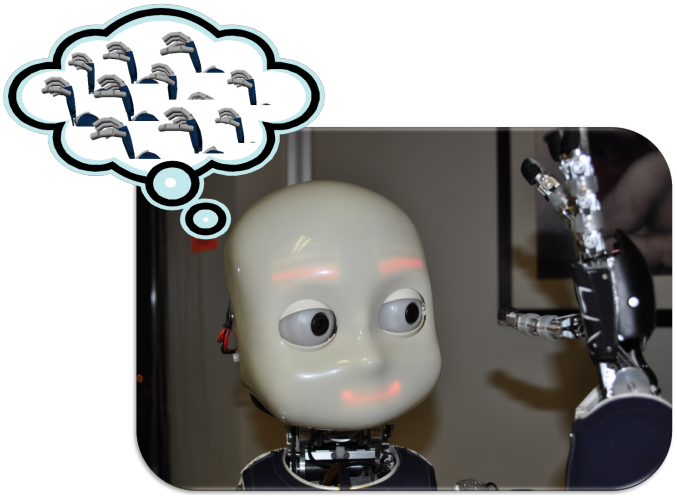
TÉCNICO LISBOA



Real time graphical simulation for visual based pose estimation and self-calibration of a humanoid robotic arm

Pedro Emanuel Antunes Vicente

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Professor Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Professor João Fernando Cardoso Silva Sequeira
Supervisor: Professor Alexandre José Malheiro Bernardino
Member of the Committee: Professor Paulo Jorge Coelho Ramalho Oliveira

October 2014

To my fiancee and my family.

Acknowledgments

Firstly, to God. For His amazing grace and endless love.

Secondly, I would like to express my gratitude to my fiancee for her love and for pushing me further. Thank you for your motivation, incentive and encouragement to continue in the battle, without you this Thesis would be just a dream unaccomplished. To my family for their support and love and for their unceasing presence in my life, and thank you Joana, for the reviews of the thesis.

Thirdly, to my supervisor Alex for the given opportunity to work at Vislab and to have a research experience, when I was just his graduated student. For his trust, his guidance in the thesis and for the general advices and conversations.

I would like to thank also to CEES: Ricardo, Filipe, Ana, Sofia, José e João for the técnico lunch time during the course. And to all my other friends who contributed directly or indirectly for my personal identity.

To Moutinho, Afonso, Giovanni, Miguel, Ricardo Nunes and all my lab mates for the exciting environment at the lab. To Ricardo Ferreira and Lorenzo Jamone for their help in reviewing the thesis. Last but not least, I would like to stress that this work was partially supported by FCT [PEst-OE/EEI/LA0009/2013] and the EU Project POETICON++ [FP7-ICT-288382].

Resumo

Nesta tese propomos um método para adaptação em tempo real da cinemática de um braço de um robô humanoide, usando os seus sensores visuais e proprioceptivos. O modelo interno do robô (esquema corporal interno) é representado por um modelo/simulador gráfico baseado num mecanismo de jogo: Unity3d®.

Um movimento típico para alcançar um objecto começa com uma fase balística em malha aberta para trazer a mão para a sua vizinhança. Durante esta fase, assim que a mão do robô entra no campo de visão de uma das suas cameras, um método de estimação da pose 3D da mão baseado em visão alimenta um filtro de particulas que, gradualmente, ajusta os parâmetros cinemáticos do braço. O nosso método usa um modelo CAD 3D da mão e do braço do robô (geometria e textura) em que a predição da sua posição na imagem é comparada (usando computação em GPU) com a informação vinda das cameras em cada iteração. Quando a mão chega perto do objecto, os erros cinemáticos foram reduzidos significativamente e um melhor controlo para o agarrar pode eventualmente ser alcançado.

Este método foi testado tanto em simulação como no robô real e verificou-se uma redução do erro de um factor de 3 durante a duração típica do movimento de aproximação.

Palavras-chave: Adaptação online, aprendizagem do modelo interno, seguimento baseado num modelo 3D usando GPU, aproximação para manipulação, robô humanoide

Abstract

In this thesis we propose a method for the online adaptation of a humanoid robot's arm kinematics, using its visual and proprioceptive sensors. The internal model of the robot (internal body schema) is represented by a graphical model/simulator based in the game engine Unity3d®.

A typical reaching movement starts with a ballistic open-loop phase to bring the hand to the vicinity of the object. During this phase, as soon as the hand of the robot enters the field of view of one of its cameras, a vision based 3D hand pose estimation method feeds a particle filter that gradually adjusts the arm kinematics' parameters. Our method makes use of a 3D CAD model of the robot hand and arm (geometry and texture) whose predicted position in the image is compared (with GPU enabled computation) at each time step with the cameras' incoming information. When the hand gets close to the object, the kinematic errors have reduced significantly and a better control of grasping can eventually be achieved.

We have tested the method both in simulation and with the real robot and verify error decreases by a factor of 3 during a typical reaching time span.

Keywords: Online adaptation, internal model learning, 3D model based tracking using GPU, reaching, humanoid robot

CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

$\#$	Quantity of

Greek symbols

β	Joint Offset.
θ	Encoders readings.
θ	Angle of one joint.

Roman symbols

bel	Belief.
I	Identity Matrix.
I_L	Left Image.
I_R	Right Image.
K	Kernel.
T	Transformation matrix
d	Distance.
E	Expected value or expectation.
R	Region of pixels in an image.
R	Rotation matrix

Subscripts

e, err	Error component.
i, t, k	Computational indexes.

r Real component.

x, y, z Cartesian components.

Superscripts

ˆ Estimated value.

T Transpose.

GLOSSARY

CAD Computer-aided design (CAD) is the use of computer systems to assist in the creation, modification, analysis, or optimization of a design.

CG Computer graphics is a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

CPU A central processing unit (CPU) is the hardware within a computer that carries out the instructions of a computer program by performing the basic arithmetical, logical, control, and input/output operations of the system.

CUDA Compute Unified Device Architecture is a parallel computing platform and programming model created by NVIDIA and implemented by the graphics processing units (GPUs) that they produce

CV Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions.

DBN Dynamic Bayesian Network is a Bayesian Network which relates variables to each other over adjacent time steps.

DIRECT DIViding RECTangles is a optimization algorithm and it was motivated by a modification to Lipschitzian optimization.

DOF Degrees of Freedom

EKF Extended Kalman Filter is an extension of KF to non-linear systems

EU European Union is a politico-economic union of 28 member states that are primarily located in Europe.

FPS Frames per second

GPU A graphics processing unit (GPU), also occasionally called visual processing unit (VPU), is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display.

HMM Hidden Markov Model is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved

IMU Inertial Measurement Unit is an electronic device that measures velocities, orientations, and gravitational forces, using a combination of accelerometers, gyroscopes and magnetometers

KDE Kernel Density Estimation is a non-parametric way to estimate the probability density function of a random variable.

KF Kalman Filter is an algorithm that uses a series of measurements observed over time, containing noise and produces estimates of unknown variables that tend to be more precise than those based on a single measurement alone.

MIT Massachusetts Institute of Technology is a private research university in Cambridge, Massachusetts

OpenCV Open Source Computer Vision is a library of programming functions mainly aimed at real-time computer vision

OpenGL Open Graphics Library is a cross-language, multi-platform application programming interface for rendering 2D and 3D vector graphics.

PSO Particle swarm optimization is a computational method that optimizes a problem by iteratively trying to improve a candidate solution

RGB-D Cameras with 4 channels - Red, Green and Blue colors and Depth data

Unity3d® Unity3d® is a game development ecosystem: a powerful rendering engine fully integrated with a complete set of intuitive tools and rapid workflows to create interactive 3D and 2D content;

YARP Yet Another Robot Platform (YARP) is an open-source middle-ware for humanoid robotics

CHAPTER 1

INTRODUCTION

Contents

Figure 1.1: The iCub humanoid robot performing a reaching task.

Humans acquire body awareness through a process of sensorimotor development that starts in early infancy [3], or most likely, already in the womb [4]. Such awareness is supported by a neural representation of the body that can be used to infer the limbs position in space and guide motor behaviors: a body schema [5].

Considering more specifically the visual based control of reaching, a form of visual-proprioceptive calibration of the body might be performed by infants during the first months of life, as they spend significant time observing themselves while moving [6]. Until four months reaching movements seem to be just “ballistic”, thus exploiting no visual feedback, as trajectory correction is absent [7, 8]. Then, from five months, vision is used to correct the hand position and orientation during the movement [9], with performance that improves during development [10]; however, after nine months this visual guidance almost disappears, as children become able to plan a proper hand trajectory at the movement onset [11]. Bushnell claims that this decline of visually guided reaching is fundamental for the further cognitive development of the child, as it frees a big portion of visual attention that can be thus devoted to perceive and learn other aspects of the experienced situations [7]. In addition, these observations suggest that an internal model might have been learned through sensorimotor experience during the first months, and later exploited to improve the control. Indeed, a more general theory of human motor learning and control postulates that forward and inverse internal models of the limbs are learned and kept up-to-date in the cerebellum [12]. While inverse models are used to compute the muscle activations required to perform a desired movement, forward models can be used to simulate motor behaviors and to predict the sensori outcomes of specific movements [13]. These predictions are exploited in different ways: for example, they are combined with the actual sensory feedback through Bayesian integration to improve the estimation of the current state of the system [14].

Clearly, endowing artificial agents with similar capabilities is a major challenge for cognitive developmental robotics.

From a wider perspective, having an accurate and robust model of the controlled system is fundamental for any robotic application. In case of complex robots (e.g. humanoids) it is typically very difficult to obtain an accurate analytical model of the system, due to hard-to-model aspects (e.g. elasticity) and

changes that might occur over time (e.g. unalignment of a joint rotation axis); therefore, learning from data is becoming a more and more popular approach to equip robots with the necessary adaptation capabilities (see [15, 16] for recent surveys).

The aim of this thesis is to improve the accuracy of an analytical model of the robot using visual information and Bayesian estimation techniques. In particular, we consider a visual based reaching scenario using the iCub humanoid robot [17], depicted in Figure 1.1. Instead of learning an internal model from scratch, we exploit the *iKin* kinematic model of the robot [18] provided within the YARP/iCub software framework, and we adapt it online during reaching movements in order to cope with the modeling inaccuracies, allowing the robot to precisely reach for a desired position and orientation.

Our solution draws some inspiration from human development and learning, as: i) the internal model is updated online based on the visual feedback of the hand (something that infants supposedly do between four and eight months), and ii) the estimation of the hand pose results from the Bayesian integration of the sensory (visual) feedback and the prediction made by an internal model (a strategy that seems to characterize human perception as well [14]).

1.1 State-of-the-art

One of the key components of our approach relies on the detection and tracking of the robot hand and its comparison with predictions formed by the current internal model. Several approaches have been proposed to track human hands with visual information [19], a complex problem due to the large number of degrees of freedom of a human hand. In [20] an approach is proposed to track and estimate the 26-DOF of a human hand model. It combines skin color segmentation and edge maps to evaluate hypotheses that are optimized with Particle Swarm Optimization methods. The method is computationally expensive but with custom GPU implementations, quasi-real time performance can be achieved. To simplify the matching problem [21] proposes the user to wear colored gloves and develops a method based on efficient search of a database of examples. The previous methods attempt to estimate the pose of the hand in an arbitrary configuration. In our case, using the model of the robot hand and forward kinematics, we have a good approximation of the hand pose and appearance, so the problem is more constrained and we can rely on local search techniques. In [22] an algorithm was proposed for estimating the pose of a human hand with a specific gesture. Because the hand posture is known, the problem reduces to a 6D search, which is further reduced to 3D search on an orientation database, since translation can be computed analytically using image moments. Our observation model is similar to that one, but we use it in a particle filtering framework to update the robot's internal kinematics model. A few recent works investigated visual detection of a robotic hand using machine learning techniques [23, 24]. Both systems are marker-free and model-free, and they employ either Online Multiple Instance Learning [23] or Cartesian Genetic Programming [24] to learn from visual examples how to detect the robot hand inside an image. In [23] information coming from arm motor encoders and visual optic flow is integrated to autonomously label the training images, thus obtaining an unsupervised learning system. However, both solutions deal only with the hand position in the image, and not its 6D pose in task

space. Different solutions have been proposed for the automatic calibration of the eye-head-arm-hand kinematic chain, that can allow accurate visual based reaching (some of them are reviewed in [15]). In [25], an upper humanoid torso is calibrated, including the sensor relative pose and the angle offsets and elasticity parameters of the kinematic chain. The method requires special markers in the wrist of the robot and operates offline with non-linear least squares optimization of data acquired during 5 minutes of robot specific movements. Online learning and adaptation of the kinematic model has been proposed as well [26, 27], but still using markers to visually detect the hand. Marker-free arm tracking has been investigated in [28], where RGB-D data from commercial depth sensors is used to correct the robot kinematics.

1.2 Contributions

In terms of visual estimation of the hand pose, our approach is to combine information coming from different channels in a Bayesian way: vision, proprioception, a model of the hand appearance and an internal model of the robot kinematics. Concerning the robot calibration, our objective is to exploit the visual estimation of the hand pose to incrementally correct the kinematic model of the robot during the movements. For this purpose, we develop a particle filter to track the hand based on a likelihood metric that compares a prediction of the visual observation of the robot hand according to the current internal model, with the real images acquired by the RGB cameras in the eyes of our platform, the iCub robot. The internal model is based in a visual simulator and the comparisons are made in the GPU, increasing the speed of the algorithm.

1.3 Dissertation Outline

In the remainder of this thesis we propose and explain an online visual based pose estimation system for humanoid robotic arms. It is organized in the following manner:

- In Chapter 2 a brief overview of the problem and the main steps to achieve the solution are introduced. Moreover, the mathematical models are presented.
- Chapter 3 contains the analysis of various filtering methods and explanation of the design decisions made due to the restrictions of the estimation problem.
- In Chapter 4 the experimental setup is explained in detail and the developed visual simulator (on GPU) is shown.
- The results of this thesis are presented in Chapter 5. The experiments were made in simulation and in the real robot.
- In Chapter 6 we express the achievements of this thesis making also references to future contributions in reaching and grasping environments.

CHAPTER 2

VISUAL BASED POSE ESTIMATION

Contents

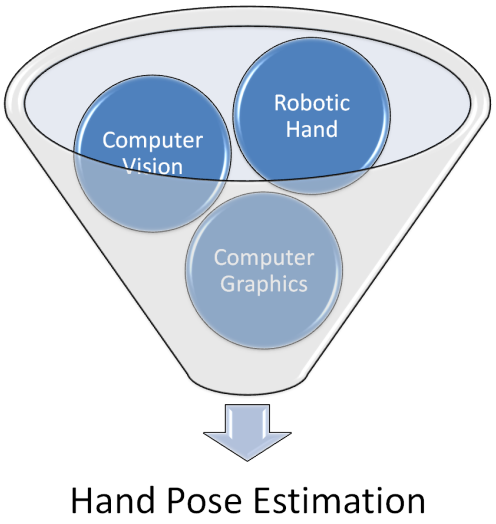


Figure 2.1: Estimation of the robotic hand pose combining Computer Vision and Computer Graphics

2.1 Problem formulation - an overview

In this thesis we propose to integrated Computer Vision and Computer Graphics fields in the estimation of the pose of a humanoid robotic hand (See Fig. 2.1) using a filtering strategy.

There are five main steps in order to achieve the final pose estimation goal. The first step, under computer vision field (CV), is the image segmentation of the robotic vision in order to maintain just the hand in the image, deleting the background environment. Computer graphics (CG) is present in the second and third stages where simulation hypotheses are generated in order to estimate the best hand pose, using image processing (from CV) to compare the real segmented image with the generated ones. The temporal estimation of our problem will be addressed using a mathematical filtering method (step 4). All together, we can estimate the hand pose of the robot (step 5).

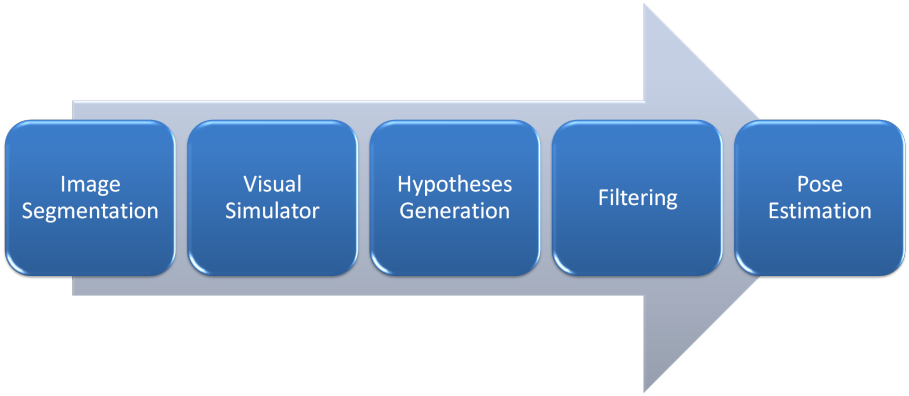


Figure 2.2: Pipeline of our strategy

2.1.1 Image segmentation

Image segmentation consists in the process of partitioning an image in various segments. Normally, these segments can represent different objects of interest or just different parts of an image. In the last case, images are usually divided in two groups: foreground, segments of the image to keep, and background, parts of the image with useless information. In the literature, segmentation can be based on color , objects' contours , edge detectors or image gradient and many other strategies in order to have image divided in segments as humans do in their daily life.

Image segmentation is one of the biggest challenges in computer vision. The solution presented in [29] behaves like the human visual system. A fixation point is used in the image to find the corresponding surrounding object combining monocular cues (color/intensity/texture). This solution was tested in our particular problem using as fixation an image point inside the robot hand from the estimated kinematic chain. Although, for our goal of performing a real-time approach, this segmentation is computationally heavy and not suitable for our case.

Actually, our option was to have a colored background during reaching and perform a color based segmentation. This approach is a simple solution for the segmentation problem and is also suitable to our particular case and a good starting point. The main focus of this thesis is to prove that we can estimate and calibrate a robotic arm using a graphical simulator, therefore the image segmentation method is a minor step and can be further improved for more uncontrolled environments.

2.1.2 Visual simulator

The majority of robots, particularly humanoid robots, are often equipped with a simulator used to test developed methodologies before applying them in the real robotic platform. Normally, this simulator includes not just the robot's kinematics model but its visual appearance, as well. Our goal is to utilize this visual simulator to perform comparisons with the real robot in real-time. Humanoid robots, in order to emulate the human eyes, have two cameras in the eyeballs to capture the world environment. We want to compare these incoming images with the ones generated in simulation, in order to estimate the hand pose and calibrate the robot arm. The robot proprioception (encoders) will be also used as a prior in the hand pose estimation jointly with the kinematic model of the torso, head and arm.

In our particular case we will use a visual simulator of the iCub robot. The simulator will be explained in detail in Section 4.2 of this thesis.

2.1.3 Hypotheses generation

The hypotheses will be generated with the visual simulator concept described before in section 2.1.2 and, for our particular setup, the simulator in Section 4.2. We will generate multiple hypotheses around estimated pose, according to the proprioception information from the real robot. The hypotheses will be compared with the robot stereo vision and weighted in order to choose the most suitable pose.

2.1.4 Filtering

The main goal of this step is to perform some dependency of the images and encoders over time. We want to estimate the pose of the hand and with this filtering strategy we will choose the best hypothesis that explains the majority of the evaluated frames and sensor values.

The filtering strategies proposed in the literature and our design solutions for the current problem will be analyzed in Chapter 3. The mathematical models used in this thesis in order to translate the problem concept in a mathematical way will be explain in Section 2.2 of the current Chapter.

2.1.5 Pose Estimation

The output of our approach will be the final pose estimate of the robotic hand concerning all the stages listed above. We do not want a distribution of hypotheses but an unique solution using the knowledge from two main fields - Computer Vision and Computer Graphics. The estimated pose is used to calibrate the robot kinematic chain.

2.2 Mathematical models

2.2.1 Joint Model

The online adaptation of the internal model consists in estimating joint offsets to the angles of the arm joints. The angular position of each joint of a generic robot can be modeled by:

$$\theta = \theta_r + \beta + \eta \tag{2.1}$$

where θ is the value read by the encoder, θ_r the real value of the joint position, β is a systematic offset, and η is zero mean Gaussian noise with covariance $\mathbf{Q}$; $\eta \sim N(0, \mathbf{Q})$. It is known that analytical kinematic models are not perfect and differences can occur between theory and practice. In our case, the goal is to estimate β from visual feedback of the robot images (right and left cameras), assuming negligible mechanical errors in the rotation axes alignment and link lengths. The estimated β for a particular reaching movement will incorporate information about all the errors in the kinematic chain, learning the best offsets that explain the observations, not necessarily the real offsets present in the joints. Actually, this approximation causes two different reaching movements with different final poses to have distinct β estimations. For this reason the Equation 2.1 can be re-written in a more general manner:

$$\theta = \theta_r + \beta(\theta_r) + \eta \tag{2.2}$$

where $\beta(\theta_r)$ is an offset dependent on the working area of the joint space. Furthermore, we assume joint errors to be independent, meaning that the covariance matrix $\mathbf{Q}$ is diagonal. Our humanoid robotic arm has seven (7) rotation joints each with a single degree of freedom. Our encoder readings ($\boldsymbol{\theta}$) are the seven joints of each arm defining: $\boldsymbol{\theta} = [\theta_0 ... \theta_6]$.

2.2.2 State Model

The state is a variable or a vector of variables used to characterize the environment. The robot's pose, velocities and accelerations are the most common states. In our problem the offsets in Equation (2.1) define the state vector of an unobserved Markov process as $\mathbf{x} = [\beta_0\ \beta_1\ \beta_2\ \beta_3\ \beta_4\ \beta_5\ \beta_6]^T$ where β_i is the offset in joint i of one of the arms of the humanoid robot.

The Markov assumption, also known as the complete state assumption, postulates that past and future data are independent if we know the current state of the system [2] [1]. With this assumption of the system we can use this equation as a simplification of the state's transition:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) \tag{2.3}$$

where $\mathbf{x}$ is the state vector, $\mathbf{y}$ the measurements and $\mathbf{u}$ the input controls.

In our particular case, since the joint offsets do not depend explicitly on the control the vector $\mathbf{u}$ will be empty. The measurements will be explained in detail in Section 2.2.3.

Our state has an initial distribution $p(\mathbf{x}_0)$ and a known state transition distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t)$. To allow for small changes in $\mathbf{x}$ we introduce a state transition noise $\mathbf{w}$ and model the system state transition as:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{w} \tag{2.4}$$

Here $\mathbf{w} \sim N(0, \mathbf{K})$ is zero mean Gaussian noise with a given covariance $\mathbf{K} = \sigma_s^2\mathbf{I}_7$, where σ_s is the standard deviation.

2.2.3 Observation Model

At each time we have two sources of information, the encoder readings ($\boldsymbol{\theta}$) and the left and right camera images, $\mathbf{I}_L$ and $\mathbf{I}_R$, respectively. The observation vector will be a concatenation of the values of the two images defined as $\mathbf{y} = [\mathbf{I}_L\ \mathbf{I}_R]$ with a distribution of $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_t)$. Given a certain state vector $\mathbf{x}_t$ and some given encoder readings $\boldsymbol{\theta}_t$, we can form a prediction on the observed images ($\hat{\mathbf{I}}_L$ and $\hat{\mathbf{I}}_R$). Let $[\hat{\mathbf{I}}_L\hat{\mathbf{I}}_R] = f(\boldsymbol{\theta}, \mathbf{x})$ be a function that receives an angular position of the joints, here defined as the composition of $\boldsymbol{\theta}$ and $\mathbf{x}$, and creates two images ($\hat{\mathbf{I}}_L$ and $\hat{\mathbf{I}}_R$) of the corresponding visible pose on the left and right eye, respectively. We implement this function (see Fig. 2.3) using a simulator (described in 4.2) that includes the forward kinematics of the robot and an image generation model. Note that f is nonlinear and two different sets of angles can generate the same image, or images with imperceptible differences. Redundancy in the joint angles may lead to different states ($\mathbf{x}$) with the same final hand pose. Therefore, the estimated $\mathbf{x}$ will be just one set of offsets that can explain our pose in the image.

To compute the image measurement probability $p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_t)$, we use the Hammoude metric [30] as a distance metric between the predicted and the real images. It is defined as:

$$d_{HMD}(y_1, y_2) = \frac{\#(\mathbf{R}_{y1} \cup \mathbf{R}_{y2}) - \#(\mathbf{R}_{y1} \cap \mathbf{R}_{y2})}{\#(\mathbf{R}_{y1} \cup \mathbf{R}_{y2})} \tag{2.5}$$

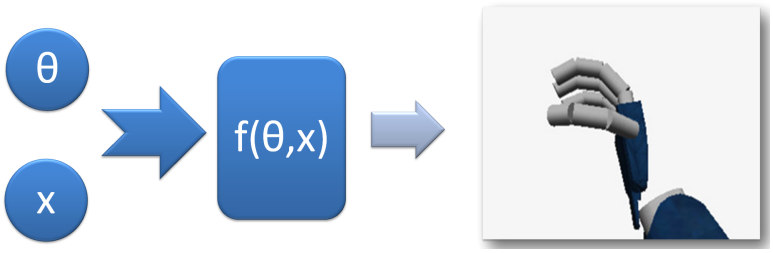


Figure 2.3: Image prediction model. We use the joint position ($\boldsymbol{\theta}$) in the iCub simulator to generate the predicted images corresponding to possible states.

where $\mathbf{R}_{y1}$ represents the region of predicted silhouette of the hand and $\mathbf{R}_{y2}$ is the region of the real silhouette. Since, the Hammoude distance has a range between [0,1] (where a 0 value represents two equal images and the value 1, two images with no intersection), we choose the likelihood to be proportional to:

$$p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_t) \propto 1 - d_{HMD}(\mathbf{y}_t, f(\boldsymbol{\theta}, \mathbf{x})) \tag{2.6}$$

This likelihood measures how reasonable is the approximation of y_2 by y_1. When it is equal to 1, the two binary images are equal (the Hammoude distance is Zero).

CHAPTER 3

STATE ESTIMATION AND FILTERING

Contents

In this chapter we will analyze possible solutions to our main goal and explain our design decisions regarding this particular problem. The state $\mathbf{x}$ will be estimated employing the observation model described before (Sec. 2.2.3). The observations are not directly accessible, just a likelihood score is available. Furthermore, we want to estimate the state during movement. The proposed solution must be able to preserve and use the previous state estimation in order to increase the accuracy of $\mathbf{x}$. Another issue is in the likelihood function of the state (offsets) which changes from an iteration $(i-1)$ to another (i) due to its dependency of the encoders readings. Additionally, the likelihood scoe also changes due to illumination, occlusion and other visual artifacts, thus preventing a direct association between its absolute value and the value of the pose error. Besides, an observability problem was identified: two similar poses can be generated by two different sets of offsets and different poses can have the same likelihood. These observability problems will be mitigated by using a sequential estimation process that integrate information along time.

To sum up, we have the following criteria to evaluate the estimation approach:

c.(i) Observations are likelihood score based.

c.(ii) Ability to use the previous estimation in step $i-1$ to improve the estimation in step i.

c.(iii) Likelihood function changes in each step.

c.(iv) Observability problem.

Different algorithms will be presented in the following sections in order to evaluate which is suitable to our case and does not violate the criteria above. For instance, Kalman Filter is one of the most popular algorithms used in estimation problems, although violates the criterion c.(i).

3.1 Bayes Filters

Bayes filters or *Sequential Bayes filters* is one group of methods that can be used to estimate the best state value in our problem. First of all, a Markov assumption of the system (eq. 2.3) is required. In our case, we also have a hidden Markov model (HMM) or dynamic Bayes Network (DBN), where we observe the measurements, but the state is not directly accessible (it is hidden).

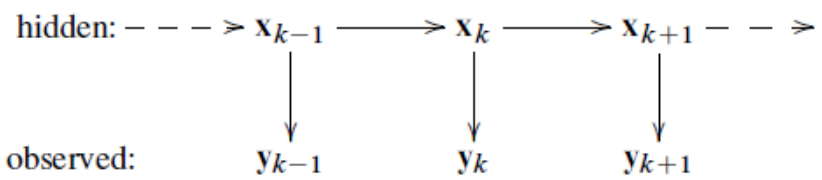
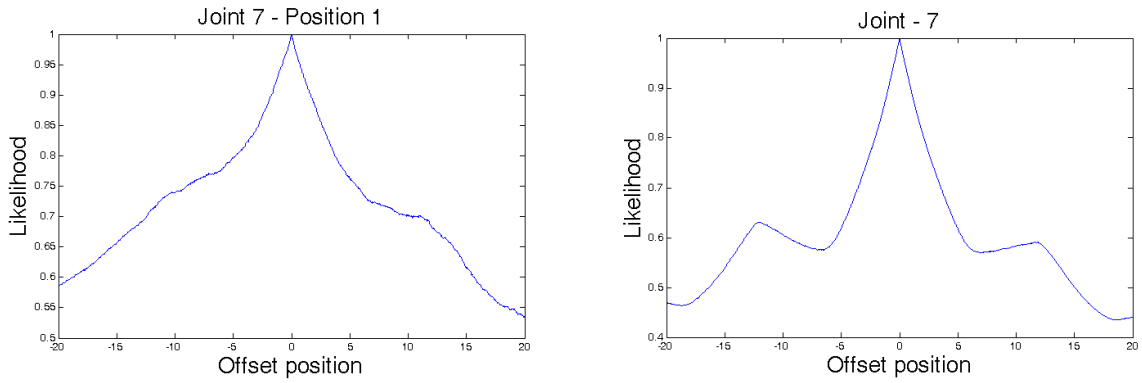


Figure 3.1: Schematic of an unobserved Markov process. Based on [1].

In these Bayesian filter methods the probabilistic Laws are the fundamental key for generating the state $\mathbf{x}$ and for progression of the algorithm. Assuming a Markov chain, the following equation is used for the Bayes filter evolution:



(a) Position 1 (b) Position 2

Figure 3.2: Value of the observation likelihood function as a function of the 7th joint angle, for two different arm configuration. Note the different forms of the likelihood function for the two configuration, actually multi-modal in the second case.

$$p(\mathbf{y}_t|\mathbf{x}_{0:t}, \mathbf{y}_{1:t-1}, \mathbf{u}_{1:t-1}) = p(\mathbf{y}_t|\mathbf{x}_t) \tag{3.1}$$

the probability $p(\mathbf{y}_t|\mathbf{x}_t)$ in eq. 3.1 is called the *measurement probability* and it quantifies the possibility that the measurements were generated by the state $\mathbf{x}$.

Another important concept is a *belief*. A belief, normally represented by a distribution, is the internal knowledge of the state based in the data. We can use the following nomenclature to define a belief over a state variable $\mathbf{x}$:

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{y}_{1:t}) \tag{3.2}$$

The principal advantage of the Bayesian methods is that they are recursive (sequential), therefore we can calculate $bel(\mathbf{x}_t)$ from $bel(\mathbf{x}_{t-1})$.

Bayesian filters strategies can further be divided in two main groups: the Gaussian and the Non-parametric Filters [2]. The first one assumes a belief with a Gaussian distribution and in the second one there is no *a priori* information about the posterior distribution.

3.1.1 Kalman Filter

The well known Kalman Filter (KF) method belongs to the Gaussian filters group and was introduced by Rudolf Kalman in 1960 [31]. In order to estimate the state vector it requires a linear system and the ability to model noise as a Gaussian distribution. The linear demand of this method can be worked around for non-linear systems using an approximation, the Extended Kalman Filter (EKF). EKF is used to non-linear systems and performs a linearization in the prediction and innovation steps, but the Gaussian distribution of the noise is still required. EKF is not suitable to our problem due to the non-linear observation model that leads to a multi-modal likelihood function (see Figure 3.2). To illustrate the multi-modal nature of the likelihood function we chose two reference positions and changed 0.1 degrees in the seventh joint (wrist yaw - θ_6). We can see that, for the two different initial positions, we get a dif-

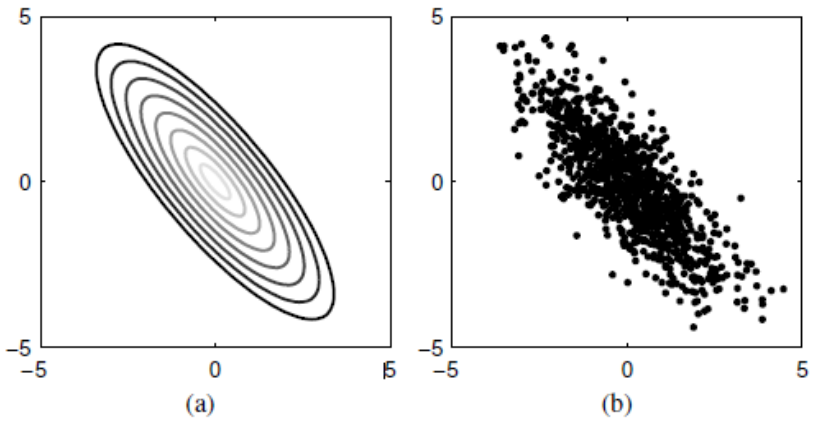


Figure 3.3: (a) A example of a Gaussian distribution with two dimensions (b) Monte Carlo representation. Based on [1].

ferent likelihood function with respect to the offsets in the joint. In the second case the likelihood has a multi-modal form. This multi-modal form causes the failure of the filter's convergence. Unfortunately, a Kalman filtering approach also requires an signed error between a hypothesis and a real observation instead of a likelihood score for the update step, which is not available violating criterion c.(i) (ignoring other smaller limitations such as the Gaussian uncertainty distribution requirements).

3.1.2 Particle Filter

Particle filter belongs to a group of Bayesian methods called *Non-Parametric Filters* and it is also considered a Monte Carlo method. The non-parametric filters do not assume a specific posterior distribution form as other Bayesian filters do. Particle filter, in particular, uses a set of random samples called *particles* to represent this distribution (the *belief*). The particle set can be denoted by:

$$\mathcal{X}_t := x_t^{[1]}, x_t^{[2]}, x_t^{[3]}.., x_t^{[M]} \tag{3.3}$$

It is also known as Sequential Monte Carlo method, since Monte Carlo methods approximate the target distribution based in a set of samples. In figure 3.3 we can see an approximation of a Gaussian distribution using a Monte Carlo method.

A general formulation of the particle filter algorithm can be seen in Algorithm 1. The filter receives the previous set of particles $\mathbf{x}_{t-1}$ and the actual measurements $\mathbf{y}_t$.

The particle filter has four stages: Prediction, Observation, Update and Re-sampling. The prediction Step is perform in Line 4, where we sample and predict using the transition probability (in our case equation 2.4) and the previous state. The Observation step uses the measurement probability 3.1 to weight each particle (Line 5). Update is carried out in Line 6 of the algorithm, where we improve the set with the new weight. The most important stage is from Line 8 to 11, where the Re-sampling is made. Some authors postulate that there is the core of the algorithm and where the real "trick" occurs.[2]

Re-sampling stage is based in a probabilistic and statistic concept called *importance sampling*. Importance sampling consists in estimating a distribution f (*target distribution*) with samples generated

Algorithm 1 Particle Filter Algorithm

procedure PARTICLE FILTER($\mathbf{x_{t-1}}, \mathbf{y_t}$)
  $\hat{\mathbf{x}}_t = \mathbf{x}_t = \emptyset$
  for $m \leftarrow 1, M$ **do** ▷ M is the number of particles
    sample $x_t^{[m]} \sim p(\mathbf{x}_t | x_{t-1}^{[m]})$
    $w_t^{[m]} = p(\mathbf{y}_t | x_t^{[m]})$
    $\hat{\mathbf{x}}_t = \hat{\mathbf{x}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  end for
  for $m \leftarrow 1, M$ **do** ▷ Re-sampling Stage
    draw x_i with probability $\propto w_t^{[i]}$
    add x_i to $\mathbf{x}_t$
  end for
  return $\mathbf{x_t}$
end procedure

from a different distribution g (*proposal distribution*). The goal is to weight the particles with the ratio $\frac{f(x)}{g(x)}$, measuring how close the sample x_t from $g(x)$ is to $f(x)$. In equation 3.4 we can see the expectation of f using this method:

$$\begin{aligned} E_f[I(x \in A] \quad &= \int f(x) I(x \in A) dx \\ &= \int \frac{f(x)}{g(x)} g(x) I(x \in A) dx \\ &= E_g[w(x) I(x \in A] \\ w(x) &\equiv \frac{f(x)}{g(x)} \end{aligned} \tag{3.4}$$

where $I(.)$ is an indicator function; it is 1 if its argument is true, and 0 otherwise. One graphical example can be seen in Figure 3.4, where in blue are represented the evolution of particles. In particles filter, the distribution after the *importance sampling* corresponds to a belief estimation, where we approximate the real belief distribution by the weights resulting from the *importance sampling*.

3.2 Other strategies

Some of the most used algorithms for estimation/optimization in this area are, as said in Sec.1.1, the particle swarm optimization (PSO), the Kalman filter method (KF) or the DIRECT optimization algorithm. Kalman filter was analyzed is section 3.1.1 and it is not suitable to our approach. A brief explanation of the other strategies will be carried out in this section.

3.2.1 Particle Swarm Optimization

PSO simulates the social behavior with a population (swarm) of candidates (particles) in a given problem. The basic idea is that each particle is searching for the optimum value and they will co-operate with each other to obtain it. Each individual keeps its personal best (the position where it had the best result so far) and it is "travelling" with a certain velocity and has a neighborhood of particles that can

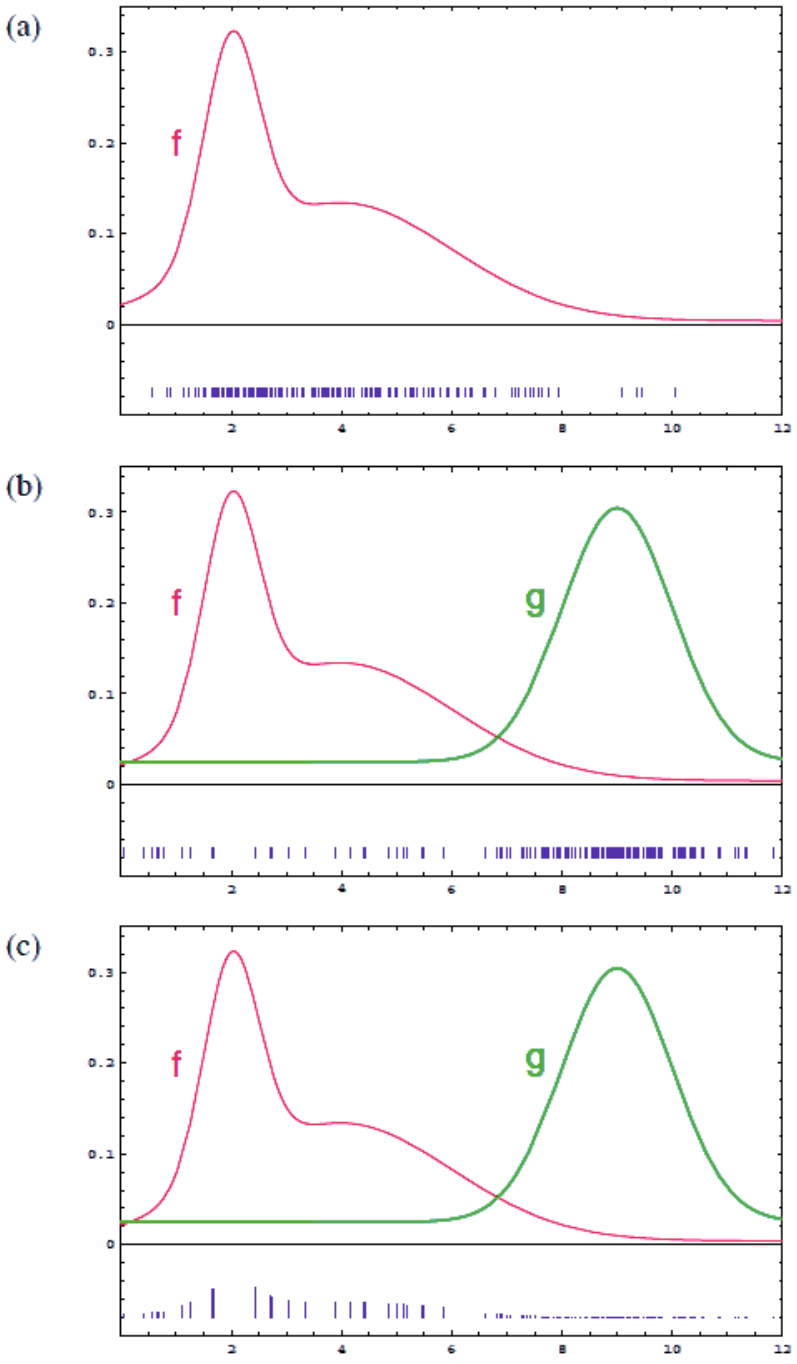


Figure 3.4: Importance sampling example. (a) f is the target distribution, we want to sample directly from f, but f is not accessible. (b) We generated particles/samples (in blue) for the distribution g . (c) We obtain approximated samples for f based in Eq. 3.4 and in the weight $w(x) = \frac{f(x)}{g(x)}$. Based on [2]

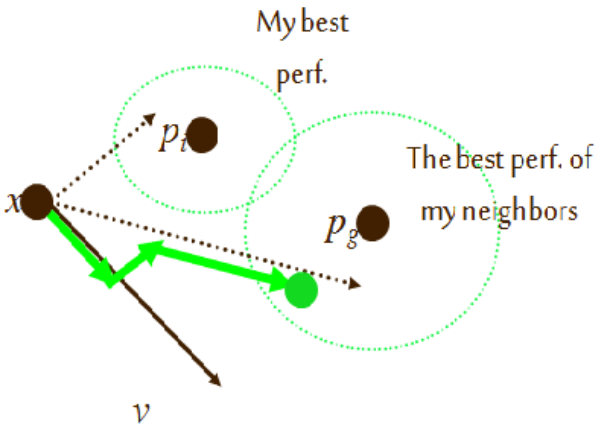


Figure 3.5: Explanation of the Particle Swarm Optimization(PSO) update. Where X is a particle.

be, for instance, geographical (nearest neighbors) or social (just a list of neighbors, regardless where they are). The adjustment of their position is based on a "Psychosocial compromise" between what an individual is comfortable with, and what society reckons. The velocity of a particle i will be updated as follows:

$$v_{i+1} = C_0 v_i + C_1 U(p_{best} - p_i) + C_2 U(pG_{best} - p_i) \tag{3.5}$$

where C_0, C_1 and C_2 are constants and p_i and v_i are the position and velocity of the particle i, respectively. p_{best} is the personal best and G_{best} the group best (the best position of the neighborhood so far). U is an uniform distribution between [0,1].

The position of each particle in the next step $(i + 1)$ will be updated as:

$$p_{i+1} = p_i + v_i \tag{3.6}$$

They will update their position in the search-space according to the current velocity, the personal best and the best position of the neighbors.

Our goal is to perform an "optimization" of the state $\mathbf{x}$ maximizing the similarities between the images (real and generated) as described in Section 2.2.3. The PSO breaks the criteria c.(iii) and c.(iv) since the personal best and the group best position can be no longer valid in further steps of the algorithm. For instance, in the case of c.(iv) the personal/group best can be a position that can not explain every image.

3.2.2 DIRECT algorithm

DIRECT stands for "DIviding RECTangles" it is a sampling algorithm. It was developed by Jones [32] in 1993. It divides the search-space recursively to get the optimum value of an objective function f. In our case we can achieve the minimum difference between the incoming and segmented images. The problem can be stated as:

$$\underset{\mathbf{x}\in\Omega}{\textit{minimize}} \quad f_0(\mathbf{x}) \tag{3.7}$$

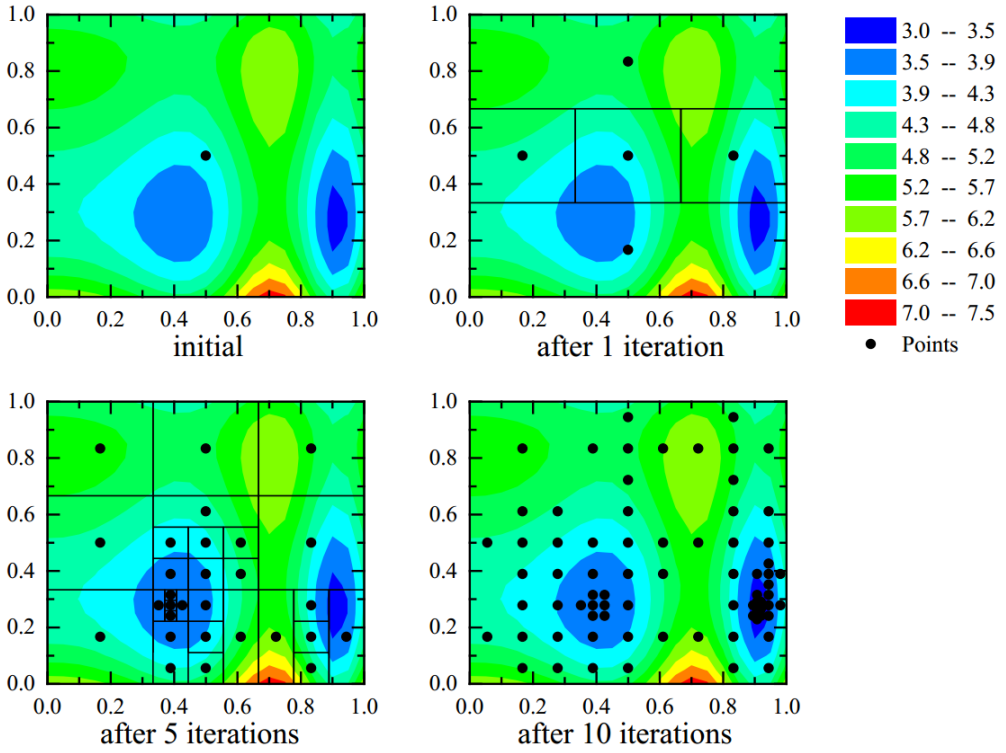


Figure 3.6: Direct algorithm example - First steps minimizing a objective function in a hypercube

where

$$f_0 : \mathbb{R}^n \rightarrow \mathbb{R}$$

and

$$\Omega = \{\mathbf{x} \in \mathbb{R}^n : l \leq \mathbf{x} \leq u\}$$

Firstly we normalize the design space D (in our case the joints' offset space) to be the unit hypercube, defining a lower(l) and an upper(u) bound in the space, and sample the center point c_i. In Figure 3.6 can be seen the first steps of the algorithm minimizing an objective function inside the hypercube.

In spite of being computationally heavy we could use DIRECT to achieve the best $\mathbf{x}$ for a single image (iteration). DIRECT fails in criterion c.(ii) and consequently in c.(iii) and c.(iv).

3.3 Particle Filter with GPU enabled likelihood computation

We have chosen the Particle Filter approach for this work. Particle Filters do not rely on any particular belief distribution, as said before, and they can estimate a belief with a multi-modal form. This is an important feature of this method since we have a non-gaussian and multi-modal likelihood function as can be seen in Figure 3.2.

We perform an adaptation of the general method described in Algorithm 1 and in Section 3.1.2 regarding our particular problem. In section 2.2.2 and 2.2.3 we have defined our state and observation model, respectively, that will be used to build our Particle Filter. In our case, we re-defined the standard belief as:

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{y}_{1:t}, \boldsymbol{\theta}_{1:t}) \tag{3.8}$$

which is the distribution of the state $\mathbf{x}$, at time t, conditioned on all past observations $\mathbf{y}_{1:t}$ and encoder readings $\boldsymbol{\theta}_{1:t}$. As said before, particle filters can compute recursively the *a posteriori* distribution ($bel(\mathbf{x}_t)$) using the previous estimation ($bel(\mathbf{x}_{t-a})$) and the observation model ($p(\mathbf{y}_t|\mathbf{x}_t, \boldsymbol{\theta}_{1:t})$), due to the fact that a Markov assumption is present.

Concerning the prediction step, our case is quite simple owing to the fact that the state transition equation (See Equation 2.4) is an equality with addition of Gaussian noise.

In the observation stage we generate an image for the state $\mathbf{x}$ using our internal model (visual Simulator) and compute the likelihood of each particle according to the observation model defined before in 2.2.3. The weight of each particle will be proportional to the computed likelihood, in other words, the weight will be proportional to the *measurement probability*.

In the update stage, we use the computed likelihood in the previous step to re-weight the particles, updating its importance in the estimation.

The re-sampling step is probably the most important stage in the filter for its convergence. In [2] they show some issues related to the re-sampling step. In order to mitigate this, we will use a *low variance sampling* algorithm, where the basic idea is not to choose particles independent of each other as in the usual re-sampling methods. One of the solutions is to measure the variance of the particles' weight and decide when to re-sample. If the weights are identical, the variance is zero and we shouldn't resample.

We use the systematic re-sampling method [33] that ensures the particles with a weight greater than 1/M to be always re-sampled, where M is the number of particles used. This method is a *low variance sampling* method and its computational simplicity and good empirical performance was a key feature for our choice. Many comparisons between different resampling methods were performed [33][34][35] and this method is better than the common "Multinomial Resampling" used in the bibliography.

First we generate M ordered random numbers:

$$u_k \qquad = \bar{u} + \frac{k-1}{M}, \text{with } \bar{u} \sim \ U[0, \frac{1}{M}] \tag{3.9}$$

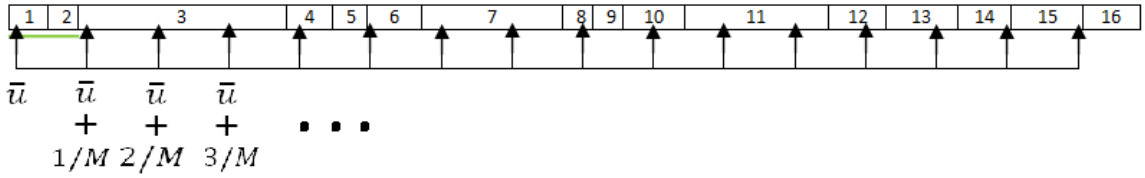
where u_k is an auxiliary variable to sample the new particle and k a index variable between 1 and M (number of particles to generate).

The new set particle $\mathbf{x}^*$ will be chosen from the previous set $\mathbf{x}$ as follow:

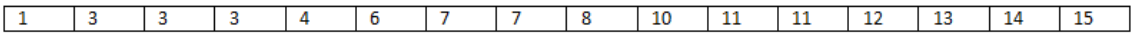
$$x_k^* = x(F^{-1}(u_k)) \tag{3.10}$$

where F^{-1} is a generalized inverse of the cumulative probability distribution of the normalized particles' weight. In other words, u_k points to exactly one particle of the set, namely the particle i for which:

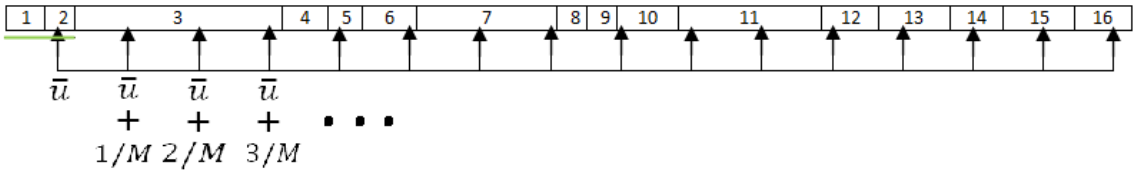
$$i = \arg\min_j \sum_{m=1}^{j} w_t^{[m]} \geq u_k \tag{3.11}$$



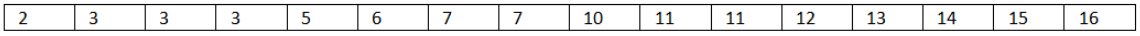
(a) CASE 1: Resampling Step



(b) CASE 1: Set after Resampling



(c) CASE 2: Resampling Set



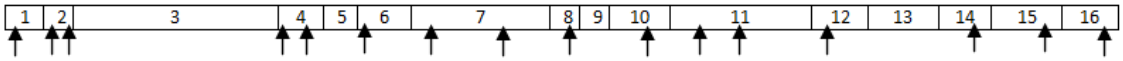
(d) CASE 2: Set after resampling

Figure 3.7: Example of a Systematic Resampling in a set of sixteen particles. The green line represents the possible values to initialize $\bar{u}$. The initial value of $\bar{u}$ is different in a) and c). In a) and c) the width of the particles is proportional to their weight. Particle 3 have the higher weight and it is the most resampled one. In b) and d) the particles have the same weight,although the sets are different because of $\bar{u}$ value.

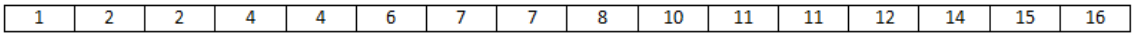
In Figure 3.7 an illustration of the method can be seen. The main difference between the two cases(a) and c)) is the generation of $\bar{u}$. The particle set is the same and the particles have the same weight in the two cases. In other words, particle 7 in Fig. 3.7 a) has the same weight as particle 7 in Fig. 3.7 c). The green line represents the possible values to initialize $\bar{u}$ and the arrows the selected particle to be resampled. We can observe that particles with a higher weight will have a higher probability of being resampled and due to the fixed step ($\frac{1}{M}$) between two consecutive samplings it is impossible to discard a particle with a weight greater than $\frac{1}{M}$. For instance in the given example, we would choose particle number 3 at least once. In Figure 3.8 a random sampling of the particles is performed. One can see that particle 3, the heaviest particle, was not resampled due to the independent distribution of the random numbers in the set.

One important remark of this method is that if the weights are identical the new set $\mathbf{x}^*$ will be equivalent to the previous one $\mathbf{x}$. This characteristic is important to ensure the convergence of the filter.

After re sampling we spread the particles using a normal distribution with zero mean and standard deviation σ_s, defined in section 2.2.2. This component of our particle filter solution will be, if necessary, adaptive to cope with the particle deprivation problem. When one tries to perform an estimation in a high-dimensional space, it is possible that no particles are generated in the neighborhood of the true state, this concept is called *particle deprivation*. Actually, adapting the spread of the particles over time along with our resampling choice could be a solution for this disadvantage of particle filters. We will decrease the standard deviation when the variance of the particle set reduces.



(a) Resampling Step



(b) Set after Resampling

Figure 3.8: Example of a Resampling using a uniform distribution in the same set of sixteen particles as in Fig. 3.7. Particle 3 has the highest weight in the set, despite it was not resample with this Re-sampling method

The designed solution is suitable to be processed in parallel, for instance in the GPU unit. The particles (x_i) are independent of each other and the generated images can be produced and evaluated simultaneously, in parallel.

This feature is an add-on for the particle filter approach. The likelihood, based on the observation model, is computed comparing the real image and the filter hypotheses within the GPU unit.

3.4 Kernel density estimation

Although the state is represented at each time step as a distribution approximated by the particles, for evaluation purposes we must compute our best guess of the value of the state. For this purpose, we use a kernel density estimation (KDE) to smooth the weight of the particles according to the information of their neighbor position and choose the particle with the highest weight (W_i) as our state estimate:

$$W_i = L_i + \alpha * KDE_i; \tag{3.12}$$

where L_i is the particle likelihood, α is a smoothing parameter enabling to sum two different entities/metrics and KDE_i is the influence of the neighbors:

$$KDE(\mathbf{x}) = \frac{1}{n}\sum_{i=0}^{n} L_i * K(\mathbf{x} - \mathbf{x_i}) \tag{3.13}$$

where n is the number of particles of the filter, $\mathbf{x} - \mathbf{x_i}$ is the distance (in offsets space) between the particle we are smoothing $\mathbf{x}$, and a neighbor $\mathbf{x_i}$. K is a kernel specifying the influence of one particle in other based on their distance. We use a Gaussian Kernel in our experiments:

$$K(\mathbf{x_1}, \mathbf{x_2}) = \frac{1}{\sqrt{2\pi|\Sigma|}}\, e^{[-\frac{1}{2}(\mathbf{x_1}-\mathbf{x_2})^T \Sigma^{-1}(\mathbf{x_1}-\mathbf{x_2})]} \tag{3.14}$$

where Σ is the co-variance matrix and $|\Sigma|$ its determinant.

We assume the joints are independent of each other, so Σ will be a diagonal matrix $\Sigma = \sigma_{KDE}^2 I_7$, where σ_{KDE} is the standard deviation in each joint, which we assume to be equal.This parameter defines if two particles are close or not. If we have a higher σ_{KDE} all particles will be “close” to each other. On the other hand if we have a small σ_{KDE} all particles will be “alone” in the world.

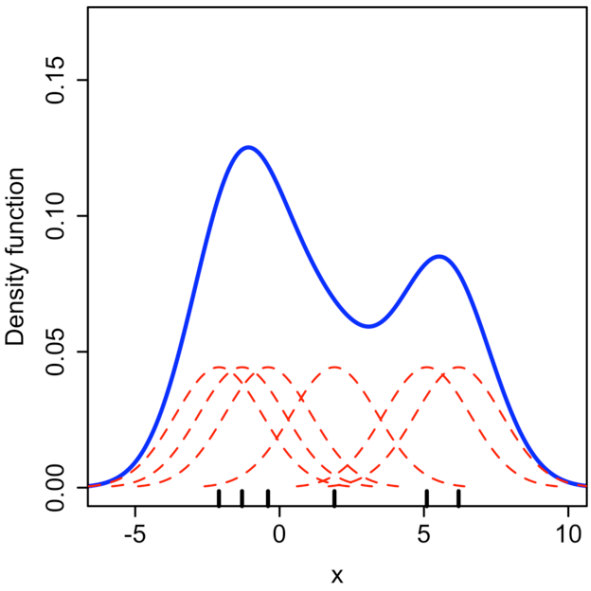


Figure 3.9: Estimation of the distribution based on the particles

In Fig. 3.9 we use, as example, six particles with only one dimension and with the same likelihood (L_i in Eq. 3.12). We can see the estimated distribution in blue using this KDE method. The particle chosen as the state estimate will be the closest to the maximum.

CHAPTER 4

EXPERIMENTAL SETUP

Contents

Figure 4.1: iCub Robot at Computer and Robot Vision Laboratory

4.1 The robotic platform

The iCub (see Figure 4.1) is a humanoid robot for research in embodied cognition, developed in the context of the EU project RobotCub (from 2005 to 2010) and subsequently adopted by more than 25 laboratories worldwide. It has 53 motors that move the head, arms and hands, waist, and legs; it has the average dimensions of a 3 years old child. It is equipped with stereo vision (cameras in the eyeballs), proprioception (motor encoders), touch (artificial skin and tactile fingertips) and vestibular sensing (IMU on top of the head).

iCub arm has three rotation joints in the shoulder (pitch, roll and yaw), one in the elbow and three in the wrist (wrist pronosupination, pitch and yaw). In Figure 4.2 an explanation of the human wrist movement with the respective angles in the iCub robot can be seen. The wrist pronosupination (θ_4) corresponds to pronation and supination movement, the pitch (θ_5) to the flexion and extension of the wrist and finally yaw (θ_6) is the radial and ulnar deviation.

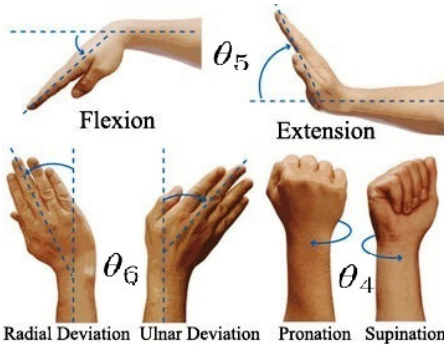


Figure 4.2: Wrist movements of the human hand with the respective encoders of the iCub robot

YARP (Yet Another Robotic Platform), developed by MIT and LIRA-Lab, is an open-source *middleware* for humanoid robotics. It is used in this robotic platform and it is responsible for the communication protocol between modules and different machines.

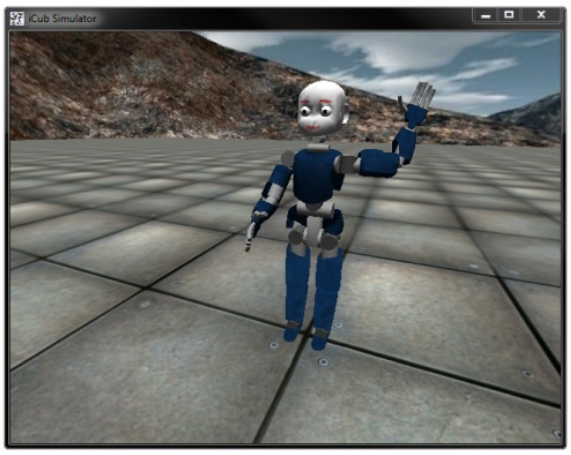


Figure 4.3: World view of the iCub simulator environment

The robot is equipped with a dynamic simulator [36] (See Fig. 4.3) that has been used to generate the predictions of the observations that are compared to the real images acquired by the cameras. The simulator behaves similarly to the real robot, emulating the encoders, motors and motor controllers. It was developed with an embedded physics engine (ODE) and turning it off is non viable without remaking all the simulation from scratch. For this reason, the iCub Simulator is not able to generate more than 30 to 60 *frames per second*.

In order to have a real-time application we developed, afterwords, a geometric model based on a game engine (Unity3D®) generating more than 500 *frames per second*. This geometric model uses a *collada* model of the iCub Robot (design by a student at VisLab) and we combine, for the first time, the YARP platform (for communication purposes), the OpenCV library (for image processing) and OpenGL and CUDA programming (to use the capabilities of the GPU) within Unity3D®. This model was able to generate poses (positions and orientations) based in motor commands, like the iCub robot, and to generate the right and left images, from the cameras in the eyeballs. In Section 4.2 we will address in detail the developed visual simulator.

4.2 Visual Simulator

As said before in Section 4.1, the iCub Simulator developed within the RobotCub European Project and available for several platforms is not suitable to our real-time demand. This simulator is based in OpenGL for the graphics pipeline and ODE for the physics emulation among other libraries.

We develop this new geometric simulator using C# programming, OpenGL and CUDA. In Fig. 4.4 the workspace of Unity3D® with the geometric model of iCub can be seen.

Every piece composing the iCub was imported from its CAD model and organized in a hierarchical way. This hierarchy is used to cope with “relationships” between joints rotation and graphical object’s poses in the virtual world, for instance, when one joint of the shoulder rotates every object attached to it (arm and hand) must perform a correspondent movement.

Despite Unity3D® has a build-in physics engine and we can use it to affect objects with gravity, forces

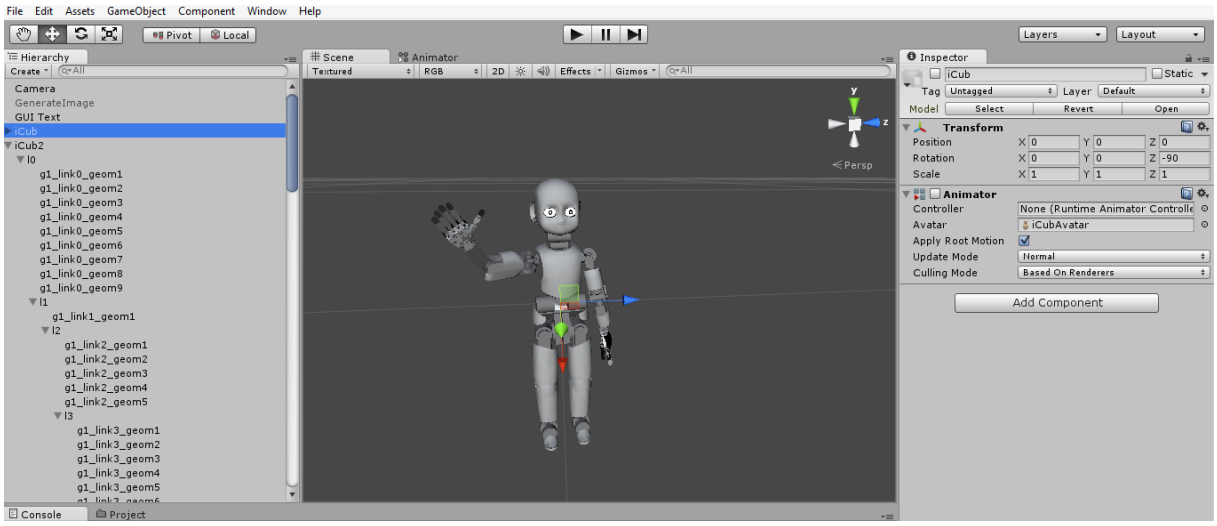


Figure 4.4: View of Unity3D® workspace with iCub model

and torques, we decide to turn off this feature in order to increase the speed of image generation. Our intention was to develop a geometric simulator capable to generate several images per second and turn-off the physics engine saves useful time.

The GameManager class/object (inside iCub object) was responsible to receive information, via YARP, about the encoders' position and the real images to compare with and to send back the likelihood score of each generated pose (particle) via another YARP port. The GameManager class communicates with each joint class sending the position command. Joints update their position and the game engine renders the actual state viewed by the cameras in the eyeballs. The observation model uses binary images comparison thus we don't define any source light in the simulator in order to generate black and white images.

The integration with OpenGL, CUDA and OpenCV took placed when we needed to compare the generated image and the received one from the real Robot. We used a rendered texture in OpenGL where Unity3D® renders the camera views and with CUDA programming we manage to copy this information to OpenCV GPU image class, providing methods to count non-zero pixels and to merge images. All the comparisons between the generated and real images as well as the likelihood computation were made in GPU increasing the speed of the data processing.

4.3 General flowchart

In this section we will provide some general view of the program work flow and how the several units communicate amongst them. Moreover, the particular work flow of each unit will be explained in order to understand its main goal.

In Fig. 4.5 we can see the work flow of one iteration of our method and four entities are present: Real Robot (1), Main Program (2), Simulator (3) and Estimated Offsets (4) and three communications procedures (I), II) and III)).

The Real Robot (1) can be either the real robot itself, or the simulation working as the Real robot, in

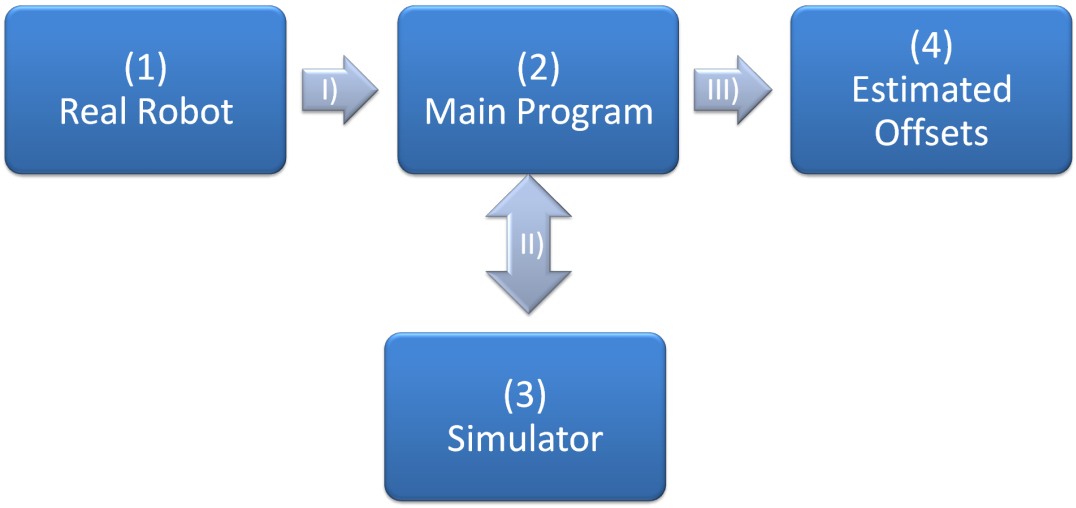


Figure 4.5: General Work Flow of the our approach using a simulator to generate hypotheses

other words, in the latter case we want to estimate the artificial errors (offsets) inserted in the simulation experiments. The robot is performing a reaching movement in a controlled environment seeing his hand in at least one of the cameras and it sends the encoder readings and the right and left images to the main program through the communication protocol I).

In the main program we generate particles according to our state vector (the offsets), that can be the initialization of the algorithm or the result of a previous re-sampling step. Jointly with the encoder readings we send through II) the information of the poses to be generated in the simulator. The images from the Real robot were processed in order to remove the background and kept just the Robot's hand. The background will be a colored plane which can be easily removed with a color-based segmentation method.

The simulator receives via II) the encoders position and generates one new image for each particle received to compare with the segmented image. In one hand, if the simulator (2) is the iCub Simulator in Fig. 4.3 only the encoders will be sent and the simulator will return the generated images to the main program which will compute the likelihood. In the other hand, if the simulator (2) is the unity simulator (Fig. 4.4) it will receive the encoders and the segmented image and return the likelihood score of each particle.

The main program, now with the likelihood of each particle, will implement our particle filter approach explained in 3.3 sending through III) the best estimated offsets for this iteration i.

4.3.1 CPU and GPU usage

The proposed algorithm equipped with the developed simulator works with CPU and GPU computation. In Fig. 4.6 the steps made from the robot to the final "Best estimated state" can be seen. As said before in Section 4.2 all the comparisons between images were made in GPU decreasing the computational time of the algorithm. The most considerable generation and evaluation of data is made in GPU

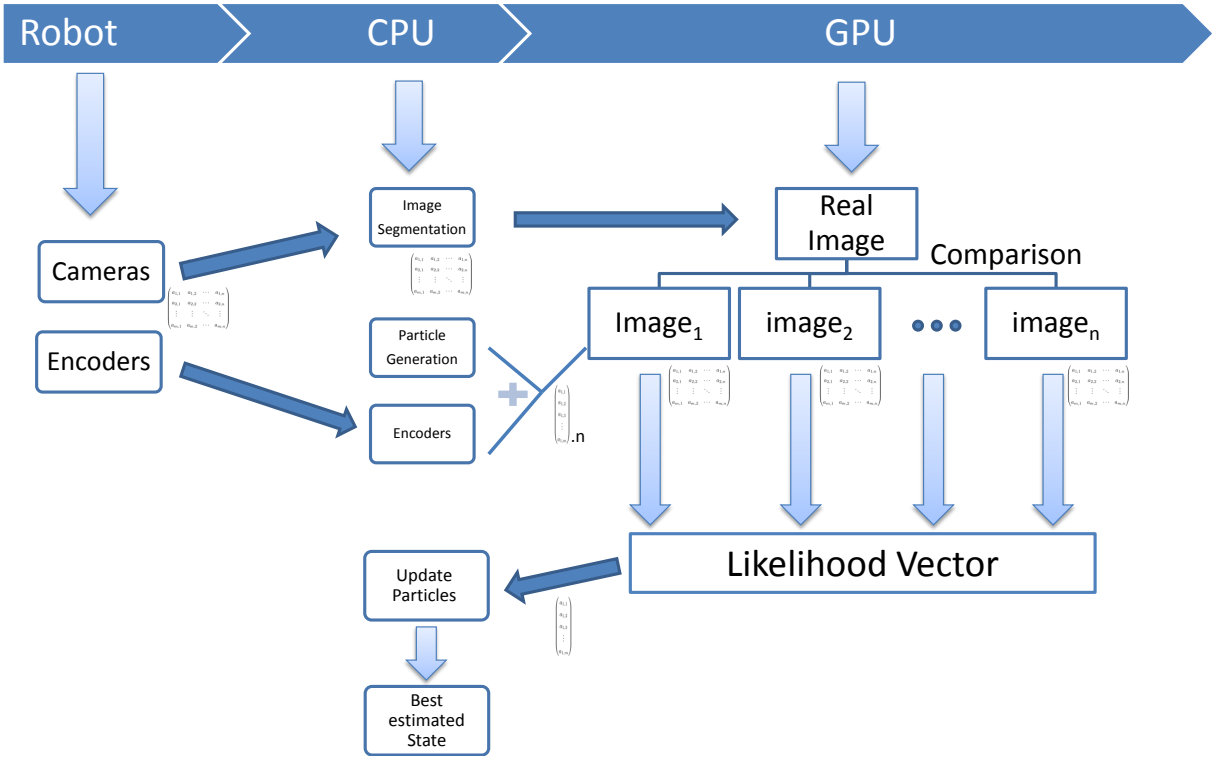


Figure 4.6: Scheme showing the operations made in CPU, GPU and at the robot

with parallel processing. The CPU is used to communicate with the robot receiving the data from the cameras and encoders and to do minor calculations as particles' generation implementing our particle filter method.

4.4 Error metrics

In order to evaluate the accuracy of our method we compute both the position and orientation errors between the real and estimated poses. The orientation error is defined as:

$$d(R_r, R_e) = \sqrt{\frac{\|logm(R_r^T R_e)\|^2}{2}} \frac{180}{\pi} \quad [^\circ] \tag{4.1}$$

where, R_r is the real rotation matrix from the eye frame to the end-effector and R_e is the estimated one. The principal matrix logarithm, $logm$, implements the usual distance on the group of rotations, one of the defined in [37].

The position error between the two positions is computed by euclidean distance:

$$d(P_r, P_e) = \sqrt{(x_r - x_e)^2 + (y_r - y_e)^2 + (z_r - z_e)^2} \tag{4.2}$$

where P_r is the real position of the end-effector in the eye reference frame in 3D Cartesian space and P_e is the estimated one.

CHAPTER 5

RESULTS

Contents

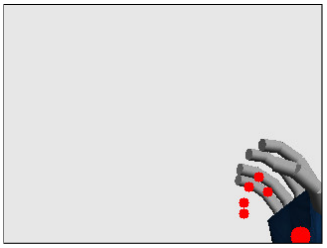


Figure 5.1: Estimated positions of the fingers tips (in red dots) at filter initialization. Note the difference to the real pose show in the simulator.

In this chapter we will show the results of various experiments. We have performed three types of tests. In the first test we use a simulated robot with artificial offsets in the arm joints and evaluate the convergence and accuracy of the filter. We consider observations taken both from one single camera and the stereo pair. In the second test we also use a simulated robot but this time we introduce also artificial offsets (calibration errors) in the head joints. Still, only the arm offsets are estimated by the filter. The head offsets are used to assess the robustness of the method to unmodelled sources of uncertainty. Finally in the third experiment we use the real robotic platform and estimate the offsets of the real arm. Despite no ground truth being available, we compare the predicted and real camera images to assess the level of precision attained.

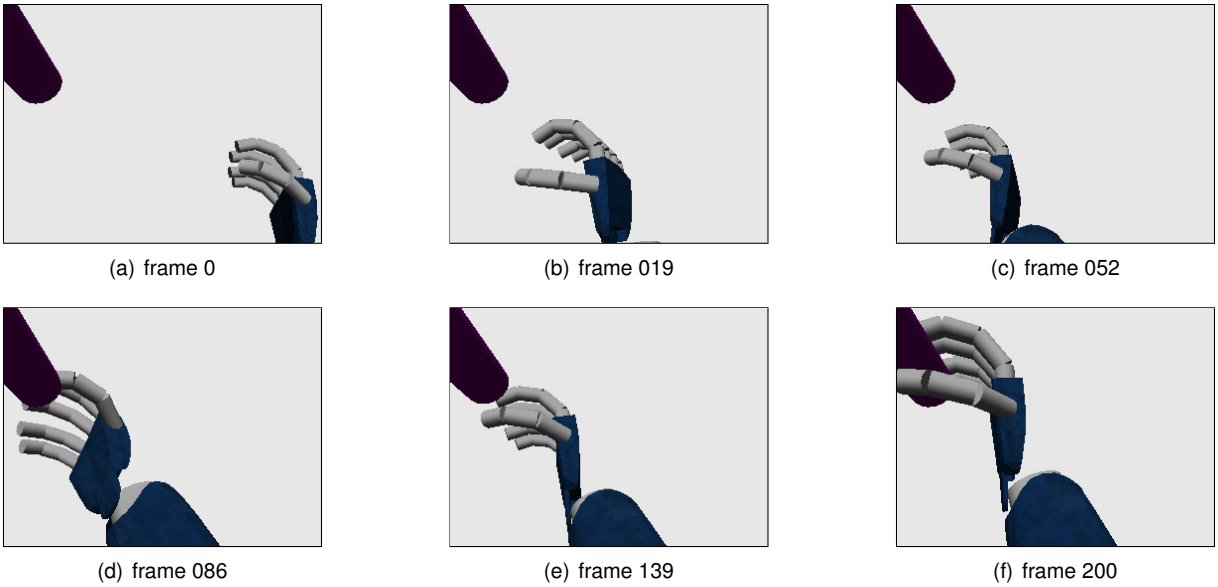
In all experiments we used a pre-defined shape for the fingers. This posture is often used in a grasping context. This shape can be different, but for simplicity and comparison of the various experiments we keep it constant. Also the filter parameters are kept constant. We initiated the filter with a normal distribution $p(x_0)$ with zero mean and with standard deviation σ_0 for all the joints. The filter is composed by 200 particles, which is not an ideal value, although it's a good balance between our real-time constraint and the time needed to generate an image. The other design parameters are $\alpha = 500$, $\sigma_{KDE} = 1.0$ and $\sigma_0 = 5.0$. To spread the particles after resampling we use σ_s decaying over time and with a lower bound. σ_s start in 3 degrees and decays 20% in every new iteration/frame with a lower limit of 0.10° , with this we achieve great precision with a reduced number of particles.

5.1 Simulation Results

5.1.1 First experiment - Offsets in Arm

In this experiment we use the simulated robot with artificial offsets on every joint of the right arm chain. These offsets are the ground truth to evaluate the filter performance and have the following values: $\beta_i = [5, 4, 3, -2, +3, -7, +3]^\circ$. In Figure 5.1 we can see an illustration of the filter initial state. We plot the projection, in red, of the predicted position of the finger tips overlapped in the acquired image. We can see that the difference in the observed *vs* predicted images is significant. Despite the artificial offsets may seem large at first sight, their magnitude is of the same order in the real robot, as will be discussed in Fig. 5.7.

While the robot hand is in the field of view of the cameras, we iterate the particle filter to improve the



(a) frame 0 (b) frame 019 (c) frame 052

(d) frame 086 (e) frame 139 (f) frame 200

Figure 5.2: Example of a reaching task used in the first experiment.

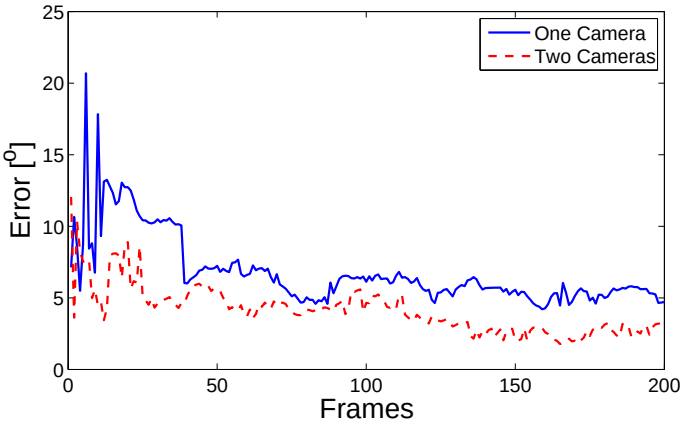
	With Filter		**Without filter**	
	Mean	Standard Deviation	Mean	Standard Deviation
Angular Error [$^\circ$]	4.5495	2.032	14.01	1.65
Position Error [mm]	3.3357	1.5163	42.67	5.47

Table 5.1: Mean and Variance of the final orientation and position errors over 10 different experiments, for two cameras. With and without filter.

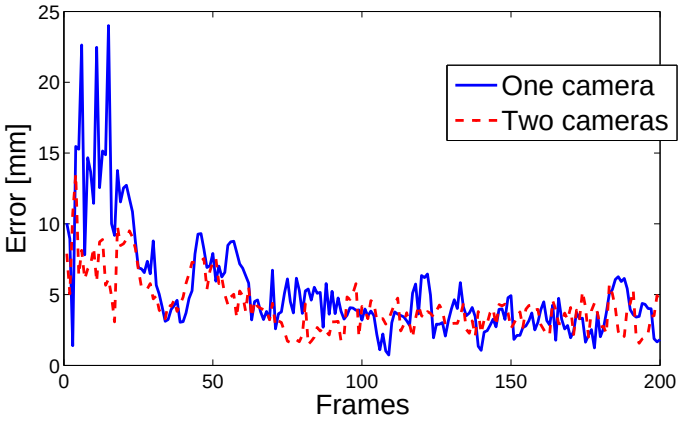
estimate of the offsets. The experiments use a reaching like trajectory of the arm as illustrated in Fig. 5.2. The hand goes from the right bottom to the left top of the image. We perform 10 such experiments, with different initial and final poses. To quantitatively evaluate the accuracy of the filter we use the Cartesian position and orientation errors (See Eq. 4.1 and 4.2). We do not compare the state estimates directly with the ground truth joint offsets because there are redundancies in the kinematics that may lead to the estimation of offsets different from ground truth but that correspond to very similar Cartesian poses. We show in Fig. 5.3 the evolution of the position and orientation errors of the hand base frame. The following observations can be made: (i) both errors reduce their magnitude about $3\times$ during the trial; (ii) convergence is achieved in less than 100 frames; and (iii) the orientation error is lower in the stereo case but the position error does not change significantly with the addition of the stereo information. The mean and standard deviation of the errors for the 10 experiments are show in Table 5.1 for the stereo case.

5.1.2 Second Experiment - Offsets in Arm and Head

For the second experiment we maintain the parameters of the filter and the errors in the right arm, but we added some errors in the head chain. The artificial offsets introduced in the head were: -4° in the neck pitch, 6° in the roll, -2° in the yaw and -1° in the eyes tilt. The executed trajectory is illustrated in Fig. 5.4, where we can also observe the predicted position of the fingertips (dots in red) during the convergence of the filter. Notice that the position of the fingertips converge to the real values during the

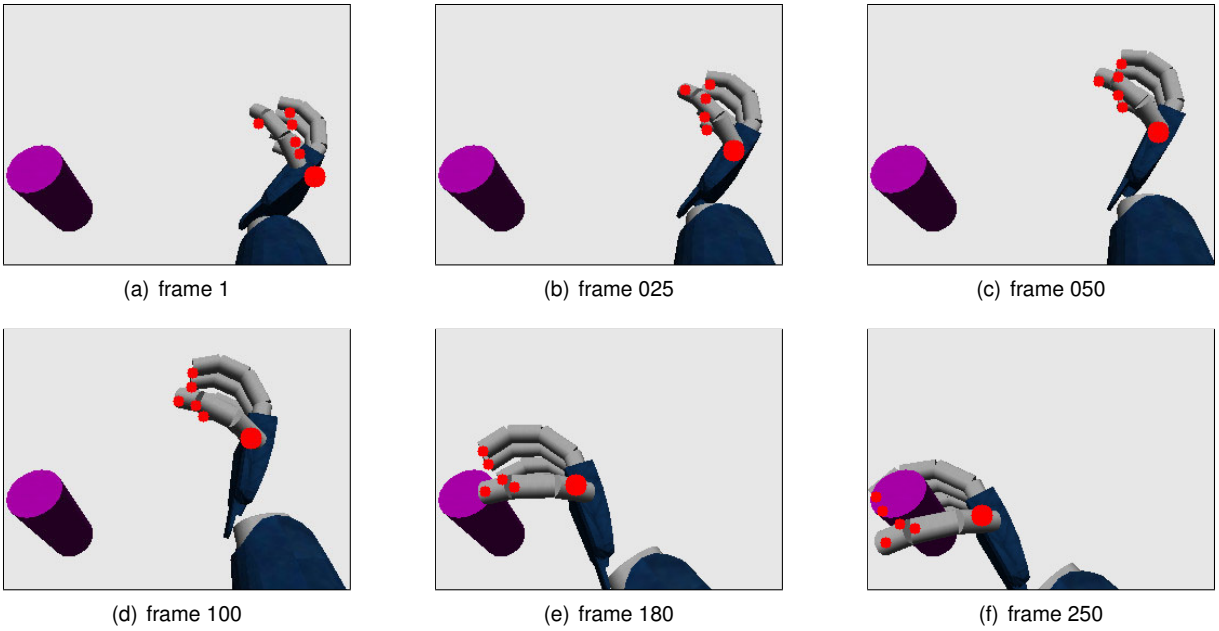


(a) Orientation



(b) Position

Figure 5.3: Orientation and position error over frames/time performing a reaching movement (Fig. 5.2) using one and two cameras. We can see the improvement in the orientation error with the use of two cameras



(a) frame 1 (b) frame 025 (c) frame 050

(d) frame 100 (e) frame 180 (f) frame 250

Figure 5.4: Frames of 2nd Experiment - We have errors in right Arm and Head chains. The estimated positions of the fingers tips, in red, improved during the movement

reaching time span. In Fig. 5.5 we plot the numerical values of the orientation and position errors along time, measured on the right eye reference frame. Note that the achieved accuracy is almost the same as when errors were only introduced in the arm chain (first experiment).

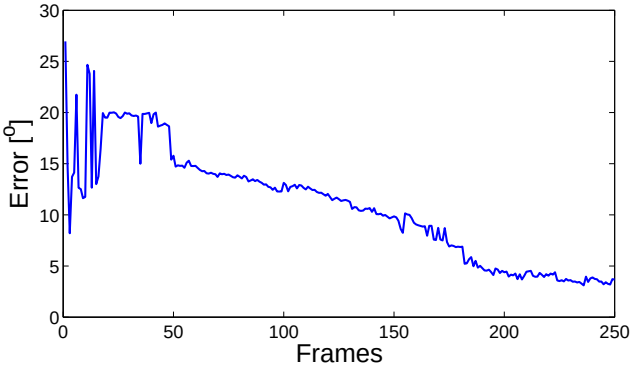
5.1.3 Generalization

Our approach uses offsets in the joints to estimate the error in the final pose. Another hypothesis is to estimate the error transformation matrix defined as:

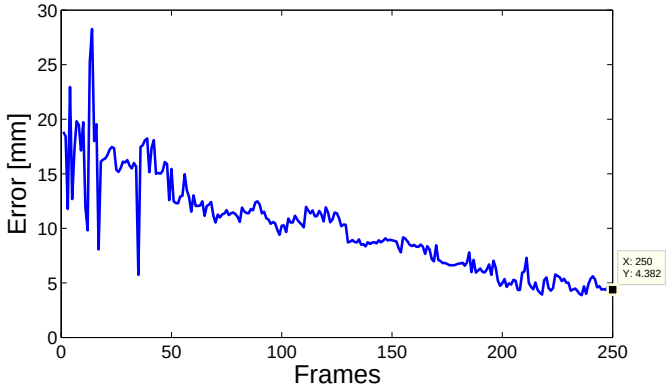
$$T_{err} = T_r^* * T_{enc}^{-1} \tag{5.1}$$

where, T_r^* is the estimation of the real transformation and T_{enc} the transformation defined by the encoders joints. The T_{err} will be the transformation to apply in order to correct the final pose of the hand.

In [38] the utilization of the transformation matrix method was chosen to estimate the error in the head chain. In order to compare the two methods, we ran our approach with a movement to a final position and tested it in five different positions (See Figure 5.6). In the final position, the estimated matrix T_r^* using the offsets or the matrix T_{err} is the same, therefore the errors are equal. In the other positions, we insert the offsets in the joints and apply the transformation T_{err}. In the table 5.2 we can see the final errors in these new positions between T_r^* and the ground truth transformation. Our method is better in all the cases of the generalization, apart for a few ones in which the orientation error is bigger (position 2 and 5). The position error is always better using the estimated offsets.

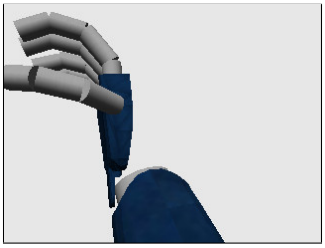


(a) Angular error

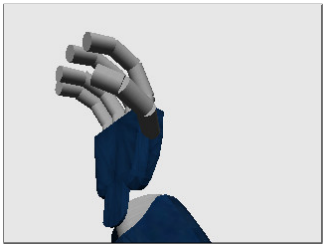


(b) Euclidean error

Figure 5.5: Angular and Euclidean error over frames/time during reaching movement (Fig. 5.4). Despite the errors in the head we can estimate the pose of the hand and have almost the same final errors.

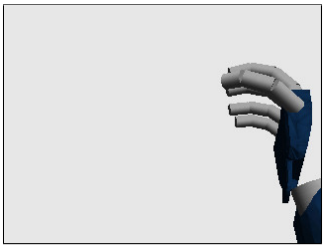


(a) Final training Pose

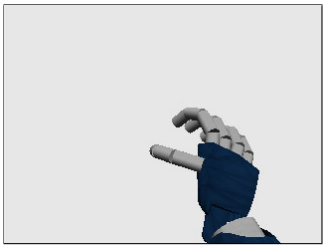


(b) Testing pose 1

(c) Testing pose 2



(d) Testing pose 3



(e) Testing pose 4

(f) Testing pose 5

Figure 5.6: Training in a trajectory with a final pose and testing in different poses. The position and orientation of the hand are very distinguished.

	Erro Angular [$^\circ$]		Erro de Posição [mm]	
	Offsets	*Transformação*	*Offsets*	*Transformação*
Posição 1	2.5602	3.8892	2.2221	4.9162
Posição 2	4.7247	3.9889	9.6751	13.3802
Posição 3	2.3720	4.3778	4.1777	10.4285
Posição 4	4.0699	8.6078	6.4442	7.6157
Posição 5	3.8914	0.8117	8.5436	16.1736

Table 5.2: Differences between Transformation and Offsets methods. The position error is better in all the cases with the offsets method.

5.2 Real Robot

5.2.1 First Experiment - w/Real Robot

In this section we will show an estimation of the offsets of the right arm with real data. The robot performed a set of arbitrary movements before approaching the target pose. The temporal evolution of the offsets values are shown in Fig. 5.7. We can observe that after about 100 frames, the estimate reaches a steady state. As mentioned previously, the accuracy is better evaluated using the Cartesian error, but since in this case ground truth is not available we measure the error in the image plane. This can be observed in Fig. 5.8, where the hand of the real robot and the predicted ones, with and without filter, are shown side by side. It's clear that the filtered pose is closer to the real one. Quantitatively, we have measured an average pixel difference between the fingertips of around 13 pixel with filtering and 25 pixel without, which at the average distance of the hand ($\sim$0.5m) corresponds more or less to 2cm and 4cm, respectively.

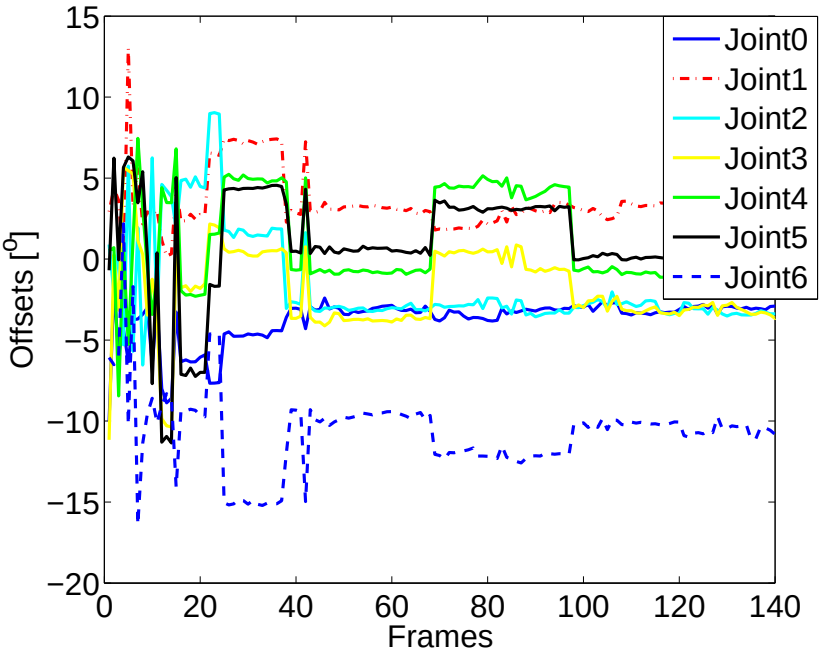
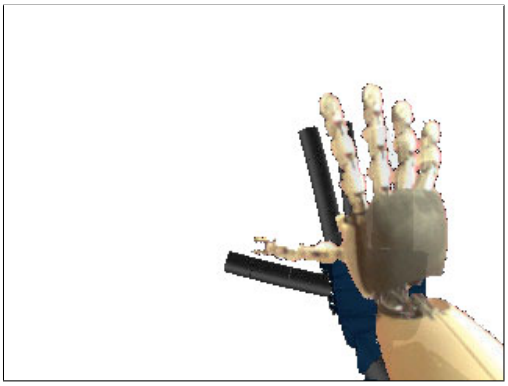
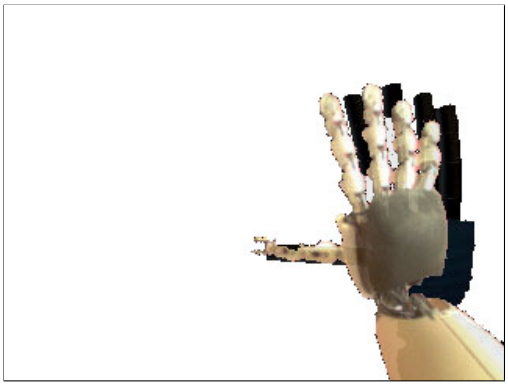


Figure 5.7: Offsets estimation of the Real Robot Right Arm. The offsets converge to a value



(a) Without filter



(b) With filter

Figure 5.8: Estimation of the pose of the hand with the real robot. a) the simulated hand without correction; b) the output estimation of our approach. Moreover, we see the similarities between the real and the simulated hand

CHAPTER 6

CONCLUSIONS

6.1 Achievements

We presented a method to estimate the pose of a robotic hand, based on a particle filter framework. We have shown its convergence and the errors obtained during the motion. We used the encoder readings and the cameras of the iCub to compute a visual-proprioceptive calibration. In our simulated experiments, the position error of the hand was always below 5mm, which is a good error for grasping purposes. The orientation error was below 5° which is a reasonable value due to the errors present in each joint. In the real robot experiment, the offsets converged and despite the geometrical differences between the real robot and simulation, the error decreases in image pixels by a factor of 2.

6.2 Future Work

In the future we plan to use the devised methodology to perform Visual Servoing (Vision-Based Robot Control) in grasping tasks.

Grasping is one of most important skills that humans perform every day. Newborn infants extend their arms to objects of interest, but only at 3-4 months old do they reach and grasp static objects intentionally [39]. As said in Chapter 1, before that age reaching movements in infants seem to be just “ballistic”, thus exploiting no visual feedback, as trajectory correction is absent [7, 8]. Then, from five months, vision is used to correct the hand position and orientation during the movement [9], with performance that improves during development [10]. A holistic approach inspired in developmental neuroscience can be designed at an architectural level using our method to estimate the hand position and orientation.

In the other hand, nowadays, real time reaching and grasping tasks in robots are performed, normally, without any visual feedback control approach [40] [41], mainly to speed-up the reaching process. Nevertheless, if the robot’s internal model is not accurate enough, grasping will not be successful. In

[42] grasping is performed in kitchenware objects using a very precise robotic arm, however some of the grasping experiments failed due to “contact between the hand and the object before grasping”. The premature contact can be mitigated with a visual control approach. According to [43] very few methods of grasping take advantage of vision to correct hand poses and in [44] catching objects in flight, using only the inverse kinematics of the robotic arm, satisfactory results were achieved. The iCub robot was used for simulated experiments, although in the real scenario another robot, with a more precise position control, was required.

Visual servoing is a feedback close-loop control strategy based on visual data [45]. Most of the visual servoing methodologies are based in eye-in-hand control, where the cameras are attached on the robot hand [46, 47], but in other systems, e.g. human inspired ones, the cameras are outside the hand and this procedure is not suitable. In the eye-to-hand approach, the camera is fixed in the world [48] or in a humanoid robot in the head reference frame. Some works were developed using eye-to-hand technique in humanoid robots nevertheless they utilize markers in the hand in order to estimate its pose. In [49] the utilization of a single camera and a land-mark (a light bulb emitting red) in the hand, cooperatively with a reflection of a flat mirror, improved the 3D estimation of the hand position and orientation in the world. *Yoshimi et al.* [50] have stereo calibrated cameras but also have a black point to identify the robot hand. In [51] a red ball is attached on the robot wrist and along with stereo calibrated vision it can perform precise grasping tasks. The humanoid robot REEM was used in [52] jointly with a body schema simulator to perform grasping tasks with visual feedback. They also use markers on the hand and on objects, and it is possible to see the differences between the real robot and the estimation from the internal model, showing the errors present in the robot perception of itself.

All in all, the present thesis can be used to fill a gap in reaching and grasping movements in humanoid robots using vision to improve the pose estimation of a robotic hand with high complexity (several degrees of freedom) and performing a visual servoing methodology in order to accomplish a more precise grasping pose.

BIBLIOGRAPHY

[1] S. Särkkä, *Bayesian Filtering and Smoothing*. Cambridge University Press, 2013.

[2] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[3] C. von Hofsten, "An action perspective on motor development," *Trends in Cognitive Sciences*, vol. 8, pp. 266–272, 2004.

[4] R. Joseph, "Fetal brain behavior and cognitive development," *Developmental Review*, vol. 20, pp. 81–98, 2000.

[5] G. Berlucchi and S. Aglioti, "The body in the brain: neural bases of corporeal awareness," *Trends in Neurosciences*, vol. 20, pp. 560–564, 1997.

[6] P. Rochat, "Self-perception and action in infancy," *Exp. Brain Res.*, vol. 123, pp. 102–109, 1998.

[7] E. Bushnell, "The decline of visually guided reaching during infancy," *Infant Behavior and Development*, vol. 8, pp. 139–155, 1985.

[8] C. V. Hofsten, "Structuring of early reaching movements: a longitudinal study," *Journal of Motor Behavior*, vol. 23, pp. 280–292, 1991.

[9] A. Mathew and M. Cook, "The control of reaching movements by young infants," *Child Development*, vol. 61, pp. 1238–1257, 1990.

[10] D. Ashmead, M. McCarty, L. Lucas, and M. Belvedere, "Visual guidance in infants' reaching toward suddenly displaced targets," *Child Development*, vol. 64, pp. 1111–1127, 1993.

[11] J. J. Lockman, D. H. Ashmead, and E. W. Bushnell, "The development of anticipatory hand orientation during infancy," *Journal of Experimental Child Psychology*, vol. 37, pp. 176–186, 1984.

[12] D. M. Wolpert, R. C. Miall, and M. Kawato, "Internal models in the cerebellum," *Trends in Cognitive Sciences*, vol. 2, pp. 338–347, 1998.

[13] R. C. Miall and D. M. Wolpert, “Forward models for physiological motor control,” *Neural Networks*, vol. 9, pp. 1265–1279, 1996.

[14] K. P. Körding and D. M. Wolpert, “Bayesian integration in sensorimotor learning,” *Nature*, vol. 427, pp. 244–247, 2004.

[15] M. Hoffmann, H. Marques, A. Hernandez Arieta, H. Sumioka, M. Lungarella, and R. Pfeifer, “Body schema in robotics: A review,” *IEEE Transactions on Autonomous Mental Development*, vol. 2, no. 4, pp. 304–324, 2010.

[16] D. Nguyen-Tuong and J. Peters, “Model learning for robot control: a survey,” *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[17] G. Metta, L. Natale, F. Nori, G. Sandini, D. Vernon, L. Fadiga, C. von Hofsten, K. Rosander, M. Lopes, J. Santos-Victor, A. Bernardino, and L. Montesano, “The icub humanoid robot: an open-systems platform for research in cognitive development,” *Neural Networks*, vol. 23, 2010.

[18] U. Pattacini, “Modular cartesian controllers for humanoid robots: Design and implementation on the icub,” Ph.D. dissertation, Italian Institute of Technology, 2011.

[19] A. Erol, G. Bebis, M. Nicolescu, R. D. Boyle, and X. Twombly, “Vision-based hand pose estimation: A review,” *Computer Vision and Image Understanding*, vol. 108, pp. 52–73, 2007.

[20] I. Oikonomidis, N. Kyriazis, and A. Argyros, “Markerless and efficient 26-dof hand pose recovery,” in *10th Asian Conference on Computer Vision*, Queenstown, New Zealand, November 2010.

[21] R. Y. Wang and J. Popović, “Real-time hand-tracking with a color glove,” *ACM Transactions on Graphics*, vol. 28, no. 3, 2009.

[22] D. Periquito, J. Nascimento, A. Bernardino, and J. Sequeira, “Vision-based hand pose estimation: A mixed bottom-up and top-down approach,” in *8th International Conference on Computer Vision Theory and Applications (VISAPP)*, Barcelona, Spain, February 2013.

[23] C. Ciliberto, F. Smeraldi, L. Natale, and G. Metta, “Online multiple instance learning applied to hand detection in a humanoid robot,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011, pp. 1526–1532.

[24] J. Leitner, S. Harding, M. Frank, A. Forster, and J. Schmidhuber, “Humanoid learns to detect its own hands,” in *IEEE Congress on Evolutionary Computation (CEC)*, 2013, pp. 1411–1418.

[25] O. Birbach, B. Bäuml, and U. Frese, “Automatic and self-contained calibration of a multi-sensorial humanoid’s upper body,” in *Intl. Conf. on Robotics and Automation*, Saint Paul, Minnesota, USA, May 2012.

[26] S. Ulbrich, V. de Angulo, T. Asfour, C. Torras, and R. Dillmann, “Rapid learning of humanoid body schemas with kinematic bézier maps,” in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2009, pp. 431–438.

[27] L. Jamone, L. Natale, F. Nori, G. Metta, and G. Sandini, “Autonomous online learning of reaching behavior in a humanoid robot,” *International Journal of Humanoid Robotics*, vol. 09, no. 03, p. 1250017, 2012.

[28] M. Klingensmith, T. Galluzzo, C. Dellin, M. Kazemi, J. A. Bagnell, and N. Pollard, “Closed-loop servoing using real-time markerless arm tracking,” in *IEEE-RAS International Conference on Robotics and Automation (ICRA) - Humanoids Workshop*, 2013.

[29] A. Mishra, Y. Aloimonos, and C. L. Fah, “Active segmentation with fixation,” in *Computer Vision, 2009 IEEE 12th International Conference on*, Sept 2009, pp. 468–475.

[30] A. Hammoude, “Computer-assisted endocardial border identification from a sequence of two-dimensional echocardiographic images,” Ph.D. dissertation, 1988.

[31] Kalman, Rudolph, and Emil, “A New Approach to Linear Filtering and Prediction Problems,” *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[32] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, “Lipschitzian optimization without the lipschitz constant,” *J. Optim. Theory Appl.*, 1993.

[33] J. D. Hol, T. B. Schon, and F. Gustafsson, “On Resampling Algorithms for Particle Filters,” 2006.

[34] R. Douc, “Comparison of resampling schemes for particle filtering,” in *In 4th International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2005.

[35] D. Crisan and A. Doucet, “A survey of convergence results on particle filtering methods for practitioners,” 2002.

[36] V. Tikhanoff, P. Fitzpatrick, F. Nori, L. Natale, G. Metta, and A. Cangelosi, “The icub humanoid robot simulator,” in *IROS Workshop on Robot Simulators*, 2008.

[37] D. Q. Huynh, “Metrics for 3d rotations: Comparison and analysis,” *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10851-009-0161-2

[38] X. Gratal, J. Romero, J. Bohg, and D. Kragic, “Visual servoing on unknown objects,” *Mechatronics*, vol. 22, no. 4, pp. 423 – 435, 2012.

[39] M. M. Daum and G. Gredebäck, “The development of grasping comprehension in infancy : Covert shifts of attention caused by referential actions,” *Experimental Brain Research*, 2011.

[40] M. Ciocarlie, K. Hsiao, E. G. Jones, S. Chitta, R. B. Rusu, and I. A. Sucan, “Towards Reliable Grasping and Manipulation in Household Environments,” in *Intl. Symposium on Experimental Robotics (ISER)*, New Delhi, India, 2010.

[41] A. Saxena, J. Driemeyer, and A. Y. Ng, “Robotic grasping of novel objects using vision,” *Int. J. Rob. Res.*, vol. 27, no. 2, pp. 157–173, Feb. 2008.

[42] R. Figueiredo, A. Shukla, D. Aragao, P. Moreno, A. Bernardino, J. Santos-Victor, and A. Billard, "Reaching and grasping kitchenware objects," in *Proceedings of International Symposium on System Integration (SII)*, 2012.

[43] M. A. A. T. Bohg, J. and D. Kragic, "Data-driven grasp synthesis - a survey," *IEEE Transactions on Robotics*, 2014.

[44] A. S. Seungsu Kim and A. Billard, "Catching objects in flight," *IEEE Transactions on Robotics*, 2014.

[45] Chaumette and S. Hutchinson, "Visual servo control, part i: Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, pp. 82–90, 2006.

[46] T. La Anh and J.-B. Song, "Robotic grasping based on efficient tracking and visual servoing using local feature descriptors," *International Journal of Precision Engineering and Manufacturing*, vol. 13, no. 3, pp. 387–393, 2012.

[47] G. Ma, Q. Huang, Z. Yu, X. Chen, L. Meng, M. Sultan, W. Zhang, and Y. Liu, "Hand-eye servo and flexible control of an anthropomorphic arm," in *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, Dec 2013, pp. 1432–1437.

[48] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 651–670, 1996.

[49] C. Kulpate, M. Mehrandezh, and R. Paranjape, "An eye-to-hand visual servoing structure for 3d positioning of a robotic arm using one camera and a flat mirror," in *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, Aug 2005, pp. 1464–1470.

[50] B. H. Yoshimi and P. Allen, "Closed-loop visual grasping and manipulation," in *IEEE International Conference Robotics and Automation*, 1996, pp. 1353–1360.

[51] N. Vahrenkamp, S. Wieland, P. Azad, D. Gonzalez, T. Asfour, and R. Dillmann, "Visual servoing for humanoid grasping and manipulation tasks," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, Dec 2008, pp. 406–412.

[52] D. Agravante, J. Pages, and F. Chaumette, "Visual servoing for the reem humanoid robot's upper body," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 5253–5258.