SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza – Università di Roma

FACOLTÀ DI INGEGNERIA
Corso di Laurea Specialistica in Ingegneria Informatica

TESI DI LAUREA SPECIALISTICA

Object Manipulation from Simplified Visual Cues

Candidato:
Giovanni Saponaro

Relatore:
Prof. Daniele Nardi

Correlatore:
Prof. Alexandre Bernardino

Anno Accademico 2007–2008

Sommario

La robotica umanoide in generale, e l'interazione uomo–robot in particolare, stanno oggigiorno guadagnando nuovi e vasti campi applicativi: la robotica si diffonde sempre di più nella nostra vita. Una delle azioni che i robot umanoidi devono poter eseguire è la manipolazione di cose (avvicinare le braccia agli oggetti, afferrarli e spostarli). Tuttavia, per poter fare ciò un robot deve prima di tutto possedere della *conoscenza* sull'oggetto da manipolare e sulla sua posizione nello spazio. Questo aspetto si può realizzare con un approccio percettivo.

Il sistema sviluppato in questo lavoro di tesi è basato sul *tracker* visuale CAMSHIFT e su una tecnica di ricostruzione 3D che fornisce informazioni su posizione e orientamento di un oggetto generico (senza modelli geometrici) che si muove nel campo visivo di una piattaforma robotica umanoide. Un oggetto è percepito in maniera *semplificata*: viene approssimato come l'ellisse che racchiude meglio l'oggetto stesso.

Una volta calcolata la posizione corrente di un oggetto situato di fronte al robot, è possibile realizzare il *reaching* (avvicinamento del braccio all'oggetto). In questa tesi vengono discussi esperimenti ottenuti col braccio robotico della piattaforma di sviluppo adottata.

Abstract

Humanoid robotics in general, and human–robot interaction in particular, is gaining new, extensive fields of application, as it gradually becomes pervasive in our daily life. One of the actions that humanoid robots must perform is the manipulation of things (reaching their arms for objects, grasping and moving them). However, in order to do this, a robot must first have acquired some *knowledge* about the target object and its position in space. This can be accomplished with a perceptual approach.

The developed system described in this thesis is based on the CAMSHIFT visual tracker and on a 3D reconstruction technique, providing information about position and orientation of a generic, model-free object that moves in the field of view of a humanoid robot platform. An object is perceived in a *simplified* way, by approximating it with its best-fit enclosing ellipse.

After having computed where an object is currently placed in front of it, the robotic platform can perform reaching tasks. Experiments obtained with the robot arm of the adopted platform are discussed.

Acknowledgements

First of all, I would like to thank my daily supervisor in this project for his uninterrupted support and patience during my eight months of stay in VisLab and Institute for Systems and Robotics, Instituto Superior Técnico, Lisbon. Thank you very much, Prof. Alexandre Bernardino. That, plus... passionately discussing algorithms while eating a Portuguese *doce* and sipping a coffee together (several times) is priceless.

I also wish to express my gratitude to my home advisor: Prof. Daniele Nardi of Sapienza University of Rome. Not only has he provided me with the chance to do my thesis research abroad, but he has been helpful, encouraging and available for suggestions at all times.

I would like to thank Prof. José Santos-Victor of Instituto Superior Técnico for his confidence in hosting me, and for making VisLab such an enjoyable environment to work at, ultimately making it a breeze to do research there.

This work was partially supported by EC Project IST-004370 RobotCub and by the Portuguese Government – Fundação para a Ciência e Tecnologia (ISR/IST pluriannual funding) through the POS_Conhecimento Program that includes FEDER funds. This is gratefully acknowledged.

In Lisbon I found plenty of nice people since day one. I am glad to have met the *Italian gang* of IST, dearest friends and valued teachers to me: Alessio, Giampiero and Matteo. And then I certainly wish to thank many more colleagues for the fruitful discussions that ensued and for the fun (apologies if I forgot anybody): Christian, Daniel, Daniela, Dario, Prof. Gaspar, Ivana, Jonas the Superschwiizer, Jonas the Swede, Luisão, L. Vargas, Manuel, Marco, Mário, Matthijs, Plinio, Ricardo, Rubén, Samuel, Verica.

Thank you Aleka, Andrea, Patty, Sabrina, Valentin and everybody else back from my Erasmus year. Cheers to Cimpe and Claudione, crazy Portugal-lovers. Oh, thanks to PierFrok for the tip.

Nico (e Tigas), obrigado por tudo, desde 2005©. Obrigado à *trindade*: Carmen, Joana, Lara, Leonor. Agradeço também ao Gustavo, ao Piçarra, ao Paulo e a todos os portugueses que andam por Roma.

Naturalmente grazie a tutti coloro con cui ho condiviso esperienze in Italia. A Balerio per l'amicizia antica. A Sofia, *sister of a lifetime*. Ai compagni di scuola: Ilardo, Il Giulio, Jessica, Lorenzo, Nausicaa, Valerione Picchiassi. Ai compagni di università: la *crew* "Gli anni '12" e lo staff di foruming.

Grazie ad Aurora, a Ludovico e ai gatti tutti: anche se preferisco i cani, faccio un'eccezione.

Grazie a tutti gli altri amici relativamente recenti ma già ottimi e abbondanti: Aldo Rock, Mauro, Simone, Riccardo.

E poi, grazie davvero a *tutta* la mia famiglia. A mia madre Paola e a mio padre Francesco per il loro amore infinito. A mio fratello Davide che è sempre stato fonte di ispirazione; ad Arianna e a Carlo.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms and Abbreviations

ADC	Analogue-to-Digital Converter
AI	Artificial Intelligence
API	Application Programming Interface
BLAS	Basic Linear Algebra Subprograms
CAMSHIFT	Continuously Adaptive Mean Shift
CCD	Charge-Coupled Device
CLAWAR	Climbing and Walking Robots and the Support Technologies for Mobile Machines
CV	Computer Vision
CMY	Cyan Magenta Yellow
DH	Denavit-Hartenberg
DOF	Degree of Freedom
DSP	Digital Signal Processor
EM	Expectation Maximization
FPGA	Field-Programmable Gate Array
FPS	Frames per Second
GSL	GNU Scientific Library
HSV	Hue Saturation Value
IBVS	Image-Based Visual Servoing
IPC	Inter-Process Communication
MDH	Modified Denavit-Hartenberg
NMC	Networked Modular Control
ROI	Region of Interest

PBVS	Position-Based Visual Servoing
PSC	Propagation of Surfaces under Curvature
PUI	Perceptual User Interface
RGB	Red Green Blue
RobotCub	Robotic Open-Architecture Technology for Cognition, Understanding and Behaviour
SDH	Standard Denavit-Hartenberg
SOM	Self-Organizing Map
YARP	Yet Another Robot Platform

Chapter 1

Introduction

1.1 Motivation

A current field of research in humanoid robotics is the study of *interactions* between a robot and its human users, focusing on topics such as perception, learning and imitation. Neurosciences and developmental psychology, which study the inner mechanisms of the human brain, also contribute to these matters as they try to understand key cognitive issues: how to learn sensory-motor coordination, which properties of observed objects or of the world we learn, how human beings imitate each other, and how they recognize actions.

The reason why interactions are relevant and worth studying in robotics is twofold. First, they allow us to progress within the underlying scientific disciplines: robotics, image processing, Computer Vision (CV), Artificial Intelligence (AI), signal processing and control theory.

Secondly, by improving human-robot interactions and better understanding our brain, we also contribute to specific applications that have an increasing *social impact*, namely rescue operations, emergencies, visual monitoring of urban areas, as well as robotic assistants that improve quality of life for the elderly or disabled people [HJLY07].

Grasping and manipulation are among the most fundamental tasks to be considered in humanoid robotics. Fig. 1.1 shows two examples of service robotics platforms that possess enough tools, appliances and flexibility to potentially adapt to human tasks.

Just like humans distinguish themselves from other animals by having highly skilled hands, so can humanoid robots: dexterous ability must be considered as a key component in practical applications such as service robotics or personal robot assistants.

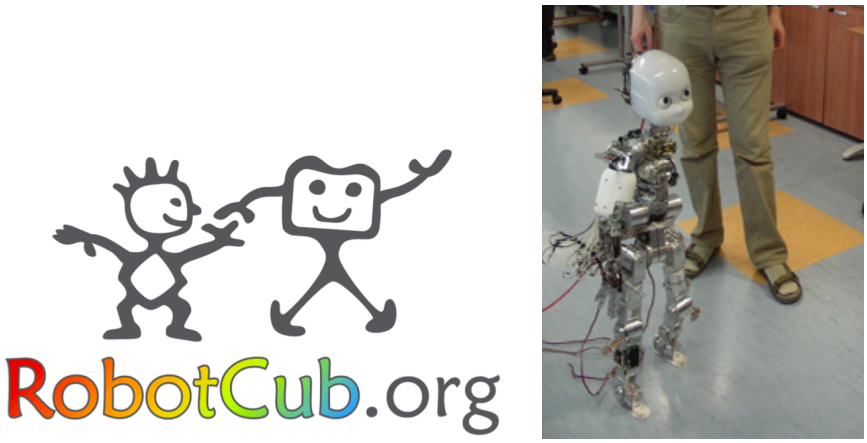
The high dexterity that characterizes human manipulation does not come for granted at birth. Instead, it arises gradually during a complex developmental process which spans different stages. After recognizing things surrounding them by the means of *vision*, babies first attempt to reach for these things, with a very limited precision. Then, at some point they start to adapt their hands to the shape of objects, initially letting these objects fall on the ground because of incorrect grasping procedures. Only after some years are they finally able to master their arm and hand skills.

Figure 1.1: Two examples of service robots, built by the University of Karlsruhe (Germany) and Fujitsu, respectively.

Furthermore, perception develops in parallel with these manipulation skills, in order to incrementally increase the performance in detecting and measuring those object features that are important for touching, grasping and holding something. Along time, interactions with objects of diverse shapes are successfully performed by applying various possible reaching and manipulation techniques. Salient effects are produced (e.g., an object moves, it is deformed, or it makes a sound when squeezed), perceived and associated to actions. An agent thus learns object affordances [MLBSV08], i.e., the relationships among a certain manipulation action, the physical characteristics of the object involved, and the observed effects. The way of reaching for an object evolves from a purely position-based mechanism to a complex behaviour which depends on target size, shape, orientation, intended usage and desired effect.

Framed within the Robotic Open-Architecture Technology for Cognition, Understanding and Behaviour (RobotCub) Project [MVS05], this thesis aims at providing *simple* 3D object perception for enabling the development of manipulation skills in a humanoid robot, by approximating a perceived object with its best-fit enclosing ellipse.

This work addresses the problem of reaching for an object and preparing the grasping action, according to the *orientation* of the objects to interact with. The proposed technique is not intended to have very accurate measurements of object and hand postures, but merely the necessary quality to allow for successful object–hand interactions. Precise manipulation needs to emerge from experience by optimizing action parameters as a function of the observed effects. To have a simple enough model of object and hand shapes, they are approximated as 2D ellipses located in a 3D space. An underlying assumption is that objects have a sufficiently *distinct colour*, in order to facilitate segmentation from the image background. Perception of object orientation in 3D is provided by the second-order moments of the segmented areas in left and right images, acquired by a humanoid robot active vision head.



(a) RobotCub Consortium logo.

(b) The iCub robot platform standing.

Figure 1.2: RobotCub logo and a prototype of its baby robot prototype, the iCub.

This thesis will describe: the humanoid robot platform "Baltazar" that was used for research and tests, the adopted CV techniques, a simple method to estimate the 3D orientation of a target object, strategies for the reaching and grasping tasks, experimental results and future work.

1.2 RobotCub and the Development of a Cognitive Humanoid Robot

The RobotCub Consortium [Rob] is a five-year-long project, funded by the European Commission through Unit E5 ("Cognition") of the Information Society Technologies priority of the Sixth Framework Programme (FP6).

RobotCub is a project to study cognition through robotics. Its objective is to create a completely open design for a humanoid robot — "open hardware, open software, open mind". All the RobotCub hardware designs and software are free and open source. The RobotCub Consortium is composed of 16 partners: 11 from Europe, 3 from Japan and 2 from the USA. LIRA-Lab [LIR] at the University of Genoa, Italy, is the coordinator.

Inspired by recent results in neurosciences and developmental psychology, the objective of RobotCub is to build an open-source humanoid platform for original research on cognitive robotics, with a focus on developmental aspects. One of the tenets of the project is that manipulation plays a key role in the development of cognitive ability.

The iCub is a humanoid baby robot designed by the RobotCub consortium. The iCub, shown in Fig. 1.2b, is a full humanoid robot the size of a two-year-old child. Its total height is around 90 cm, and it has 53 Degree of Freedoms (DOFs), including articulated hands to be used for manipulation and gesturing; in addition, the iCub is equipped with an inertial system in its hand, stereo audition, and the ability to perform facial expressions.

At the time of writing this thesis, a study is being conducted for determining if and how many DOFs are minimally required to produce and generate plausible facial expressions. The iCub robot should eventually be able to crawl and sit (to free the hands from supporting the body) and autonomously transition from crawling to sitting and vice-versa.

This thesis work was carried out at the Computer and Robot Vision Laboratory [Vis], Institute for Systems and Robotics, IST, Lisbon (Portugal) during 2008. At the time of working on this project, the arm-hand system of a full iCub prototype was still being assembled. Therefore, for this work another humanoid robot platform was used: Baltazar [LBPSV04, LSV07]. It consists of a robotic torso and a binocular head, built with the aim of understanding and performing human-like gestures, mainly for biologically inspired research.

1.3 Thesis Background

Manipulation skills at macro- and micro-scales are very important requirements for robot applications. This is valid both in industrial robotics and in less-traditional fields. As far as industry is concerned, robot manipulators hold a key role in many scenarios; to name a few:

- handling;
- food;
- fabrics;
- leather.

Similarly, in less structured domains of robotics, manipulation still play a relevant part:

- surgery;
- space;
- undersea.

Manipulation and grasping systems are thus a vital part of industrial, service and personal robotics, they are employed in various applications and environments, not just in advanced manufacturing automation as one may intuitively think.

Actuating a robotic limb is not merely commanding it to a given position: the issue is also where and how to move it, and for this purpose CV (the ability for machines to understand images) is a powerful tool that can assist humanoid robotics broadly. In particular, CV is used for robot manipulation tasks: reaching for something, touching or grasping it.

A major difficulty for humanoid robots in order to successfully perform a grasping task is the *variety of objects* they have to interact with: a robot should be able to see and understand any shape and size, including never-before-seen objects. To do this, deploying simple *model-free* methods (i.e., not enforcing any model) is often the sensible choice to follow, including in the approach presented in this thesis.

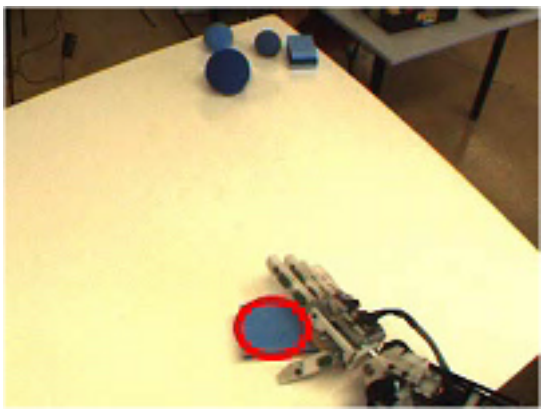


Figure 1.3: Humanoid robot Baltazar operating in its workspace, as seen from one of its cameras. The anthropomorphic hand of Baltazar is reaching for a visually-tracked object and is about to grasp it.

In particular, our objective is to approximate an object positioned in front of a humanoid robot with its smallest *enclosing ellipse*. In addition, the target object may not necessarily be static, so the developed techniques must also cope with following where this target is moving. This is done by the means of Continuously Adaptive Mean Shift (CAMSHIFT) trackers.

A computer vision technique to estimate the 3D position and orientation of a moving target object placed in front of a humanoid robot, equipped with a stereo rig, will be presented. The information inferred will subsequently be used by the robot to better *interact* with the object, that is, to manipulate it with its arm: an approach for real-time preparation of grasping tasks is described, based on the low-order moments of the target's shape on a stereo pair of images acquired by an active vision head.

To reach for an object, two distinct phases are considered [LSV07]:

1. an open-loop ballistic phase to bring the manipulator to the vicinity of the target, whenever the robot hand is not visible in the robot's cameras;

2. a closed-loop visually controlled phase to accomplish the final alignment to the grasping position.

The open-loop phase (reaching preparation) requires the knowledge of the robot's inverse kinematics and a 3D reconstruction of the target's posture. The target position is acquired by the camera system, where the hand position is measured by the robot arm joint encoders. Because these positions are measured by different sensory systems, the open-loop phase is subject to mechanical calibration errors. The second phase (grasping preparation), operates when the robot hand is in the visible workspace. 3D position and orientation of target and hand are estimated in a form suitable for Position-Based Visual Servoing (PBVS) [CH06, HHC96]. The goal is to make the hand align its posture with respect to the object. Since both target and hand postures are estimated in the same reference frame, this methodology is not prone to significant mechanical calibration errors.

1.4 Problem Statement

The objective is to estimate the 3D position and orientation of an object placed in front of a humanoid robot, in order for make it possible to interact and manipulate such object. This estimation is done in two visual processing steps: tracking the object shape as it (possibly) moves within the robot stereo cameras' 2D vision, then combining the inferred information through 3D reconstruction methods.

Computation must be real-time, so that the understanding of a dynamic scene and the interactions with objects are accurate but also usable in practical experiments. This constraint calls for a simplified, model-free vision approach. An object position and orientation will be approximated with its best-fit enclosing ellipse.

Furthermore, the employed software components must be clearly decoupled, in order to make it possible to adapt them to other robotic platforms and scenarios (such as a full iCub robot; see p. 3).

Finally, the geometric measurements acquired so far should to be used for the two phases of a reaching task, necessary for object manipulation: (i) an initial phase whereby the robot positions its hand close to the target with an appropriate hand orientation, and (ii) a final phase where a precise hand-to-target positioning is performed using Visual Servoing methods.

1.5 Thesis Structure

The thesis is organized as follows:

- Chapter 2 lists the existing techniques for segmentation, stereo vision (in particular 3D reconstruction) and manipulation tasks for humanoid robots;
- Chapter 3 describes the structure of the robot used in our development and experiments, "Baltazar", as well as the software libraries and tools that make up this work;
- Chapter 4 details the CAMSHIFT tracker implementation, the 3D reconstruction software module and the proposed manipulation approach;
- Chapter 5 displays the behaviour and results obtained with the developed work;
- Chapter 6 draws the concluding remarks and it outlines some possible ways to continue the research work started in this thesis.

Chapter 2

Related Work

Research in the field of robot manipulation in human-robot environments has emphasized the importance for a robot to constantly *sense* its environment, instead of referring to internal, predefined models. Results provided by the Edsinger Domo [Eds] upper torso, developed at MIT CSAIL [CSA], suggest to use sparse perceptual features to capture just those aspects of the world that are relevant to a given task. Many manipulation tasks can be designed and planned through the perception and control of these features, and the Domo system includes strategies to reduce the uncertainty that is inherent in human-robot scenarios, addressing the following aspects:

- generalization across objects within an object class;
- variability in lighting;
- cluttered backgrounds;
- no prior 3D models of objects or the environment.

Another important contribution was given by Tomita *et al.* [TYU+98]: the Versatile 3D Vision system "VVV" can construct the 3D geometric data of any scene when two or more images are given, by using structural analysis and partial pattern matching. It can also recognize objects, although under the assumption that their geometric models are known. This is a relevant difference from our proposed approach, which, instead, is free of geometric models. A general user of VVV can make a task-oriented system whose ability is at least the same of a specialized robotic system (one that was built for a very specific action).

Within this chapter we will summarize the techniques that are related to the modules that compose our proposed architecture. Section 2.1 will outline the main image segmentation techniques existing in literature; Section 2.2 will give a brief insight on stereo vision and 3D reconstruction; Section 2.3 will present the existing approaches for visual-based robot manipulator control; Section 2.4 will introduce the object affordances learning tool, to improve manipulation performances.

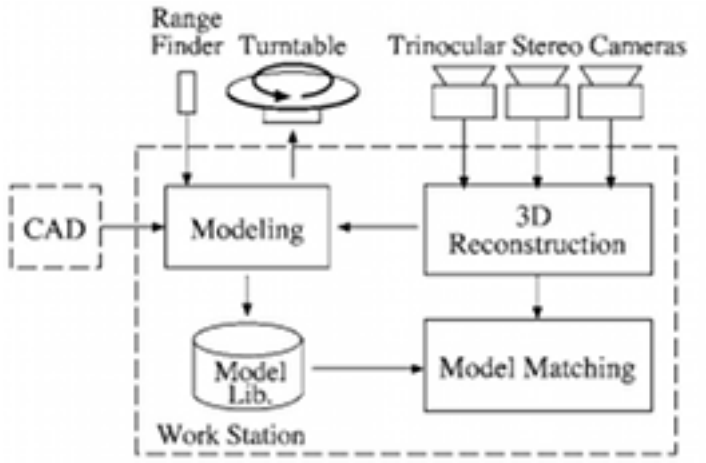


Figure 2.1: Block diagram of the "VVV" 3D vision robot system [SKYT02].

Figure 2.2: Why segmentation is difficult: these are three surfaces whose image information can be organized into meaningful assemblies easily and successfully by the human eye, but it is not straightforward to do so for machines.

2.1 Image Segmentation Techniques

The human eye is powerful because it is extremely *versatile*: it can recognize objects nearly instantly, it can follow (track) their motion, recognize the mood of living beings from their facial expressions, decide when a circumstance is dangerous, compute the number and speed of objects, and more. It is because of this breadth of skills that the human eye is *difficult to emulate*.

Recent CV systems, on the other hand, are normally highly specialized; furthermore, they usually work well only under certain hypotheses. Some relevant fields of research inside CV are tracking and image segmentation.

The purpose of image segmentation is to better organize the way we process a picture, by separating the interesting or relevant parts from those which are not useful for a given problem. Image segmentation, or simply "segmentation" for brevity, consists of dividing an image into two or more subset regions that cover it, according to a certain criterion, in order to simplify the representation and focus the attention on relevant parts of the image [FvDFH95].

One of the major difficulties in object recognition problems is knowing *which* pixels we have to recognize, and which to ignore [FP02]. For example, it is

difficult to tell whether a pixel lies on the surfaces in Fig. 2.2 simply by looking at the pixel. Specifically, it is difficult to tell whether a pixel lies on the three surfaces in the picture simply by looking at the pixel; the solution to the problem is to work with a *compact representation* of the "interesting" image data that emphasize relevant properties. Segmentation is the process of obtaining such compact representation [SS01].

The goal in many tasks is for the regions to represent *meaningful* areas of the image, such as crops, urban areas, and forests of a satellite image. Each of the resulting subsets contains those pixels of the image which satisfy a given condition: for instance, having the same colour, the same texture, or having rough edges. Furthermore, changing the representation of an image to something more meaningful means that it also becomes *easier to analyze* and process.

Practical applications of image segmentation include:

- recognizing a face or any other trait that is *salient* in a given situation;
- traffic controlling systems;
- medical imaging:
  - revealing, diagnosing and examining tumours;
  - studying other pathologies and anatomical structures;
  - measure the volume of tissues;
  - computer-guided surgery;
- locate objects in satellite images;
- fingerprint recognition.

Traditionally [SS01, ch. 10], segmentation has had two objectives:

1. to decompose the image into parts for further analysis;
2. to perform a change of representation.

The importance of the first objective is straightforward to understand and is discussed thoroughly in texts like [FvDFH95, FP02, SS01, TV98]. The second one, however, is more subtle: the pixels of an image must be organized into higher-level units that are either more meaningful or more efficient for further analysis, or both. A critical issue is whether or not segmentation can be performed for many different domains using bottom-up methods that do not use any special domain knowledge.

Since there is no general solution to the image segmentation problem, several different approaches to implement segmentation have been proposed in literature, each having its advantages and drawbacks. A brief list of the most common techniques will now be given. Further down, the CAMSHIFT algorithm, which belongs to the class of histogram-based segmentation approaches (see Section 2.1.4) and is the technique adopted in this thesis, will be detailed.

2.1.1 Clustering Segmentation

In pattern recognition, *clustering* is the process of partitioning a set of pattern vectors into subsets called clusters [TK03]. Several types of clustering algorithms have been found useful in image segmentation.

One way to look at the segmentation problem is thus to attempt to determine which components of a data set naturally "belong together" in a cluster. Generally, two approaches for clustering are considered in literature [FP02]:

partitioning: carving up a large data set according to some notion of the association between items inside the set, i.e., decomposing the set into pieces that are good with regards to a model. For example:

- decomposing an image into regions that have coherent colour and texture inside them;
- decomposing an image into extended blobs, consisting of regions that have coherent colour, texture and motion and look like limb segments;
- decomposing a video sequence into shots—segments of video showing about the same scene from about the same viewpoint.

grouping: starting from a set of distinct data items, collect sets of these items that make sense *together* according to a model. Effects like occlusion mean that image components that belong to the same object are often separated. Examples of grouping include:

- collecting together tokens that, when taken together, form a line;
- collecting together tokens that seem to share a fundamental matrix.

The key issue in clustering is to determine a suitable representation for the problem in hand.

A famous clustering algorithm is k-Means, first described in [Mac67]. Basic pseudocode is shown in Algorithm 1. k-Means is an iterative technique to partition an image into k clusters. The algorithm works by randomly selecting centroids, finding out which elements are closest to the centroid, then working out the mean of the points belonging to each centroid, which becomes the new centroid. Region membership is checked again, and the new centroids are computed again. This operation continues until there are no points that change their region membership.

Algorithm 1 Basic k-Means

1: Place k points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2: Assign each object to the group that has the closest centroid.
3: When all objects have been assigned, recalculate the positions of the k centroids.
4: Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

While the k-Means algorithm is guaranteed to converge, it has a few drawbacks:

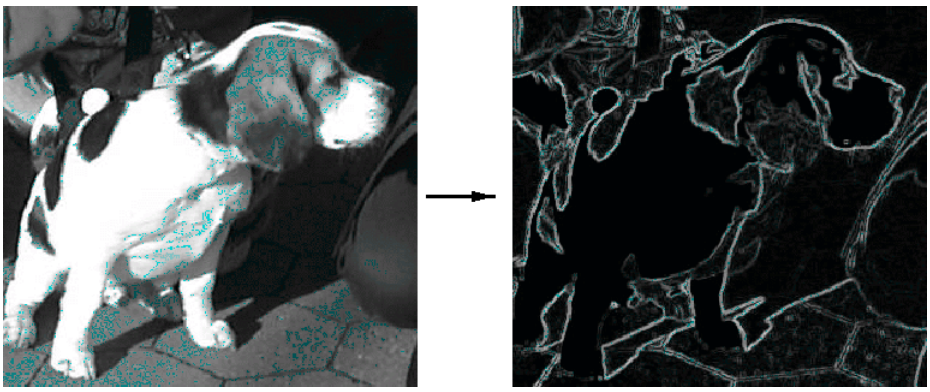


Figure 2.3: An example of edge-based segmentation, showing the computed edges of the left input image on the right.

- it may occur that k-Means converges to a local, possibly not global, solution;
- the algorithm is significantly sensitive to the initial randomly-selected cluster centres (to reduce this effect, one can execute k-Means multiple times, but this is costly);
- basic k-Means relies on the assumption that the number of clusters k is known in advance. Alternatives have been proposed to overcome this limitation of the initial setting, for example "Intelligent k-Means" by Mirkin [Mir05, p. 93].

2.1.2 Edge Detection Segmentation

Edge points, or simply "edges", are pixels at or around which the image values undergo a sharp variation.

Edge detection techniques have been used as the base of several segmentation techniques, exploiting the tight relationship that exists between region boundaries and edges—the sharp adjustment in intensity at the region boundaries, as in Fig. 2.3.

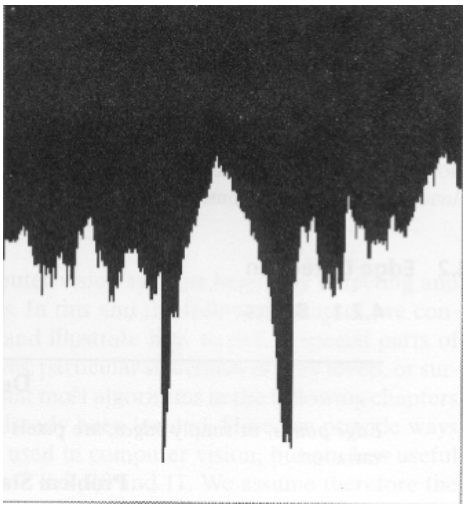
The edge detection problem can be formulated like this: given an image that has been corrupted by acquisition noise, locate the edges most likely to be generated by scene elements (not by noise). Fig. 2.4 shows an example of edge detection computations.

Though, the edges identified by edge detection are often disconnected. To segment an object from an image, however, one needs closed region boundaries. Discontinuities are normally bridged if the distance between the two edges is within some predetermined threshold.

The Canny edge detection algorithm [Can86] is known as the optimal edge detector. It follows a list of three criteria with the aim of improving previous methods of edge detection:

good detection: the algorithm should mark as many real edges in the image as possible. This is the first and most obvious principle, that of low error

(a) A 325×237-pixel image, with scanline $i = 56$ (over the baby's forehead) highlighted.



(b) Intensity profile along the highlighted scanline.

Figure 2.4: An intensity image (left) and the intensity profile along a selected scanline (right). The main sharp variations correspond to significant contours.

rate. It is important that edges occurring in images should not be missed and that there be no responses to non-edges.

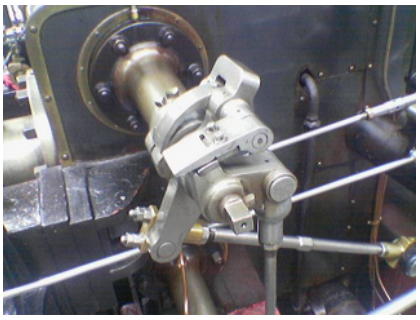
good localization: the edges marked should be as close as possible to the edges in the real image. This second criterion suggests that the edge points should be well localized, in other words, that the distance between the edge pixels as found by the detector and the actual edge is to be at a minimum.

minimal response: a given edge in the image should only be marked once, and, where possible, image noise should not create false edges. The third criterion is thus to have only one response to a single edge; this was implemented because the first two criteria were not sufficient to completely eliminate the possibility of multiple responses to an edge.

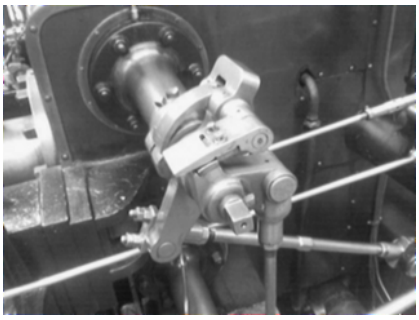
To satisfy these requirements, the calculus of variations (a technique which finds the function which optimizes a given functional) is employed [Can86]. The optimal function in Canny's detector is described by the sum of four exponential terms, but can be approximated by the first derivative of a Gaussian. See Fig. 2.5 for an example.

2.1.3 Graph Partitioning Segmentation

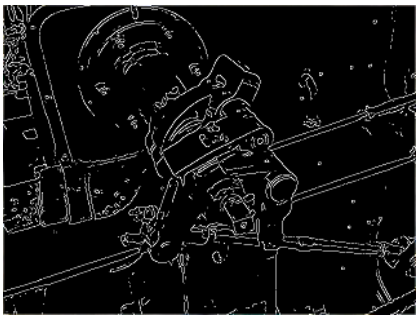
Various algorithms exist in this class of methods, all of which model (groups of) pixels as vertices of a graph, while graph edges define the correlation or similarity among neighbouring pixels.



(a) Input image: a colour photograph of a steam engine.



(b) After having passed a 5×5 Gaussian mask across each pixel for noise reduction, the input image becomes slightly blurred.



(c) Result of Canny edge detector.

Figure 2.5: Canny edge detector example.

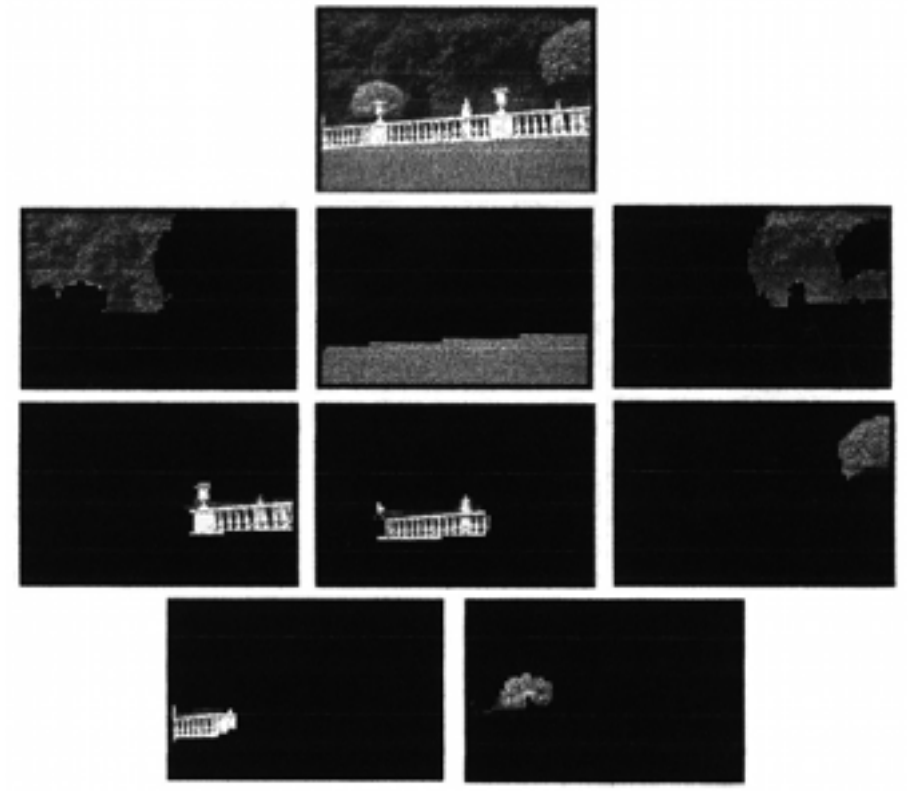


Figure 2.6: The image on top being segmented using the normalized cuts framework into the components shown below. The affinity measure used involved both intensity and texture. Thanks to having a texture measure, the railing shows as three reasonable coherent segments, which would not have happened with other approaches such as k-Means.

Popular graph partitioning segmentation techniques include: normalized cuts, random walker, minimum mean cut, and minimum spanning tree-based algorithms.

Originally proposed by Shi and Malik [SM97], the normalized cuts method involves the modelling of the input image as a weighted, undirected graph. Each pixel is a node in the graph, and an edge is formed between every pair of pixels; the weight of an edge is a measure of the *similarity* between the pixels.

The image is partitioned into disjoint sets (called "segments"; see Fig. 2.6) by removing the edges that connect the segments. The optimal partitioning of the graph is the one that minimizes the weights of the edges that were removed (the "cut"). The algorithm seeks to minimize the "normalized cut", which is the ratio of the cut to all of the edges in the set.

2.1.4 Histogram-Based Segmentation

In this class of techniques, a histogram is computed from *all* of the image pixels, taking into account colour or intensity values. Then, the peaks and valleys of

such histogram are used to locate meaningful clusters in the image. By doing so, clusters are directly obtained after building the histogram.

See Fig. 2.7 for a generic scheme of these approaches. X, Y, and Area are relative to the colour probability distribution representing the tracked object (compare Section 4.2). Area is proportional to Z, i.e., the distance from the camera. Roll (inclination) is also tracked, as a fourth degree of freedom. For each video frame, the raw image is first converted to a colour probability distribution image via a colour histogram model of the colour being tracked (e.g., flesh for face tracking). The centre and size of the colour object are found via the CAMSHIFT algorithm operating on the colour probability image. The current size and location of the tracked object are reported and used to set the size and location of the search window in the next video image. This process is iterated for continuous tracking.

Histogram-based segmentation methods present an important advantage compared to other techniques: *high efficiency*. Typically, these techniques require only one pass through the image pixels. For this reason, we will choose the CAMSHIFT algorithm, which belongs to this class, because we require a fast, simple, efficient tracker in our perceptual humanoid robotics framework.

Building the histogram is a critical phase; as mentioned above, one can choose different types of measures like colour or intensity. This fact is important when envisioning a perceptual framework [Bra98], as is the case of our project and humanoid robotics.

A Perceptual User Interface (PUI) is one in which a machine is given the ability to send and produce analogs of human senses, such as allowing computers to perceive and produce localized sound and speech, giving robots a sense of touch and force feedback, or the ability to see.

2.1.5 Level Set Segmentation

In general, level set segmentation is a method for tracking the evolution of contours and surfaces. Originally proposed in [OS88], this technique uses Propagation of Surfaces under Curvature (PSC) schemes. The idea is to move surfaces under their curvature, propagating the surfaces towards the lowest potential of a cost function.

This framework has several advantages:

- level sets yield a useful representation of regions and their boundaries on the pixel grid without the need of complex (and costly) data structures. Therefore, optimization is simplified, as variational methods and standard numerics can be employed;

- level sets can describe topological changes in the segmentation, i.e., parts of a region can split and merge;

- it is possible to describe the image segmentation problem with a variational model, thus increasing flexibility (and permitting the introduction of additional features, shape knowledge, or joint motion estimation and segmentation).

On the other hand, level set segmentation has a problem: a level set function is restricted to the separation of only two regions. Brox and Weickert [BW04]

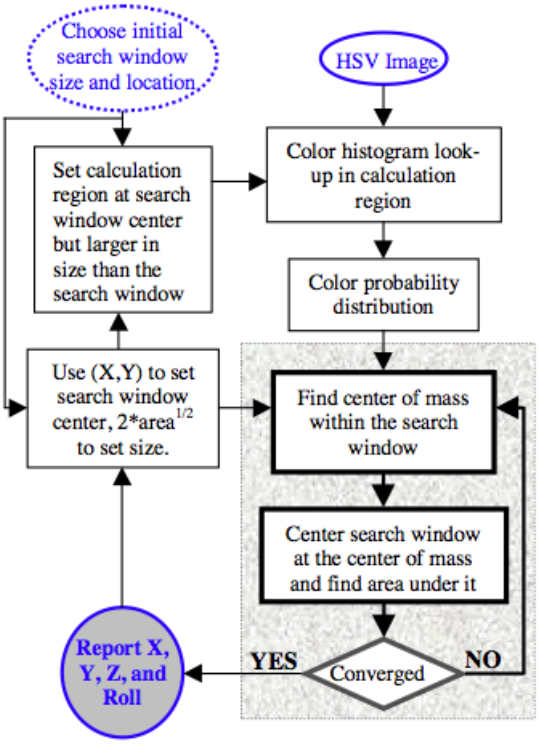
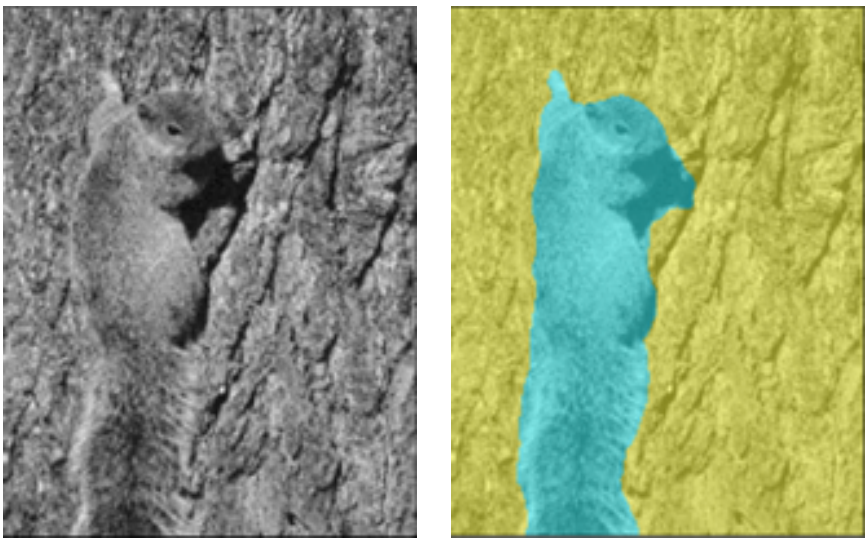


Figure 2.7: Block diagram of histogram-based object tracking. The grey box is the Mean Shift algorithm.



(a) Input image. (b) Level set segmentation.

Figure 2.8: Level set segmentation of a squirrel image: two regions have been detected.

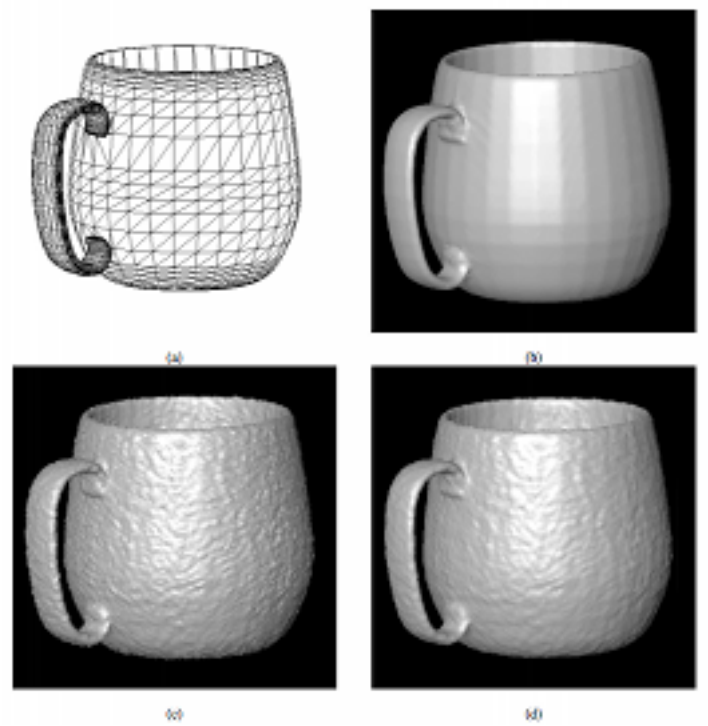


Figure 2.9: Level set based 3D reconstruction of a mug, using synthetic data generated from a 3D model (a). Without noise, the reconstruction (b) is limited only by the resolution of the model, $140 \times 140 \times 140$. With noise, the surface appears rough (c). Including a prior improves the appearance of the reconstruction (d).

proposed a new formulation of the potential function to minimize, taking into account the number of regions, too.

What is more, level set segmentation is well suited to generate 3D reconstructions of objects [Whi98]. See Fig. 2.9 for a sample run of Whitaker's algorithm. The strategy applied is as follows: construct a rather coarse volume that is the solution to a linear problem, i.e., the zero-level sets of a function, without the prior. This volume will serve as initialization for a level set model which moves towards the data given by range maps while undergoing a second-order flow to enforce the prior. After the rate of deformation slows to below some predefined threshold, the resolution is increased, the volume resampled, and the process repeated (in an attempt to avoid convergence to local minima).

Note that this last strategy employs predefined models of shapes: the approach has much to share with model-based segmentation methods, explained below.

2.1.6 Model-Based Segmentation

Model-based segmentation approaches (or knowledge-based segmentation), commonly adopted in medical imaging, rely on the assumption that structures of

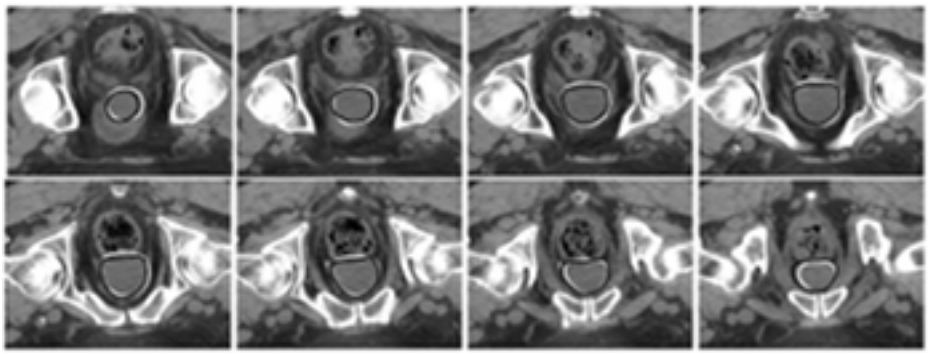


Figure 2.10: Model based segmentation results using a prostate model, to detect carcinoma cell masses. The white contour shows the result at convergence; the black contour shows the hand-drawn ground-truth contours supplied by a radiation oncologist.

interest have a repetitive form of geometry.

State of the art methods in the literature for knowledge-based segmentation [FRZ+05] involve active shape and appearance models, active contours and deformable templates (see Fig. 2.10 for an example). Note that there is an intersection with level set segmentation methods (refer to Section 2.1.5).

One can seek a probabilistic model that explains the variation of the shape, for instance, of an organ and then, when segmenting an image, impose constraints using this model as a prior. Specifically, such a task involves:

1. registration[1] of the training examples to a common pose;

2. probabilistic representation of the variation of the registered samples; and

3. statistical inference between the model and the image.

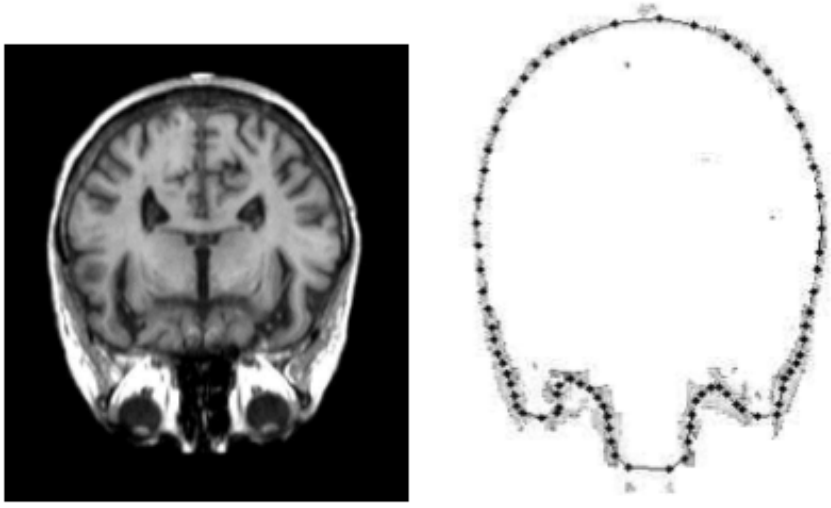
So, these algorithms are based on matching probability distributions of photometric variables that incorporate learned shape and appearance models for the objects of interest. The main innovation is that there is no need to compute a pixel-wise correspondence between model and image. This allow for fast, principled methods.

2.1.7 Neural Networks Segmentation

Neural network image segmentation typically relies on processing *small areas* of an image using an unsupervised neural network (a network where there is no external teacher affecting the classification phase) or a set of neural networks.

After such processing is completed, the decision-making mechanism marks the areas of an image accordingly to the category recognized by the neural network, as exemplified in Fig. 2.11. A type of network well designed for these purposes [RAA00] is the Kohonen Self-Organizing Map (SOM).

[1] Registration fits models that are previously known; 3D reconstruction extracts models from images.



(a) Human head Magnetic Resonance input image.

(b) Neural Networks segmentation of MR image.

Figure 2.11: Neural Networks segmentation.

2.1.8 Region Growing Thresholding Segmentation

Just like edge detection (see Section 2.1.2) is implemented by quite different processes in photographs and range data, segmenting image into *regions* presents a similar situation.

Region growing is an approach to image segmentation in which neighbouring pixels are examined and added to a region class if no edges are detected [FP02]. This process is iterated for each boundary pixel in the region. If adjacent regions are found, region-merging techniques are used in which weak edges are dissolved and strong edges are left intact.

This method offers several advantages over other techniques:

- unlike edge detection methods (such as gradient and Laplacian), the borders of regions found by region growing are thin –since one only adds pixels to the exterior of regions– and connected;

- the algorithm is stable with respect to noise: the resulting region will never contain too much of the background, as long as the parameters are defined correctly;

- membership in a region can be based on multiple criteria, allowing us to take advantage of several image properties, such as low gradient or gray level intensity value, at once.

There are, however, disadvantages to region growing. First and foremost, it is very *expensive* computationally: it takes both serious computing power (processing power and memory usage) and a decent amount of time to implement and run the algorithms efficiently.

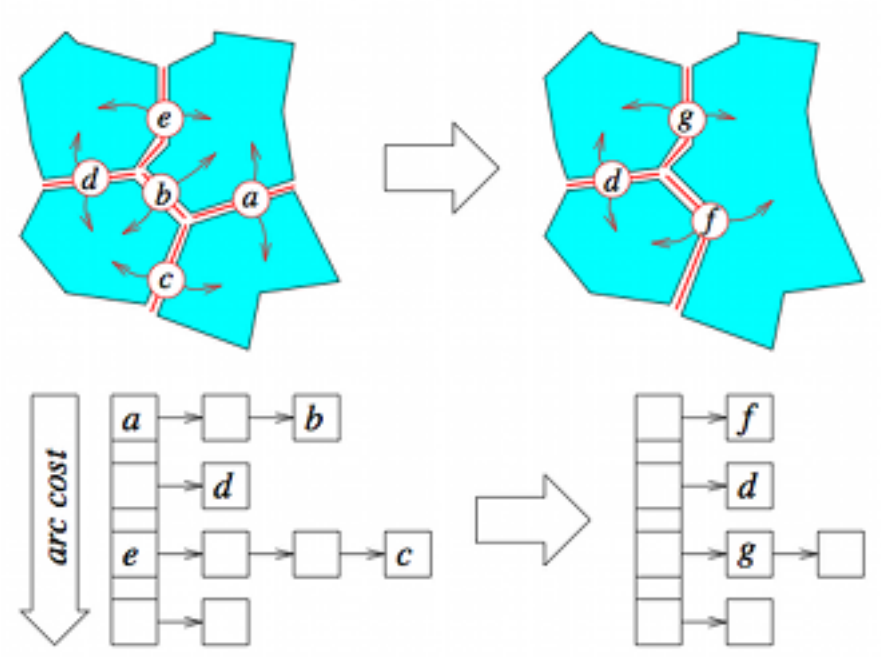


Figure 2.12: One iteration of the region growing process during which the two patches incident on the minimum-cost arc labelled a are merged. The heap shown in the bottom part of the figure is updated as well (which bears a considerable computational cost): the arcs a, b, c and e are deleted, while two new arcs f and g are created and inserted in the heap.

An example of region growing thresholding is [FH86]. This algorithm iteratively merges planar patches by maintaining a graph whose nodes are the patches and whose arcs (edges) are associated with their common boundary link adjacent patches. Each arc is assigned a cost, corresponding to the average error between the points of the two patches and the plane best fitting these points. The best arc is always selected, and the corresponding patches are merged. Note that the remaining arcs associated with these patches must be deleted while new arcs linking the new patch to its neighbors are introduced. The situation is illustrated by Fig. 2.12.

2.1.9 Scale-Space Segmentation

Scale-space segmentation, also known as multi-scale segmentation, is based on the computation of image descriptors at multiple scales of smoothing. It is a general technique used for signal and image segmentation (see Fig. 2.13 and Fig. 2.14).

The main type of scale-space is the linear (Gaussian) scale-space, which has wide applicability as well as the attractive property of being possible to derive from a small set of scale-space axioms. The corresponding scale-space framework encompasses a theory for Gaussian derivative operators, which can be used as a basis for expressing a large class of visual operations for computerized systems

(a) $t = 0$, corresponding to the original image f.

(b) $t = 1$.

(c) $t = 4$.

(d) $t = 16$.

Figure 2.13: Space-scale representation $L(x, y; t)$ for various t scales. As the parameter t increases above 0, L is the result of smoothing f with a larger and larger filter.

that process visual information. This framework also allows visual operations to be made scale-invariant, which is necessary for dealing with the size variations that may occur in image data, because real-world objects may be of different sizes and in addition the distance between the object and the camera may be unknown and may vary depending on the circumstances.

For a two-dimensional image $f(x, y)$, its linear (Gaussian) scale-space *representation* is a family of derived signals $L(x, y; t)$ defined by the convolution of $f(x, y)$ with the Gaussian kernel

$$g_t(x, y) = \frac{1}{2\pi t} e^{-(x^2+y^2)/2t} \tag{2.1}$$

such that

$$L(x, y; t) = (g_t * f)(x, y), \tag{2.2}$$

where the semicolon in the argument of L implies that the convolution is performed only over the variables x, y, while the scale parameter t after the semicolon just indicates which scale level is being defined. This definition of L works for a continuum of scales , but typically only a finite discrete set of levels in the scale-space representation would be considered.

In Fig. 2.14b, each "x" identifies the position of an extremum of the first derivative of one of 15 smoothed versions of the signal (red for maxima, blue for

minima). Each "+" identifies the position that the extremum tracks back to at the finest scale. The signal features that persist to the highest scale (smoothest version) are evident as the tall structures that correspond to the major segment boundaries in the figure above.

2.1.10 Semi-Automatic Livewire Segmentation

In this segmentation method, the user outlines the Region of Interest (ROI) with mouse clicks, then an algorithm is applied so that the path that best fits the edge of the image is shown. It is based on the lowest cost path algorithm by Dijkstra.

The user sets the starting point clicking on an image pixel. Then, as he starts to move the mouse over other points, the smallest cost path is drawn from the starting point to the pixel where the mouse is over, changing itself if the user moves the mouse. If the user wants to choose the path that is being displayed, he will simply click the image again.

One can easily see in Fig. 2.15 that the places where the user clicked to outline the desired ROI are marked with a small square. It is also easy to see that Livewire has snapped on the image borders.

2.2 Stereopsis

Since a considerable portion of this thesis work deals with how a humanoid robot can perceive object positions and orientations in 3D space by using a binocular head (see Section 4.3), some introductory theoretical background is first due.

Stereopsis, also known as stereo vision or simply "stereo", allows two-dimensional images to be interpreted in terms of 3D scene structure and distance [TV98].

Humans have an uncanny ability to perceive and analyze the structure of the 3D world from visual input, operating effortlessly and with little or no idea of what the mechanisms of visual perception are.

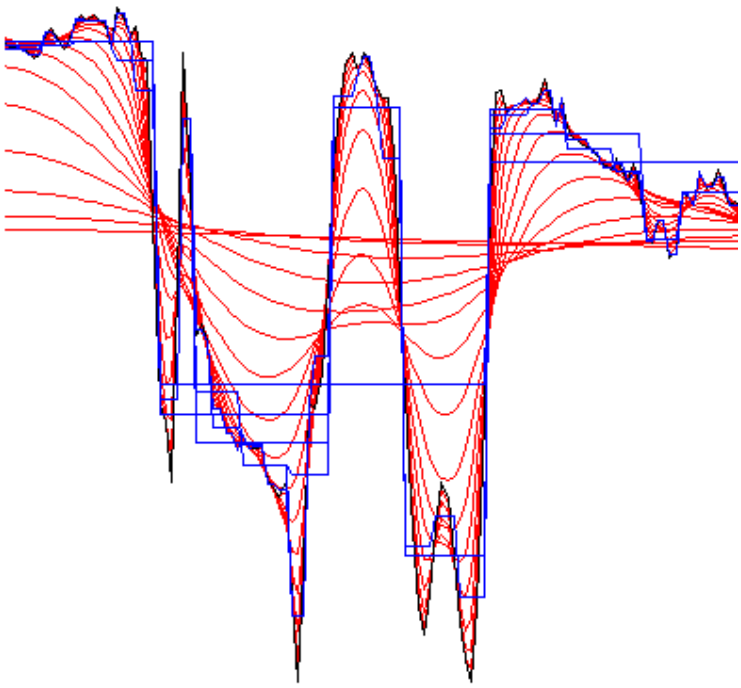
Depending on the nature of the features we wish to observe (2D or 3D, points or lines on the surface of the object, etc.), different formulations and algorithms come into play. However, the underlying mathematics has much in common: all the different cases can be formulated in such a way that they require solutions of simultaneous transcendental, polynomial, or linear equations in multiple variables which represent the structure of the object and its 3D motion as characterized by rotation and translation.

In particular, what is inferred is a sensation of depth from the two slightly different projections of the world onto the retinas of the two eyes. The differences in the two retinal images are called horizontal disparity, retinal disparity, or binocular disparity. The differences arise from the eyes' different positions in the head.

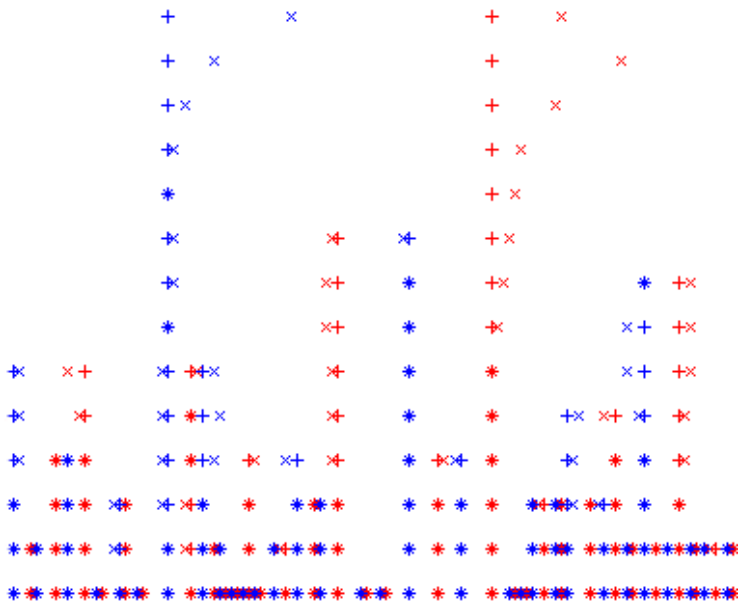
Stereo vision involves two processes:

- the binocular fusion of features observed by the two eyes;
- the actual reconstruction of the features observed in the world.

They can be translated into two problems:



(a) A signal (black), various multi-scale smoothed versions of it (red) and some segment averages (blue).



(b) Dendrogram resulting from the segmentation in Fig. 2.14a.

Figure 2.14: Space-scale segmentation example.

Figure 2.15: Example run of a semi-automatic Livewire technique applied on a picture.

correspondence: which parts of the left and right images are projections of the same scene element?

reconstruction: given a number of corresponding parts of the left and right image, and possibly information on the geometry of the stereo system, what can we say about the 3D location and structure of the observed objects?

The correspondence problem is out of the scope of this project. In our proposed approach in Section 4.3, we will focus on 3D reconstruction.

2.3 Object Manipulation with Visual Servoing

Existing visual-based robot control approaches [CH06, CH07, HHC96], summarized below, solve the issue of representing a gripper–object relationship by handling *models* of gripper and object in memory. This approach, while accurate and powerful, presents two drawbacks:

- the object model may be poor; besides, in several circumstances it may not be available at all (as addressed by Malis and Chaumette [MC02] or by Dufournaud *et al.* [DHQ98]);
- computational cost is high due to the manipulation program having to memorize and compute comparison operations to such models.

Visual Servoing [Cor97] is a multi-disciplinary approach to the control of robots based on visual perception, involving the use of cameras to control the position of the robot relative to the environment as required by the task. This technique uses visual feedback information extracted from a vision sensor, to

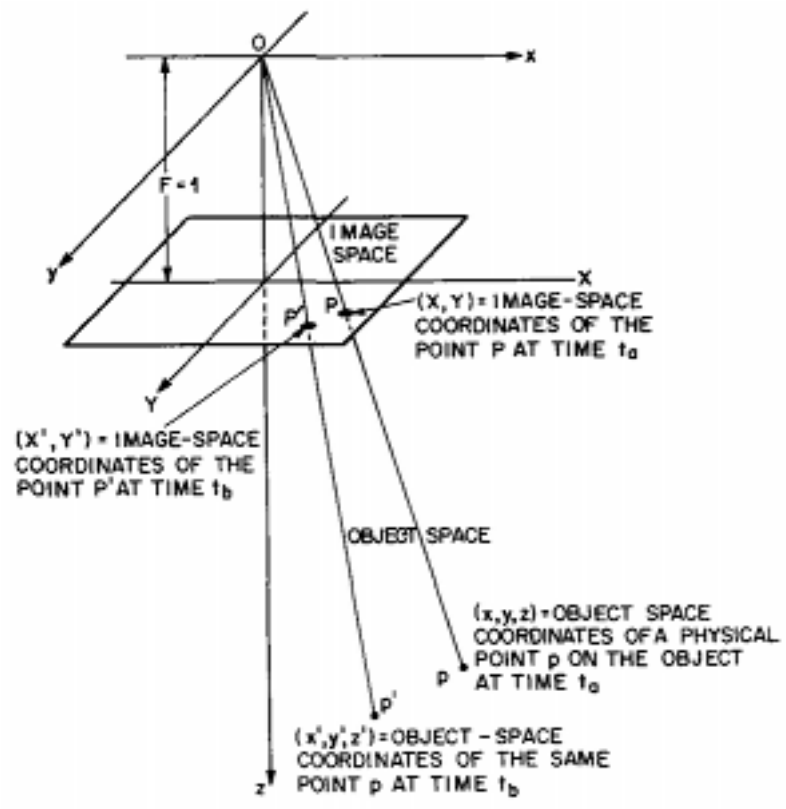


Figure 2.16: Basic perspective geometry for imaging. Lower case letters refer to coordinates in the object space, upper case letters to coordinates on the image plane. Focal length (denoted here with F) is assumed to be 1.

control the motion of robots. This discipline spans CV, robotics, control, and real-time systems.

The task in Visual Servoing is to control the pose (3D position and orientation) of a robot's end-effector, using visual information (features) extracted from images.

Visual Servoing methods are commonly classified as image-based or position-based[2], depending on whether image features or the camera position define the signal error in the feedback loop of the control law.

2.3.1 Image-Based Visual Servoing

Image-Based Visual Servoing (IBVS) is a feature-based technique, meaning that it employs features that have been extracted from the image to directly provide a command to the robot (without any computation by the robot controller). Typically for IBVS, all the information extracted from the image features and used in control, occurs in a 2D space. In most cases this coincides with the image coordinates' space. Despite this 2D information, because of which the approach is also known as "2D servoing control", the robot still has the capability to move in 3D.

IBVS involves the estimation of the robot's velocity screw, $\dot{\mathbf{q}}$, so as to move the image plane features, $\mathbf{f}^c$, to a set of desired locations, $\mathbf{f}^*$ [MC02]. IBVS requires the computation of the *image Jacobian* (or *interaction matrix*). The image Jacobian represents the differential relationships between the scene frame and the camera frame (where either the scene or the camera frame is attached to the robot):

$$\mathbf{J}(\mathbf{q}) = \left[\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right] = \begin{bmatrix} \frac{\partial f_1(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial f_1(\mathbf{q})}{\partial q_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k(\mathbf{q})}{\partial q_1} & \cdots & \frac{\partial f_k(\mathbf{q})}{\partial q_m} \end{bmatrix} \tag{2.3}$$

where $\mathbf{q}$ represents the coordinates of the end-effector in some parameterization of the task space $\mathcal{T}$, $\mathbf{f} = [f_1, f_2, \ldots, f_k]$ represents a vector of image features, m is the cardinality of the task space $\mathcal{T}$, and k is the number of image features.

2.3.2 Position-Based Visual Servoing

PBVS is traditionally a model-based technique. The pose of the object of interest is estimated with respect to the camera frame, then a command is issued to the robot controller, which in turn controls the robot. In this case, the image features are extracted as well, like in IBVS, though the feature information is used to estimate the 3D object pose information in Cartesian space.

PBVS is usually referred to as "3D servoing control", since image measurements are used to determine the pose of the target with respect to the camera and some common world frame. The error between the current and the desired pose of the target is defined in the task (Cartesian) space of the robot; hence, the error is a function of pose parameters, $\mathbf{e}(\mathbf{x})$. Fig 2.17 shows possible example tasks of PBVS.

[2]Some "hybrid approaches" have also been proposed: 2-1/2-D Servoing, Motion Partition Based Servoing, and Partitioned DOF Based Servoing.

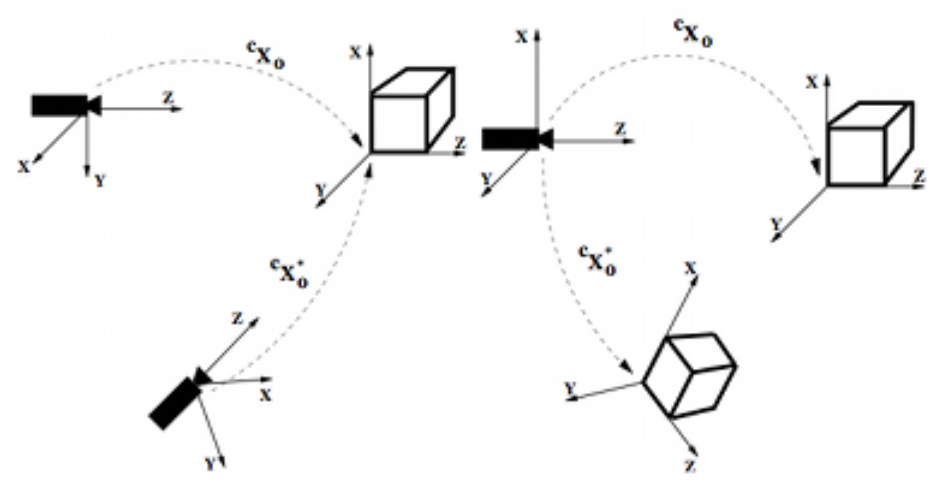


Figure 2.17: Two examples of PBVS control. Left: eye-in-hand camera configuration, where the camera/robot is servoed from ${}^c\mathbf{x}_0$ (current pose) to ${}^c\mathbf{x}_0^*$ (desired pose). Right: a monocular, standalone camera system used to servo a robot-held object from its current to the desired pose.

Fig 2.18 illustrates the general working scheme of PBVS, where the difference in pose between the desired and the current pose represents an *error* which is then used to estimate the velocity screw for the robot, $\dot{\mathbf{q}} = [\mathbf{V}; \boldsymbol{\Omega}]^T$, in order to minimize that error.

2.4 Object Affordances

Object affordances, or simply "affordances" for brevity, are a way to encode the relationships among actions, objects and resulting effects [MLBSV08].

The general tool adopted to capture the dependencies in affordances (see Fig. 2.19) is that of Bayesian networks. Affordances make it possible to infer

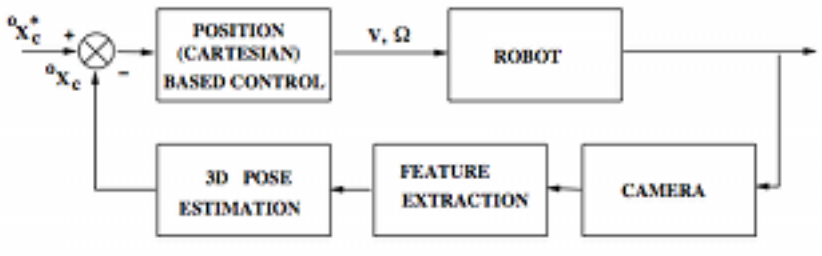


Figure 2.18: Block diagram of PBVS [HHC96]. The estimated pose of the target, ${}^c\mathbf{x}_0$, is compared to the desired reference pose, ${}^c\mathbf{x}_0^*$. This is then used to estimate the velocity screw, $\dot{\mathbf{q}} = [\mathbf{V}; \boldsymbol{\Omega}]^T$, for the robot so to minimize the error.

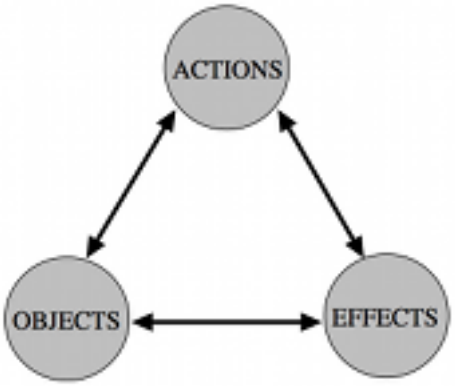


Figure 2.19: Object affordances represent the relationships that take place among actions (A), objects (O) and effects (E).

Table 2.1: Object affordances can be used for different purposes: to predict the outcome of an action, to plan the necessary actions to achieve a goal, or to recognize objects/actions.

input	output	function
(O, A)	E	predict effect
(O, E)	A	action recognition and planning
(A, E)	O	object recognition and selection

causality relationships by taking advantage of the intervention of a robot and the temporal ordering of events. Table 2.1 lists the basic purposes of affordances.

Contrary to similar approaches, in the affordances framework the dependencies shown in Fig. 2.19 are *not* known in advance (in which case we would learn a mapping between paris of actions and objects, or use supervised learning). Not assuming any prior knowledge on these dependencies, with affordances we try to infer the graph of the network directly from the exteroceptive and proprioceptive measurements. In addition, the affordances model allows the robot to tune the free parameters of the controllers.

This framework combines well with a developmental architecture whereby the robot incrementally develops its skills. In this sense, affordances can be seen as a bridge between

- sensory–motor coordination, and
- world understanding and imitation.

Results on the learning and usefulness of object affordances for robots that use monocular vision (one camera only) are discussed in [MLBSV08]. With the (future) work of this thesis we intend to combine affordances with information obtained from stereo vision.

Chapter 3

Robot Platform and Software Setup

In 2004, Computer and Robot Vision Laboratory [Vis] at IST in Lisbon developed a humanoid robot platform, "Baltazar" [LBPSV04], which was used for this work. Baltazar, shown in Fig. 3.1, is an anthropomorphic torso that features a binocular head as well as an arm and a hand. It was built as a system that mimics human arm-hand kinematics as closely as possible, despite the relatively simple design.

Baltazar is well suited (and was designed) for research in imitation, skill transfer and visuomotor coordination. The design of Baltazar was driven by these constraints:

- the robot should resemble a human torso;
- the robot kinematics should be able to perform human-like movements and gestures, as well as to allow a natural interaction with objects during grasping;
- payload should be at least 500 g (including the hand);
- force detection should be possible;
- the robot should be easy to maintain and be low cost (it contains regular DC motors with reduced backlash and off-the-shelf mechanical parts).

In this section we will summarize mechanical and kinematic details of Baltazar, its sensors and technology. More details (such as the design of the 11-DOF hand of Baltazar, not actually used in this thesis work as we focus on grasping *preparation*) can be found in [Lop06, p. 113].

A software library used to develop programs for Baltazar, called Yet Another Robot Platform (YARP), is used extensively in the whole RobotCub developers community and in the implementation of this thesis, therefore YARP and its middleware mechanisms will also be described; other secondary software tools that we used will be cited.

Figure 3.1: Baltazar humanoid robot in relax position.

3.1 Kinematic Description of Baltazar

Robot kinematics studies the *motion* of robots, thus taking into account the positions, velocities and accelerations of robot links, but without regard to the forces that actually cause the motion (whose study belongs to robot dynamics): robot kinematics uses just geometrical constraints. The kinematics of manipulators involves the study of the geometric- and time-based properties of motion, in particular how the various links move with respect to one another and with time.

The vast majority of robots belong to the *serial* link manipulator class, which means that it comprises a set of bodies called *links* in a chain, connected by *joints*[1]. Each joint has one DOF, either translational or rotational; for example, the anthropomorphic arm used for the work of this thesis has 6 rotational DOFs.

For a manipulator with n joints numbered from 1 to n, there are $n+1$ links, numbered from 0 to n. Link 0 is the base of the manipulator, while link n carries the end-effector. Joint i connects links i and $i-1$.

The kinematic model of a robot expresses how its several components move among themselves, achieving a transformation between different configuration spaces. The "spaces" mentioned here are the ones of the Cartesian geometric world workspace, as opposed to less-intuitive spaces that are directly associated with the robot's joint parameters, which are usually [Cra05, SS00] denoted as a vector $\mathbf{q}$.

The following types of kinematics approaches are commonly studied in robotics:

forward kinematics computes the position of a point in space (typically, that of the end-effector), given the values of the joint parameters (lengths and angles);

inverse kinematics computes all the joint parameters, given a point in space that the end-effector must lie on;

forward velocity kinematics (or forward differential kinematics) computes the velocity of a point in space, given the derivatives of the joint parameters;

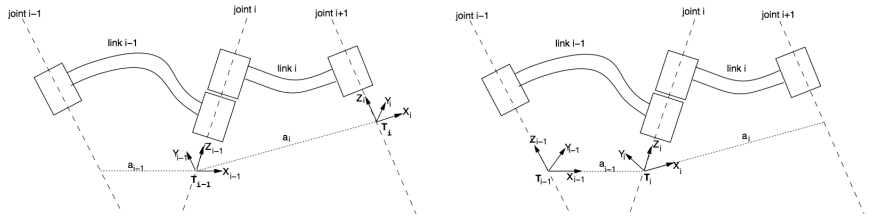
inverse velocity kinematics (or inverse differential kinematics) computes the derivatives, i.e., velocities of joint parameters, given spatial velocities.

Forward kinematics (also known as direct kinematics) is the problem of transforming the joint positions of a robot to its end-effector pose. In other words, it is the computation of the position and orientation of a robot's end-effector as a function of its joint angles. For example, given a serial chain of n links and letting θ_i be the angle of link i, then the reference frame of link n relative to link 0 is

$$^0\mathbf{T}_n = \prod_{i=1}^{n} {}^{i-1}\mathbf{T}_i(\theta_i)$$

where ${}^{i-1}\mathbf{T}_i(\theta_i)$ is the transformation matrix from the frame of link i to that of link $i-1$.

[1] To be more accurate, parallel link and serial/parallel hybrid structures are theoretically possible, although they are not common.



(a) Standard Denavit-Hartenberg (SDH) convention.

(b) Modified Denavit-Hartenberg (MDH) convention.

Figure 3.2: Different forms of DH notation. Note: a_i always indicates the length of link i, but the displacement it represents is between the origins of frame i and frame $i+1$ in the standard form, between frames $i-1$ and i in the modified form.

3.1.1 Kinematic Notation

A word of advice on the study and notation of kinematics: in robotics literature, at least two related but different conventions to model serial manipulator kinematics go by the name DH, however they actually vary in a few details related to the assignment of reference frames to the rigid bodies (links) of robots.

These differences among DH parameterizations are rarely acknowledged (with the exception of [Cor96]). Typically, an author chooses one of the existing DH notations, writes down "this is the Denavit-Hartenberg convention" then sticks to it from that moment on. One should, however, pay attention to which DH formulation is being used and understand it.

Two different methodologies, shown in Fig. 3.2, have been established to assign coordinate frames:

1. Standard Denavit-Hartenberg (SDH) form: frame i has its origin along the axis of joint $i+1$.

2. Modified Denavit-Hartenberg (MDH) form, also known as "unmodified" DH convention: frame i has its origin along the axis of joint i.

MDH is commonly used in the literature on manipulator mechanics, and the (forward and inverse) kinematics approaches for Baltazar that exist so far, have in fact followed the MDH form. However, further on we will see cases where it is practical to make a change of representation to SDH to model the arm of Baltazar by the means of publicly-available robotics simulation tools.

Difference between SDH and MDH.

The difference between SDH and MDH is the following. In SDH, we position the origin of frame i along the axis of joint $i+1$. With MDH, instead, frame i has its origin along the axis of joint i.

A point to stress is that the choices of the various reference frames to assign (with SDH or with MDH) are *not unique*, even under the constraints that need to be enforced among consecutive links. For example, the origin of the first reference frame O_0 can be arbitrarily positioned anywhere on the first joint axis. Thus, it is possible to derive different, equally valid, coordinate frame assignments for the links of a given robot. On the other hand, the final matrix

Table 3.1: Joint angles of Baltazar robotic head.

angle	description
θ_l	left eye camera vergence
θ_r	right eye camera vergence
θ_p	pan (neck rotation)
θ_t	tilt (head rotation)

that transforms from the base to the end effector, ${}^{n}\mathbf{T}_0$, must be the same—regardless of the intermediate frame assignments.

For a detailed description of DH conventions and the meaning of the four parameters (a: link length; α: link twist; d: link offset; θ: joint angle) refer, for example, to:

- [SHM05], which explains SDH thoroughly; Chapter 3 is publicly available[2];
- [Cra05] for MDH; or
- [Cor96] (both parameterizations).

If we use the SDH representation, the following 4×4 homogeneous transformation matrix

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

represents each link's coordinate frame with respect to the previous link's coordinate system, i.e.,

$$ {}^{0}\mathbf{T}_i = {}^{0}\mathbf{T}_{i-1} \quad {}^{i-1}\mathbf{A}_i \tag{3.2} $$

where ${}^{0}\mathbf{T}_i$ is the homogeneous transformation describing the pose (position and orientation) of coordinate frame i with respect to the world coordinate system 0.

With MDH, Eq. 3.2 still holds, however the homogeneous transformation matrix assumes the following form (instead of Eq. 3.1):

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & d_i \sin\alpha_{i-1} \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i \cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3} $$

3.1.2 Head Structure

The mechanical and geometrical structure of the robotic head used for this thesis work can be seen in Fig. 3.3, which shows the four DOFs of the head, all of which are rotational: neck rotation (pan), head elevation (tilt), left eye vergence, and right eye vergence.

[2] `http://www.cs.duke.edu/brd/Teaching/Bio/asmb/current/Papers/chap3-forward-kinematics.pdf`

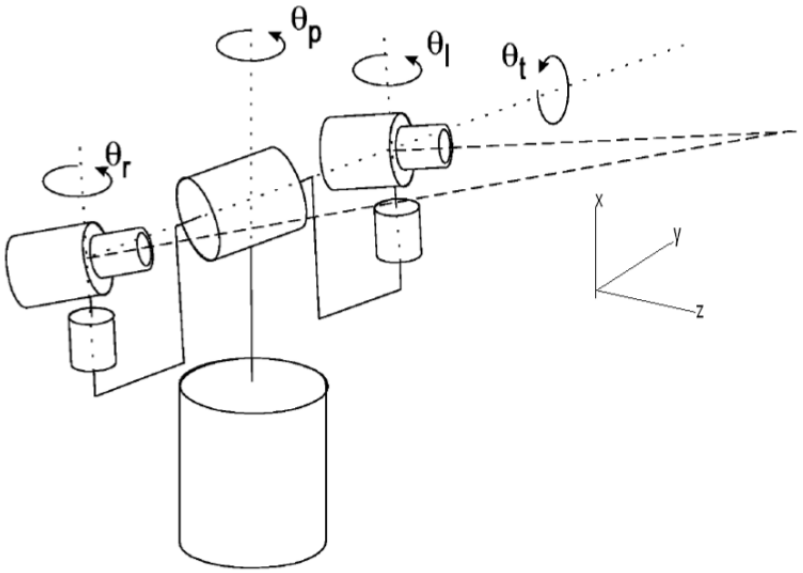


Figure 3.3: Scheme of Baltazar robotic head, code-named "Medusa" [BSV99]. The meaning of the four joint angles $\theta_l, \theta_r, \theta_p$ and θ_t is explained in Table 3.1.

Figure 3.4: Real Baltazar robotic head.

Table 3.2: A possible MDH parameterization of the left eye of the binocular head of Baltazar, taken from [LBPSV04]. B is the baseline distance between the two eyes.

Joint i	a_{i-1} [cm]	d_i [cm]	α_{i-1} [°]	θ_i [°]
1	0	0	0	θ_p
2	15	0	$-\pi$	θ_t
3	0	$B/2$	π	0
4	0	0	π	θ_r

A view of the real Baltazar robotic head, along with its two cameras, can be seen in Fig. 3.4.

Manual adjustments can be made to align the vergence and elevation axes of rotation with the optical centres of the cameras. Inter-ocular distance (baseline) can also be modified. MDH parameters of the head are displayed in Table 3.2.

Let ${}^{2}\mathbf{P}$ denote the 3D coordinates of point $\mathbf{P}$ expressed in eye coordinates. If we denote by ${}^{A}\mathbf{P}$ the coordinates of $\mathbf{P}$ expressed in the arm base (shoulder) coordinate system, this relation holds:

$$ {}^{A}\mathbf{P} = {}^{A}\mathbf{T}_{H} \quad {}^{1}\mathbf{T}_{0} \quad {}^{1}\mathbf{T}_{2} \quad {}^{2}\mathbf{P} \tag{3.4} $$

where the head–arm transformation ${}^{A}\mathbf{T}_{H}$ is given by

$$ {}^{A}\mathbf{T}_{H} = \begin{bmatrix} 0 & 0 & 1 & -27 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 29.6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5} $$

and the translation values of the first three rows of the last column of Eq. 3.5 are expressed in centimetres.

3.1.3 Baltazar and Its Anthropomorphic Arm

Baltazar has an anthropomorphic arm inspired from human arms. However, given the complexity of articulations that a human arm can present, it is still not viable to reproduce one from a technology standpoint. Thus, some simplifications are due, and unfortunately they bring along a loss of maneuverability. The anthropomorphic arm of Baltazar is a fair compromise between complexity and imitation of a human arm.

Fig. 3.5 shows the robotic platform Baltazar in its entirety, whereas a scheme and a picture of the arm can be seen in Fig. 3.6 and in Fig. 3.7, respectively.

The intersection between the two last motor axes, at the base of the wrist, is considered the end-effector for Baltazar.

Forward and inverse kinematics of the robotic arm are taken into account. This arm, which aims to replicate a human one, consists of 6 joints:

- 2 joints are associated with the shoulder;
- 2 with the elbow; and

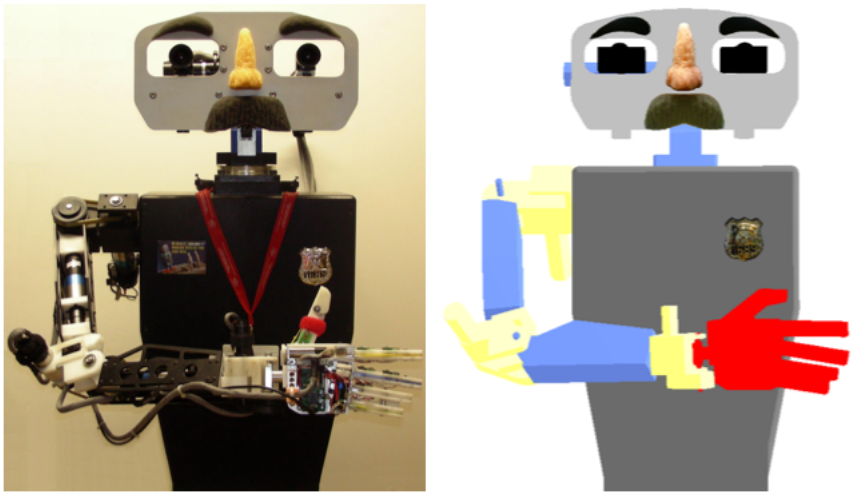


Figure 3.5: Real Baltazar and its CAD model (obtained with the Webots robot simulator [Web]).

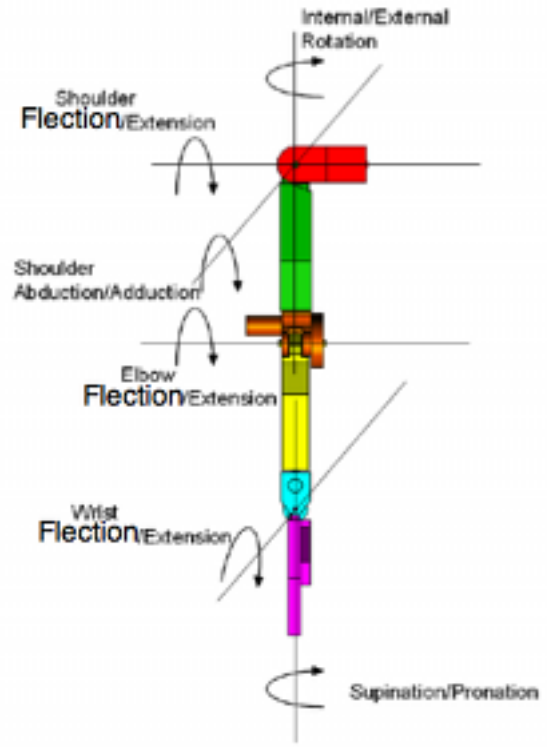


Figure 3.6: Scheme of Baltazar anthropomorphic arm with available rotation DOFs.

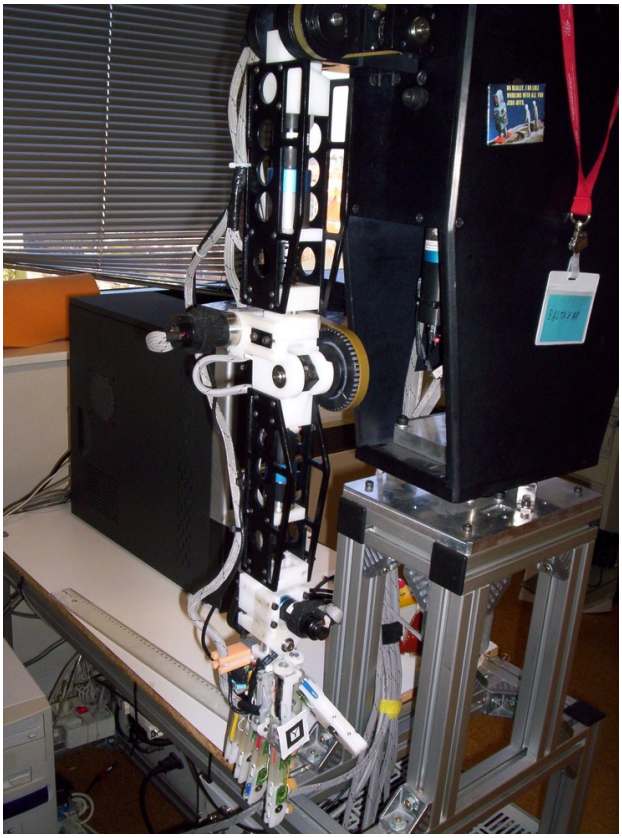


Figure 3.7: Real anthropomorphic arm of Baltazar.

Table 3.3: A possible SDH parameterization of the anthropomorphic arm of Baltazar, derived during this thesis work in order to make use of existing robotics tools.

Link i	a_i [cm]	d_{i+1} [cm]	α_i [°]	θ_{i+1} [°]
0	0	0	$\pi/2$	$-\pi/2$
1	0	0	$\pi/2$	$\pi/2$
2	2.82	29.13	$\pi/2$	$\pi/2$
3	2.18	0	$\pi/2$	π
4	0	26.95	$\pi/2$	$\pi/2$
5	0	0	0	$-\pi/2$

Table 3.4: A possible MDH parameterization of the anthropomorphic arm of Baltazar, taken from [LBPSV04].

Joint i	a_{i-1} [cm]	d_i [cm]	α_{i-1} [°]	θ_i [°]
1	0	0	0	0
2	0	0	π	$\pi/2$
3	0	29.13	π	$\pi/2$
4	2.82	0	π	0
5	-2.18	26.95	$-\pi$	$\pi/2$
6	0	0	π	0

- 2 with the wrist.

As far as this work is concerned, forward kinematics will be used as an extra tool or constraint to the iterative inverse kinematics solution which will be detailed later. The purpose is to exclude those solutions that do not respect a specific restriction imposed on the position of some joints. This is done to operate the robot easily and with no risk of damage when it is close to other objects, such as a table.

3.1.4 Anthropomorphic Arm Forward Kinematics

A possible SDH parameterization of Baltazar 6-DOF arm, written down for this thesis work in a way similar to how the iCub arm kinematics was derived[3], is shown in Table 3.3. Similarly, an MDH parameterization for the anthropomorphic arm is shown in Table 3.4.

[3] `http://eris.liralab.it/wiki/ICubForwardKinematics`

3.1.5 Anthropomorphic Arm Inverse Kinematics

The inverse kinematics problem is that of computing the joint angles[4] that a robot configuration should present, given a spatial position and orientation of the end-effector. This is a useful tool for manipulator path planning, more so than forward kinematics.

One problem is that, in general, the inverse kinematic solution is non-unique, and for some manipulators no closed-form solution exists at all. If the manipulator possesses more DOFs than the number strictly necessary to execute a given task, it is called *redundant* and the solution for joint angles is under-determined. On the other hand, if no solution can be determined for a particular manipulator pose, that configuration is said to be *singular*. Typically, a singularity is due to an alignment of axes reducing the effective DOFs.

We address inverse kinematics in two steps. First, we set the anthropomorphic arm to the desired position (positioning of wrist); then, we change its orientation to a suitable one (orientation of hand).

Let $\mathbf{P}$ denote the desired position of the wrist, and $\mathbf{Z}$ be a null vector. In homogeneous coordinates, this means that:

$$\mathbf{P} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T, \tag{3.6}$$

$$\mathbf{Z} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T. \tag{3.7}$$

The position of the wrist, $\mathbf{P}$, can be related to the various joint angles by cascading the different homogeneous coordinate transformation matrices:

$$\mathbf{P} = \prod_{i=0}^{5} \quad {}^{i}\mathbf{T}_{i+1} \quad \mathbf{Z}, \tag{3.8}$$

where, as per Eq. 3.2, ${}^{i}\mathbf{T}_{i+1}$ denotes homogeneous transformation between frames $i+1$ and i.

In order to achieve a desired 3D location, the first four joints of the arm (counting from the shoulder) must be set to a specific position. Like [LBPSV04], we will use the following transcendental result iteratively in order the determine these positions. Equation

$$a\cos(\theta) + b\sin(\theta) = c \tag{3.9}$$

has solutions

$$\theta = 2\arctan\left(\frac{b \pm \sqrt{a^2 + b^2 + c^2}}{a + c}\right). \tag{3.10}$$

Eq. 3.10 is useful for determining the joint angles of an inverse kinematics problem. Notice that this equation has two solutions: the desired joint position can be chosen accordingly to the physical limits of a joint and/or by using additional criteria (comfort, least change).

To position the arm wrist to a given position $\mathbf{P}$ in space, we need to determine the corresponding values of joints $\theta_1, \theta_2, \theta_3, \theta_4$. Given the kinematic structure of the anthropomorphic arm of Baltazar, the distance ρ from the base

[4] For a robot whose joints are all rotational, like the one used in this work.

to the wrist (end-effector) depends only on θ_4. Using Eq. 3.8, the following constraint holds:

$$a\cos(\theta_4) + b\sin(\theta_4) = \rho^2 - (a_2^2 + l_2^2 + l_1^2 + a_1^2), \tag{3.11}$$

where

$$\begin{cases} a = 2(-a_2 a_1 + l_2 l_1) \\ b = -2(l_2 a_1 + a_2 l_1). \end{cases} \tag{3.12}$$

Since Eq. 3.11 is compatible with the transcendental Eq. 3.10, we can determine the value of θ_4.

The solution of θ_2 and a constraint on θ_3 are obtained from the z component (third column) of $\mathbf{P}$, obtained from Eq. 3.8. In order for the parameters in Eq. 3.10 to permit the existence of a θ_2 solution, we need θ_3 to be such that:

$$a^2 + b^2 + c^2 > 0 \tag{3.13}$$

$$\begin{cases} a = d_2\cos(\theta_4) + l_2\sin(\theta_4) - d_1\sin(\theta_3) \\ b = -d_1\sin(\theta_4) + l_2\cos(\theta_4) + l_1 \\ c = z. \end{cases} \tag{3.14}$$

The algorithm that computes inverse kinematics consists in initializing θ_3 in such a way that the constraint of Eq. 3.13 holds, subsequently allowing the computation of the remaining joint angles [Car07].

All the computed angle variables are tested against the two solutions of Eq. 3.10, so that we can verify if the values are coherent with the physical joint limits of the anthropomorphic arm (see Table 3.5 on p. 43).

As far as hand orientation is concerned, it is sufficient to constrain the solutions to a specific *plane*, by specifying a normal vector to the plane as an input of the inverse kinematics software solver. Note that, in this way, one DOF is still free (hand palm up or hand palm down).

3.2 Hardware Devices of Baltazar

3.2.1 "Flea" Cameras

Two colour cameras are attached on each of the eyes of Baltazar. These are "Flea" cameras, manufactured by Point Grey Research and displayed in Fig. 3.8. They are equipped with an IEEE-1394 FireWire interface and the following characteristics:

- very compact size: $30 \times 31 \times 29$ mm;
- 1/3" Sony Charge-Coupled Device (CCD) sensor;
- high processing speed, up to 640×480 resolution at 60 Frames per Second (FPS);
- external trigger, strobe output;
- 12-bit Analogue-to-Digital Converter (ADC).

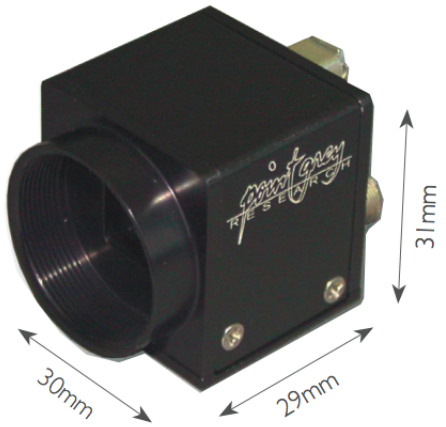


Figure 3.8: Point Grey "Flea" camera. The small dimensions of these cameras is worth noting, making it possible to employ them as (moving) humanoid eyes.

3.2.2 Controller Devices

The four DOFs of the head of Baltazar correspond to four axes or encoders; these are managed by a National Instruments PCI-7340 motion control board, shown in Fig. 3.10.

The 7340 controller is a combination servo and stepper motor controller for PXI, Compact PCI, and PCI bus computers. It includes programmable motion control for up to four independent or coordinated axes of motion, with dedicated motion I/O for limit and home switches and additional I/O for general-purpose functions. Servo axes can control:

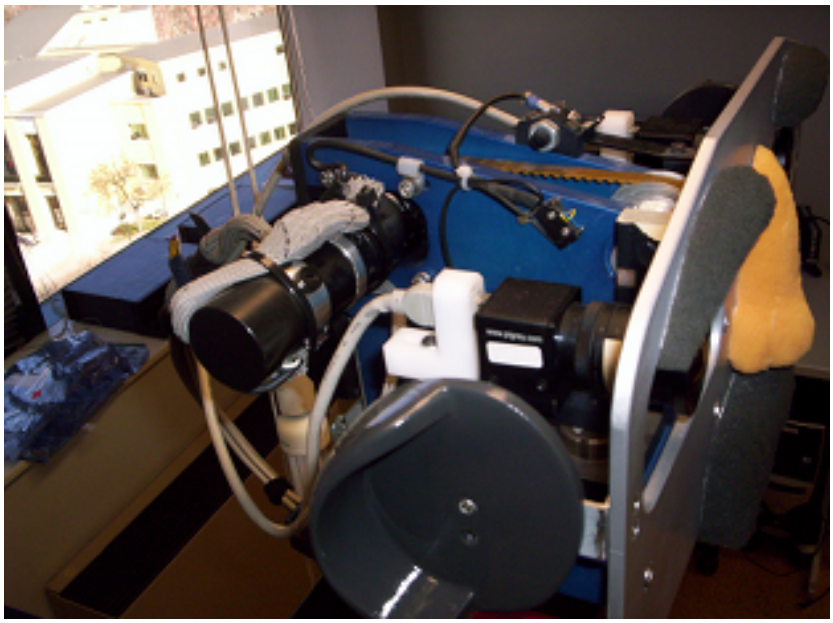
- servo motors;
- servo hydraulics;
- servo valves and other servo devices.

Servo axes always operate in closed-loop mode. These axes use quadrature encoders or analog inputs for position and velocity feedback and provide analog command outputs with a standard range of ± 10 V.

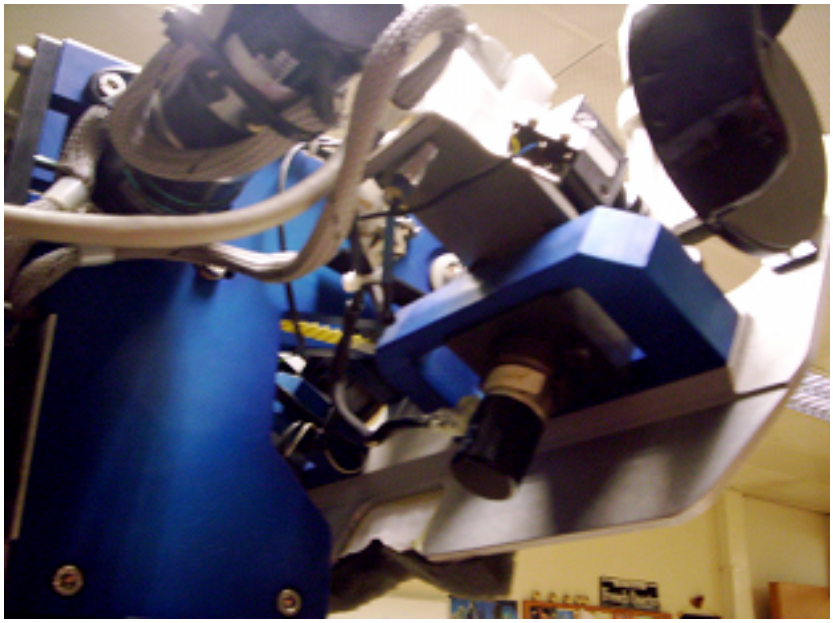
Stepper axes of the 7340 controller, on the other hand, can operate in open- or closed-loop mode. In closed-loop mode, they use quadrature encoders or analogue inputs for position and velocity feedback (in closed-loop only), and they provide step/direction or clockwise/counter-clockwise digital command outputs. All stepper axes support full, half, and microstepping applications.

The 7340 controller reflects a dual-processor architecture that uses a 32-bit CPU, combined with a Digital Signal Processor (DSP) and custom Field-Programmable Gate Arrays (FPGAs), all in all providing good performance.

With regards to application software for this controller, the bundled tool NI-Motion is used. NI-Motion is a simple high-level programming interface (API) to program the 7340 controller. Function sets are available for LabVIEW [Lab] and other programs.



(a) "Flea" camera seen from above.



(b) "Flea" camera seen from below.

Figure 3.9: Right eye Baltazar camera as seen from two perspectives.

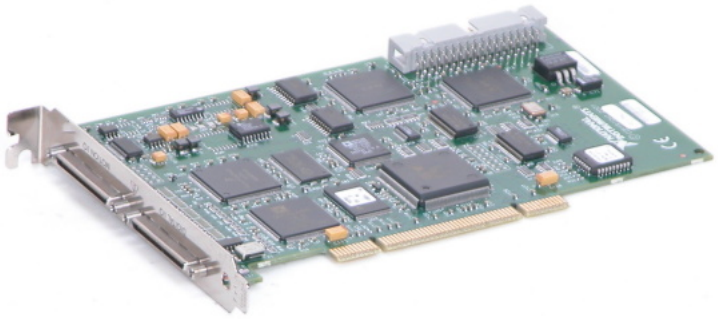


Figure 3.10: National Instruments 7340 Stepper/Servo Motion Controller.

Table 3.5: Joint angles as available in Baltazar arm YARP server. The "physical limits" are the actual angle limitations of the real robot joints, while the "original bounds" column indicates the limits that had been theoretically planned in Lopes *et al.* [LBPSV04], for the sake of historical reference.
All angles are expressed in degrees.

encoder	description	arm joint	physical limits	original bounds
1	shoulder abduction/adduction	1	[-45 35]	[-45 135]
2	shoulder extension/flection	2	[-40 5]	[-110 10]
3	not used	-	-	-
4	torso rotation	-	-	-
5	shoulder external/internal rotation	3	[-90 0]	[-90 0]
6	elbow extension/flection	4	[-90 0]	[-90 0]
7	arm pronation/supination	5	[-80 80]	[-90 90]
8	wrist extension/flection	6	[-29 45]	[-45 45]

LabVIEW is a platform and development environment for a visual programming language (also from National Instruments) called "G".

In order to actuate the anthropomorphic arm and hand of Baltazar, another control board is mounted on the platform. The Networked Modular Control (NMC) communication protocol[5] is used to control the six joints of the limbs of the robot.

3.3 Software Setup

During the development of a piece of software, particularly if it is a project involving different people and institutions as well as operating systems and hardware, one should keep in mind certain basic principles of software engineering at all times:

[5] `http://www.jrkerr.com/overview.html`

- high cohesion;
- low coupling
- explicit interfacing;
- information hiding.

Some software libraries that were largely employed in the development of this project (chiefly the YARP set of libraries) will now be briefly presented.

3.3.1 YARP

The iCub software (and other projects developed under the "umbrella" of the RobotCub Consortium, such as this thesis) is potentially parallel and distributed. Apart from Application Programming Interfaces (APIs) that speak directly to the hardware, the upper layers might require further support libraries, as is often necessary when programming robot systems immersed in various computer networks. In fact, many software solutions are already available [CCIN08, Table 1]. In the case of RobotCub, these missing libraries include *middleware* mechanisms and were custom developed: their suite is called YARP.

YARP is open source and, as such, it is suitable for inclusion of newly developed iCub code. The rationale in this choice lays in the fact that having the source code available and, especially, well understood, can potentially simplify the software integration activity.

In order to facilitate the integration of code, clearly the simplest way would be to lay out a set of standards and to ask developers to strictly follow them. In a large research project like RobotCub, the community should also allow for a certain freedom to developers, so that ideas can be tested quickly. These two requirements are somehow conflicting. Especially, they are conflicting when different behaviours are to be integrated into a single system and the integrator is not the first developer.

To allow developers to build upon the already developed behaviours, the researchers of RobotCub chose to layer the software and release-packaged behaviours in the form of APIs. The idea is to produce behaviours that can be used without necessarily getting into the details of the middleware code employed. While for lower levels there is no much alternative than following a common middleware approach, higher levels and user level code can be developed by considering a less demanding scenario. In the latter case, modules are distributed with interfaces specified in an API—possibly a C++ class hierarchy.

Internally, each module will unleash a set of YARP processes and threads whose complexity will be hidden within the module. Various levels of configuration are possible. In one case, the given module would be capable of running on a single processor machine. This is a tricky and difficult choice since in many cases the behaviour of the robot relies explicitly on timing, synchronization, and performances of its submodules. Considering that eventually each module is a very specialized controller, issues of real-time performances have to be carefully evaluated. The modules' APIs will include tests and indications on the computational timing and additional requirements in this respect, to facilitate proper configuration and use.

Fig. 3.11 exemplifies the iCub software architecture. The lowest level of the software architecture consists of the level-0 API which provides the basic

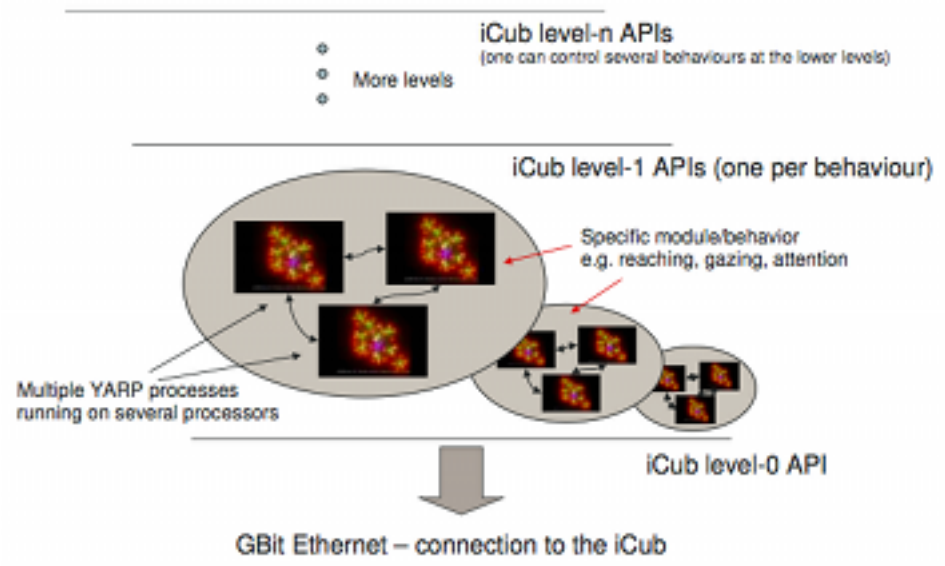


Figure 3.11: Software architecture of the iCub (and RobotCub).

control of the iCub hardware by formatting and unformatting IP packets into appropriate classes and data structures. IP packets are sent to the robot via a Gbit Ethernet connection. For software to be compliant to the iCub, the only requirement is to use this and only this API. The API is provided for both Linux and Windows. The iCub behaviours/modules/skills will be developed using YARP to support parallel computation and efficient Inter-Process Communication (IPC). YARP is both open source and portable (i.e., OS independent), so it fits the requirements of RobotCub in this sense. Each module can thus be composed of several processes running on several processors.

YARP is an open source framework for efficient robot control, supporting distributed computation and featuring a middleware infrastructure [YAR, MFN06].

YARP was created for a number of reasons:

- computer multitasking: it is useful to design a robot control system as a set of processes running on different computers, or several central processing units (CPUs) within a single system;
- making communication between different processes easy;
- code decoupling and modularity: it is a good practice to maintain and reuse small pieces of code and processes, each one performing a simple task. With YARP it is easy to write location-independent modules, which can run on different machines without code changes whatsoever;
- possibility to redistribute computational load among CPUs, as well as to recover from hardware failures.

In particular, as far as communication is concerned, YARP follows the *Observer* design pattern (also known as *publish/subscribe*; see [GHJV00]). One or

more objects ("observers" or "listeners") are registered to observe an event that may be raised by the observed object (the "subject").

Several *port* objects deliver messages to any number of observers, i.e., to their ports.

3.3.2 Other Software Libraries

Besides YARP, other libraries used for the implementation that are worth mentioning are GNU Scientific Library (GSL), its Basic Linear Algebra Subprograms (BLAS) interfaces, and OpenCV.

GSL is a software library written in C for numerical calculations. Among other things, GSL includes an implementation of the BLAS interface.

BLAS [BLA] is a set of routines for linear algebra, useful to efficiently perform operations with vectors and matrices. They are divided into:

- Level 1 BLAS for vector-vector operations;
- Level 2 BLAS for matrix-vector operations;
- Level 3 BLAS for matrix-matrix operations.

During the development of this thesis, GSL and BLAS were used to make computations between matrices fast and robust (preventing memory leaks and segmentation faults, thus improving security).

OpenCV [Ope] is a multi-purpose CV library originally developed by Intel. Nowadays it is free for commercial and research use (under a BSD license). This library has two characteristics that it shares with RobotCub research and that made us choose it:

- being cross-platform;
- having a focus on real-time image processing. If OpenCV finds Intel's Integrated Performance Primitives (IPP) on the working system, it will use these commercial optimized routines to accelerate itself.

Chapter 4

Proposed Architecture

In RobotCub and in this thesis project too, manipulation is seen as a means to assist perception under uncertainty. Often, vision alone is not enough to reliably segment distinct objects from the background. Overlapping objects or similarity in appearance to the background can confuse many visual segmentation algorithms. Rather than defining an object in terms of its appearance or shape as a predefined model, we propose a simple framework, where only the position and orientation of a tracked object are taken into consideration. This will be done by *estimating the orientation as the major axis of the best-fit enclosing ellipse that surrounds the object.*

In this section we will describe the proposed visual processing (a segmentation based on colour histograms), our 3D reconstruction technique, and the applications to object manipulation tasks.

4.1 Visual Processing

Using CV to control grasping tasks is natural, since it allows to recognize and to locate objects [DHQ98]. In particular, stereopsis can help robots reconstruct a 3D scene and perform visual servoing.

As discussed in Section 2.1, the problems of object tracking and image segmentation can be handled from several different perspectives. There exist elaborate methods which implement tracking, based, for example, on: contour tracking by the means of *snakes*; association techniques and matrix Eigenvalues (Eigenspace matching); maintaining large sets of statistical hypotheses; computing a convolution of the image with predefined feature detector patterns. Most of these techniques, though, are computationally expensive and not suited for our framework, which must be simple enough and be able to run in real time, such as at 30 FPS.

So, CV algorithms that are intended to form part of a PUI, such as our problem and all RobotCub research in general, must be fast and efficient. They must be able to *track in real time*, yet not absorb a major share of the computational resources that humanoid robots have.

The majority of the available segmentation algorithms in literature do not cover all the needs of our objective optimally, because they do not possess all of our requirements:

- being able to track similar objects;
- being robust to noise;
- working well and reasonably *fast*, since the tracking algorithm is going to be run in parallel with a considerable set of computationally heavy tasks, so that the robot arm is able to intercept and grasp objects;
- in particular, we want to run several concurrent instances of the algorithm at the same time, so performance is an important requirement.

A possible approach we initially thought of was to interpret motion statistically: by performing a Bayesian interpretation of the problem, the solution of motion estimation could be translated to inferring and comparing different similarity functions, generated from different motion models.

Alternatively, we could have used the Expectation Maximization (EM) algorithm[1] to choose the best model. This method is frequently employed in statistical problems, and it consists of two phases, shown in Algorithm 2.

Algorithm 2 Expectation Maximization (EM)

1: (E step) Extrapolate a parameterized likelihood.
2: (M step) Maximize the expected likelihood found in the E step.

More precisely, in the E step we associate all the points that correspond to a randomly chosen model, and in the M step we update the model parameters basing on the points that were assigned to it. The algorithm is iterated over and over, until the model parameters converge.

The solutions recalled so far in this chapter deal with a problem that is similar to the one we want to address—on the one hand, we need to establish *a priori* the set of motion models that the object can present; on the other hand we wish to choose the best possible motion model.

To sum up, several different approaches for segmentation are possible; though, since the development of a tracking algorithm is not the main objective of this work (it is but an initial component of it), and performance is a strong requirement, we opted for using a simple, ready-made solution from OpenCV (Intel Open Source Computer Vision Library, [Ope]), which is a library of C/C++ functions and algorithms frequently used in image processing. The method that we chose was CAMSHIFT, detailed further down. However, some custom modifications were applied by us on the OpenCV version of CAMSHIFT. The most relevant of them are these two:

- we compute the enclosing ellipse of an object by focusing the attention on the axes and centroid of such an ellipse (rather than memorizing and transmitting a whole rectangular area, we just handle, for instance, the major axis of the ellipse);
- we add networking capabilities to the OpenCV implementation of CAMSHIFT, by encapsulating it into the YARP module system (see Section 3.3.1) and

[1] A variant of k-Means (see Section 2.1.1).

using a middleware mechanism. This makes it possible to run several instances of the tracker in parallel, for example two trackers to track an object with stereopsis, or multiple objects.

4.2 CAMSHIFT Module

The Continuously Adaptive Mean Shift algorithm, or CAMSHIFT for short, is based on Mean Shift [CM97], which in turn is a robust, non-parametric iterative technique for finding the mode of probability distributions. Interestingly, the Mean Shift algorithm was not originally intended to be used for tracking, but it proved effective in this role nonetheless (see [Bra98]).

CAMSHIFT is fast and simple; as such, it fulfils the requirements of our project. It is a technique based on colour, however, contrary to similar algorithms, CAMSHIFT does *not* take into account colour correlation, blob growing, region growing, contour considerations, Kalman filter smoothing and prediction (all of which are characteristics that would place a heavy burden on computational complexity and speed of execution).

The complexity of most colour-based tracking algorithms (other than CAMSHIFT) derives from their attempt to deal with *irregular object motion*, which can be due to:

- perspective (near objects to the camera seem to move faster than distal ones);
- image noise;
- distractors, such as other shapes present in the video scene;
- occlusion by hands or other objects;
- lighting variation.

Indeed, all of the above are serious problems that are worth studying and being modelled for certain practical applications, however the main trait of CAMSHIFT is that it is a fast, computationally-efficient algorithms that mitigates those issues "for free", i.e., during the course of its own execution.

Algorithm 3 Mean Shift

1: Choose a search window size.
2: Choose the initial location of the search window.
3: Compute the mean location in the search window.
4: Centre the search window at the mean location computed in step 3.
5: Repeat steps 3 and 4 until convergence (or until the mean location moves less than a preset threshold).

At the beginning of this section, we mentioned that in general Mean Shift (see Algorithm 3) operates on probability distributions. Therefore, in order to track coloured objects in a video frame sequence, the colour image data has to be represented as a probability distribution [CM97] — to do this, *colour histograms* are used.

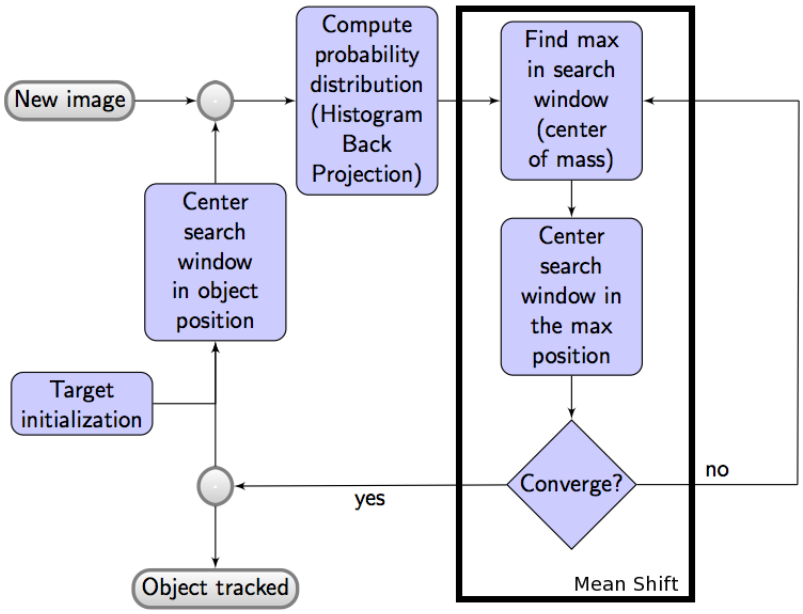


Figure 4.1: Block diagram of CAMSHIFT.

Colour distributions that are derived from video image sequences may change over time, so the Mean Shift algorithm must be modified to dynamically adapt to the probability distribution that it is tracking at a given moment. It is here that the new, modified algorithm –CAMSHIFT– bridges the gap (also see 2.7).

Given a colour image and a colour histogram, the image produced from the original colour image by using the histogram as a lookup table is called *back-projection image*. If the histogram is a model density distribution, then the back-projection image is a probability distribution of the model in the colour image. CAMSHIFT detects the mode in the probability distribution image by applying Mean Shift while dynamically adjusting the parameters of the target distribution. In a single image, the process is iterated until convergence—or until an upper bound on the number of iterations is reached.

A detection algorithm can be applied to successive frames of a video sequence to track a single target. The search area can be restricted around the last known position of the target, resulting in possibly large computational savings. This type of scheme introduces a feedback loop, in which the result of the detection is used as input to the next detection process. The version of CAMSHIFT applying these concepts to tracking of a single target in a video stream is called Coupled CAMSHIFT.

The Coupled CAMSHIFT algorithm as described in [Bra98] is demonstrated in a real-time head tracking application, which is part of the Intel OpenCV library [Ope].

4.2.1 CAMSHIFT and HSV Conversion

In order to use a histogram-based method to track coloured objects in a video scene, a probability distribution image of the desired colour present in the video sequence must first be created. For this, one first creates a model of the desired *hue* by using a colour histogram.

The reason why Hue Saturation Value (HSV) space is better suited for our proposed perceptual interface is the following. Other colour models like Red Green Blue (RGB), Cyan Magenta Yellow (CMY), and YIQ are hardware-oriented [FvDFH95, p. 590]. By contrast, Smith's HSV [Smi78] is user-oriented, being based on the intuitive, "artistic" approach of tint, shade and tone.

In general, the coordinate system of HSV is cylindrical; however, the subset of space within which the model is defined is a hexcone[2], as in Fig. 4.2b. The hexcone model is intended to capture the common notions of hue, saturation and value:

- Hue is the hexcone dimension with points on it normally called red, yellow, blue-green, etc.;
- Saturation measures the departure of a hue from achromatic, i.e., from white or gray;
- Value measures the departure of a hue from black (the colour or zero energy).

These three terms are meant to represent the artistic ideas of hue: tint, shade and tone.

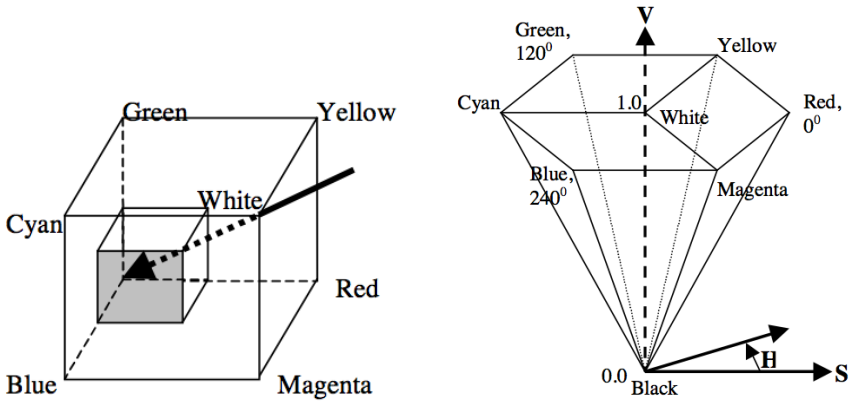
The top of the hexcone in Fig. 4.2b corresponds to $V = 1$, which contains the relatively bright colours. Descending the V axis gives smaller hexcones, that correspond to smaller (darker) RGB subcubes in Fig. 4.2a.

The HSV colour space is particularly apt to capture senses and perception, more so than RGB. HSV corresponds to projecting standard Red, Green, Blue colour space along its principle diagonal from white to black [Smi78], as seen looking at the arrow in Fig. 4.2a. As a result, we obtain the hexcone in Fig. 4.2b. HSV space separates out Hue (colour) from Saturation (i.e., how concentrated the colour is) and from brightness. In the case of CAMSHIFT, we create our colour models by taking 1D histograms (with 16 bins) from the H channel in HSV space.

CAMSHIFT is designed for dynamically-changing distributions. These occur when objects in video sequences are being tracked and the object moves so that the size and location of the probability distribution changes in time. The CAMSHIFT algorithm adjusts the search window size in the course of its operation. Instead of a set or externally adapted window size, CAMSHIFT relies on the zeroth moment information, extracted as part of the internal workings of the algorithm, to *continuously* adapt its window size within or over each video frame.

The zeroth moment can be thought of as the distribution "area" found under the search window [Bra98]. Thus, window radius, or height and width, is set to a function of the zeroth moment found during search. CAMSHIFT, outlined in Algorithm 4, is then calculated using any initial non-zero window size.

[2]Six-sided pyramid.



(a) RGB colour cube.

(b) HSV colour system.

Figure 4.2: RGB and HSV colour spaces. In the single-hexcone HSV model, the $V = 1$ plane contains the RGB model's $R = 1, G = 1$, and $B = 1$ planes in the regions shown.

Algorithm 4 CAMSHIFT

1: Choose the initial location of the search window.
2: Perform Mean Shift as in Algorithm 3, one or more times. Store the zeroth moment.
3: Set the search window size equal to a function of the zeroth moment found in Step 2.
4: Repeat Steps 2 and 3 until convergence (mean location moves less than a preset threshold).

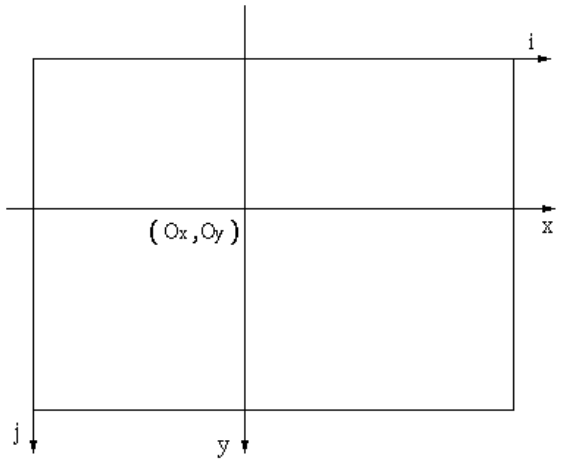


Figure 4.3: Image coordinates.

4.3 3D Reconstruction Approach

In order for a humanoid robot to be able to do things in the world, it requires to have a tridimensional perception, which is what we want to accomplish in this module, in a precise yet simple and efficient (fast) way.

The anthropomorphic head of Baltazar (Fig. 3.4, p. 34) has two cameras, mounted in a way similar to the eyes of a human being (Fig. 3.9, p. 42).

The notion of *depth* is thus obtained from the combination of information that comes from the two cameras. In this section, we will explain a method to determine the coordinates of one point in the area that is visible from both cameras at a given moment. In particular, we want to reconstruct the 3D coordinates of two points for each object, corresponding to the two extremities of the major axis of the best-fit ellipse to the object. This visual simplification facilitates real-time performances and at the same time it opens the way for manipulation tasks.

4.3.1 From Frame Coordinates to Image Coordinates

A digitalized image is usually stored in a framebuffer which can be seen as a matrix of pixels with W columns (from "width") and H rows (from "height").

Let (i, j) be the discrete frame coordinates of the image with origin in the upper-left corner, (O_x, O_y) be the focal point of the lens (the intersection between the optical axes and the image plane) in the frame coordinates, and (x, y) be the image coordinates, as illustrated in Fig 4.3.

Image coordinates relate to frame coordinates in this way:

$$x = (i - O_x) \cdot S_x \tag{4.1}$$

$$y = (j - O_y) \cdot S_y \tag{4.2}$$

where S_x, S_y are the horizontal and vertical distances of two adjacent pixels in the framebuffer.

An *a priori* hypothesis is that we know the relative displacement of the two cameras (rotation and translation) at all times. This makes sense, as we can continuously update the angle values in our software module, receiving instantaneous values from the robot encoders, for example at a frequency of one update per second.

The matrix of intrinsic parameters of a camera, which sets the relationship between a 3D point and the pixels in a sensor, will not be explicitly computed in this project. In other words, we avoid the calibration phase. The only aspects that we will consider are:

- image resolution;
- focal distance; and
- pixel size.

In stereo analysis, *triangulation* is the task of computing the 3D position of points in the images, given the disparity map and the geometry of the stereo setting. The 3D position (X, Y, Z) of a point $\mathbf{P}$ can be reconstructed from the perspective projection (see Fig. 2.16, p. 25) of $\mathbf{P}$ on the image planes of the cameras, once the relative position and orientation of the two cameras are known.

For example, if we choose the 3D world reference frame to be the left camera reference system, then the right camera is translated and rotated with respect to the left one, therefore six parameters will describe this transformation.

In the most general case, the right camera can be rotated with respect to the left one (or vice versa) in three directions.

For 3D reconstruction, we use a pinhole camera model like the one in Fig. 4.4.

The relationships that exist between the world coordinates of a point $\mathbf{P} = (X, Y, Z)$ and the coordinates on the image plane (x, y) in a pinhole camera are

$$x = F \cdot X/Z \tag{4.3}$$

$$y = F \cdot Y/Z \tag{4.4}$$

where lower-case letters refer to *image* position, upper-case ones to *world* position (in metres), and F is the focal distance of the lens (also in metres).

Considering the two cameras and referring as B to the baseline (distance between the optical centres of them), we can now obtain the missing coordinate Z:

$$y_L = \frac{FY_L}{Z_L} \tag{4.5}$$

$$Y_R = Y_L - B \tag{4.6}$$

$$Z_L = Z_R = Z \tag{4.7}$$

$$y_R = \frac{FY_R}{Z_R} = F\frac{Y_L - B}{Z} \tag{4.8}$$

$$y_L - y_R = F\frac{B}{Z} \Longleftrightarrow Z = \frac{FB}{y_L - y_R}. \tag{4.9}$$

We will now analyze the case in which the two cameras are in an arbitrary relative orientation. A description of various vergence situations can be consulted in [BSV99].

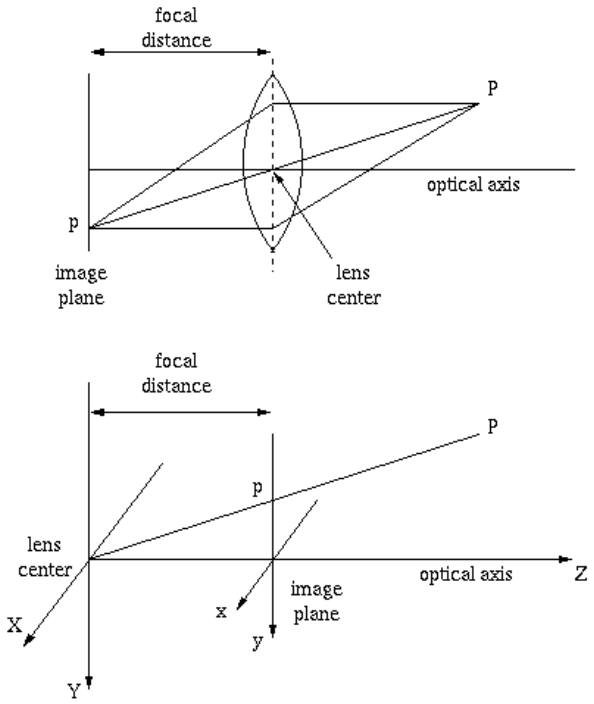


Figure 4.4: Pinhole camera model.

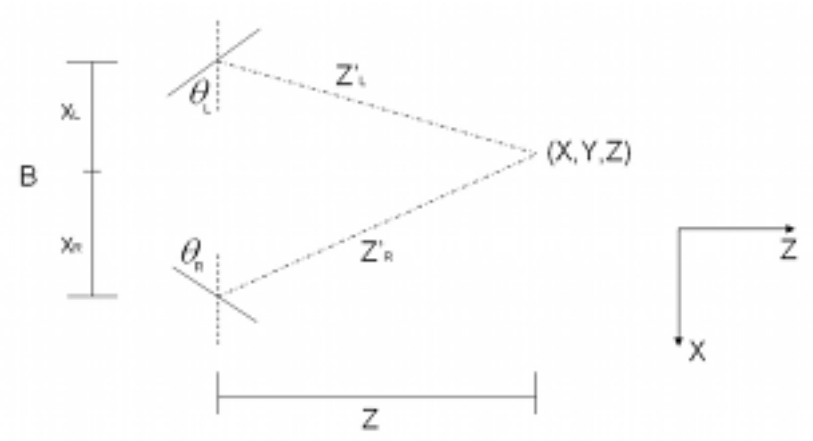


Figure 4.5: 3D reconstruction scheme for a stereo pair of cameras with arbitrary vergence.

Looking at Fig. 4.5, we can thus write down the following equations for the Z coordinate:

$$\tan(\theta_L + x_L/F) = \frac{B/2}{Z} \tag{4.10}$$

$$\tan(\theta_R + x_R/F) = \frac{B/2}{Z} \tag{4.11}$$

$$Z = \frac{B}{\tan(x_L/F + \theta_L) - \tan(x_R/F + \theta_R)}. \tag{4.12}$$

As for the X coordinate, we obtain:

$$\tan(\theta_L + x_L/F) = \frac{X_L}{Z} \tag{4.13}$$

$$\tan(\theta_R + x_R/F) = \frac{X_R}{Z} \tag{4.14}$$

$$X = -\frac{Z}{2}\big[\tan(x_L/F + \theta_L) + \tan(x_R/F + \theta_R)\big]. \tag{4.15}$$

Finally, the 3D reconstructed coordinate Y is obtained like this:

$$Z'_R = Z/\cos\theta_R \tag{4.16}$$

$$Z'_L = Z/\cos\theta_L \tag{4.17}$$

$$Y = \frac{Zy_R}{2F\cos\theta_R} + \frac{Zy_L}{2F\cos\theta_L} \tag{4.18}$$

In order to determine the coordinates of an object in a fixed reference frame, it is necessary to consider the forward kinematics of the head of Baltazar (see Section 3.1.2). From coordinates that are expressed in the image plane, we want to obtain coordinates in a fixed reference frame (a frame attached to the robot torso, which does not move).

Recall the scheme of Baltazar head, illustrated in Fig. 3.3. The expression given in Eq. 4.19 was obtained from geometrical analysis, and it provides a relationship between coordinates in the image plane (of one of the two cameras) with the coordinates expressed in a fixed frame.

$$\mathbf{P}_l = \mathbf{R}_l(\mathbf{P} - \mathbf{t}_l). \tag{4.19}$$

For the sake of simplicity, Eq. 4.19 refers to the left camera case, thus the l subscript. The rotation and translation matrices are, in particular, equal to

$$\begin{aligned} \mathbf{R}_l &= \begin{bmatrix} c_p c_l - c_t s_p s_l & -s_p s_t & c_t c_l s_p + c_p s_l \\ -s_t s_l & c_t & c_l s_t \\ -c_l s_p - c_p c_t s_l & -c_p s_t & c_p c_t c_l - s_p s_l \end{bmatrix}; \\ \mathbf{t}_l &= \begin{bmatrix} -B' c_l - t_Y s_t s_l + t_Z(-c_l s_p - c_p c_t s_l) \\ t_Y c_t - t_Z c_p s_t \\ t_Y c_l s_t - B' s_l + t_Z(c_p c_t c_l - s_p s_l) \end{bmatrix} \end{aligned} \tag{4.20}$$

where $B' = B/2$ (half the baseline distance).

From a geometrical point of view, the camera sensors mounted on the Baltazar head simply measure *relative positions*; these two cameras are used to calculate the position of objects within their workspace, relative to their optical

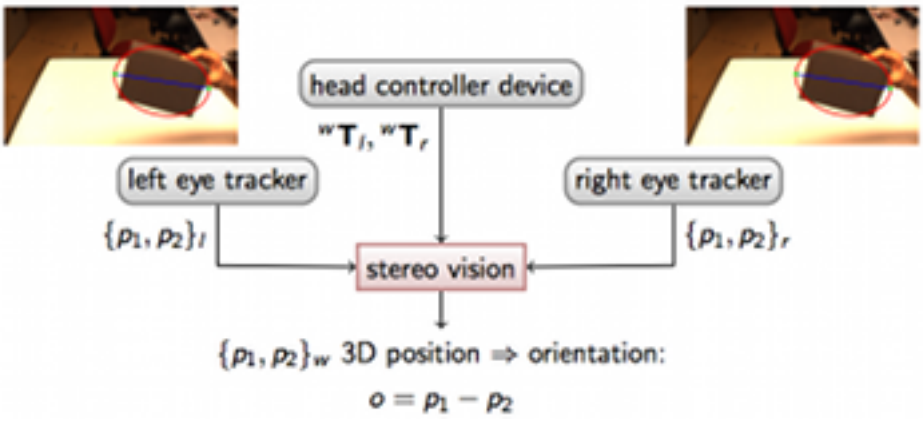


Figure 4.6: 3D reconstruction software module scheme, outlining the data that are passed as inputs/outpus among YARP modules.

centre. Thus, a 3D point is mapped onto a two-dimensional space, by the means of its projection on the image plane. It is precisely in this way that we obtain the 2D coordinates of a pair of stereo cameras: the resulting coordinates derive from a 3D point located in the surrounding of Baltazar.

4.3.2 3D Pose Estimation

Consider now a *target object* placed in front of the robot; tracking is accomplished by running two CAMSHIFT processes. Let points $\{p_1, p_2\}_l^{\text{target}}$ and $\{p_1, p_2\}_r^{\text{target}}$ be the extremities of the major ellipse axis expressed in the 2D coordinate frame of the left and right tracker, respectively[3].

A 3D reconstruction process receives the coordinates of the four points $\{p_1, p_2\}_{l,r}$ as inputs, along with the instantaneous head joint angle values of the robot, used to compute the time-varying extrinsic camera parameter matrices: not just the target object, but also the robot cameras may be moving during experiments. Transformation matrices ${}^{w}\mathbf{T}_l$ and ${}^{w}\mathbf{T}_r$ represent the roto-translations occurring, respectively, from the left and right camera reference frame to the world (torso) reference frame, as shown in Fig. 4.7.

Fig. 4.6 shows how the 3D reconstruction module works. It receives inputs from two CAMSHIFT trackers (one per each eye), it receives the instantaneous head joint angles (with which it builds transformation matrices), then finally it computes estimated coordinates and orientation of a tracked object.

Thanks to how YARP is designed, we can easily run various concurrent instances of this modules in parallel. Specifically, in the grasping preparation (visual servoing) phase we will be interested in 3D reconstructing the target object and the robot hand at the same time.

Once the reconstruction is computed, 3D coordinates of $\{p_1, p_2\}$ are obtained. The difference vector $p_1 - p_2$ encodes the orientation of the target.

[3]The same considerations apply for stereo tracking and 3D reconstruction of the robot *hand*, but for the sake of simplicity only the target object case is explained here. From now on, the "target" exponent in the notation is therefore omitted.

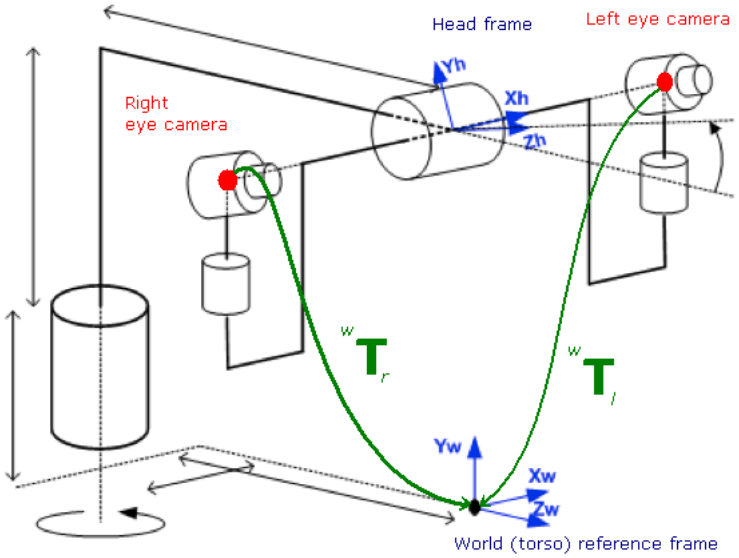


Figure 4.7: Mechanical structure of Baltazar head, its reference frames and points of interest. Transformation matrices are highlighted in green.

4.4 Object Manipulation Approaches

As mentioned in p. 5, in this work we consider two distinct phases for a manipulation task:

reaching preparation: this phase aims at bringing the robot hand to the vicinity of the target. It is applied whenever a target is identified in the workplace but the hand is not visible in the cameras. The measured 3D target position is used, in conjunction with the robot arm kinematics, to place the hand close to the target. Inevitably, there are mechanical calibration errors between arm kinematics and camera reference frames, so the actual placement of the hand will be different from the desired one. Therefore, the approach is to command the robot not to the exact position of the target but to a distance safe enough to avoid undesired contact both with the target and the workspace.

grasping preparation: in this phase, both target and hand are visible in the camera system and their posture can be obtained by the methods previously described. The goal is now to measure the position and angular error between target and hand, and use a PBVS approach to make the hand converge to the target. The features used in such an approach are 3D parameters estimated from image measurements—as opposed to IBVS, in which the features are 2D and immediately computed from image data.

Reaching preparation is relatively easy, since in this phase we just position the arm to the vicinity of the target object (within a threshold of 20 cm). The arm starts moving from a predefined position outside of the field of view of the

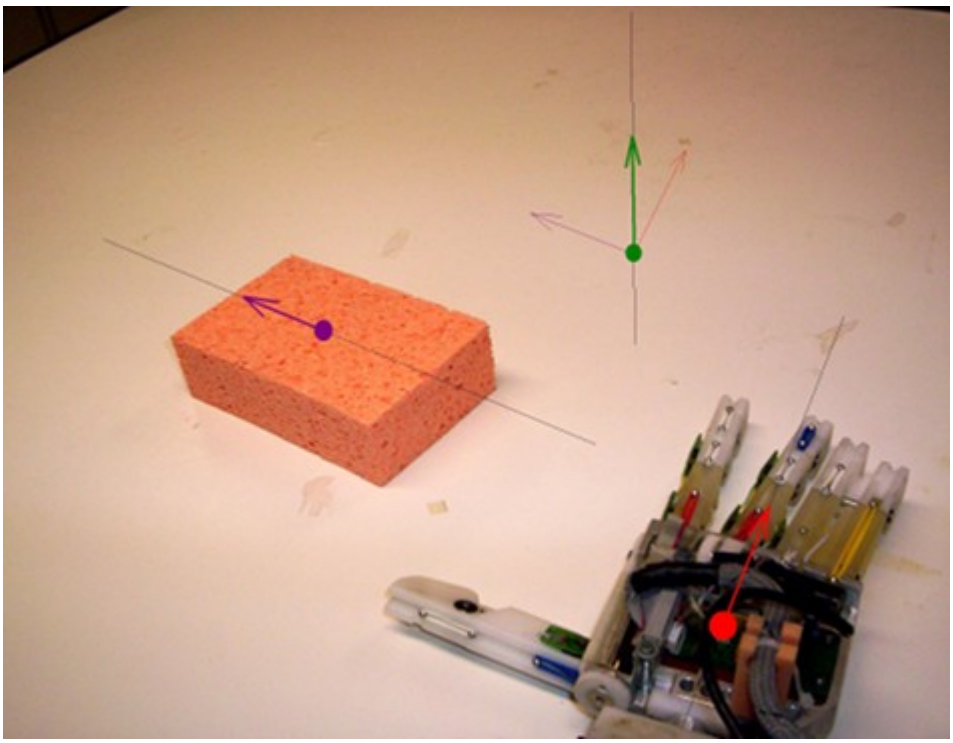


Figure 4.8: Unit vector of the target object along its orientation axis (purple); unit vector and orientation of robot hand (red); third axis resulting from their cross product, and corresponding unit vector (green).

two cameras, and it reaches the position estimated by the inverse kinematics solver.

On the other hand, there are two peculiarities in the presented grasping preparation approach:

- Normally, PBVS requires the 3D model of the observed object to be known [HHC96], but in our framework one gets rid of this constraint: by using the stereo reconstruction technique explained above, the only condition to prepare the servoing task is that the CAMSHIFT trackers are actually following the desired objects—whose models are *not* known beforehand.

- Classical PBVS applications consider that target and end-effector positions are measured by different means, e.g. target is measured by the camera and end-effector is measured by robot kinematics. This usually leads to problems due to miscalibrations between the two sensory systems. Instead, in this work, target and hand positions are measured by the camera system in the same reference frame, therefore the system is robust to calibration errors.

Having computed 3D position and orientation of both a target object and of the robot hand, features suitable for the application of the PBVS technique

must be obtained. As described in [CH06], the robot arm can be controlled by the following law:

$$\begin{cases} \mathbf{v} = -\lambda \left((\mathbf{t}_{\text{target}} - \mathbf{t}_{\text{hand}}) + [\mathbf{t}_{\text{hand}}]_{\times} \vartheta \mathbf{u} \right) \\ \omega = -\lambda \vartheta \mathbf{u} \end{cases} \tag{4.21}$$

where $\mathbf{v}$ and ω are the arm linear and angular velocities, λ establishes the trajectory convergence time, $\mathbf{t}_{\text{target}}$ and $\mathbf{t}_{\text{hand}}$ are the target and hand positions, $[\cdot]_{\times}$ is the associated skew-symmetric matrix of a vector; $\vartheta, \mathbf{u}$ are the angle-axis representation of the rotation required to align both orientations. Other control laws can be applied to this problem, but they normally rely on an angle–axis parameterization of the rotation.

It is possible to calculate the required angle ϑ and axis $\mathbf{u}$ by applying a simple *cross product rule* between the normalized hand and target orientation vectors:

$$\begin{cases} \mathbf{u} = \mathbf{o}_{\text{target}} \times \mathbf{o}_{\text{hand}} \\ \vartheta = \arcsin \|\mathbf{u}\|_2 \end{cases} \tag{4.22}$$

where $\| \cdot \|_2$ is the Euclidean norm, $\mathbf{o}_{\text{target}}$ is a unit vector in the direction of the target object's reconstructed orientation and $\mathbf{o}_{\text{hand}}$ is a unit vector in the direction of the hand's reconstructed orientation (see Fig. 4.8).

Chapter 5

Experimental Results

This chapter contains experimental results obtained by testing the programs that were written for this thesis. The whole project was carried out at Computer and Robot Vision Laboratory [Vis], Institute for Systems and Robotics, Instituto Superior Técnico, Lisbon (Portugal) for eight months during 2008.

All the tests were performed on the machine that is attached to the Baltazar robotic platform: a personal computer with a Dual Intel Xeon 3.20GHz processor, 1GB of RAM, running Microsoft Windows XP Pro SP2.

The development environment adopted was Microsoft Visual Studio 2005, also providing a graphical debugging interface for C++.

5.1 Segmentation and Tracking

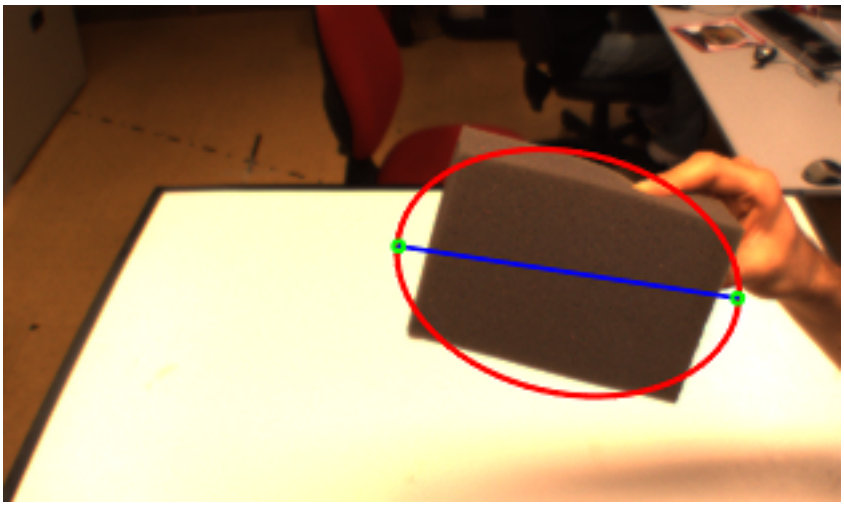
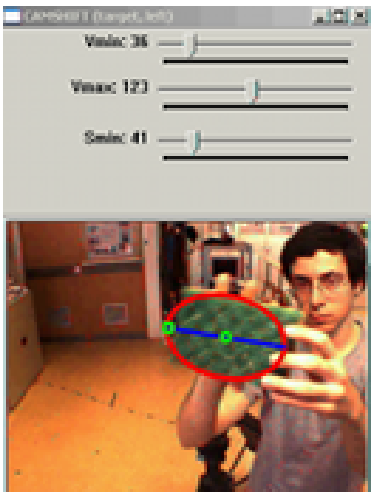
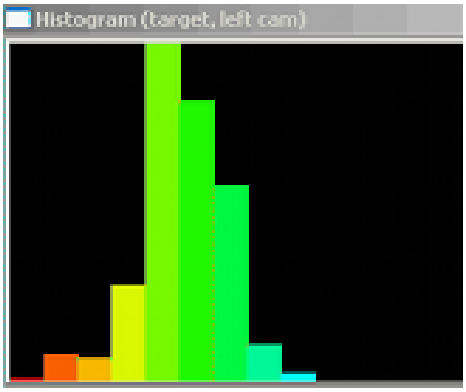


Figure 5.1: CAMSHIFT tracking experiment, with a grey sponge.

Fig. 5.1 shows an early version of the custom modified CAMSHIFT tracker while running during a video experiment. The best-fit enclosing ellipse is drawn in red, the major axis in blue.



(a) Second CAMSHIFT tracking experiment screenshot.



(b) 16-bin CAMSHIFT colour histogram used for iterative searches during all of the experiment of Fig 5.2a.

Figure 5.2: Another CAMSHIFT tracking experiment, with a green sponge.

The major axis of the ellipse represents an estimation of the target object orientation. One can see that such axis is not completely parallel to the long edges of the cuboid sponge: this is due to previous motion and frames that cause an oscillatory nature of the ellipse (axes).

Furthermore, we can see the estimated extremities of the major axis (p_1 and p_2) marked with green circles. Note that this type of experiments did not yet take into account the manipulation (reaching and grasping) of objects, thus the object centroid is neither estimated nor visually marked.

All in all, this first experiment (which involved only one instance of CAMSHIFT tracker running at a time—no stereo vision yet) proved quite successful. The tracked object, a grey sponge, was tracked continuously for several minutes. When the object was shaken (rotated) very fast in the experimenter's hand, it was still tracked. This means that each iteration of CAMSHIFT managed to use the colour histogram successfully for its search. On the other hand, there were some stability issues with the major ellipse of the axis or when the object was occluded for a few seconds (the tracker would not completely lose it, but its tracked region would shrink to the very small visible portion of the sponge during the occluded sequence, predictably making the axis shake between various orientations).

Fig. 5.2 shows the execution of another CAMSHIFT tracking task. Here, we are computing (and displaying) not the reconstructed extremities p_1 and p_2 of the major axis of the ellipse, but rather p_1 and the estimated object centroid. Fig. 5.2b shows the 16-bin colour histogram after it has been initialized for this experiment (green is the colour the tracker will look for, at every iteration). Note that the histogram remains constant during the whole experiment, also providing some robustness against occlusions.

5.2 3D Reconstruction

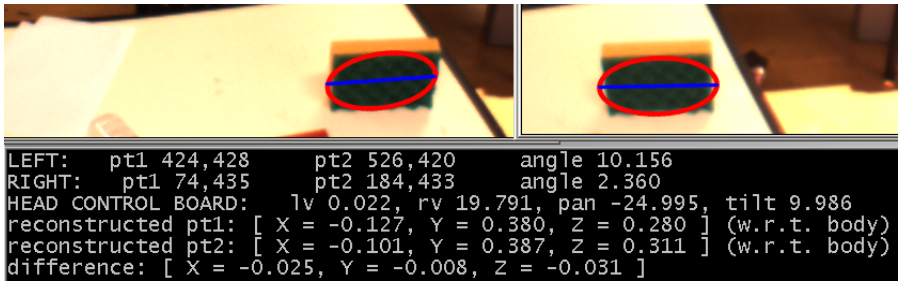


Figure 5.3: 3D reconstruction experiment.

In Fig. 5.3 we can see the output of the 3D reconstruction module during an experiment. Three windows are visible:

- left eye CAMSHIFT tracker;
- right eye CAMSHIFT tracker; and
- 3D reconstruction module output.

As far as the two trackers are concerned, we can see an optimal behaviour and drawing of the ellipses with their respective major axes. This contrasts with previous tracking experiments such as Fig. 5.1 for a number of reasons: first of all, the tracked sponge is not moving now. Because we were interested in measuring and judging the quality of our 3D reconstruction, in this experiment we chose a static scenario (there was some minor flickering and oscillation in the camera views during the video, but it was a negligible phenomenon, as the tracked axis kept stable throughout the whole experiment). Secondly, the tracked object was completely in front of the white table that is in front of Baltazar—greatly facilitating colour segmentation.

The most important part of this experiment, though, is the behaviour of the 3D reconstruction module. It is the black window with white text at the bottom of Fig. 5.3. The first three lines of text, written in capital letters, contain numerical values used internally by the 3D reconstruction module: coordinates in pixels, angles in degrees and lengths in metres. Then, the coordinates of the two extremities of the reconstructed axis (p_1 and p_2) are printed, in a (X, Y, Z) world reference frame, where Z is positive in the direction in front of the face of Baltazar. This means, for example, that a coordinate of

$$Z = 0.280$$

(metres) corresponds to 28 cm in front of the robot torso, on the white table.

The 3D reconstructed coordinates are correct in this experiment: they were checked with a ruler and the error along all the three dimensions was small (less than 5 cm).

Finally, the orientation of the target object (encoded as the different between reconstructed p_1 and reconstructed p_2) is printed.

5.3 Object Manipulation Tasks

5.3.1 Reaching Preparation

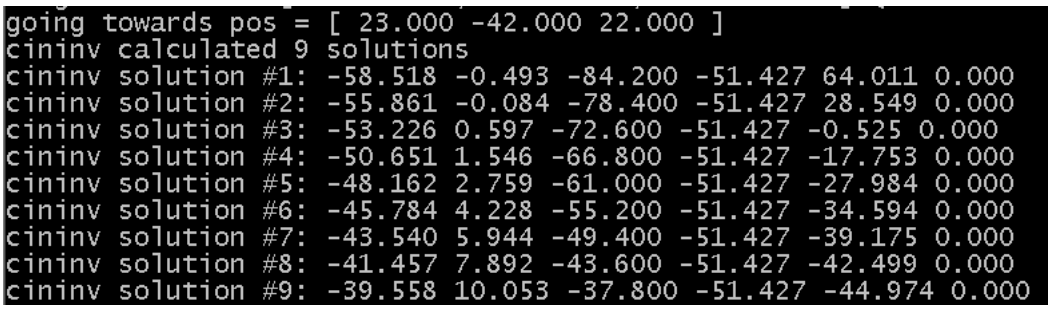


Figure 5.4: Object manipulation: inverse kinematics solver experiment.

Recall (p. 5) that in accordance to our perceptual framework we have split the reaching task in two distinct phases:

reaching preparation: an open-loop ballistic phase to bring the manipulator to the vicinity of the target, whenever the robot hand is not visible in the robot's cameras;

grasping preparation: a closed-loop visually controlled phase to accomplish the final alignment to the grasping position.

We shall now focus on the first problem, reaching preparation. In this phase our aim is to position the anthropomorphic arm of Baltazar, initially outside of the cameras' field of view, in the "vicinity" of the target. To start off, we define this vicinity as the 3D reconstructed coordinates of the centroid of the target object, minus a safety threshold of 20 cm along the horizontal axis (parallel to the table and to the ground, directed towards the right of the robot, from its own point of view). So, for this phase to be successful, we need to position the robot wrist at 20 cm from the object centroid. A necessary condition to accomplish this, is to solve the robot arm inverse kinematics for the desired hand position coordinates.

Fig 5.4 shows a number of inverse kinematics solutions found for Cartesian coordinates

$$X = 23, Y = -42, Z = 22 \quad \text{[cm]}.$$

Specifically, each of the 9 inverse kinematics solutions is a vector of 6 joint angles $\mathbf{q} = [q_1, q_2, q_3, q_4, q_5, q_6]$, expressed in degrees. The 6 angles correspond to the 6 encoder values that are actually streamed by the YARP arm server of Baltazar (see Table 3.5). Upon inspection of the computed results, two things immediately strike one's attention:

- q_4 (elbow) has the same constant value ($-51.427°$) for all the solutions;
- q_6 (wrist) is constantly zero.

The output of the fourth column, q_4, has always the same value because there exists a redundancy between this joint and the hand ones (several possible inverse kinematics solutions, depending on how "high" the elbow is). In particular, these joints only affect the orientation of the *hand*, so we applied a simplification here, to reduce the number of solutions. Also, recall that the end-effector of Baltazar is designed to be the base of the wrist (p. 35), not the palm of the hand or the tip of any finger.

Moving on from inverse kinematics to actual arm actuating, Fig. 5.5 shows an experiment of a reaching preparation task. Initially (Fig. 5.5a) the arm is positioned outside of the robot cameras' field of view, at a predefined position. Then it is moved within the field of view. Finally (Fig. 5.5c), the robot arm is correctly positioned in the vicinity of the 3D estimated object centroid coordinates, within a safety threshold of 20 cm from it.

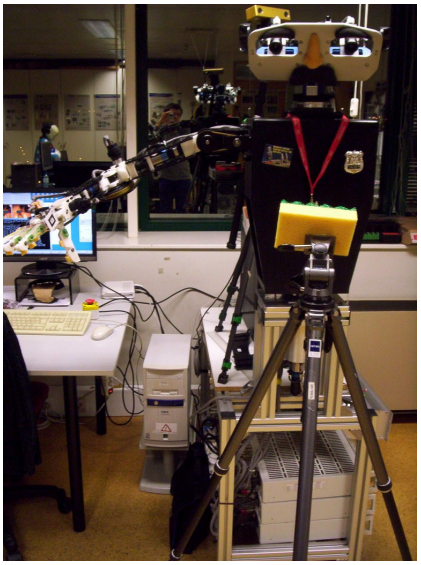
5.3.2 Grasping Preparation

The second and last phase of the reaching task is the "grasping preparation" phase. Contrary to reaching preparation, this task uses closed-loop feedback control. At the beginning of this task, hand and target object are already quite close (for example at a distance of 20 cm, in accordance to the desired safety threshold that we have imposed during the previous phase). The objective of grasping preparation is a more precise alignment of hand and target, thanks to a control law.

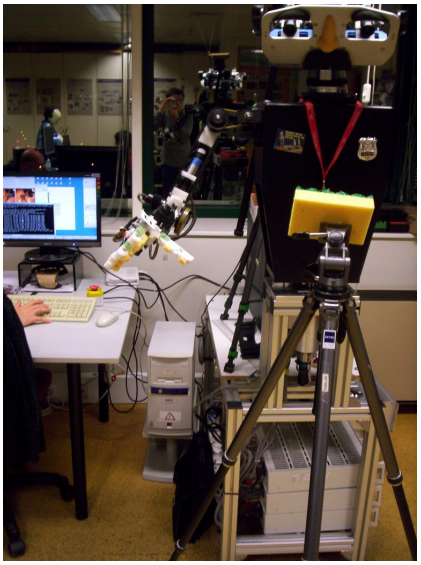
At the time of writing this thesis, experiments for this section were still at an initial stage. However, we did test the necessary arm control law computations and show here some results.

Fig 5.6 shows the hand of Baltazar wearing a latex glove in the vicinity of the target object. We applied this glove in an attempt to make colour segmentation (of the hand) more robust, thanks to a more uniform colour to be found by the search histogram of CAMSHIFT. This glove does not impede finger movement, so it is not a problem for the grasping itself.

Fig. 5.7 displays some tests done when tracking and 3D reconstructing both a target object and a robot hand at the same time. The relative angle–axis alignment is thus computed, as per Eq. 4.21. The initial results on this part are promising, as the three obtained angle values are similar to the real ones: 0, 45, 90 degrees, respectively.



(a) Arm is at a predefined position out of the robot field of view.

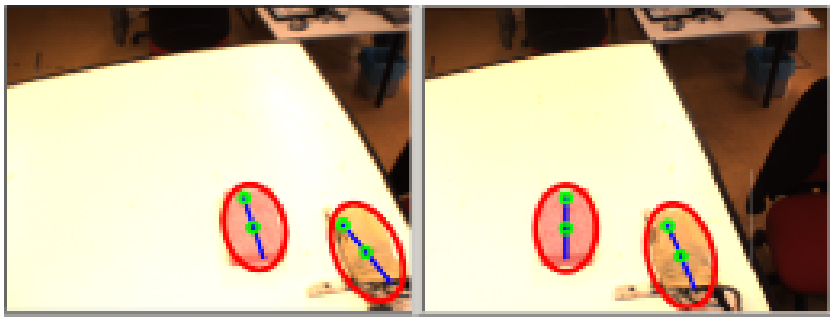


(b) Robot arm is now within the field of view of cameras.

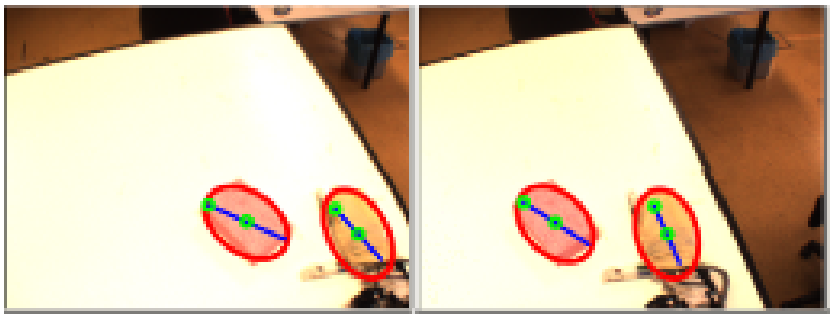
(c) Hand is finally positioned at target (minus a safety horizontal threshold).

Figure 5.5: Reaching preparation task experiment: the robot arm moves gradually towards the estimated centroid position of the target object.

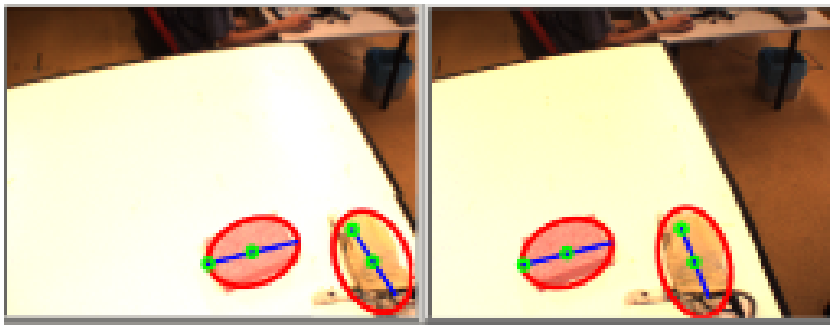
Figure 5.6: Baltazar robot hand wearing a glove, in order to make its colour more homogeneous, thus facilitating CAMSHIFT tracking and 3D reconstruction of the hand.



(a) Object and hand are parallel; $\mathbf{u} = (X = -0.006, Y = -0.062, Z = -0.041), \theta = 4.285°$.



(b) Object and hand have a relative slope of roughly 45°; $\mathbf{u} = (X = -0.129, Y = 0.737, Z = -0.129), \theta = 49.413°$.



(c) Orthogonality scenario; $\mathbf{u} = (X = -0.146, Y = 0.919, Z = 0.362), \theta = 87.408°$.

Figure 5.7: Evaluated axis $\mathbf{u}$ and angle θ between tracked object and hand in several scenarios in three different stereo pairs.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis presented an approach to perform manipulation tasks with a robot by the means of stereopsis clues and certain desired characteristics: using simple, generic features (best-fit ellipses) so that we can handle many different object that the robot has not dealt with before, and managing real-time performance.

We have addressed the problem of reaching for an object and preparing the grasping action, according to the *orientation* of the objects that a humanoid robot needs to interact with. The proposed technique is not intended to have very accurate measurements of object and hand postures, but merely the necessary quality to allow for successful object–hand interactions and learning with *affordances* (Section 2.4). Precise manipulation needs to emerge from experience by optimizing action parameters as a function of the observed effects.

To have a simple model of object and hand shapes, we have approximated them as 2D ellipses located in a 3D space. An assumption is that objects have a sufficiently *distinct colour*, in order to facilitate segmentation from the image background. Perception of object orientation in 3D is provided by the second-order moments of the segmented areas in left and right images, acquired by a humanoid robot active vision head.

As far as innovations are concerned, the Versatile 3D Vision system "VVV" (Tomita *et al.*, [TYU+98]) presents some analogies with our approach, in fact it can construct the 3D geometric data of any scene when two or more images are given, by using structural analysis and partial pattern matching. However, it works under the strong assumption that the geometric CAD models of objects are known beforehand in a database. This is a relevant difference from our proposed approach, which, instead, is model-free.

The Edsinger Domo (p. 7) is also similar to our proposed approach, in the sense that it emphasizes the importance for a robot to constantly perceive its environment, rather than relying on internal models. While the Edsinger Domo focuses on sparse perceptual features to capture just those aspects of the world that are relevant to a given task, we focus specifically on simplified object features: best-fit enclosing ellipses of objects and their estimated orientation in

3D.

6.2 Future Work

With regards to visual processing and tracking, the combined CAMSHIFT and 3D reconstruction approach can potentially be made more stable by using not just two, but more points to characterize each object (for example, by taking into account the minor axis of every ellipse in addition to its major one). However, this modification could increase computational cost and its viability needs to be verified.

As for manipulation, future work intends more thorough testing of the two phases (reaching preparation and grasping preparation), in particular of the latter.

Another improvement will be the combination of this work with the object affordances framework, thus adding a learning layer to the approach (for example by iterating many grasping experiments and assigning points to successful tests).

Appendix A

CLAWAR 2008 Article

Figure A.1: The logo of CLAWAR Association.

We now include a copy of the original paper [SB08] published in the proceedings of the 11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR 2008) held in Coimbra, Portugal on 8–10 September 2008.

Pose Estimation for Grasping Preparation from Stereo Ellipses

Giovanni Saponaro[1], Alexandre Bernardino[2]

[1] *Dipartimento di Informatica e Sistemistica "Antonio Ruberti" Sapienza - Università di Roma, via Ariosto 25, 00185 Rome, Italy*

[2] *Institute for Systems and Robotics - Instituto Superior Técnico Torre Norte, Piso 7, Av. Rovisco Pais, 1049-001 Lisbon, Portugal*

giovanni.saponaro@gmail.com, alex@isr.ist.utl.pt

This paper describes an approach for real-time preparation of grasping tasks, based on the low-order moments of the target's shape on a stereo pair of images acquired by an active vision head. The objective is to estimate the 3D position and orientation of an object and of the robotic hand, by using computationally fast and independent software components. These measurements are then used for the two phases of a reaching task: (i) an initial phase whereby the robot positions its hand close to the target with an appropriate hand orientation, and (ii) a final phase where a precise hand-to-target positioning is performed using Position-Based Visual Servoing methods.

Keywords: Reaching, Grasping, 3D Pose Estimation, Stereo, Visual Servoing.

1. Introduction

Grasping and manipulation are among the most fundamental tasks to be considered in humanoid robotics. Like humans distinguish themselves from other animals by having highly skilled hands, humanoid robots must consider dexterous manipulation as a key component of practical applications such as service robotics or personal robot assistants.

The high dexterity present in human manipulation does not come for granted at birth, but it arises from a complex developmental process across many stages. Babies first try to reach for objects, with very low precision; then they start to adapt their hands to the shape of the objects, and only at several years of age they are able to master their skills. Together with the manipulation, perception develops in parallel in order to incrementally increase performance in detecting and measuring the important object features for grasping. Along time, interactions with objects of diverse shapes

are performed, applying many reaching and manipulation strategies. Eventually, salient effects are produced (e.g. the object moves, deforms, makes a sound when squeezed), perceived and associated to actions. An agent learns the object affordances,[10] i.e. the relationships between a certain manipulation action, the physical characteristics of the object and the observed effect. The way of reaching for an object evolves from a purely position-based mechanism to a complex behavior which depends on target size, shape, orientation, intended usage and desired effect.

Framed by the context of the EU project RobotCub,[9] this work aims at providing simple 3D object perception for enabling the development of manipulation skills in a humanoid robot. The objective of the RobotCub project is to build an open-source humanoid platform for original research on cognitive robotics, focusing especially on developmental aspects. Inspired by recent results in neurosciences and developmental psychology, one of the tenets of the RobotCub project is that manipulation plays a key role in the development of cognitive ability.

This work puts itself in an early stage of this developmental pathway and will address the problem of reaching for an object and preparing the grasping action according to the orientation of the objects to interact with. It is not intended to have a very precise measurement of object and hand postures, but merely the necessary quality to allow for successful interactions with the object. Precise manipulation will emerge from experience, by the optimization of action parameters as a function of the observed effects.[10] To have a simple enough model of object and hand shape, they are approximated as 3D ellipses. The only assumption is that objects have a sufficiently distinct color to facilitate segmentation from the background. Perception of object orientation in 3D is provided by the second-order moments of the segmented areas in the left and right images, acquired in the humanoid robot active vision head.

The paper will describe the humanoid robot setup, computer vision techniques, 3D orientation estimation, the strategy to prepare the reaching and grasping phases, and experimental results.

2. Robotics setup

The robotic platform of RobotCub, called the *iCub*, has the appearance of a three-year-old child, with an overall of 53 degrees of freedom (see Fig. 1). However, the iCub's arm-hand system is still under development and for this work the robot *Baltazar*[7] was used: it is a robotic torso built with the aim of understanding and performing human-like gestures, mainly for

biologically inspired research (see Fig. 1).

To reach for an object, two distinct phases are considered:[8] (i) an open-loop ballistic phase is used to bring the manipulator to the vicinity of the target, whenever the robot hand is not visible in the robot's cameras; (ii) a closed-loop visually controlled phase is used to make the final alignment to the grasping position. The open-loop phase (reaching preparation) requires the knowledge of the robot's inverse kinematics and a 3D reconstruction of the target's posture. The target position is acquired by the camera system, where the hand position is measured by the robot arm joint encoders. Because these positions are measured by different sensory systems, the open-loop phase is subject to mechanical calibration errors. The second phase, grasping preparation, operates when the robot hand is in the visible workspace. 3D position and orientation of target and hand are estimated in a form suitable for Position-Based Visual Servoing (PBVS).[4,6] The goal is to make the hand align its posture with respect to the object. Since both target and hand postures are estimated in the same reference frame, this methodology is not prone to mechanical calibration errors.

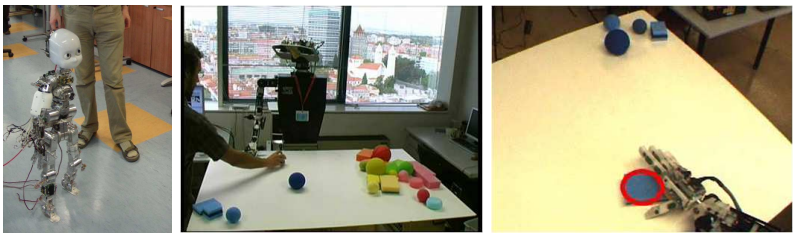


Fig. 1. Left: RobotCub humanoid platform iCub. Middle: humanoid robot Baltazar in its workspace. Right: view from one of Baltazar's eyes during a grasping task.

2.1. *Software architecture*

The software architecture used in this project is based on YARP[a], a cross-platform, open-source, multitasking library, specially developed for robotics. YARP facilitates the interaction with the devices of humanoid robot Baltazar, as well data exchange among the various software components (middleware). Other libraries used are OpenCV[b] for image pro-

[a]Yet Another Robot Platform: `http://eris.liralab.it/yarp`.
[b]Open Computer Vision Library: `www.intel.com/technology/computing/opencv`.

cessing, and GSL[c] for efficient matrix computation, especially in the 3D reconstruction part (see Sec. 3.2).

Particular care was put into designing the several components of the project as distributed. YARP takes care of inter-process communication (IPC), while the several concurrent instances of the CAMSHIFT tracker (left and right view of the target object, left and right view of the robot hand) can run on different machines or CPU cores: as modern processors sprout an increasing number of cores, the code can thus take advantage of the extra power available and improve real-time performance.

3. Visual processing

Using computer vision to control the grasping task is natural, since it allows to recognize and to locate objects (see Ref. 5 and Ref. 6). In particular, stereo vision can help robots reconstruct the 3D scene and perform visual servoing. In this work, the CAMSHIFT tracking algorithm[2,3] was used extensively. A brief outline of it is given in the next section.

3.1. *CAMSHIFT algorithm*

Originally designed for the field of perceptual user interfaces and face tracking,[3] CAMSHIFT is a method based on color histograms and MeanShift,[1] which in turn is a robust, non-parametric and iterative technique that finds the mode of a probability distribution, in a manner that is well suited for real-time processing of a live sequence of images.

A sketch of the algorithm logic and a sample execution are presented in Fig. 2. For this project, a modified version of the CAMSHIFT implementation publicly available in OpenCV was used. The inputs are the current original image obtained from the camera and its color histogram in the HSV (hue, saturation, value) space. The output of each iteration of CAMSHIFT is a "back projected" image, produced by the original image by using the histogram as a lookup table. When it converges, a CAMSHIFT tracker returns not only the position, but also the size and 2D orientation of the best-fit ellipse to the segmented target points. Then, the boundary points in the ellipse along its major axis are computed.

Consider a *target object* placed in front of the robot; tracking is accomplished by running two CAMSHIFT processes. Let points $\{p_1, p_2\}_l^{target}$ and

[c]GNU Scientific Library: `http://www.gnu.org/software/gsl/`.

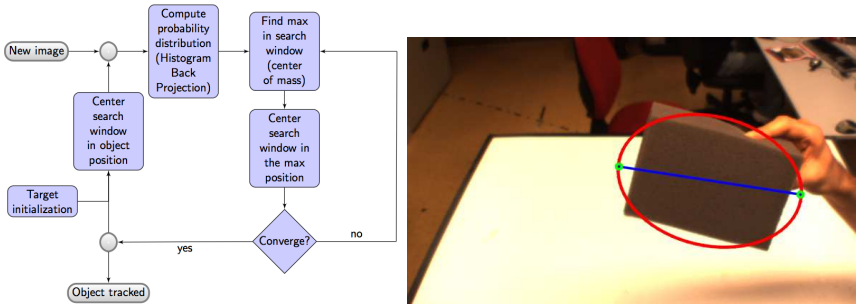


Fig. 2. Left: Flux diagram of the CAMSHIFT object tracking algorithm. Right: CAMSHIFT tracking of an object. The approximating best-fit ellipse is drawn in red, the major axis in blue, and the extremities of the axis are small green circles.

$\{p_1, p_2\}_r^{target}$ be the extremities of the major ellipse axis expressed in the 2D coordinate frame of the left and right tracker, respectively[d].

3.2. *3D reconstruction*

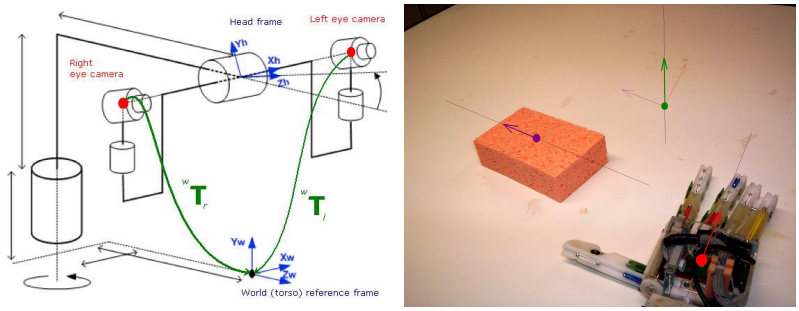


Fig. 3. Left: mechanical structure of Baltazar's head, reference frames and points of interest; transformation matrices are highlighted in green. Right: unit vector of the target object along its orientation axis (purple), versor and orientation of robot hand (red), and third axis resulting from their cross product, and corresponding unit vector (green).

A 3D reconstruction process receives the coordinates of the four points $\{p_1, p_2\}_{l,r}$ as inputs, along with the instantaneous head joint angle values of the robot, used to compute the time-varying extrinsic camera parameter

[d]The same considerations apply for stereo tracking and 3D reconstruction of the robot *hand*, but for the sake of simplicity only the target object case is explained in this paper. From now on, the "target" exponent in the notation is therefore omitted.

matrices: not just the target object, but also the robot cameras may be moving during experiments. Transformation matrices ${}^{w}\mathbf{T}_{l}$ and ${}^{w}\mathbf{T}_{r}$ represent the roto-translations occurring, respectively, from the left and right camera reference frame to the world (torso) reference frame, as shown in Fig. 3.

Once the reconstruction is computed, 3D coordinates of $\{p_1, p_2\}$ are obtained. The difference vector $p_1 - p_2$ encodes the orientation of the target.

4. Reaching and grasping preparation

As mentioned in Sec. 2 and Ref. 8, two distinct phases in reaching and grasping preparation are considered.

Reaching preparation: this first phase aims at bringing the robot hand to the vicinity of the target. It is applied whenever a target is identified in the workplace but the hand is not visible in the cameras. The measured 3D target position is used, in conjunction with the robot arm kinematics, to place the hand close to the target. Inevitably, there are mechanical calibration errors between arm kinematics and camera reference frames, so the actual placement of the hand will be different from the desired one. Therefore, the approach is to command the robot not to the exact position of the target but to a distance safe enough to avoid undesired contact both with the target and the workspace.

Grasping preparation: in this phase, both target and hand are visible in the camera system and their posture can be obtained by the methods previously described. The goal is now to measure the position and angular error between target and hand, and use a PBVS approach to make the hand converge to the target. The features used in such an approach are 3D parameters estimated from image measurements—as opposed to Image-Based Visual Servoing (IVBS), in which the features are 2D and immediately computed from image data. There are, however, two peculiarities in the presented approach:

(1) Normally, PBVS requires the 3D model of the observed object to be known,[4,6] but in this project one gets rid of this constraint: by using the stereo reconstruction technique explained in Sec. 3.2, the only condition to prepare the servoing task is that the CAMSHIFT trackers are actually following the desired objects—whose models are *not* known beforehand.

(2) Classical PBVS applications consider that target and end-effector positions are measured by different means, e.g. target is measured by the camera and end-effector is measured by robot kinematics. This usually leads to problems due to miscalibrations between the two sensory systems. Instead, in this work, target and hand positions are measured by the camera

system in the same reference frame, therefore the system becomes more robust to calibration errors.

Having computed 3D position and orientation of both a target object and of the robot hand, features suitable for the application of the PVBS technique must be obtained. As described in Ref. 4, the robot arm can be controlled by the following law:

$$\begin{cases} \mathbf{v} = -\lambda \left((\mathbf{t}_t - \mathbf{t}_h) + [\mathbf{t}_h]_\times \vartheta \mathbf{u} \right) \\ \omega = -\lambda \vartheta \mathbf{u} \end{cases} \tag{1}$$

where $\mathbf{v}$ and ω are the arm linear and angular velocities, λ establishes the trajectory convergence time, $\mathbf{t}_t$ and $\mathbf{t}_h$ are the target and hand positions; $\vartheta, \mathbf{u}$ are the angle-axis representation of the rotation required to align both orientations. Other control laws can be applied to this problem, but most of them rely on an angle-axis parameterization of the rotation. In this case, it is possible to calculate the required angle ϑ and axis $\mathbf{u}$ by applying a simple cross-product rule between the normalized hand and target orientation vectors:

$$\mathbf{u} = o_{target} \times o_{hand} \qquad \text{and} \qquad \vartheta = \arcsin \|\mathbf{u}\|_{L^2} \tag{2}$$

where $\| \cdot \|_{L^2}$ is the Euclidean norm, o_{target} is a unit vector in the direction of the target object's reconstructed orientation and o_{hand} is a unit vector in the direction of the hand's reconstructed orientation (see Fig. 3).

5. Experiments and results

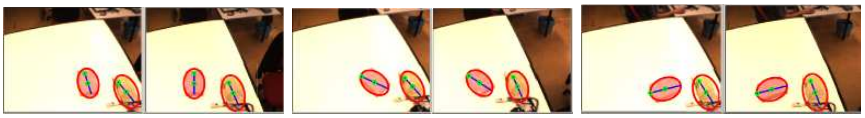


Fig. 4. Evaluated axis $\mathbf{u}$ and angle ϑ between tracked object and hand in several scenarios, in three different stereo pairs. Left: object and hand are parallel – $\mathbf{u} = (X = -0.006, Y = -0.062, Z = -0.041)$, $\vartheta = 4.285°$. Middle: about 45° – $\mathbf{u} = (-0.129, 0.737, -0.129)$, $\vartheta = 49.413°$. Right: orthogonality scenario – $\mathbf{u} = (-0.146, 0.919, 0.362)$, $\vartheta = 87.408°$.

Keeping in mind that the aim of this work is not high accuracy, but good qualitative estimations in order to interact with objects in front of the robot (see Sec. 1 and Ref. 10), the precision obtained is satisfactory. Fig. 4 shows the obtained results, estimated through Eq. (2).

6. Conclusions and future work

A simple algorithm for reaching and grasping preparation in a humanoid robot was presented in this paper. The method does not assume any particular shape model for the hand and objects, and it is robust to calibration errors. Although not relying on high precision measurements, the method will provide a humanoid robot with the minimal reaching and grasping capabilities for initiating the process of learning object manipulation skills from self-experience.

Future work includes evaluating the proposed technique with actual servoing and grasping experiments, as well as improving the pose estimation method by using the minor axis of ellipses in addition to the major one.

Acknowledgments

Work supported by EC Project IST-004370 RobotCub, and by the Portuguese Government - Fundação para a Ciência e Tecnologia (ISR/IST pluriannual funding) through the POS_Conhecimento Program that includes FEDER funds. The authors also want to thank Dr. Manuel Lopes for his guidance on visual servoing.

References

1. A. J. Abrantes, J. S. Marques, *The Mean Shift Algorithm and the Unified Framework*, ICPR, p. I: 244–247, 2004.
2. J. G. Allen, R. Y. D. Xu, J. S. Jin, *Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces*, 2003 Pan-Sydney Area Workshop on Visual Information Processing, Vol. 36, pp. 3–7, 2004.
3. G. R. Bradski, *Computer Vision Face Tracking for Use in a Perceptual User Interface*, Intel Technology Journal, 2nd Quarter 1998.
4. F. Chaumette, S. Hutchinson, *Visual Servo Control, Part I: Basic Approaches*, IEEE Robotics & Automation Magazine, Vol. 13, Issue 4, 2006.
5. Y. Dufournaud, R. Horaud, L. Quan, *Robot Stereo-hand Coordination for Grasping Curved Parts*, BMVC, pp. 760–769, 1998.
6. S. Hutchinson, G. D. Hager, P. I. Corke, *A Tutorial on Visual Servo Control*, IEEE Transactions on Robotics and Automation, Vol. 12, Issue 5, 1996.
7. M. Lopes, R. Beira, M. Praça, J. Santos-Victor, *An anthropomorphic robot torso for imitation: design and experiments*, IROS 2004, Japan, 2004.
8. M. Lopes, A. Bernardino, J. Santos-Victor, *A Developmental Roadmap for Task Learning by Imitation in Humanoid Robots: Baltazar's Story*, AISB 2005 Symposium on Imitation in Animals and Artifacts, UK, 12-14 April 2005.
9. G. Metta *et al.*, *The RobotCub Project: An Open Framework for Research in Embodied Cognition*, IEEE-RAS ICHR, December 2005.
10. L. Montesano, M. Lopes, A. Bernardino, J. Santos-Victor, *Learning Object Affordances: From Sensory-Motor Maps to Imitation*, IEEE Transactions on Robotics, Special Issue on Bio-Robotics, Vol. 24(1), February 2008.

Appendix B

Trigonometric Identities

Formulas for rotation about the principal axes by θ:

$$\mathbf{R}_X(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \tag{B.1}$$

$$\mathbf{R}_Y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \tag{B.2}$$

$$\mathbf{R}_Z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{B.3}$$

Identities having to do with the periodic nature of sine and cosine:

$$\sin\theta = -\sin(-\theta) = -\cos(\theta + 90^\circ) = \cos(\theta - 90^\circ), \tag{B.4}$$

$$\cos\theta = \cos(-\theta) = \sin(\theta + 90^\circ) = -\sin(\theta - 90^\circ). \tag{B.5}$$

The sine and cosine for the sum or difference of angles θ_1 and θ_2, using the notation of [Cra05]:

$$\cos(\theta_1 + \theta_2) = c_{12} = c_1c_2 - s_1s_2, \tag{B.6}$$

$$\sin(\theta_1 + \theta_2) = s_{12} = c_1s_2 + s_1c_2, \tag{B.7}$$

$$\cos(\theta_1 - \theta_2) = c_1c_2 + s_1s_2, \tag{B.8}$$

$$\sin(\theta_1 - \theta_2) = s_1c_2 - c_1s_2. \tag{B.9}$$

The sum of the squares of the sine and cosine of the same angle is unity:

$$c^2(\theta) + s^2(\theta) = 1. \tag{B.10}$$

If a triangle's angles are labeled a, b and c, where angle a is opposed side A, and so on, then the *law of cosines* is

$$A^2 = B^2 + C^2 - 2BC\cos a. \tag{B.11}$$

The *tangent of the half angle* substitution:

$$u = \tan\frac{\theta}{2}, \tag{B.12}$$

$$\cos\theta = \frac{1-u^2}{1+u^2}, \tag{B.13}$$

$$\sin\theta = \frac{2u}{1+u^2}. \tag{B.14}$$

To rotate a vector $\mathbf{Q}$ about a unit vector $\hat{\mathbf{K}}$ by θ, we use *Rodrigues's formula* which yields the rotated $\mathbf{Q}'$:

$$\mathbf{Q}' = \mathbf{Q}\cos\theta + \sin\theta(\hat{\mathbf{K}} \times \mathbf{Q}) + (1-\cos\theta)(\hat{\mathbf{K}} \cdot \hat{\mathbf{Q}})\hat{\mathbf{K}}. \tag{B.15}$$

Bibliography

[AM04] Arnaldo J. Abrantes and Jorge S. Marques. The Mean Shift Algorithm and the Unified Framework. In *International Conference on Pattern Recognition*, volume I, pages 244–247, August 2004.

[AXJ04] John G. Allen, Richard Y. D. Xu, and Jesse S. Jin. Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces. In Massimo Piccardi, Tom Hintz, Sean He, Mao Lin Huang, and David Dagan Feng, editors, *2003 Pan-Sydney Area Workshop on Visual Information Processing (VIP2003)*, volume 36 of *CRPIT*, pages 3–7, Sydney, Australia, 2004. ACS.

[Bei07] Ricardo Beira. Mechanical Design of an Anthropomorphic Robot Head. Master Thesis in Design Engineering, Instituto Superior Técnico, Lisbon, Portugal, December 2007.

[Bra98] Gary R. Bradski. Computer Vision Face Tracking for Use in a Perceptual User Interface. *Intel Technology Journal*, 2nd Quarter 1998.

[BSV99] Alexandre Bernardino and José Santos-Victor. Binocular Visual Tracking: Integration of Perception and Control. *IEEE Transactions on Robotics and Automation*, 15(6):1080–1094, December 1999.

[BW04] Thomas Brox and Joachim Weickert. Level Set Based Image Segmentation with Multiple Regions. In *DAGM*, Lecture Notes in Computer Science, pages 415–423. Springer, 2004.

[Can86] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[Car07] Paulo Carreiras. PREDGRAB – Predição de Trajectórias de Alvos Móveis. Master Thesis in Electrical and Computer Engineering, Instituto Superior Técnico, Lisbon, Portugal, September 2007. In Portuguese.

[CCIN08] Daniele Calisi, Andrea Censi, Luca Iocchi, and Daniele Nardi. OpenRDK: A Modular Framework for Robotic Software Development. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, September 2008.

[CH06] François Chaumette and Seth Hutchinson. Visual Servo Control, I: Basic Approaches. *IEEE Robotics & Automation Magazine*, 13(4):82–90, December 2006.

[CH07] François Chaumette and Seth Hutchinson. Visual Servo Control, II: Advanced Approaches. *IEEE Robotics & Automation Magazine*, 14(1):109–118, March 2007.

[CM97] Dorin Comaniciu and Peter Meer. Robust Analysis of Feature Spaces: Color Image Segmentation. In *CVPR*, pages 750–755. IEEE Computer Society, 1997.

[Cor96] Peter I. Corke. A Robotics Toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, March 1996.

[Cor97] Peter I. Corke. *Visual Control of Robots: High-Performance Visual Servoing*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[Cra05] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Prentice Hall, 3rd edition, 2005.

[DHQ98] Yves Dufournaud, Radu Horaud, and Long Quan. Robot Stereo-Hand Coordination for Grasping Curved Parts. In *British Machine Vision Conference*, pages 760–769, 1998.

[FH86] Olivier D. Faugeras and Martial Hebert. The Representation, Recognition, and Locating of 3-D Objects. *International Journal of Robotics Research*, 5(3):27–52, 1986.

[FP02] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, August 2002.

[Fra04] Alexandre R. J. François. CAMSHIFT Tracker Design Experiments with Intel OpenCV and SAI. Technical Report IRIS-04-423, Institute for Robotics and Intelligent Systems, University of Southern California, July 2004.

[FRZ+05] Daniel Freedman, Richard J. Radke, Tao Zhang, Yongwon Jeong, D. Michael Lovelock, and George T. Y. Chen. Model-Based Segmentation of Medical Imagery by Matching Distributions. *IEEE Transactions on Medical Imaging*, 24:281–292, 2005.

[FvDFH95] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1995.

[GHJV00] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2000.

[GML+08] Nicola Greggio, Luigi Manfredi, Cecilia Laschi, Paolo Dario, and Maria Chiara Carrozza. Real-Time Least-Square Fitting of Ellipses Applied to the RobotCub Platform. In Stefano Carpin, Itsuki Noda, Enrico Pagello, Monica Reggiani, and Oskar von Stryk,

editors, *SIMPAR – Simulation, Modeling and Programming for Autonomous Robots, First International Conference, Venice, Italy*, volume 5325 of *Lecture Notes in Computer Science*, pages 270–282. Springer, November 2008.

[HHC96] Seth Hutchinson, Gregory D. Hager, and Peter I. Corke. A Tutorial on Visual Servo Control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, October 1996.

[HJLY07] Huosheng H. Hu, Pei Jia, Tao Lu, and Kui Yuan. Head Gesture Recognition for Hands-Free Control of an Intelligent Wheelchair. *Industrial Robot: An International Journal*, 34(1):60–68, 2007.

[HN94] Thomas H. Huang and Arun N. Netravali. Motion and Structure from Feature Correspondences: A Review. *Proceedings of the IEEE*, 82(2):252–268, February 1994.

[LBPSV04] Manuel Lopes, Ricardo Beira, Miguel Praça, and José Santos-Victor. An Anthropomorphic Robot Torso for Imitation: Design and Experiments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, 2004.

[Lin08] Tony Lindeberg. Scale-space. In Benjamin Wah, editor, *Encyclopedia of Computer Science and Engineering*, volume IV, pages 2495–2504. John Wiley and Sons, September 2008.

[Lop06] Manuel Lopes. *A Developmental Roadmap for Learning by Imitation in Robots*. PhD thesis, Instituto Superior Técnico, Lisbon, Portugal, May 2006.

[LSV07] Manuel Lopes and José Santos-Victor. A Developmental Roadmap for Learning by Imitation in Robots. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37(2):308–321, April 2007.

[Mac67] James B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In Lucien M. Le Cam and Jerzy Neyman, editors, *Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[MC02] Ezio Malis and François Chaumette. Theoretical Improvements in the Stability Analysis of a New Class of Model-Free Visual Servoing Methods. *IEEE Transactions on Robotics and Automation*, 18(2):176–186, April 2002.

[MFN06] Giorgio Metta, Paul Fitzpatrick, and Lorenzo Natale. YARP: Yet Another Robot Platform. *International Journal on Advanced Robotics Systems, Special Issue on Software Development and Integration in Robotics*, 3(1), March 2006.

[Mir05] Boris Mirkin. *Clustering for Data Mining: A Data Recovery Approach*. Chapman & Hall/CRC, 2005.

[MLBSV08] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and José Santos-Victor. Learning Object Affordances: From Sensory-Motor Maps to Imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

[MVS05] Giorgio Metta, David Vernon, and Giulio Sandini. The RobotCub Approach to the Development of Cognition. *Lund University Cognitive Studies*, 123:111–115, January 2005.

[Nes07] Torindo Nesci. Mean Shift Tracking. Master Thesis in Computer Engineering, Sapienza – Università di Roma, Rome, Italy, 2007. In Italian.

[OS88] Stanley Osher and James A. Sethian. Fronts Propagating with Curvature Speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[RAA00] Constantino Carlos Reyes-Aldasoro and Ana Laura Aldeco. Image Segmentation and Compression Using Neural Networks. In *Advances in Artificial Perception and Robotics CIMAT*, pages 23–25, 2000.

[SB08] Giovanni Saponaro and Alexandre Bernardino. Pose Estimation for Grasping Preparation from Stereo Ellipses. In Lino Marques, Aníbal T. de Almeida, M. Osman Tokhi, and Gurvinder S. Virk, editors, *11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR 2008)*, pages 1266–1273, Coimbra, Portugal, September 2008. World Scientific Publishing.

[SHM05] Mark W. Spong, Seth Hutchinson, and Vidyasagar Mathukumalli. *Robot Modeling and Control*. John Wiley & Sons, Inc., 2005.

[SKYT02] Yasushi Sumi, Yoshihiro Kawai, Takashi Yoshimi, and Fumiaki Tomita. 3D Object Recognition in Cluttered Environments by Segment-Based Stereo Vision. *International Journal of Computer Vision*, 46(1):5–23, January 2002.

[SM97] Jianbo Shi and Jitendra Malik. Normalized Cuts and Image Segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 1997)*, June 1997.

[Smi78] Alvy Ray Smith. Color Gamut Transform Pairs. In *SIGGRAPH 78*, volume 12, pages 12–19, August 1978.

[SS00] Lorenzo Sciavicco and Bruno Sicliano. *Robotica industriale. Modellistica e controllo di manipolatori*. McGraw-Hill, 2nd edition, 2000. In Italian.

[SS01] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2001.

[TK03] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 2nd edition, 2003.

[TV98] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

[TYU+98] Fumiaki Tomita, Takashi Yoshimi, Toshio Ueshiba, Yoshihiro Kawai, and Yasushi Sumi. R&D of versatile 3D vision system VVV. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*, volume 5, pages 4510–4516, San Diego, CA, USA, October 1998.

[UPH07] Ranjith Unnikrishnan, Caroline Pantofaru, and Martial Hebert. Toward Objective Evaluation of Image Segmentation Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):929–944, June 2007.

[Whi98] Ross T. Whitaker. A Level-Set Approach to 3D Reconstruction from Range Data. *International Journal of Computer Vision*, 29:203–231, October 1998.

Online references

[BLA] BLAS – Basic Linear Algebra Subprograms. http://www.netlib.org/blas/.

[CLA] CLAWAR – Climbing and Walking Robots and the Support Technologies for Mobile Machines. http://www.clawar.org/.

[CSA] MIT CSAIL – MIT Computer Science and Artificial Intelligence Laboratory. http://www.csail.mit.edu/.

[Eds] Edsinger Domo robot. http://people.csail.mit.edu/edsinger/domo.htm.

[Lab] LabVIEW – Laboratory Virtual Instrumentation Engineering Workbench. http://www.ni.com/labview/.

[LIR] LIRA-Lab – Laboratory for Integrated Advanced Robotics, Genoa, Italy. http://www.liralab.it/.

[Ope] OpenCV – Open Source Computer Vision Library. http://opencvlibrary.sf.net/.

[Rob] RobotCub – An Open Framework for Research in Embodied Cognition. http://www.robotcub.org/.

[Vis] VisLab – Computer and Robot Vision Laboratory, Institute for Systems and Robotics, Instituto Superior Técnico, Lisbon, Portugal. http://vislab.isr.ist.utl.pt/.

[Web] Webots. http://www.cyberbotics.com/.

[YAR] YARP – Yet Another Robot Platform. http://eris.liralab.it/yarp/.

Index