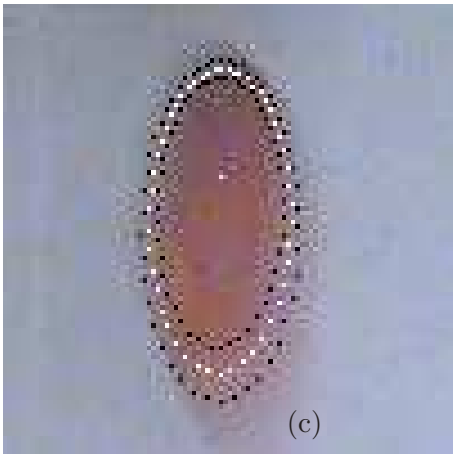
In this method we project the sample-based hypothetical contour of an hypothesis onto the image plane, model it with a B-spline and select inner and outer boundary pixels along normals to the spline.



(a)



(b)



(c)

Figure 4.5: (a) 3D points modeling the hypothetical ball contour (red) and 3D points used to select inner and outer boundary pixel sets (black). (b) and (c) Examples of the pixel selection resulting from projecting the 3D model onto an image (after rotation and translation). White pixels represent the projected contour, black pixels are the ones selected to represent the color in the contour inner and outer boundaries.

A B-spline has the following representation:

$$f(t) = \sum_{i=0}^{k-m-1} c_i \mathcal{B}_i^m(t), \quad t \in [t_m, t_{k-m}] \tag{4.1}$$

where $\{\mathcal{B}_k^m(t), k = 0, \ldots, k-m-1\}$ is the set of basis functions such that $\mathcal{B}_i^m(t) \geq 0$, and $\sum_i \mathcal{B}_i^m(t) = 1$; $\{c_0, c_1, \ldots, c_{k-m-1}\}$ is the set of coefficients and $[t_{i-1}, t_i]$ is an interval in which the spline functions are polynomial and exhibit a certain degree of continuity at the knots.

Planar curves are simply the $\mathbb{R}^2$ version of Equation 4.1:

$$\mathbf{v}(t) \equiv [x(t) \;\; y(t)] = \sum_{i=0}^{k-m-1} c_i \mathcal{B}_i^m(t) \tag{4.2}$$

A discretized spline is a set of N equispaced samples of $\mathbf{v}(t)$ collected as the $N \times 2$ vector:

$$\mathbf{v} = [\mathbf{v}_0^T, \ldots, \mathbf{v}_{N-1}^T] = [\mathbf{x} \;\; \mathbf{y}], \quad N > k \tag{4.3}$$

If we arrange the coordinates of control points into a parameter $\boldsymbol{\theta}_{(k)}$:

$$\boldsymbol{\theta}_{(k)} = [c_0^T, \ldots, c_{k-1}^T]^T = [\boldsymbol{\theta}_{(k)}^x \;\; \boldsymbol{\theta}_{(k)}^y]^1, \tag{4.4}$$

the discretized closed spline $\mathbf{v}$ can be obtained by the matrix product

$$\mathbf{v} = \mathbf{B}_{(k)}\boldsymbol{\theta}_{(k)} \Leftrightarrow \{\mathbf{x} = \mathbf{B}_{(k)}\boldsymbol{\theta}_{(k)}^x; \;\; \mathbf{y} = \mathbf{B}_{(k)}\boldsymbol{\theta}_{(k)}^y\}, \tag{4.5}$$

where the elements of $\mathbf{B}_{(k)}$ are $[\mathbf{B}_{(k)}]_{ij} = \mathcal{B}_j(t_0 + \frac{(t_k - t_0)i}{N})$.

In computer graphics $m = 3$ or 4 is generally found to be sufficient. Herein, we use quadratic B-splines, i.e., $m = 3$.

After obtaining the 2D points $\{\mathbf{d}_{\text{boundary}}^n\}, n = 1, \ldots, N$, we convert them into a B-spline which best fits to this set. Assuming that we known the knots, the $N \times k$ matrix $\mathbf{B}_{(k)}$ can be computed. Thus, given the vector $\mathbf{d}$ with N data points and a choice of k, a $N \times k$ matrix $\mathbf{B}_{(k)}$ can be built and its pseudo-inverse $\mathbf{B}_{(k)}^\dagger$ computed. The estimated control points are given by $\widehat{\boldsymbol{\theta}}_{(k)} = \mathbf{B}_{(k)}^\dagger \mathbf{d}$, with $\mathbf{B}_{(k)}^\dagger = (\mathbf{B}_{(k)}^T \mathbf{B}_{(k)})^{-1} \mathbf{B}_{(k)}^T$ and the curve is given by $\mathbf{v} = [\mathbf{B}_{(k)}^\dagger \mathbf{x} \;\; \mathbf{B}_{(k)}^\dagger \mathbf{y}] = \mathbf{B}_{(k)}\widehat{\boldsymbol{\theta}}_{(k)}$. The regions in which the histogram is computed are defined at the points of the normal lines radiating from the discrete B-spline curve $\mathbf{v}$, i.e.

$$\mathbf{H} = \bigcup_{i=1}^{N} (\pm\Delta)\mathbf{v}(s_i)\, \bar{\mathbf{n}}(s_i) \tag{4.6}$$

[1] (k) is the total of number of control points

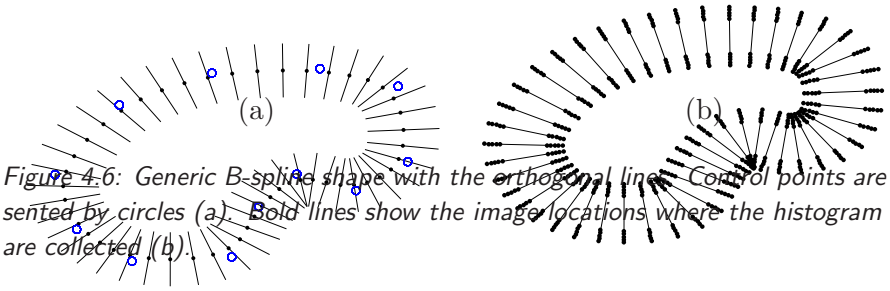


Figure 4.6: Generic B-spline shape with the orthogonal lines. Control points are represented by circles (a). Bold lines show the image locations where the histogram values are collected (b).

where the sign $+$, and $-$ distinguishes whether the inspection is performed inside ($\mathbf{H}^{\text{inner}}$) or outside ($\mathbf{H}^{\text{outer}}$) the reference contour respectively; $\bar{\mathbf{n}}(s_i)$ is the normal vector at the point s_i; $\Delta \in [0, 1]$, Figure 4.6 depicts the technique proposed herein. A quadratic B-spline of a generic shape and the orthogonal lines are shown in Figure 4.6a. Figure 4.6b displays the lines at which the histogram values are collected. The pixels selected to evaluate one hypothesis (one hypothesis shown for each image) are depicted in Figure 4.7. In some cases, i.e., when using cameras displaying color mosaicking errors at image edges, it may be convenient to avoid sampling at the middle of the line segment. In our case, however, color information at the contour pixels is quite acceptable and one can perform the inspection along the entire line segment.

4.3 Color model

We use color histograms to model the color of the object to track and to characterize the color of the two image regions surrounding an hypothetical contour.

We use the HSI (Hue, Saturation, Intensity) color space in the Matlab implementation of the tracker. We choose the HSI color space because the Hue component separates well colors of pixels belonging to the ball from colors of pixels belonging to the rest of the RoboCup environment, see Figure 4.8. We calculate the H and S values of each pixel as for the HSV color space, but prefer Intensity, I=(R+G+B)/3 to Value, V=max(R,G,B), as Intensity exploits all the available RGB information, while V discards the values of the two non-maximum RGB channels. We set the number of bins

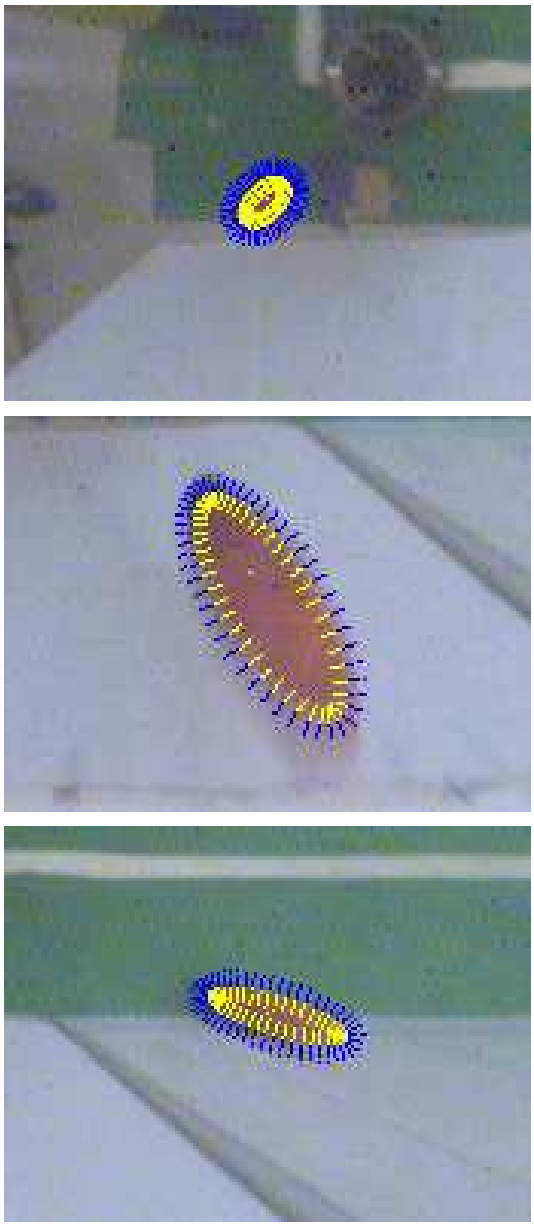


Figure 4.7: Close-up showing the pixels used to evaluate one hypothesis (one for each image), selected using B-splines. Yellow and blue pixels are used to build the inner and outer color histogram, respectively.

for the HSI histogram to $B_h = 12$, $B_s = 12$, $B_i = 4$. We set the number of bins for the Intensity channel to a smaller value than the other two in order to achieve some robustness w.r.t. illumination intensity changes. Ignoring the Intensity channel altogether would have prevented us from distinguishing white and black object from the others.

We use the YUV color space in the C++ real-time implementation of the tracker because that is the color space in which the camera provides the images. This choice allows us to save processing time, as we do not perform a color-space transformation, and maintains the desirable characteristic of a color representation in which the brightness channel (Y) is orthogonal to the others. We set the number of bins for the YUV histogram to $B_y = 4$, $B_u = 8$, $B_v = 8$. We set the number of bins for the V channel to a smaller value than the other two, in order to achieve robustness w.r.t. illumination changes, as in the previous case.

A color histogram is a non parametric statistical description of a color distribution, in terms of occurrences of different color classes. Let us denote

$$b_t(\mathbf{d}) \in \{1, \ldots, B\}, \tag{4.7}$$

the bin index associated with the color vector at pixel location $\mathbf{d}$ and frame t. Then the histogram of the color distribution of a generic set of points can be computed by a kernel density estimate

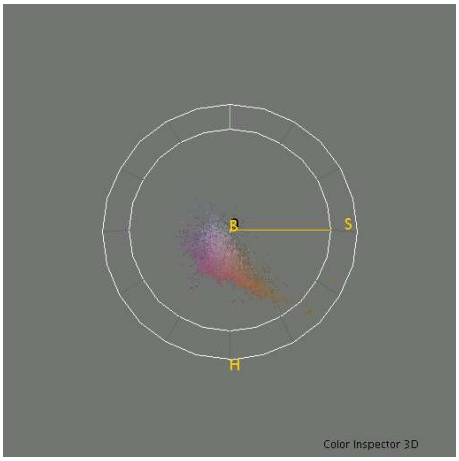
$$\mathbf{H} \doteq \{h(b)\}_{b=1,\ldots,B} \tag{4.8}$$

of the color distribution at frame t, where each histogram bin is given as in [10]

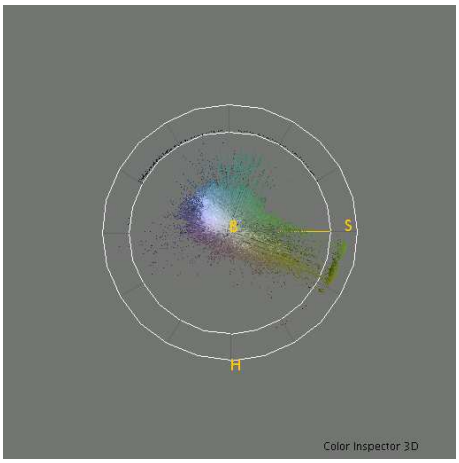
$$h(b) = \beta \sum_n \delta[b_t(\mathbf{d}^n) - b] \tag{4.9}$$

where δ is the Kronecker delta function, β is a normalization constant which ensures h to be a probability distribution $\sum_{b=1}^{B} h(b) = 1$. We encode each color histogram as a $12 \times 12 \times 4$ or a $4 \times 8 \times 8$ matrix in the HSI and YUV case respectively.

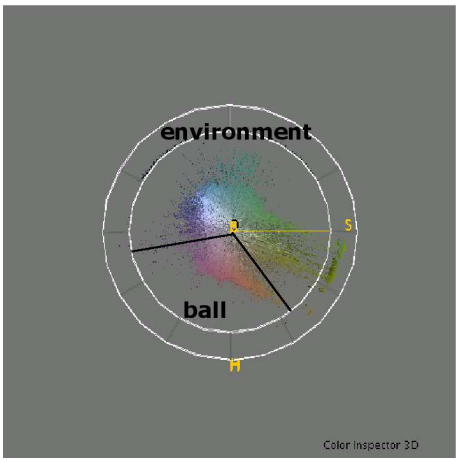
The color model for each object was built collecting a set of images in which the object is present, and calculating the color histogram on the (hand labeled) pixels belonging to the specific object. All object pixels were used to train the color histograms, whereas in run-time, only a subset of points is used.



(a)



(b)



(c)

Figure 4.8: (a) Projection in the HSB(=HSV) space of pixels belonging to the ball, taken from different images. (b) Projection in the HSB space of pixels belonging to an image depicting a typical RoboCup environment. (c) Setting two thresholds on the Hue would provide a reasonable separation between the two classes.

4.4 Likelihood

We measure the similarity between color histograms using the Bhattacharyya similarity metrics, as in [33]:

$$\mathcal{S}\left(\mathbf{H}^1, \mathbf{H}^2\right) = \sum_{b=1}^{B} \sqrt{h^1(b) \cdot h^2(b)} \tag{4.10}$$

We adopt a distance metric inspired in [32] to assess how far an observation is from the perfect one. Two quantities are taken into account: (i) distance between the object color model and the color measures inside the contour and; (ii) similarity between regions inside and outside the contour.

Defining $\mathbf{H}^{\text{model}}$, $\mathbf{H}^{\text{inner}}$ and $\mathbf{H}^{\text{outer}}$ as a reference (object) color model, the inner boundary points and the outer boundary points histogram, respectively, we will measure their pairwise similarities using (4.10). The distance metric should be high when candidate color histograms are different to the reference histogram and similar to the background. This can be expressed by the following quantity:

$$\mathcal{D} = \frac{\left(1 - \mathcal{S}(\mathbf{H}^{\text{model}}, \mathbf{H}^{\text{inner}})\right) + \kappa \mathcal{S}(\mathbf{H}^{\text{outer}}, \mathbf{H}^{\text{inner}})}{\kappa + 1} \tag{4.11}$$

This allows us to take into account the object-to-model mismatch (first term) and the object-to-background similarity (second term), see Figure 4.9 for a graphical example. The parameter κ allows to balance the two terms and was tuned manually for good performance: $\kappa = 1.5$.

The data likelihood function $\mathcal{L}$ is modeled as a Laplacian distribution over the distance metric: $p(\mathbf{y}_t \mid \mathbf{x}_t^{(i)}) \propto e^{-\frac{|\mathcal{D}|}{b}}$ In our experiments we set $b = 1/30$.

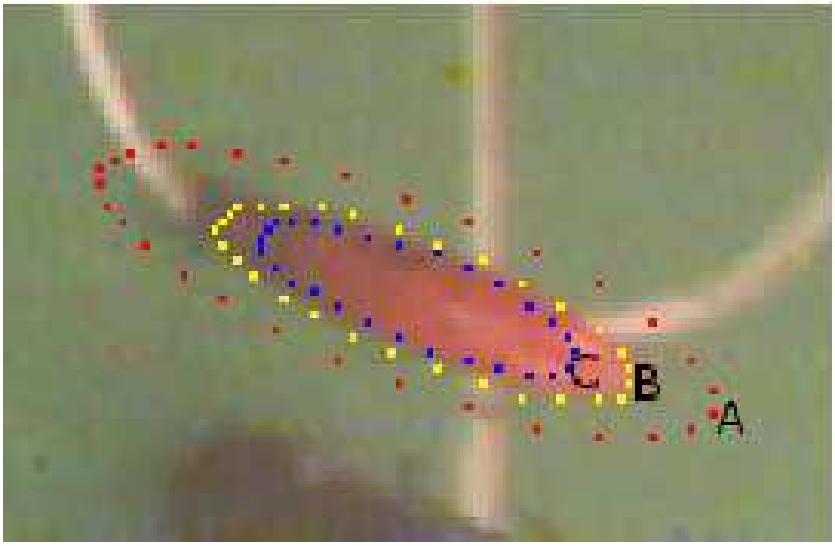


Figure 4.9: Three hypothetical contours projected on one image. Contour A has a low likelihood, because of high object-to-model mismatch (the inner boundary of the contour is not orange) and high object-to-background similarity (the colors of inner and outer boundary of the contour are pretty similar). Contour C is associated with a medium likelihood, because of low object-to-model mismatch (the inner boundary of the contour is orange), but high object-to-background similarity (also the outer boundary of the contour is quite orange). Contour B has a high likelihood because of low object-to-model mismatch and low object-to-background similarity.

Chapter 5

Tracking with particle filters

"If I could remember the names of all these particles, I'd be a botanist.[1]*."*

Enrico Fermi

In this chapter we describe the methods employed for 3D target tracking with particle filters.

5.1 Particle filter tracking.

We are interested in computing, at each time $t \in \mathbb{N}$, an estimate of the 3D pose of a target. We represent this information as a "state-vector" defined by a random variable $\mathbf{x}_t \in \mathbb{R}^{n_\mathbf{X}}$ whose distribution in unknown (non-Gaussian); $n_\mathbf{X}$ is the dimension of the state vector. In the present work we are mostly interested in tracking balls and cylindrical robots, whose orientation is not important for tracking. However, the formulation is general and can easily incorporate other dimensions in the state-vector, e.g., target orientation and spin.

Let $\mathbf{x}_t = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T$, with (x,y,z), $(\dot{x},\dot{y},\dot{z})$ the 3D cartesian position and linear velocities in a robot centered coordinate system. The state sequence $\{\mathbf{x}_t; t \in \mathbb{N}\}$ represents the state evolution along time and is assumed to be a Markov process with some initial distribution $p(\mathbf{x}_0)$ and a transition distribution $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$.

The observations taken from the images are represented by the random variable $\{\mathbf{y}_t; t \in \mathbb{N}\}$, $\mathbf{y}_t \in \mathbb{R}^{n_\mathbf{Y}}$, and are assumed to be conditionally independent given the process $\{\mathbf{x}_t; t \in \mathbb{N}\}$ with marginal distribution $p(\mathbf{y}_t \mid \mathbf{x}_t)$, where $n_\mathbf{Y}$ is the dimension of the observation vector.

[1] In this occasion Fermi was referring to subatomic particles, but he is often credited with the development of the Monte Carlo method.

In a statistical setting, the problem is posed as the estimation of the *posteriori* distribution of the state given all observations $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$. Under the Markov assumption, we have:

$$p(\mathbf{x}_t \mid \mathbf{y}_{1:t}) \propto p(\mathbf{y}_t \mid \mathbf{x}_t) \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1})\, p(\mathbf{x}_{t-1} \mid \mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1} \tag{5.1}$$

The previous expression tells us that the *posteriori* distribution can be computed recursively, using the previous estimate, $p(\mathbf{x}_{t-1} \mid \mathbf{y}_{1:t-1})$, the motion-model, $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$ and the observation model, $p(\mathbf{y}_t \mid \mathbf{x}_t)$.

To address this problem we use particle filtering methods. Particle filtering is a Bayesian method in which the probability distribution of an unknown state is represented by a set of M weighted particles $\{\mathbf{x}_t^{(i)}, w_t^{(i)}\}_{i=1}^{M}$ [13]:

$$p(\mathbf{x}_t \mid \mathbf{y}_{1:t}) \approx \sum_{i=1}^{M} w_t^{(i)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) \tag{5.2}$$

where $\delta(\cdot)$ is the dirac delta function. Based on the discrete approximation of $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$, different estimates of the best state at time t are possible to be devised. For instance we may use the Monte Carlo approximation of the expectation:

$$\hat{\mathbf{x}} \doteq \frac{1}{M} \sum_{i=1}^{M} w_t^{(i)} \mathbf{x}_t^{(i)} \approx \mathbb{E}(\mathbf{x}_t \mid \mathbf{y}_{1:t}), \tag{5.3}$$

or the maximum likelihood estimate:

$$\hat{\mathbf{x}}_{ML} \doteq \text{argmax}_{\mathbf{x}_t} \sum_{i=1}^{M} w_t^{(i)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) \tag{5.4}$$

To compute the approximation to the *posteriori distribution*, a typical tracking algorithm works cyclically in three stages:

1. *Prediction* - computes an approximation of $p(\mathbf{x}_t \mid \mathbf{y}_{1:t-1})$, by moving each particle according to the motion model;

2. *Update* - each particle's weight i is updated using its likelihood $p(\mathbf{y}_t \mid \mathbf{x}_t^{(i)})$:
$$w_t^{(i)} \propto w_{t-1}^{(i)} p(\mathbf{y}_t \mid \mathbf{x}_t^{(i)}) \tag{5.5}$$

3. *Resampling* - the particles with a high weight are replicated and the ones with a low weight are forgotten.

For this purpose, we need to model probabilistically both the motion dynamics, $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$, and the computation of each particle's likelihood $p(\mathbf{y}_t \mid \mathbf{x}_t^{(i)})$.

5.1.1 Motion model

In the system proposed herein we assume motion dynamics follow a standard autoregressive dynamic model:

$$\mathbf{x}_t = A\mathbf{x}_{t-1} + \mathbf{w}_t, \tag{5.6}$$

where $\mathbf{w}_t \sim \mathcal{N}(0, Q)$. The matrices A, Q, could be learned from a set of representative correct tracks, obtained previously (e.g., see [5]), however, we choose pre-defined values for these two matrices. Since the coordinates in the model are real-world coordinates, the motion model for a tracked object can be chosen in a principled way, both by using realistic models (constant velocity, constant acceleration, etc.) and by defining the covariance of the noise terms in intuitive metric units.

We use a constant velocity model, in which the motion equations correspond to a uniform acceleration during one sample time:

$$x_t = Ax_{t-1} + Ba_{t-1}, \quad A = \begin{bmatrix} \mathbf{I} & (\Delta t)\mathbf{I} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad B = \begin{bmatrix} (\frac{\Delta t^2}{2})\mathbf{I} \\ (\Delta t)\mathbf{I} \end{bmatrix} \tag{5.7}$$

where I is the 3×3 identity matrix and a_t is a 3×1 white zero mean random vector corresponding to an acceleration disturbance. We have set $\Delta t = 1$ for all the experiments, whereas the covariance matrix of the random acceleration vector was fixed at:

$$cov(a_t) = \sigma^2\mathbf{I}, \quad \sigma = 90\text{mm/frame}^2 \tag{5.8}$$

5.1.2 Observation model

The observation model allows us to compute the likelihood that a particular hypothesis (state vector) generated the observation (image) we have. We compute the likelihood of an object state, given the object model, the projection model for the omnidirectional camera and one omnidirectional image, as described in Chapter 4.

5.2 Initialization of the particle filter

The PF needs an initial distribution of particles to start from. In the case of the dioptric camera and the ball, we use a method designed to work coupled with a detection module (possibly based on color segmentation): it receives as input an image and a pair of image coordinates, identifying the center of the projected ball. These coordinates are now provided by a user,

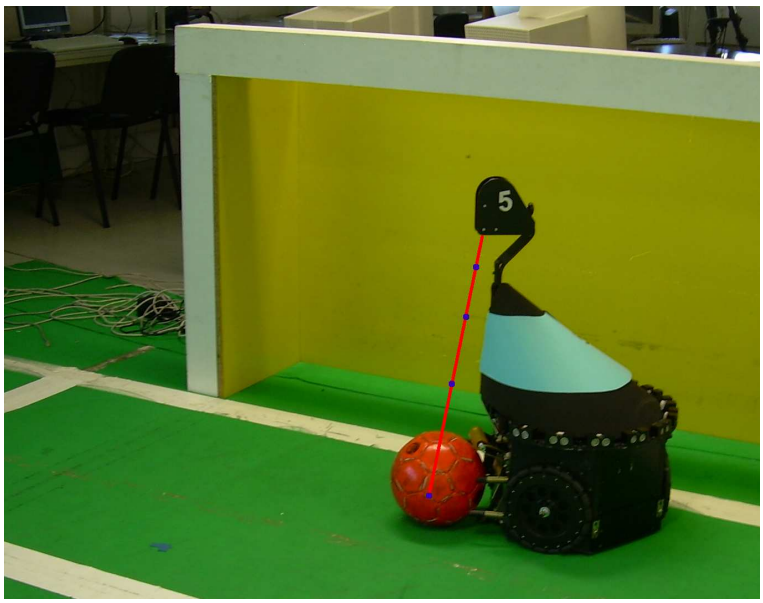


Figure 5.1: Initialization of the filter: particles are spread along a 3D ray based on a couple of image coordinates selected by a detection algorithm. The distribution used to initialize the filter will be Gaussian, centered on the particle, among the ones spread along the ray, with the highest likelihood.

which has to choose a pixel near the center of the projected ball, but should in the future be provided by a detection module. We apply the inverse equidistance projection to the coordinates and thus select the direction of the corresponding 3D ray joining the projection center to the ground plane (see Figure 5.1). We spread some particles along this ray and calculate the likelihood for every particle, based on the information contained in the image.

The distribution of particles for the PF is then initialized by a Gaussian distribution, centered at the position of the maximum likelihood particle. The velocities are also initialized by a Gaussian distribution, with mean 0.

In the other cases we initialize the particles by a Gaussian distribution with large variance, centered at rough estimates of the object's initial position and velocity. This method is not automatic, so it can not be used in an autonomous robot, unless in very specific cases. One such case is for example when robots communicate among them, so one robot can detect the ball and pass its estimates on position and speed to the others. When we track robots, both position and velocity are only bidimensional, as MSL robots do not jump yet.

Chapter 6

Experimental results

"No amount of experimentation can ever prove me right;
a single experiment can prove me wrong."

Albert Einstein

We ran several experiments to assess the accuracy and precision of the proposed tracking method: in the catadioptric setup, using UPM to model the camera projection, we tracked a ball rolling down a ramp, a ball bouncing on the floor and a robot maneuvering. We furthermore ran an experiment placing a still ball at different positions around the robot and measuring the error of the detected position with respect to the ground truth.

We compared the results obtained in the tracking and detection, using UPM and PPM to model the catadioptric sensor.

Eventually, we tracked a ball in the dioptric setup, testing the system for robustness against occlusions. We also assessed the time needed by the C++ implementation of our tracker to elaborate one image, verifying it can run in real-time.

6.1 Catadioptric setup, UPM

In the first experiment we tested the proposed method on sequences of images acquired with our catadioptric omnidirectional camera, which we model with the UPM. We used the Matlab implementation of the code, setting the number of particles to 10000. This value was intentionally very high, as we intended to test the precision of the tracker in an off-line setting. To sample the pixels in order to build the inner and outer color histograms for each hypothesis we projected a sphere with respectively 0.9 and 1.1 the radius of

the actual ball. For each projection we used 50 points, uniformly distributed on each 3D contour.

The initial position for the particles was obtained sampling a 3D Normal distribution, the mean and standard deviation of which were manually set. The initial velocity was also manually set, equal for every particle. The parameter k was set to 1.5, meaning that we wanted the difference between inner color and outer color to be more discriminative than the similarity between inner color and model color. We repeated the tracking 10 times on every image sequence, to evaluate the precision of the tracker.

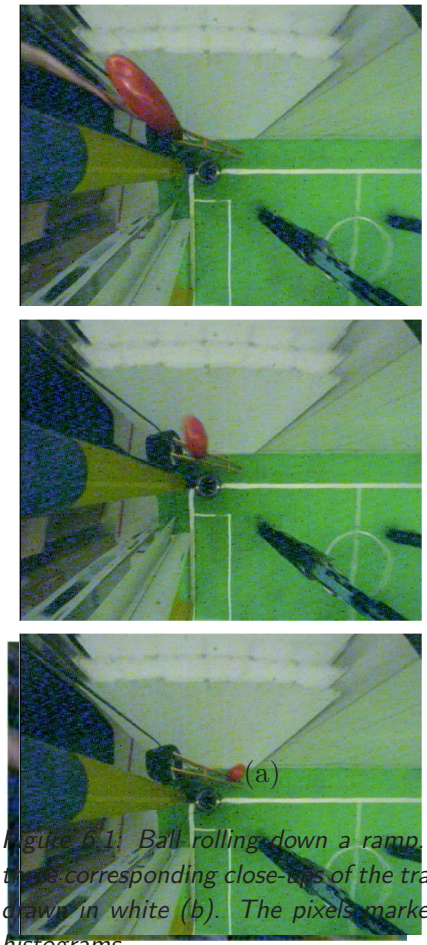
6.1.1 Ball tracking

In the first part of this experiment, we tracked a ball rolling down a two-rail ramp. The projection of the ball on the image plane changes dramatically in size after some frames (see Figure 6.1), due to the nature of the catadioptric system used. The images are affected by both motion blur and heavy sensor noise (see Figure 6.2). This image sequence was acquired with a frame rate of 20fps. The results of the tracking are visible in Figure 6.3a.

In the second part of this experiment we tracked a ball bouncing on the floor. The frame rate in this case was of 25fps. The image sequence begins with the ball about to hit an obstacle on the ground, while moving horizontally. The collision triggers a series of parabolic movements for the ball, which is tracked until it hits the ground for the fourth time. The results are visible in Figure 6.3b.

6.1.2 Robot tracking

In this experiment we tracked a robot moving along a straight line, turning by 90 degrees and continuing its motion along the new direction (Figure 6.4). The vertical position and speed of the tracked object were constrained to be null. The injected velocity noise was, thereafter, distributed as a 2D Normal. To sample the pixels in order to build the inner and outer color histogram for each hypothesis we projected the contour of an 8-sided-prism with respectively 0.75 and 1.25 the size of the actual robot. The size difference between the projected models and the actual one is greater than in the case of the ball due to the fact that the model for the robot does not exactly fit its actual shape. For each projection, 120 points were used. We repeated the tracking 10 times and results are shown in Figure 6.5.



(a) (b)

Figure 6.1: Ball rolling down a ramp. Frames 1, 11 and 21 of the sequence (a) and the corresponding close-ups of the tracked ball with the contour of the best hypothesis drawn in white (b). The pixels marked in black are the ones used to build the color histograms.

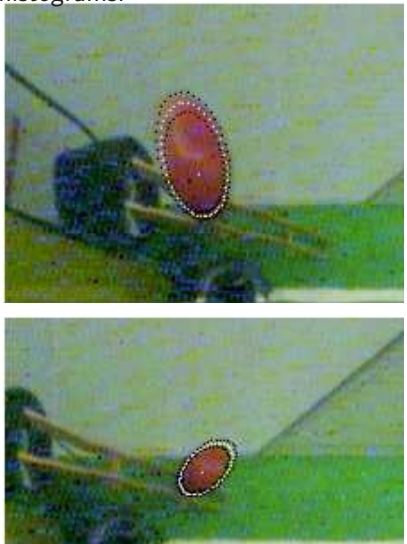


Figure 6.2: Close-ups of the ball showing motion blur and sensor noise.

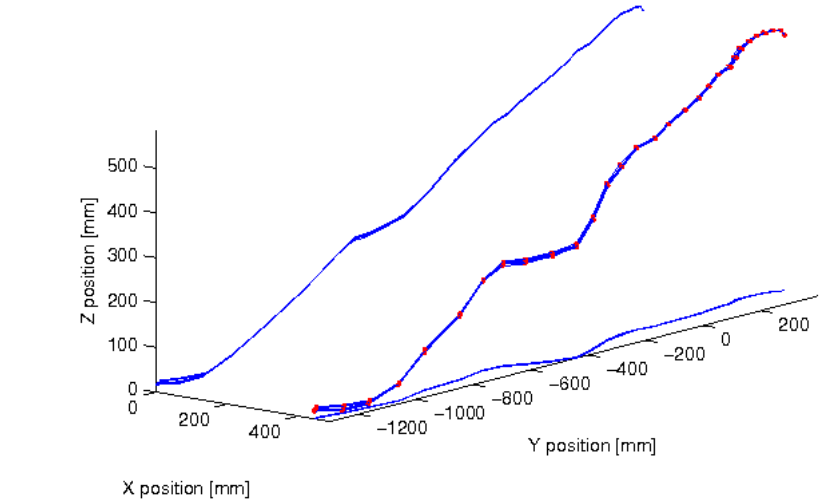
6.1.3 Error evaluation

In this experiment we placed a ball at various positions around a robot and confronted the positions measured with our system against the ground truth. The positions were either in front of the robot, on its right, on its left or behind it, and either on the floor or at a height of 340mm.

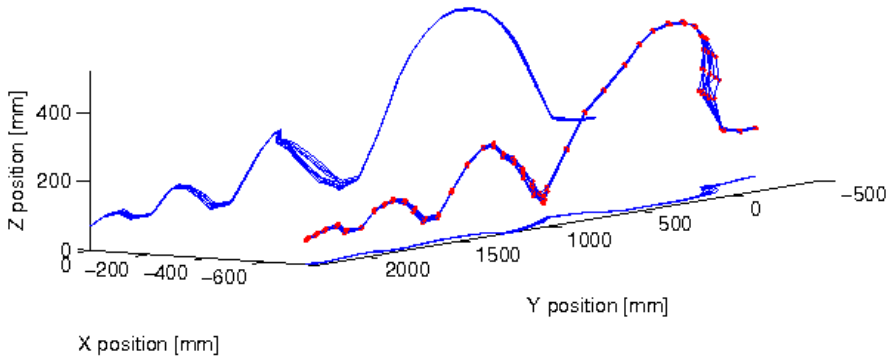
As we were interested in studying the localization error "at convergence", we run the particle filter three times on each image. In the first step, the particle filter was initialized with the ground truth position of the ball and the particles were generated with a 3D Normal p.d.f. with that position as mean and a standard deviation of 100mm. In the second step the particles were generated around the position estimated in step one, with a standard deviation of 70mm. In the third step the particles were generated around the position estimated in step two, with a standard deviation of 40mm.

Both the ground truth and the estimated ball positions are shown in Figure 6.6. It is noticeable some bias mainly in the vertical direction, due to miscalibration of the experimental setup. However, we are mostly interested in evaluating errors arising in the measurement process. Distance to the camera is an important parameter in this case, because target size varies significantly. Therefore, we have performed a more thorough error analysis evaluating its characteristics as a function of distance to the camera.

In Figure 6.7, we plot the measured error in spherical coordinates (γ–distance, ϕ–elevation, ψ–azimuth), as a function of distance to camera's virtual projection center. The first plot shows that radial error characteristics are mostly constant along distance, with a systematic error (bias) of about $46mm$, and standard deviation of about $52mm$. This last value is the



(a)



(b)

Figure 6.3: (a) Ball rolling down the ramp: plot of the tracked paths resulting from 10 runs of the algorithm performed on the same image sequence. The 10 blue lines with red points represents the 10 3D estimated trajectories of the ball, the blue lines are the projection of these trajectories on the ground and lateral plane. (b) Ball jumping: same kind of plot for the estimated trajectories of the ball in the jump image sequence.

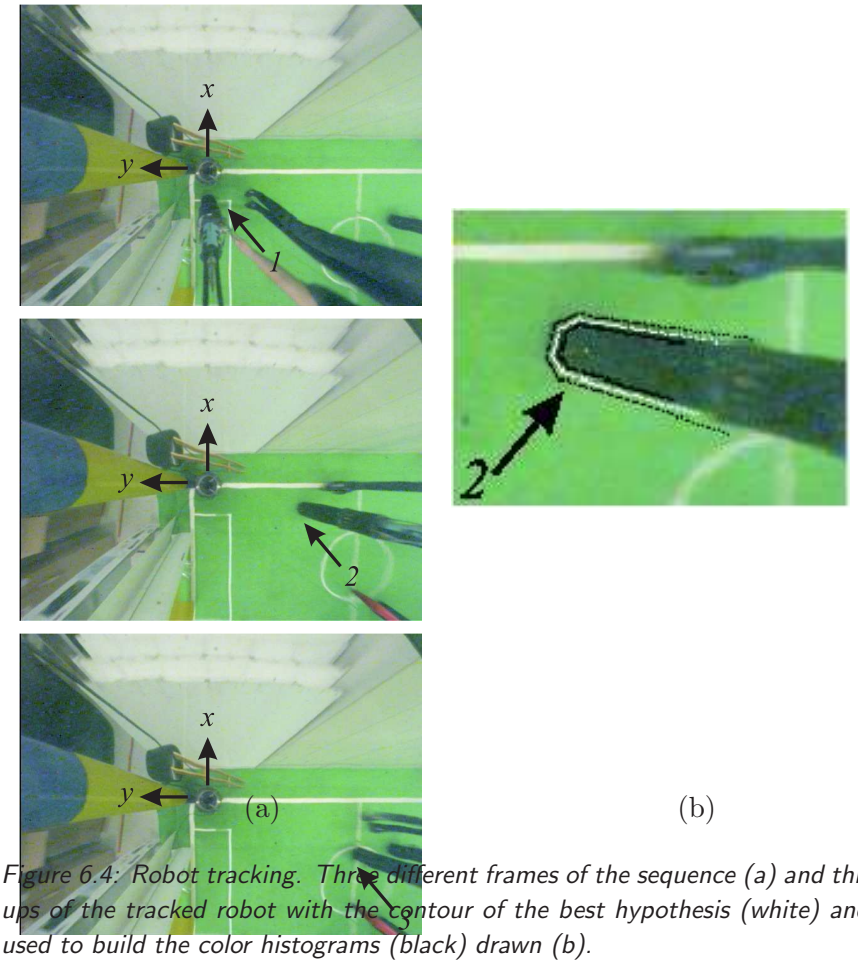


Figure 6.4: Robot tracking. Three different frames of the sequence (a) and three close-ups of the tracked robot with the contour of the best hypothesis (white) and regions used to build the color histograms (black) drawn (b).

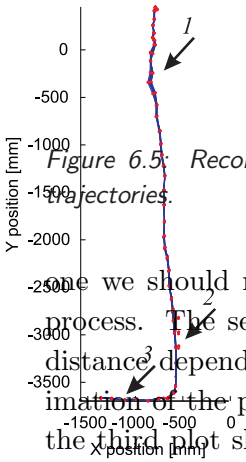


Figure 6.5: Reconstructed robot trajectory - a view from the top of 10 estimated trajectories.

one we should retain for characterizing the precision of the measurement process. The second plot shows the elevation error, where it is evident a distance dependent systematic error. This has its source on a bad approximation of the projection model for distances close to the camera. Finally, the third plot shows the azimuthal error. It can be observed that there is a larger random error component at distances close to the camera, but this just a consequence of the fact that equal position errors at closer distances produce larger angular errors.

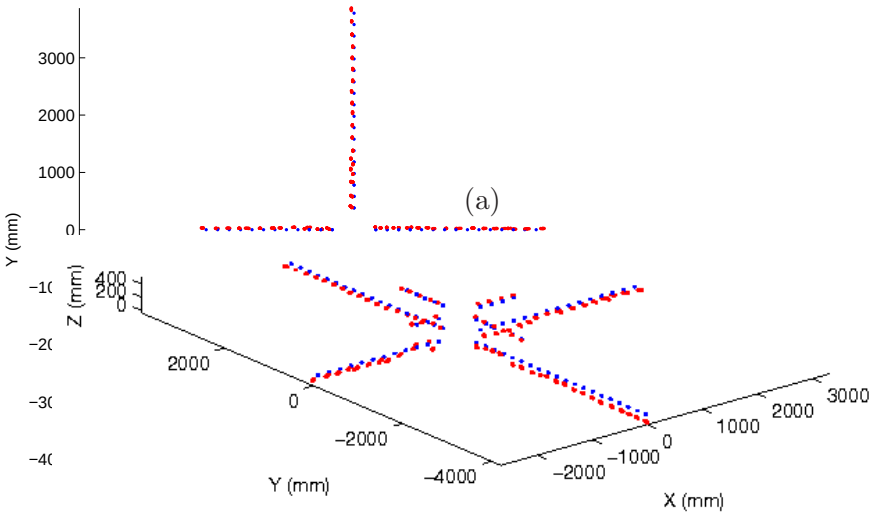
6.2 Catadioptric setup, UPM and PPM experimental comparison

In this experiment we compared the results obtained modeling the catadioptric omnidirectional camera with either UPM and PPM.

6.2.1 Error evaluation

We repeated the experiment described in Section 6.1.3, using either PPM and UPM to model the camera. Both models were calibrated with the same 3D- and imaged-points data. The color histograms in this experiment were computed based on the pixels sampled with the B-spline method, as described in Section 4.2.2.

We have computed the errors between ground truth and measurements, in spherical coordinates γ, ϕ and ψ, respectively distance, elevation and azimuth. Results for the errors' mean and standard deviation are presented



(a)

(b)

Figure 6.6: Two views of ground truth (blue) and measurements (red).

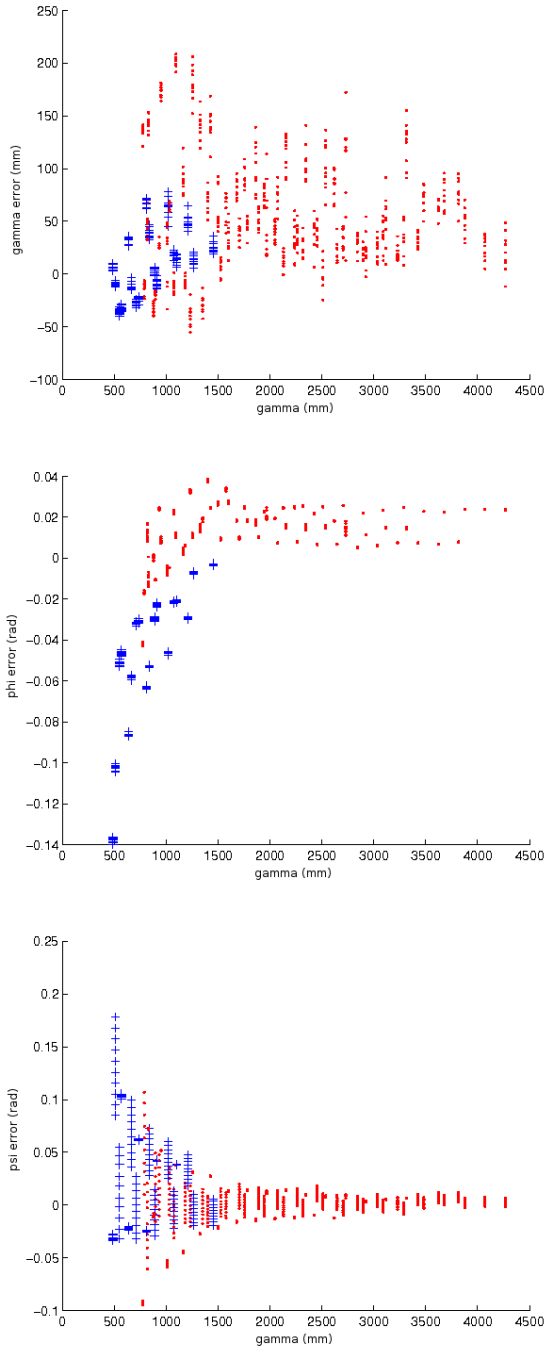


Figure 6.7: γ, ϕ, ψ error for balls laying on the ground (red dots) and flying at 340mm (blue crosses).

	UPM	PPM
mean γ error (mm)	-19.8712	-18.2616
std.dev γ error (mm)	48.1137	47.9484
mean ϕ error (rad)	0.0005	0.0006
std.dev ϕ error (rad)	0.0310	0.0276
mean ψ error (rad)	0.0072	0.0072
std.dev ψ error (rad)	0.0314	0.0312
Computation Time (ms)	11.2	6.3

Table 6.1: Comparison of errors measured with the UPM vs PPM. Last line shows the computation times for projecting 50000 points, on a P4 2.6GHz computer.

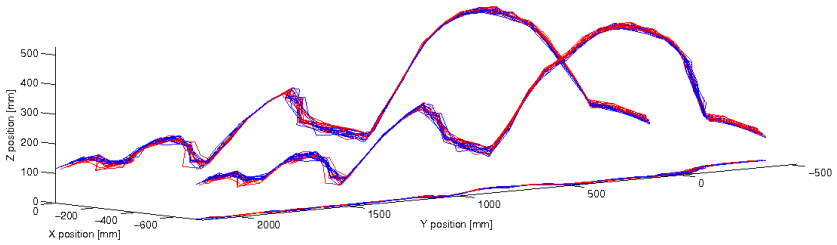


Figure 6.8: Ball jumping: plot of the tracked paths resulting from 10 runs of the algorithm performed on the same image sequence. The estimated trajectories using the PPM and the UPM models are shown, respectively, with blue and red lines. Both the 3D trajectory and its projections on the ground and lateral planes are shown.

in Table 6.1. As can be observed from the table, there are no significant differences between the two projection models. However, the computation time for PPM is about half the time taken by the UPM (last line of Table 6.1). Therefore, the usage of the PPM is advantageous for the overall 3D tracking system, since it reduces computation time for the same level of precision.

6.2.2 Ball tracking

In this experiment we repeated the tracking on the sequence of the jumping ball (see Section 6.1.1), using PPM instead of UPM. A comparison of the results obtained using the two models is shown in Figure 6.8. No noticeable differences are observed between the two models, but computational savings justify the choice of the PPM model.

6.3 Dioptric setup

In this set of experiments we tested the proposed method on sequences of images acquired with our dioptric omnidirectional camera, which we model with the EPM. We used the C++ implementation of the tracker, the one exploiting the vector and matrix operations provided by the IPP library, by Intel. To sample the pixels in order to build the inner and outer color histograms for each hypothesis we projected a sphere with respectively 0.7 and 1.3 the radius of the actual ball. This provided us with a "wider" and "flatter" likelihood function than in experiment described in Section 6.1.3.This choice allowed us to use few particles, namely 300, trading some precision for speed. For each projected contour we used 50 points, uniformly distributed on each 3D contour. With these settings, the tracker run at more than 29fps on a 2.6GHz Pentium 4 processor. The color model, a YUV histogram, was obtained from manually selected regions, belonging to an image sequence which was not used for tracking (see Figure 6.9). The image sequences for the various experiments were acquired at different frame rates, with different camera settings and different illumination conditions. The tracker was successful in all its runs, in spite of these difficulties, which made the color of the ball look different in every sequence (see Figure 6.10). The tracker also successfully handled reflexions and shadows, which made the ball look nonhomogeneous in color.

The initial distribution of the particles was set up in two steps: a detection step and a real initialization step. In the first step, particles were spread along the 3D ray corresponding to a couple of user-selected image coordinates. A 3D point was chosen in correspondence of the particle with the highest likelihood. In the second step, the set of particles was initialized by a Gaussian distribution with large variance, centered at the selected 3D point. The parameter k was, as in the other experiments, set to 1.5. We repeated the tracking 10 times on each image sequence, in order to test the precision of the tracking method.

6.3.1 Jumping ball

We tracked a ball jumping on the floor and bouncing off a wall (see Figure 6.11). This experiment shows the ability of our tracker to work with off-the-floor targets. The trajectories resulting from the 10 runs of the tracker show that, in spite of using few particles, the estimation of the position of the ball is precise (Figure 6.12). This image sequence was acquired at 25fps.

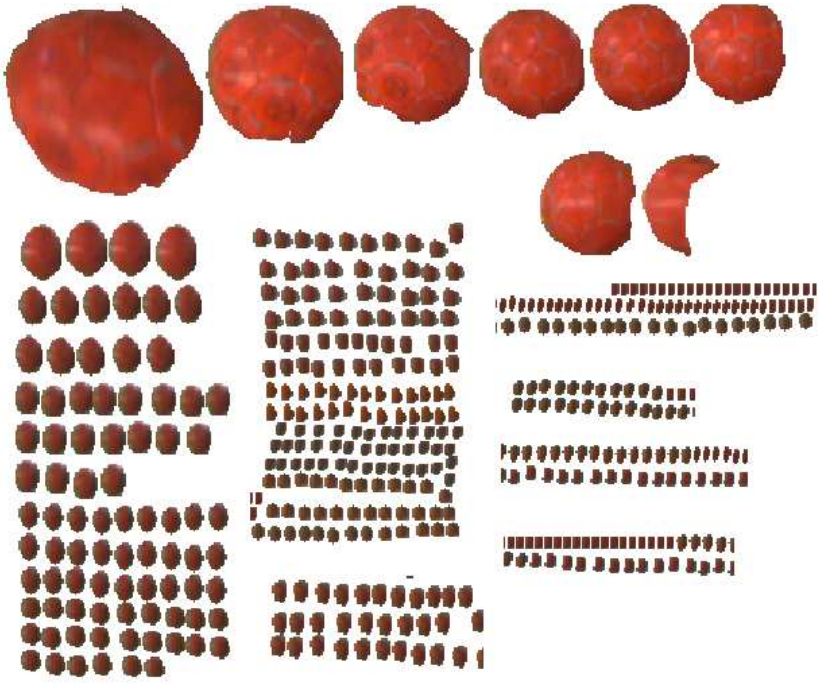


Figure 6.9: The color histogram for the experiments with the dioptric camera was built based on the non-white pixels of this image.

6.3.2 Occlusions

We tracked a ball passing behind an obstacle (see Figure 6.13 and 6.14) and a ball which gets to be halfway hidden by a robot (see Figure 6.15 and 6.16). These experiments show the robustness of our tracking method against occlusions. The method succeeds in tracking the ball in both cases, showing some inaccuracy when the ball is hidden by the robot.

6.3.3 Multiple objects

In this experiment we tracked a ball along a sequence in which its projection onto the image overlaps with that of another ball. Our tracker is not a multiple object tracker, but we show that its behavior in such a case is still sensible: it either keeps tracking the ball it was initialized on or switches to the other. The results of the tracking are visible in Figure 6.17 and 6.18.

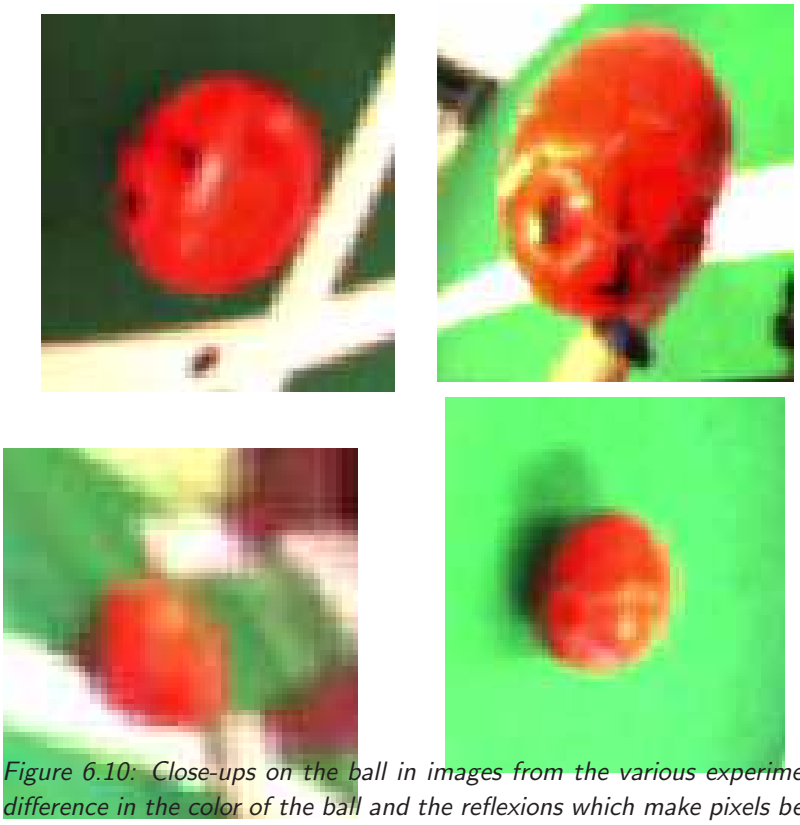
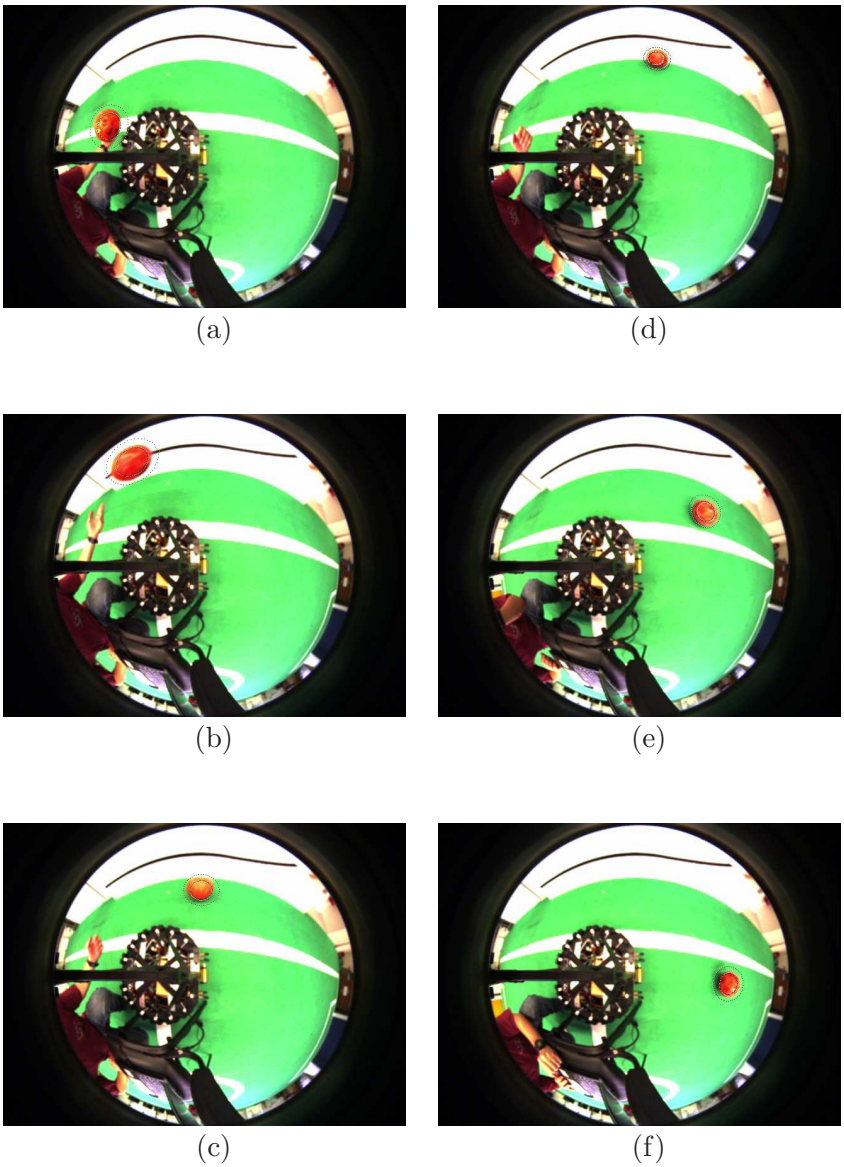


Figure 6.10: Close-ups on the ball in images from the various experiments. Note the difference in the color of the ball and the reflexions which make pixels belonging to the projection of the ball look yellow, rather than orange.



(a) (d)

(b) (e)

(c) (f)

Figure 6.11: Ball jumping. Six frames from the tracked sequence.

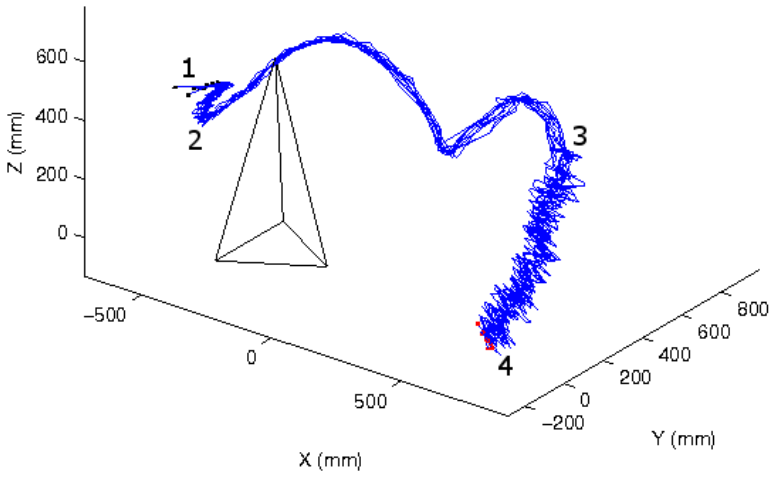
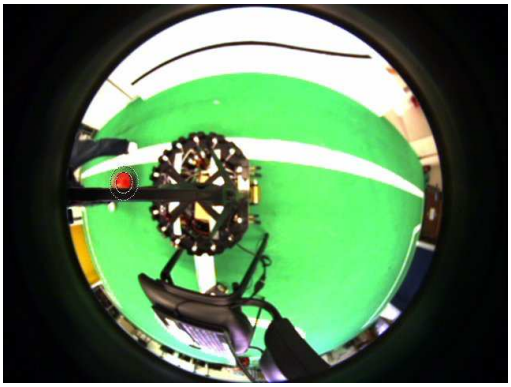
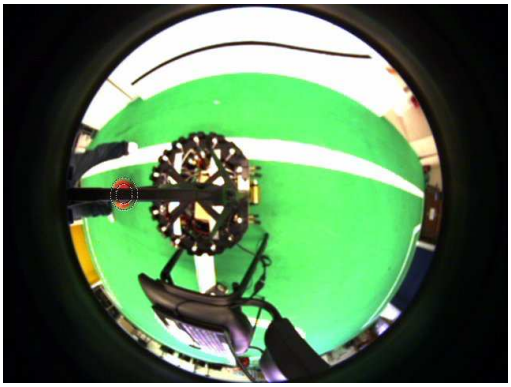


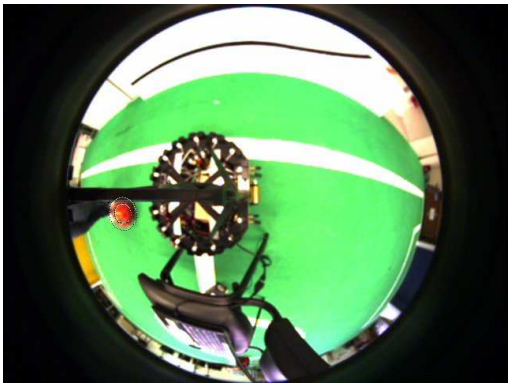
Figure 6.12: Ball jumping: plot of the tracked paths resulting from 10 runs of the algorithm performed on the same image sequence. The pyramid represents the robot acquiring the image sequence. The tracked paths start in 1. At first, the ball is held by a person who lowers it (tracks from area 1 to 2), then the ball is thrown in the air, bounces on the floor and then bounces on the floor and off the wall at the same time (3). The tracked paths end in 4.



(a)

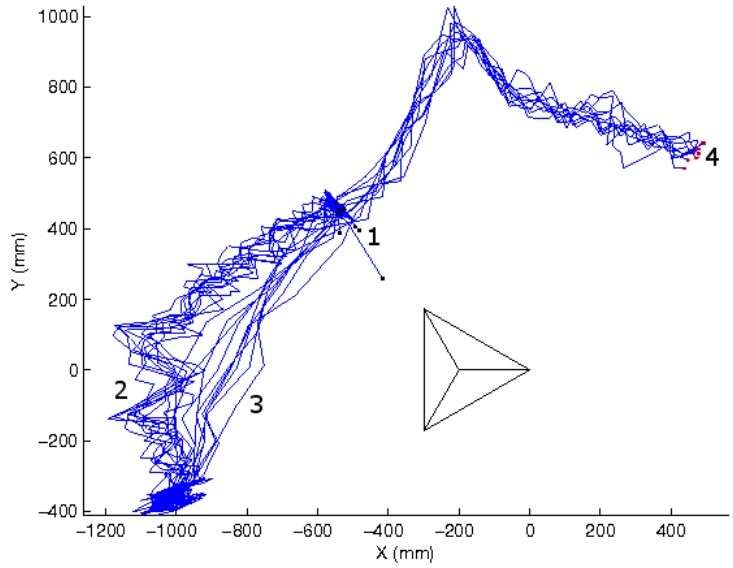


(b)

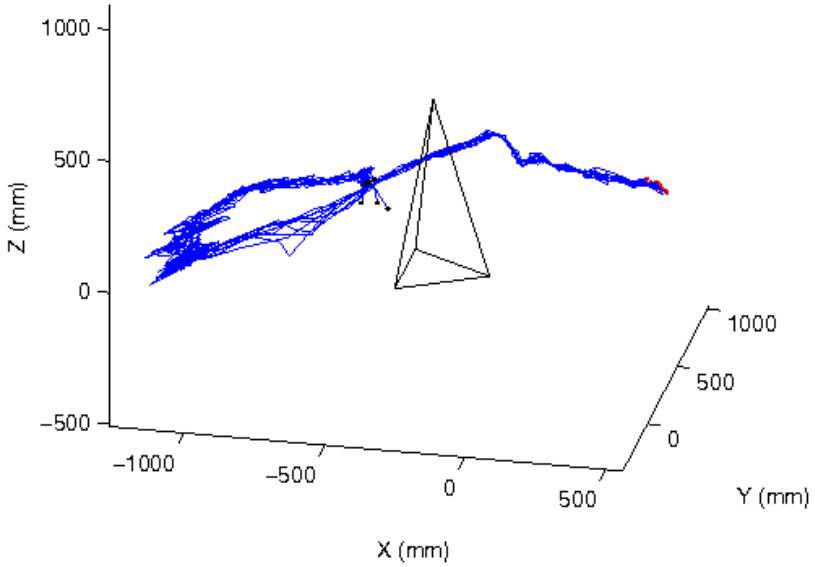


(c)

Figure 6.13: Three frames of the tracked sequence when the ball is occluded by the bar which sustains the camera.

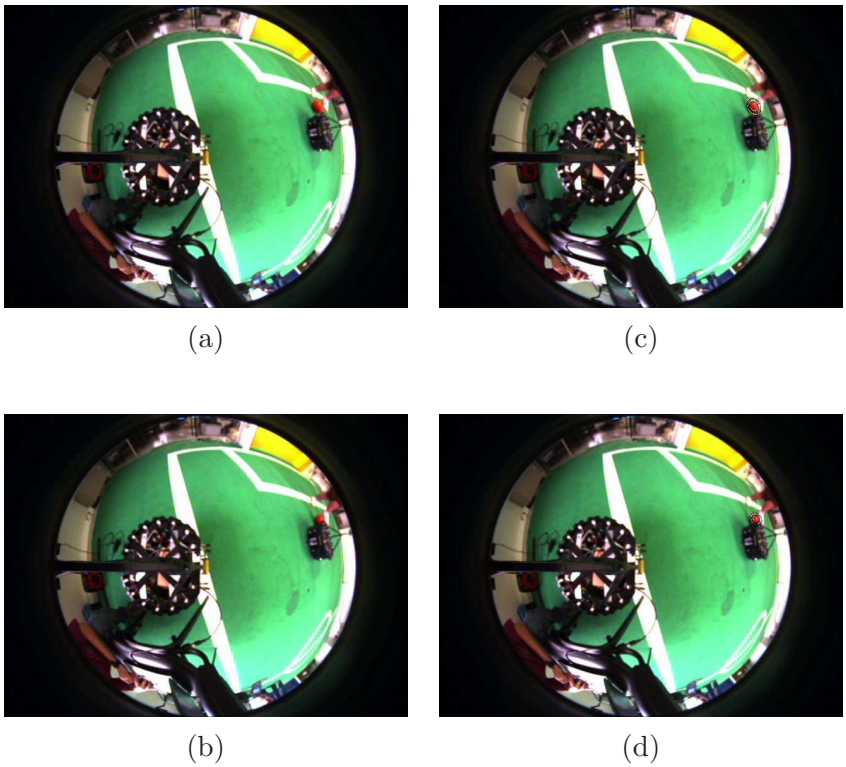


(a)



(b)

Figure 6.14: Top view (a) and 3D plot (b) of the 10 tracked trajectories. The pyramid represents the robot acquiring the image sequence. 1 denotes the beginning of the trajectories, 2 denotes the area when the ball is occluded by the bar and moving slowly, a small error in the trajectories can be noticed (in the trajectories there are two abrupt changes of direction, in correspondence with the beginning and end of the occlusion). 3 denotes another passing of the ball behind the obstacle, while 4 denotes the end of the trajectories.



(a) (c)

(b) (d)

Figure 6.15: Two frames of the tracked sequence in which the ball is occluded by another robot. In the first row the ball is still fully visible, while in the second it is halfway hidden by the robot. In the first column the original images are shown, in the second one the inner and outer contours of the maximum likelihood hypothesis are drawn in white, while those of the hypothetical position corresponding to mean of all the particles, weighted by their likelihood, are drawn in black.

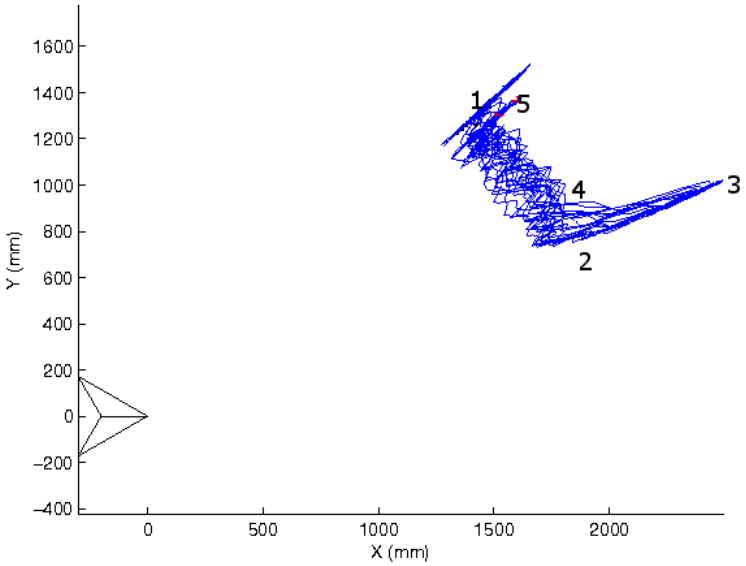
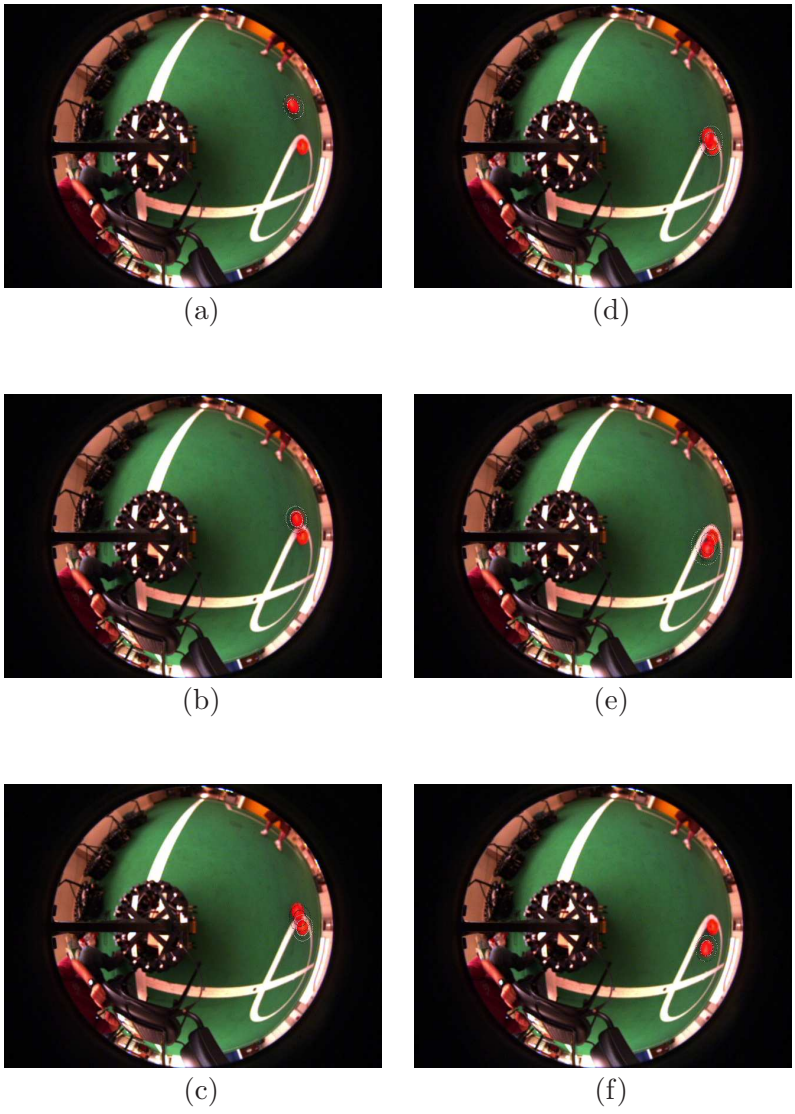


Figure 6.16: Plots of the 10 tracked trajectories of the ball. The tracked paths begin in 1. When the ball is fully visible (from area 1 to 2 and from 4 to 5) the precision of the tracker is lower than in the previous experiment. This depends on the fact that in this experience the ball is farther from the robot acquiring the image sequence. Farther objects can be localized with less precision because of the projection performed by the dioptric omnidirectional camera. When the ball is hidden (from area 2 to 4), the estimate of the position becomes less accurate, as the tracker associates higher likelihoods to smaller contours, corresponding to balls which are farther from the observing robot.



(a) (d)

(b) (e)

(c) (f)

Figure 6.17: Image frames from the sequence with two overlapping balls.

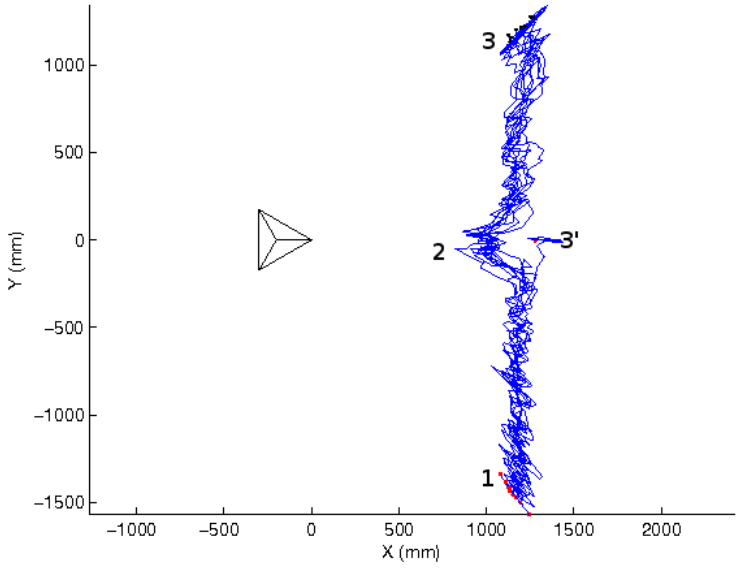


Figure 6.18: Tracked trajectories for the sequence with overlapping balls. The tracked paths start in area 1. Some inaccuracy in the localization of the tracked object can be seen in correspondence of the overlapping (2). In nine cases out of ten, the tracker followed the ball it was initialized on (ending in area 3), while in one case it switched to tracking the other ball (ending in area 3').

Chapter 7

Conclusions and future work

"The end is the beginning
of the end."

Billy Corgan, The Smashing Pumpkins.

In this thesis we presented a model-based 3D tracker for autonomous robots equipped with omnidirectional cameras. The tracker was designed to work in the structured, color-coded environment of the RoboCup MSL. The system was based on a particle filter, for robustness against occlusions and ambiguity. The chosen object model comprises a 3D shape model, a color model and a motion model. The observation model was designed to be robust against motion blur, image sensor noise and illumination changes. The system works with either a catadioptric or a dioptric omnidirectional camera.

We developed both a Matlab and a C++ implementation of the tracker.

We ran several experiments, in a RoboCup MSL scenario, to assess the accuracy and precision of the proposed tracking method with color-coded objects.

In the catadioptric setup, using UPM to model the camera projection, we tracked a ball rolling down a ramp, a ball bouncing on the floor and a robot maneuvering. These experiments on ball tracking show the robustness of the method with respect to motion blur and image sensor noise.

Still in the catadioptric setup, we ran an experiment to assess the accuracy of the system, placing a still ball at different positions around the robot and measuring the error of the detected position with respect to the ground truth.

We compared the results obtained in tracking and detection, using UPM and PPM to model the catadioptric sensor, verifying that PPM provides the

same level of precision and accuracy of UPM, but at a smaller computational cost.

Eventually, we tracked a ball in the dioptric setup, testing the system for robustness against occlusions and real-time capabilities, verifying that the C++ implementation, exploiting the vector and matrix operations provided by the IPP library, can run at 29fps on a P4 2.6GHz processor.

In future work we intend to extend the system with the ability to track multiple objects. We will introduce the use of multiple motion models, for example use a model describing the motion of a ball rolling freely and another describing the motion of a ball bouncing against an obstacle. We also intend to use physics-based motion models (e.g. applying gravitational acceleration to particles which model flying balls). We will express the target's coordinates in an inertial reference frame, instread of the robot-centered one we use now. This will be combined with ego-motion compensation: applying the inverse motion of the robot to the filter's particles. We also plan to test the tracker in a non-color-coded environment.

Bibliography

[1] George A. Bekey. *Autonomous Robots*. MIT, 2005.

[2] R. Benosman and S.B. Kang, editors. *Panoramic Vision*. Springer Verlag, 2001.

[3] ST Birchfield and S. Rangarajan. Spatiograms versus Histograms for Region-Based Tracking. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2, 2005.

[4] G.R. Bradski et al. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, 2(2):12–21, 1998.

[5] Gilles Celeux, J. Nascimento, and J. S. Marques. Learning switching dynamic models for objects tracking. *Pattern Recognition*, 37(9):1841–1853, 2004.

[6] P. Chang and J. Krumm. Object recognition with color cooccurrence histograms. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:498–504, 1999.

[7] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[8] C. Christoudias, B. Georgescu, and P. Meer. Synergism in low level vision. *International Conference on Pattern Recognition*, 4:150–155, 2002.

[9] RT Collins, Y. Liu, and M. Leordeanu. Online selection of discriminative tracking features. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1631–1643, 2005.

[10] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 142–151, 2000.

[11] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(5):564–577, 2003.

[12] I.J. Cox, S.L. Hingorani, et al. An efficient implementation of Reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[13] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods In Practice*. Springer Verlag, 2001.

[14] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975.

[15] J. Gaspar, C. Deccó, Jun Okamoto Jr, and J. Santos-Victor. Constant resolution omnidirectional cameras. *3rd International IEEE Workshop on Omni-directional Vision at ECCV*, pages 27–34, 2002.

[16] J. Gaspar, N. Winters, and J. Santos-Victor. Vision-based navigation and environmental representations with an omnidirectional camera. *IEEE Transactions on Robotics and Automation*, 16(6), 2000.

[17] T. Gevers and H. Stokman. Robust histogram construction from color invariants for object recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(1):113–118, 2004.

[18] C. Geyer and K. Daniilidis. A unifying theory for central panoramic systems and practical applications. *Conf. on Computer Vision and Pattern Recognition*, pages 445–461, 2000.

[19] Christopher Geyer, Tomas Pajdla, and Kostas Daniilidis. Short course on omnidirectional vision. ICCV, 2003.

[20] R. Hicks and R. Bajcsy. Catadioptric sensors that approximate wide-angle perspective projections. *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 545–551, 2000.

[21] C. Hue, J.P. Le Cadre, P. Perez, and R.I. IRISA. Tracking multiple objects with particle filtering. *Aerospace and Electronic Systems, IEEE Transactions on*, 38(3):791–812, 2002.

[22] M. Isard and A. Blake. Contour Tracking by Stochastic Propagation of Conditional Density. *Proceedings of the 4th European Conference on Computer Vision-Volume I-Volume I*, pages 343–356, 1996.

[23] M. Isard and A. Blake. CONDENSATION: Conditional Density Propagation for Visual Tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[24] M. Isard and J. MacCormick. BraMBLe: A Bayesian multiple-blob tracker. *International Conference on Computer Vision*, 2(5), 2001.

[25] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[26] J. Kannala and S. Brandt. A generic camera calibration method for fish-eye lenses. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 1, 2004.

[27] Z. Khan, T. Balch, and F. Dellaert. MCMC-based particle filtering for tracking a variable number of interacting targets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1805–1819, 2005.

[28] W. Koch and G. Van Keuk. Multiple hypothesis track maintenance with possibly unresolvedmeasurements. *Aerospace and Electronic Systems, IEEE Transactions on*, 33(3):883–892, 1997.

[29] V. Lepetit and P. Fua. *Monocular Model-based 3d Tracking of Rigid Objects*. Now Publishers Inc, 2005.

[30] P. Lima, A. Bonarini, C. Machado, F. Marchese, F. Ribeiro, and D. Sorrenti. Omni-directional catadioptric vision for soccer robots. 2001.

[31] K. Okuma, A. Taleghani, N. de Freitas, J.J. Little, and D.G. Lowe. A boosted particle filter: Multitarget detection and tracking. *European Conference on Computer Vision*, 1:28–39, 2004.

[32] S. Olufs, F. Adolf, R. Hartanto, and P. Ploger. Towards probabilistic shape vision in robocup: A practical approach. *RoboCup International Symposium*, 2006.

[33] P. Perez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking using unscented particle filter. *IEEE Conf. on Computer Vision and Pattern Recognition*, 2002.

[34] M. Pressigout and E. Marchand. Real-time 3D Model-Based Tracking: Combining Edge and Texture Information. *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2726–2731, 2006.

[35] M. Pupilli and A. Calway. Real-time camera tracking using known 3D models and a particle filter. *Intl. Conference on Pattern Recognition*, 2006.

[36] D. Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, 1979.

[37] Y.B. Shalom and TE Fortmann. Tracking and Data Association. *Academic-Press, Boston*, 1988.

[38] Matteo Taiana, José Gaspar, Jacinto Nascimento, Alexandre Bernardino, and Pedro Lima. 3d tracking by catadioptric vision based on particle filters. *RoboCup International Symposium*, 2007.

[39] Matteo Taiana, José Gaspar, Jacinto Nascimento, Alexandre Bernardino, and Pedro Lima. On the use of perspective catadioptric sensors for 3d model-based tracking with particle filters. *IEEE International Conference on Intelligent Robots and Systems*, Accepted for publication, 2007.

[40] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT, 2005.

[41] H. Wang, D. Suter, and K. Schindler. Effective Appearance Model and Similarity Measure for Particle Filtering and Visual Tracking. *European Conference on Computer Vision*, 2006.

[42] T. Zhao and R. Nevatia. Tracking multiple humans in complex situations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1208–1221, 2004.

Appendix A

Paper published in RoboCup Symposium 2007

Appendix B

Paper to be published in IROS 2007