Journal of Intelligent and Robotic Systems manuscript No.
(will be inserted by the editor)

Robotic hand pose estimation based on stereo vision and GPU-enabled internal graphical simulation

Pedro Vicente · Lorenzo Jamone · Alexandre Bernardino

Received: ... / Accepted: ...

Abstract Humanoid robots have complex kinematic chains whose modeling is error prone. If the robot model is not well calibrated, its hand pose cannot be determined precisely from the encoder readings, and this affects reaching and grasping accuracy. In our work, we propose a novel method to simultaneously i) estimate the pose of the robot hand, and ii) calibrate the robot kinematic model. This is achieved by combining stereo vision, proprioception, and a 3D computer graphics model of the robot. Notably, the use of GPU programming allows to perform the estimation and calibration in real time during the execution of arm reaching movements. Proprioceptive information is exploited to generate hypotheses about the visual appearance of the hand in the camera images, using the 3D computer graphics model of the robot that includes both kinematic and texture information. These hypotheses are compared with the actual visual input using particle filtering, to obtain both i) the best estimate of the hand pose and ii) a set of joint offsets to calibrate the kinematics of the robot model. We evaluate two different approaches to estimate the 6D pose of the hand from vision (silhouette segmentation and edges extraction) and show experimentally that the pose estimation error is considerably reduced with respect to the nominal robot model. Moreover, the GPU implementation ensures a performance about 3 times faster than the CPU one, allowing real-time operation.

Keywords robot hand pose estimation · robot self-calibration · humanoid robots · 3D graphical simulation · GPU programming · online reaching adaptation · computer vision

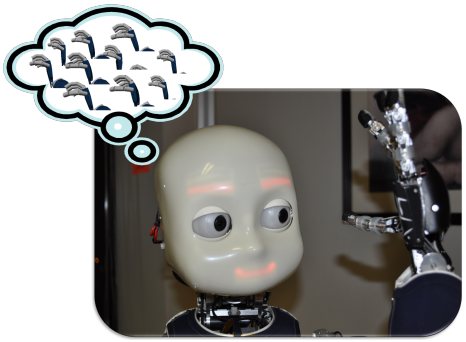


Fig. 1: The iCub humanoid robot imagining hypotheses of its hand pose.

1 Introduction

Everyday tasks like grasping an object with precision or pushing an electrical plug into a socket are quite simple for humans, who deal with these activities on a daily basis, but complex and challenging for autonomous robots. A correct kinematic model of the limbs structure is of paramount importance when performing such tasks. In particular, accurate models of the arms and hands are crucial in order to properly grasp and manipulate objects. Moreover, real time reaching and grasping tasks in robots are very often performed without any visual feedback control approach [8,38]. In that case, if the robot

Parts of this manuscript were previously presented at the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC 2015), Vila Real

P. Vicente, L. Jamone and A. Bernardino
Avenida Rovisco Pais, 1 - Torre Norte
1049-001 Lisboa, Portugal
Tel.: +351218418050
E-mail: {pvicente,ljamone,alex}@isr.ist.utl.pt

internal model is not accurate enough, the tasks will not be successful. However, a precise analytical model of the kinematic chain of a humanoid robot with several degrees of freedom can be difficult to obtain due to hard-to-model aspects (e.g. friction) and changes that might occur over time (e.g. misalignment of a joint rotation axis). Therefore, to correctly estimate the hand pose, a continuous re-calibration of the model is desirable. In general, this can be done by exploiting multisensory information coming from the robot sensors, together with advanced techniques for learning and estimation.
Indeed, if we consider human studies, it is known that five-months-old infants already use vision to correct the hand position and orientation during movements [27], with performance that improves during development [1]. However, after nine months this visual guidance almost disappears, and children become able to plan a proper hand trajectory at the movement onset [26]. This might suggest that an internal model has been learned by combining visual and proprioceptive information during the first months, and later exploited to improve the control. Moreover, motor learning theories postulate that learned internal models might be used to make predictions, that are combined with the actual sensory feedback through Bayesian integration to improve the estimation of the current state of the system [23]. Overall, these studies put forward the idea that the processes of model calibration and state estimation might support each other in natural systems, and therefore provide us with further motivation to explore similar strategies in artificial systems.
In our work we propose a novel computational framework to simultaneously i) estimate the 6D pose of the robot hand and ii) calibrate the kinematic model of the robot arm. While the robot model is exploited to facilitate the hand pose estimation, the estimates are used to calibrate the model itself. These two processes are realized online in real time during reaching movements. We apply our framework to the iCub humanoid robot [29].
In this paper we implement the internal model of the robot in a 3D graphical simulation environment using the game engine Unity®, modeling both the kinematics and appearance (texture) of the robot. The nominal kinematics is provided by the *iKin* library [36], part of the YARP/iCub software repository (`https://github.com/robotology`). The robot texture is modeled based on the open-source CAD documentation of the robot, that can be found online at `http://wiki.icub.org/wiki/RobotCub`. The internal model is used to generate hypotheses about the hand appearance in the eye cameras, based on the arm and head joint angles (i.e. proprioception). These hypotheses are compared to the actual hand appearance in the images coming from the cameras (i.e. stereo vision). Using kernel density estimation and a maximum likelihood approach, we compute the estimate of the current hand pose as well as a set of joint angles corrections to calibrate the model kinematics.
A preliminary version of the proposed framework has been presented in a previous work [44], in which we developed a CPU implementation of the method. We realized that there was a computational bottleneck in the hypotheses generation step and in the comparison to the visual input, which are operations that can be easily parallelized. To tackle this problem, in this paper, we present an efficient GPU implementation of our computational framework that makes it possible to use the system in real time. Moreover, we evaluate two different approaches for the visual estimation. The first approach is simpler, and it consists of comparing only the silhouettes of the real and simulated hand, that are obtained by color-based segmentation. In the second approach, instead, the edges extracted from the real and simulated hand are compared. The use of edges provides better robustness to clutter and illumination conditions than the first approach based on color segmentation. However, it is computationally more expensive; this further justifies the need for a GPU implementation. This paper is an extended version of the work originally presented in [45]. The major extensions consist of: a more detailed description of the GPU implementation; new experimental data; a comparison between silhouette segmentation and edges extraction for the visual pose estimation.
The rest of the paper is organized as follows. In Section 2 we report the related work in robotics and we highlight our contribution more specifically. Then in Section 3 we formulate the problem, while in Section 4 we provide the details of our proposed solution. In Section 5 we account for the implementation details as well as for the robotic platform used. Finally, in Section 6 we present the experimental results, and in Section 7 we draw our conclusions and sketch the future work.

2 Related Work

Estimating the pose of a multifingered hand from vision is a challenging problem that has been studied in Human-Computer Interaction (for human hands, see [11] for a review up to the year 2007) as well as in robotics (for robot hands). One recurring idea in human hand pose estimation is to first detect the hand and then to exploit a kinematics and appearance hand model to estimate its pose. For example, in [33,34] the hand is divided into different segments, and detected using skin color segmentation and edge maps. Then it is compared

to a 26-DOF model of the human hand using Particle Swarm Optimization, with a GPU implementation to speed-up the algorithm and allow it to run it in quasi-real time. Another example is the work in [46], where the hand detection is simplified by using a colored glove. Then a search for a corresponding pose in a database of examples is performed. A similar approach is described in [37], where the hand silhouette is matched to a previously trained database of silhouettes to track the pose of a human hand.

Since in our case we deal with the pose estimation of a robotic hand, we can use the robot kinematics model to obtain initial hypotheses of the hand pose, reducing the problem to a local search.

Concerning visual robot hand detection and representation, we evaluate two strategies in this paper: one based on silhouette segmentation and the other based on edges extraction. The use of edges to detect the hand pose from vision is inspired by the approach in [6], and it has also been used in [15] to track the Schunk Dexterous Hand in a Virtual Visual Servoing approach. A few interesting works in robotics have dealt with the problem of hand detection by using machine learning techniques. The Cartesian Genetic Programming method was used by [25] to learn from visual examples how to detect the robot hand inside an image. Online Multiple Instance Learning was used in [7] for the same task, exploiting proprioceptive information from the arm joints and visual optic flow to automatically label the training images.

To achieve real-time computation we exploit GPU programming, that has been widely used in the last years to speed-up well-known computer vision algorithms, such as Canny-Edge detection [35] or image segmentation procedures [13]. Some authors state that it is possible to achieve a speed-up of almost 10 times [40] in high-level computer vision algorithms, when more conservative ones, and in a wider perspective, claim for 2.5 times on average [24].

According to our knowledge, the work in the literature more similar to ours, with respect to visual robot hand pose estimation, is [15], where the authors use a 3D-model based approach, an edge based error function and graphics acceleration techniques. However, their optimization method is based on Virtual Visual Servoing [9] that, being a gradient based method, is prone to converge to local minima. On the contrary, we propose a particle filtering method that is robust to non-convex/non-gaussian error functions. Another key feature of our approach is that, while we exploit a model of the robot to facilitate the hand detection and pose estimation, at the same time we continuously calibrate the model using sensory data.

The calibration of a robot kinematic model (sometimes also referred as *body schema adaptation*, borrowing from human sciences) has been *per-se* a topic of considerable attention (see for example [16] for a review up to the year 2010). A way to simplify the problem is to use a marker to visually detect the end-effector (i.e. the robot hand, in the humanoid case). The method used by [2] requires five minutes of data acquisition during specific robot movements with a special marker in the robot wrist. It optimizes offline some parameters of the kinematic chain (angle offsets and elasticity) of an upper humanoid torso using non-linear least squares. Online solutions have been studied for example in [42,21,18], in which visual markers are used to easily detect the hand position. The inclusion of additional parts into the kinematic chain (i.e. tools) has been considered as well [19,20]. A few marker-free solutions have been explored as well. In [43] the kinematic chain is decomposed into smaller segments, and both offline and online learning solutions are proposed; although it seems that no markers are used, it is not described how the hand pose is measured. The research in [22] is also based on a marker-free correction of the robot kinematics using RGB-D data. In [12] eye-hand calibration is realized by performing several ellipsoidal arm movements with a predefined hand pose, tracking the tip of the index finger in the cameras, and employing optimization techniques to learn the transformation between the fingertip position obtained by stereo vision and the one computed from the forward kinematics. Such transformation is then used to calibrate the kinematics. The hand orientation is not considered.

2.1 Our contribution

Automatic calibration and hand pose estimation are important and challenging skills for humanoid robots with many degrees of freedom. In our work, we developed a computational framework to achieve both objectives simultaneously. Our solution is online, marker-free, fast, and based on on-board stereo vision and proprioception (i.e. no external sensor is required). To the best of our knowledge, this is the first system that meets all these requirements on a humanoid robot.

In previous work [44] we described a preliminary version of the system, that we improved in [45] taking advantage of GPU programming to reduce the time needed for computation. The present manuscript extends the work in [45] in four main aspects. First, we describe in full details the GPU implementation, which combines OpenCV GPU functions [4], OpenGL library [39], and GPGPU (CUDA) programming [32], and show how this combination considerably reduces the computation time.

Second, we compare two different approaches for the visual matching of hand pose hypotheses, one based on silhouette segmentation and the other based on edges extraction. Third, we replicate the experiments performed in the previous papers by using the new system, with GPU-enabled computation and edge-based hand pose estimation, and we show that we can achieve a better performance. Finally, we show experiments with the new visual matching strategy in a non-uniform background, where the previous observation model (silhouette-based) cannot be exploited.

3 Problem statement

Consider the problem of reaching for an object in the peripersonal space of a humanoid robotic platform. The object's pose is often computed using vision-based methods, so it is originally described in the camera reference frame. The conversion of this pose to any other reference frame (e.g. the hand) will inevitably accumulate the calibration errors in the serial kinematic chain from the camera to the end frame. So, if we can obtain a good estimate of the robot hand pose in the same camera reference frame, we can achieve two goals:

- to compute the relative pose between the robot hand and the object without kinematics calibration errors, since both the object and the hand pose are computed in the same sensor reference frame.
- to evaluate the error between two different estimates of the hand pose, obtained using either vision or proprioception, that will be used to correct the kinematics model so as to match the visual and the proprioceptive perceptual systems.

In this paper, we focus on the latter objective. The former will be subject of future work.

3.1 Parametrization of Calibration Errors

Let us consider the problem of estimating the robot hand pose in the left camera reference frame. The pose can be represented by a generic 4×4 roto-translation matrix $\mathbf{T}$. Using the robot kinematics function from the left camera to the hand palm $\mathscr{K}(\cdot)$ and the vector of joint encoder readings $\boldsymbol{\theta}$ (see Fig. 2), an estimate of the pose can be obtained by:

$$\hat{\mathbf{T}}_k = \mathscr{K}(\boldsymbol{\theta}) \tag{1}$$

However, several sources of error may affect this estimate. We consider the existence of calibration errors (bias) and random errors (noise). These sources of errors can be

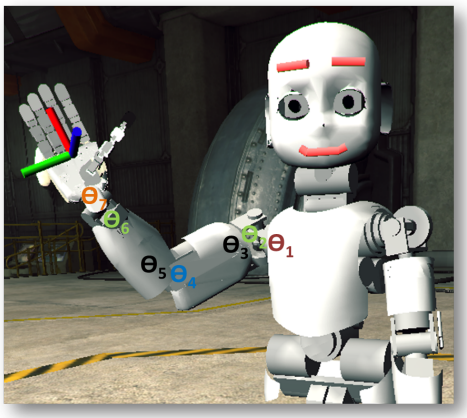


Fig. 2: The hand pose is a function of the joint angles.

encoded in many different ways. We propose to encode the errors in the robot's joint space, i.e.:

$$\boldsymbol{\theta}_r = \boldsymbol{\theta} + \boldsymbol{\beta} + \boldsymbol{\eta} \tag{2}$$

where $\boldsymbol{\theta}_r$ are the real angles; $\boldsymbol{\theta}$ are the measured angles; $\boldsymbol{\beta}$ are joint offsets representing calibration errors; and $\boldsymbol{\eta}$ represents random measurement noise. Given an estimate of the joint offsets $\hat{\boldsymbol{\beta}}$, a better hand pose estimate can be computed by:

$$\hat{\mathbf{T}}_j = \mathscr{K}(\boldsymbol{\theta} + \hat{\boldsymbol{\beta}}) \tag{3}$$

Another possibility could be to encode the errors in Cartesian space. In this case, the errors would be parametrized as roto-translation matrices that compose with the nominal kinematics function:

$$\mathbf{T} = \mathscr{K}(\boldsymbol{\theta}) \cdot \mathbf{T}_{\boldsymbol{\beta}} \cdot \mathbf{T}_{\boldsymbol{\eta}} \tag{4}$$

where $\mathbf{T}_{\boldsymbol{\beta}}$ encodes the calibration errors and $\mathbf{T}_{\boldsymbol{\eta}}$ represents the random noise. Given an estimate of the calibration error $\hat{\mathbf{T}}_{\boldsymbol{\beta}}$, the pose estimate would be given by:

$$\hat{\mathbf{T}}_c = \mathscr{K}(\boldsymbol{\theta}) \cdot \hat{\mathbf{T}}_{\boldsymbol{\beta}} \tag{5}$$

This is the approach taken in a number of works in the literature, as for example in [12,14,15]. We will show experimentally that the joint space error parametrization has advantages over the Cartesian one in terms of generalization to different parts of the workspace.

3.2 State Model

In our formulation, the adaptation of the robot internal model consists in estimating the offsets ($\boldsymbol{\beta}$) of the joint angles in the kinematic chain:

$$\boldsymbol{\beta} = \boldsymbol{\theta}_r - \boldsymbol{\theta} - \boldsymbol{\eta} \tag{6}$$

where $\boldsymbol{\eta}$ is modeled as a zero-mean Gaussian noise with covariance $\mathbf{Q}$, $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$.

The state vector $\boldsymbol{\beta}$ is composed of the offsets in the arm joints angles. In this paper, we only consider the offsets of the arm to reduce the complexity of the problem, as the head chain is assumed to be calibrated using the procedure defined in [30]. Also, miscalibration in the finger joints has a small impact on the observations since they are at the end of the kinematic chain.

The offsets in Eq. (6) define the state vector, $\boldsymbol{\beta} = [\beta_1 \ \beta_2 \ \beta_3 \ \beta_4 \ \beta_5 \ \beta_6 \ \beta_7]^T$, as an unobserved Markov process where β_i is the offset in joint i of the arm. We assume an initial distribution $p(\boldsymbol{\beta}_0)$ and a known state transition distribution $p(\boldsymbol{\beta}_{t+1}|\boldsymbol{\beta}_t)$. To cope with non-modeled errors (for instance, about the size of a link) we allow for small changes over time in $\boldsymbol{\beta}$ via a process noise vector $\mathbf{w}$ and model the system state transition as:

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t \ + \mathbf{w} \tag{7}$$

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

3.3 Observation Model

We consider two sources of information in our humanoid robot: i) the motor encoders (proprioceptive sensing) and ii) the cameras in the left and right eyes (visual sensing).

We define the observations as the images acquired from the cameras of the robot. A random variable $\mathbf{y}_t$ represents the concatenation of the left and right images acquired at time step t. Also, we can generate virtual images using the internal model of the robot, for an arbitrary value of the state vector $\boldsymbol{\beta}$ (see Fig. 3):

$$\bar{\mathbf{y}}_t = f(\boldsymbol{\theta} + \boldsymbol{\beta}) \tag{8}$$

where $\bar{\mathbf{y}}_t$ represents the concatenation of the virtual images in the left and right cameras of the robot simulator and $f(.)$ encodes the kinematics and texture information of the robot as well as the environment in the simulation scenario. The camera model (included in $f(.)$) introduces noise through the rasterization and color quantization processes. Moreover, the complex illumination model (based on multiple light sources with different intensity, color, and location in Cartesian space) produces artifacts in the simulated images like shadows, specularities, and glitter which is the typical noise present in the real images.

From the comparison between the real measurements $\mathbf{y}_t$ and the virtual measurements $\bar{\mathbf{y}}_t$, through a suitable function $g(\mathbf{y}_t, \bar{\mathbf{y}}_t)$, we can compute a likelihood of the state $\boldsymbol{\beta}$ at time t:

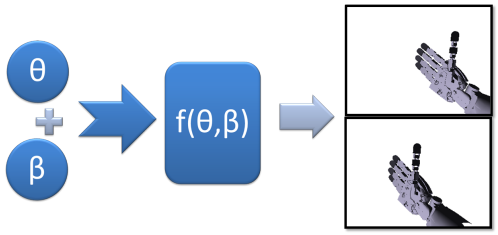


Fig. 3: Generation of virtual images in the Unity iCub simulator.

$$l(\boldsymbol{\beta}_t) = p(\mathbf{y}_t \mid \boldsymbol{\beta}_t, \boldsymbol{\theta}_t) = g(\mathbf{y}_t, \bar{\mathbf{y}}_t) \tag{9}$$

We have defined two different approaches for implementing the comparison function $g(\cdot, \cdot)$. One is based on silhouettes segmentation and the other is based on edges extraction.

It is worth noting that due to the redundancy in the robot kinematics (joints space is 7DOF while hand pose is 6DOF) different states ($\boldsymbol{\beta}$) may correspond to the same hand pose. Therefore, the likelihood function $l(\boldsymbol{\beta}_t)$ is multimodal and a particular choice of $\hat{\boldsymbol{\beta}}$ will be just one set of offsets that can explain the hand pose in the images.

3.3.1 Silhouette segmentation approach

In this approach, we make use of the segmented binary images from the real and virtual cameras (see Fig. 4). To compute the similarity between the real and virtual binary masks (silhouettes) we use the Jaccard coefficients (s_{Jc}) [10]. Let $\mathbf{R}(\mathbf{y})$ be the real silhouette and $\mathbf{R}(\bar{\mathbf{y}})$ the virtual silhouette. The Jaccard coefficients are defined as:

$$s_{Jc}(\mathbf{y}, \bar{\mathbf{y}}) = \frac{\#\left(\mathbf{R}(\mathbf{y}) \cap \mathbf{R}(\bar{\mathbf{y}})\right)}{\#\left(\mathbf{R}(\mathbf{y}) \cup \mathbf{R}(\bar{\mathbf{y}})\right)} \tag{10}$$

where # denotes the number of pixels in the region.

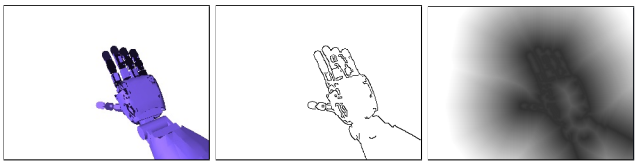
The numerator term in Eq. 10 is measuring how similar and overlaid the two silhouette regions are and the denominator is normalizing the metric to a range [0,1]. Therefore, we define the likelihood model as:

$$p(\mathbf{y}_t|\boldsymbol{\beta}_t, \boldsymbol{\theta}_t) \propto s_{\mathrm{Jc}}(\mathbf{y}_t, \bar{\mathbf{y}}_t) \tag{11}$$

This approach requires a good segmentation of the hand. For example, this can be a feasible observation model if: the background is uniformly colored and a good silhouette can be obtained by color segmentation

(a) Input image (b) Silhouette Image

Fig. 4: An example of the computation of the silhouette segmentation in a simulated image of the iCub hand.



(a) Input image (b) Edge Image (c) Distance Transform image

Fig. 5: An example of edges extraction and distance transform in a simulated image of the iCub hand.

methods; or, the head of the robot is static and a silhouette can be extracted by background segmentation methods. However, in more general cases, this approach may be not very effective due to imprecise or noisy segmentation, and more robust methods (like the one described in the following Section) would be preferable.

3.3.2 Edges extraction approach

For this approach, we compute the average distance between the edges of the real image to the closest edge in the virtual image and denote this quantity as $\bar{d}$. A perfect match between the real and virtual images would correspond to $\bar{d} = 0$ whereas bad matches would correspond to large values of $\bar{d}$. The likelihood function is thus defined as:

$$p(\mathbf{y}_t|\boldsymbol{\beta}_t, \boldsymbol{\theta}_t) \propto \exp^{-\lambda \bar{d}} \tag{12}$$

where λ is a tuning parameter to control sensitivity in the distance metric.

To compute $\bar{d}$ we make use of the distance transform [3]. The distance transform (DT) consists of the application of an edge detector to the image (e.g. [5]) and then, for each pixel, compute its distance to the closest edge point. In Fig. 5 we give an example of one input image (in this case, from the right eye camera) and the corresponding edge and distance transform images, respectively.

Let $\mathbf{D}(\mathbf{y})$ be the distance transform of the real images and $\mathbf{E}(\bar{\mathbf{y}})$ be the edge map of the virtual images (binary image indicating the edge pixels).

The average distance, $\bar{d}$, can be efficiently computed by:

$$\bar{d} = \frac{1}{k} \cdot \sum_{i=0}^{N} \mathbf{E}\left(\bar{\mathbf{y}}(i)\right) \cdot \mathbf{D}\left(\mathbf{y}(i)\right) \tag{13}$$

where k is the number of edge pixels in the virtual image, and N is the total number of pixels.

4 State Estimation Approach

In our approach, we use a particle filter as the one defined in [41]. Particle filter is a non-parametric implementation of the Bayes filter, approximating the posterior distribution of the state by a set of random samples called particles denoted by:

$$B_t := \boldsymbol{\beta}_t^{[1]}, \boldsymbol{\beta}_t^{[2]}, \boldsymbol{\beta}_t^{[3]}, ..., \boldsymbol{\beta}_t^{[M]} \tag{14}$$

where M is the number of particles, $\boldsymbol{\beta}_t^{[m]}$ (with $1 < m < M$) is one state sample and B_t is the particle set at time t. The posterior distribution is approximated by the weighted set of particles:

$$p(\boldsymbol{\beta}_t|\mathbf{y}_{1:t}, \boldsymbol{\theta}_{1:t}) \approx \sum_{m=1}^{M} \omega^{[m]} \delta(\boldsymbol{\beta}_t - \boldsymbol{\beta}_t^{[m]}) \cdot \left(\sum_{m=1}^{M} \omega^{[m]} \right)^{-1} \tag{15}$$

where $\omega^{[m]}$ is the weight of particle m and $\delta(.)$ is the Dirac delta function. In the beginning of each time step t all the particles have the same weight $\omega^{[m]} = \frac{1}{M}$. Under the Markov assumption (see Section 3.2) we can compute recursively $p(\boldsymbol{\beta}_t|\mathbf{y}_{1:t}, \boldsymbol{\theta}_{1:t})$ sampling from the previous state estimation $p(\boldsymbol{\beta}_{t-1}|\mathbf{y}_{1:t-1}, \boldsymbol{\theta}_{1:t-1})$. The four stages of the particle filter are:

1. prediction - we use the state transition equation defined in Section 3.2 (Eq.(7));
2. observation - the predicted observation $\bar{\mathbf{y}}$ is generated using the internal model and compared with to observation from the cameras, $\mathbf{y}$, by using either the silhouette or the edges approach (see Section 3.3);
3. update - the measurement likelihood defined for each approach (Eq.(11) or Eq.(12)) is used to re-weight the particles according to $\omega^{[m]} = p(\mathbf{y}_t|\boldsymbol{\beta}_t, \boldsymbol{\theta}_t)$;
4. resampling - the particles are sampled according to their weight. We use the systematic resampling method [17] to guarantee that a particle with a weight greater than $1/M$ is always resampled, where M is the number of particles.

4.1 Computing the state estimate

Although the state is represented at each time step as a distribution approximated by the weighted 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 neighbor particles, and choose the particle with the highest smoothed weight ($\omega'^{[i]}$) as our state estimate:

$$\omega'^{[i]} = \omega^{[i]} + \alpha \cdot \frac{1}{M} \sum_{m=0}^{M} \omega^{[m]} \cdot K(\boldsymbol{\beta}^{[i]}, \boldsymbol{\beta}^{[m]}) \tag{16}$$

where $\omega^{[i]}$ is the particle likelihood, α is a smoothing parameter, M is the number of particles of the filter, $\boldsymbol{\beta}^{[i]}$ is the particle we are smoothing. The sum term is the influence of the neighbors in the score of the particle i. K is a kernel specifying the influence of a particle in another one based on their distance. We use a Gaussian Kernel in our experiments:

$$K(\boldsymbol{\beta}^{[i]}, \boldsymbol{\beta}^{[j]}) = \frac{1}{\sqrt{2\pi|\Sigma|}} \, e^{[-\frac{1}{2}(\boldsymbol{\beta}^{[i]} - \boldsymbol{\beta}^{[j]})^T \Sigma^{-1} (\boldsymbol{\beta}^{[i]} - \boldsymbol{\beta}^{[j]})]} \tag{17}$$

where Σ is the co-variance matrix and $|\Sigma|$ its determinant. The term $(\boldsymbol{\beta}^{[i]} - \boldsymbol{\beta}^{[j]})$ denotes the difference in the joint space between the two particles (joint offsets).

We assume the joints offsets ($\boldsymbol{\beta}$) are independent of each other, so Σ will be a diagonal matrix $\Sigma = \sigma^2_{\text{KDE}} I_7$, where σ_{KDE} is the standard deviation in each joint, which we assume to be the same for all joints. This parameter defines if two particles are close or not. If we have a high σ_{KDE} it means that all particles are "close" to each other. On the other hand, if we have a small σ_{KDE} it means that all the particles are "far" from each other resulting in $\omega'^{[i]} \approx \omega^{[i]}$.

4.2 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:

$$d(R_r, R_e) = \sqrt{\frac{\|logm(R_r^T R_e)\|_F^2}{2}} \frac{180}{\pi} \quad [^\circ] \tag{18}$$

where R_r is the real rotation matrix from the eye reference frame to the hand reference frame and R_e is the estimated one. The principal matrix logarithm, *logm* with the Frobenius norm ($\|.\|_F$), implements the usual distance on the group of rotations.

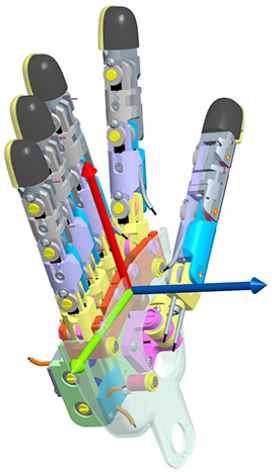


Fig. 6: The reference frame of the iCub right hand in the CAD model of the robot. Axis color notation: x-red; y-green; z-blue. Based on `http://wiki.icub.org/wiki/ICubFowardKinematics_right` (Best viewed in color)

The position error is computed by the Euclidean distance, $d(P_r, P_e)$:

$$d(P_r, P_e) = \sqrt{(x_r - x_e)^2 + (y_r - y_e)^2 + (z_r - z_e)^2} \tag{19}$$

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

5 Implementation

5.1 The robotic platform

The iCub (see Fig. 1) is a humanoid robot for research in artificial intelligence and cognition. It has 53 motors that move the legs, waist, head, arms, and hands and it has the average size of a 3-year-old child. It was developed in the context of the EU project RobotCub (2004-2010) and subsequently adopted by more than 25 laboratories worldwide. Its stereo vision system (cameras in the eyeballs), proprioception (motor encoders), touch (tactile fingertips and artificial skin) and vestibular sensing (IMU on top of the head) are important characteristics that enable the study of autonomy in humanoid robots. For the experiments reported in this paper, we used a novel simulator of the iCub that we developed with the Unity® game engine. The iCub right hand reference frame can be seen in Fig. 6. The errors presented in the results section (Section 6) are computed using this reference frame.

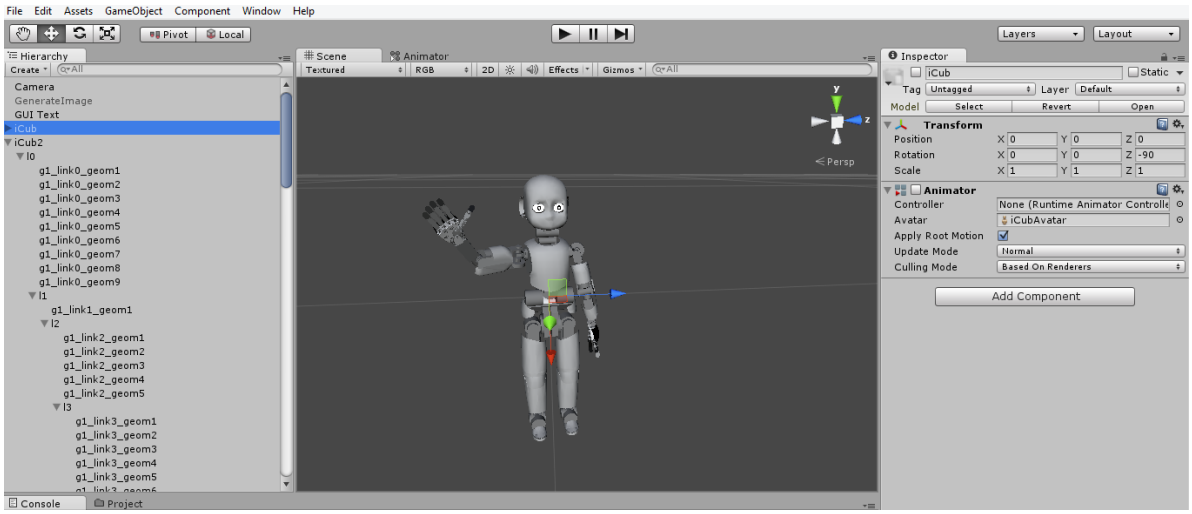


Fig. 7: View of Unity® game engine workspace with iCub model. The hierarchical tree of the components of the iCub can be seen on the left side.

5.2 The Unity® iCub Simulator

Unity® is a well-known cross-platform game engine developed by Unity Technologies. We choose this option to develop our internal graphical simulation due to its ease of use, efficiency, extensibility and programming features. It uses the C# programming language for scripting which we have used extensively to interact with the other software components in our system. We imported a previously developed model of the iCub in the COLLADA file format into the game engine. This model is based on the CAD model of the real robot. A hierarchical tree of the robot kinematics (see Fig. 7) was defined where each node has a reference frame attached and a pivot point. The rotation of this hierarchical object structure is around this pivot point. In other words, this tree represents the relationship between the several objects in the model (i.e. the robot body parts). For instance, the fingers are coupled with the robot hand, thus, if the hand moves the fingers will move along and update their absolute position in the world, maintaining the relative pose in the hand reference frame. A built-in physics engine is present and can be used to affect objects with gravity, forces, and torques. However, since our observation model is based on geometrical and appearance properties, the forces present in the environment do not influence the results, so we decided to turn the physics engine off to increase the speed of image generation. Our goal was to develop a geometric simulator capable of generating several hundreds of images per second in a desired pose. In spite of the high nominal frame rate of the game engine (about 1000 Frames Per Second, FPS) the actual one depends on the rendered scene (e.g. number of objects/triangles) and on the CPU and GPU specifications (e.g. number of cores). Another feature used to increase the generation speed was the batching method. In order to be able to use this feature, we define all the objects in the arm with the same material, decreasing the draw calls in the game engine. This means that all the body parts have the same color and the same dynamic properties (which does not affect the system behavior, since we are only considering the geometrical and appearance properties of the model, as mentioned before). Having the same color is not a problem for both observation methods: in the silhouette method, after the segmentation, we compare binary images; in the edge method, the most significant edges are generated by the shape of the bodies (and not by their color). On average, and due to the high number of different objects of the iCub body, we reduce to half the number of draw calls, from 500 to 250. The cameras in the eyeballs of the robot were defined using the intrinsic parameters of the real robot cameras. In computer graphics, the intrinsic parameters are defined through the Field Of View (FOV) instead of the focal length. The horizontal field of view (FOV_h) can be computed as:

$$\mathrm{FOV}_h = 2 \cdot \arctan\left(\frac{W}{2f}\right) \tag{20}$$

where, W is the width of the projection plane and f the focal length.

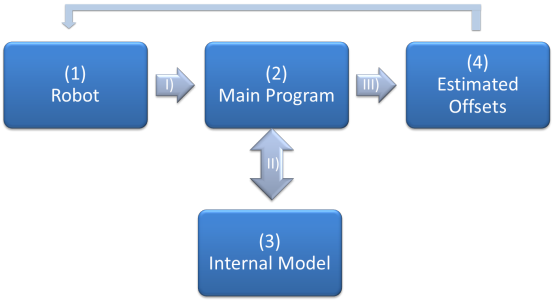


Fig. 8: The general workflow of our approach using an internal model to generate hypotheses.

5.3 General workflow

In Fig. 8 we can see the workflow of one iteration of our method. Four entities are present: Robot (1), Main Program (2), Internal Model (3) and Estimated Offsets (4) and three communications procedures: I), II) and III). In this paper we will focus on the communication channel II) and the implementation of the Main Program (2) and the Internal Model (3). Moreover, it is worth noting that in the experiments described in this paper we do not use the real iCub robot, but instead, we use the Unity® iCub simulator described in Section 5.2; indeed, the simulator is used both as Robot and as Internal Model. When using the simulator as Robot, we introduce artificial errors in the model to simulate a more realistic situation; this is explained in more details in Section 6. The main reason for using the simulator only is that we wanted to perform experiments with a precise ground truth about the hand pose, in order to validate our computational framework, and this would not have been possible by using the real robot unless external sensors were added. As we will discuss in Section 7, experiments with the real iCub robot are planned as future work; the general workflow and the software modules have been designed already to work with the real iCub robot as well.

The Main Program runs on the CPU and is responsible for the generation of the particles and for sending the information required by the Internal Model: the particles ($\boldsymbol{\beta}$); the encoders readings ($\boldsymbol{\theta}$) and the pre-processed images from the Robot ($\mathbf{y}$). This information is sent to the Internal Model through communication port II) via the YARP (Yet Another Robotic Platform) middleware [28]. The predicted images ($\bar{\mathbf{y}}$) are then generated inside the Internal Model based on the encoder readings and the best choice of particles (i.e. the current estimate of the joints offsets). Finally, the predicted images are compared to the real images and the likelihood computed with the observation model (see Section 3.3).

The Internal Model generates the virtual images on the GPU memory. In order to compare the real and virtual images we have two possibilities: (i) "download" the generated images from the GPU to CPU memory and compare them in CPU (we call this CPU/GPU approach, see Fig. 9), or (ii) "upload" the real image from CPU to GPU and compare them inside the GPU (we call this GPU approach, see Fig. 10). In this work we have implemented both approaches and compared them. The latter (GPU approach) is able to achieve up to 3 times speed-up with respect to the former (CPU/GPU approach).

We will describe in detail the proposed GPU approach, given its better efficiency. The CPU/GPU interoperation and the data structures that are shared and sent between the two processing units can be seen in Fig. 10. The CPU is used to communicate with the robot receiving the data from the cameras and encoders. It also manages the particle filter steps according to our filtering approach. The main bulk of data generation and evaluation is carried out at the GPU level by means of parallel processing, using the OpenGL, OpenCV and CUDA libraries. This interoperability will be explained in Section 5.4.

In section 6.1 we compare the two approaches. We will benchmark the time needed to generate the images (frame rate) and how many particles we can evaluate in one second (particle rate).

5.4 Interoperability between Libraries

The image processing routine necessary to compute the particles likelihood is implemented in the GPU thanks to the integration between Unity®, YARP, OpenGL, OpenCV and CUDA libraries. We use the game engine software in OpenGL mode launching it with the parameter `-force-opengl`. This feature is important for the integration of all the libraries. We render the visible scene in the cameras into a render texture structure (`RenderTexture` class) with the format ARGBHalf and get its pointer within the GPU memory, using the function `GetNativeTexturePtr()`. We initialize M render textures and generate M different scenes according to the M particles received from the particle filter. Every texture has the information of both left and right cameras.

We receive via YARP the images from the robot vision. In order to transfer the image data to an OpenGL structure we use two functions: `glBindTexture`, to define the initialized texture as the current one and `glTexImage2D` to import the data into the new structure using the information of the number of channels, the alignment, and the widthStep. The format used in OpenGL is `GL_RGBA16F`.

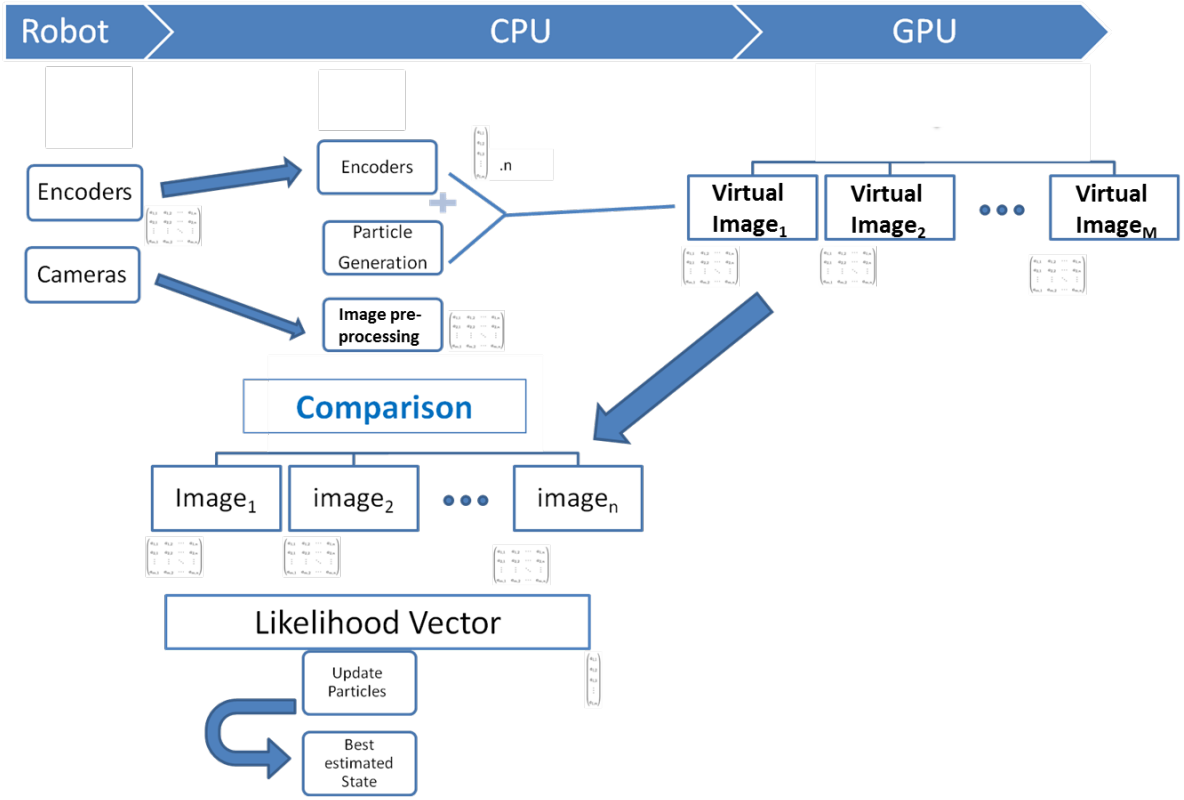


Fig. 9: Flowchart of the CPU/GPU approach, showing the operations made in CPU, GPU and at the robot level, where M is the number of particles.

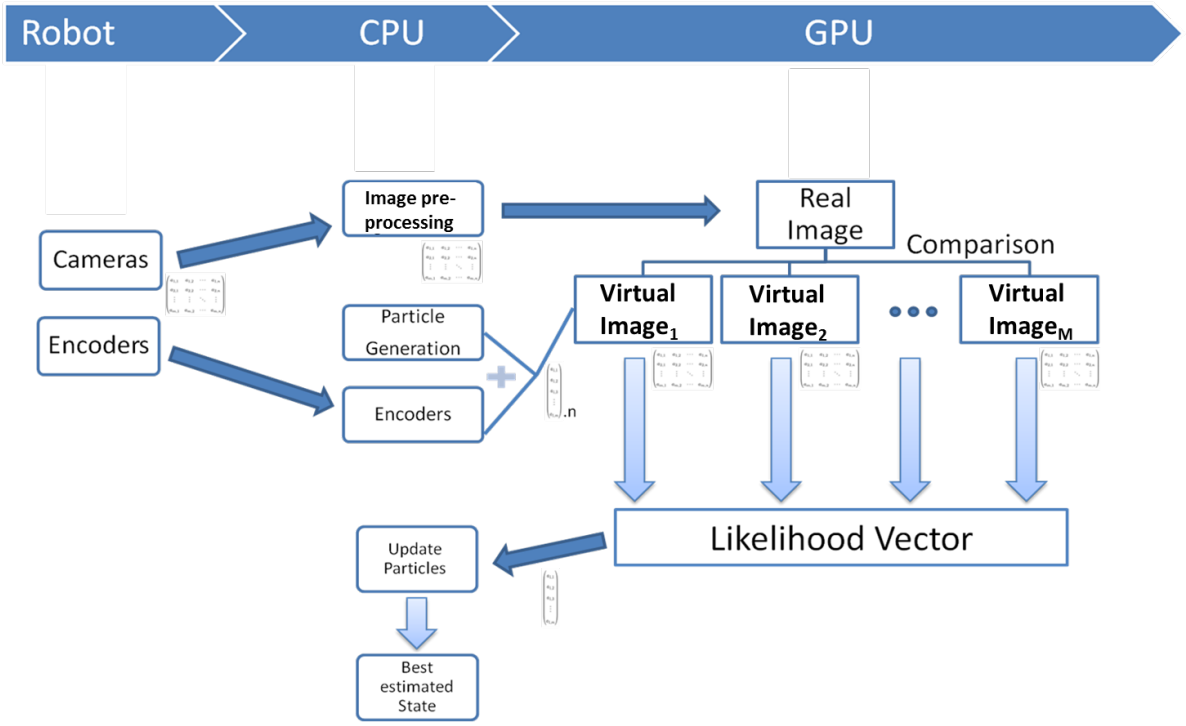


Fig. 10: Flowchart of the GPU approach, showing the operations made in CPU, GPU and at the robot level, where M is the number of particles.

In this phase, we have the concatenation of the real images (left and right) and the M generated images in OpenGL `Texture2D`. We exploit the CUDA programming language from NVIDIA in order to wrap the information from these OpenGL structures with the visible scenes to the OpenCV `GpuMat` class. We use the functions `cudaGraphicsGLRegisterImage` and `cudaGraphicsMapResources` to access the OpenGL information by CUDA. The usage of the CUDA function `cudaBindTextureToArray()` is now possible, transferring the image data from OpenGL texture to a CUDA texture. Unmap and Unregister the OpenGL structures in CUDA is an important step to guarantee a "clean" GPU memory. The GPU-accelerated Computer Vision module from the OpenCV library has some image processing functions as well as matrices and per-element operations. We have defined a CUDA kernel which takes the information in a CUDA texture and writes the image data in a `GpuMat` structure using the information about the data pointer (`GpuMat.data`), the step size (`GpuMat.step`) and the number of columns and rows (`GpuMat.cols`, `GpuMat.rows`), where each thread is responsible for copying one-pixel information to this new structure.

All the functions used to achieve the interoperability between the different libraries are loaded through a plugin written in C++.

6 Results

In this section we present the results of our experiments illustrating the following aspects: i) comparison between CPU/GPU and GPU implementation, in Section 6.1, ii) comparison between the two observation models (i.e. silhouette segmentation vs edge extraction), in Section 6.2, iii) general results of simultaneous online calibration and pose estimation, with generalization to different areas of the robot workspace, in Section 6.3 and iv) robustness to non-uniform background, in Section 6.4.
The experiments consist of reaching movements towards a target executed in the Unity® iCub simulator. To simulate the presence of modeling errors (that we would have on the real robot) we add artificial joint offsets in the arm kinematic chain of the simulator; the offsets have the same order of magnitude of the errors we typically encounter on the real robot. In the beginning of the movement, the state that we want to estimate (i.e. the joint offsets) is initialized from a Normal distribution with zero mean and standard deviation $\sigma = 5.0°$. As soon as the robot hand enters the field of view of the cameras, the particle filter starts estimating the hand pose and updating the offsets state vector. In the end of the movement, we evaluate the position and orientation errors of the estimated pose with respect to ground truth.

6.1 Comparison between CPU/GPU and GPU approaches

The GPU implementation of our computational framework is one of the main focus of this paper. Here, we compare two possible alternatives: i) CPU/GPU and ii) GPU only. We use a computer equipped with an Intel® Xeon® Processor W3503 at 2.4 GHz with 2 cores, 2 threads and a 4MB memory cache and an NVidia GeForce GTX 750 with 512 CUDA Cores, a base clock of 1020 MHz and 2048 MB of memory.

6.1.1 CPU/GPU methodology

In the CPU/GPU method we generate the virtual images (either binary or edge images in accordance with the observation model) on the GPU and "download" them into the CPU to perform the comparison with the real ones (see Fig. 9). The shortcoming of this methodology is the latency between the CPU/GPU data transfer. The data of the M generated images must be on the CPU in order to perform the comparison. This method has application in machines with a non-dedicated GPU, where GPGPU programming is not possible. Some computers have integrated graphics cards which do not support CUDA or another GPU programming language. For this reason, the study of this methodology is important as an alternative to the full GPU-enabled approach.

6.1.2 GPU methodology

The GPU method is our proposed approach for machines with GPGPU abilities (see Fig. 10). We generate and compare the virtual images inside the GPU using the developed iCub internal model. We only upload the stereo images from the real robot and download the score/likelihood of each particle, thus saving computational effort due to the decreased data transfer between CPU and GPU processing units. For this method a CUDA compatible graphics card is needed in order to perform the comparison between the observation and the generated hypotheses. In Table 1 we can see the computational efficiency of the different options in terms of particle rate (number of particles evaluated per second) and Frame rate (number of frames generated per second on the visual simulator). We can see that the particle rate is better when the image comparison is made at a GPU level. This increasing speed can be justified due

to the fact that we do not have to copy all the M generated images (M - particles) into the CPU. The only data transferred from the GPU to the CPU, as can be seen in Fig. 10, is the likelihood vector computed by the comparison between the uploaded real image and the generated hypotheses. With these results, we can argue that the GPU methodology presented before is the best way to branch the algorithm in order to increase the computational speed.

	CPU/GPU	GPU
Particle Rate	~ 100	~ 250
Frame Rate	~ 200	~ 600

Table 1: Comparison between the two different possible methodologies in particle Rate (particles per second) and frame Rate (images per second). We can see the better performance using the GPU only.

6.2 Silhouette segmentation vs Edge extraction

We performed 10 different experiments in order to compare the two proposed observation methods. We have pre-defined the duration of the reaching movement to be equal in every experience (90 frames). Each experiment consists of performing the same reaching movement twice; using either the silhouette or the edge approach as the observation method. In each experiment the initial and final poses of the hand are different. All the movements have the same duration (90 frames). In these experiments, we have a uniformly colored background to perform a better segmentation of the iCub hand in the image. The silhouette approach requires this particular background in order to be able to perform a segmentation of the robotic hand based on color. However, the edge extraction approach does not require this uniformly colored background. In order to provide a fair comparison between the two observation models, we use the same environment for both.
In Fig. 11 it is shown the mean (bold line) and standard deviation (shaded-part) of the estimation errors of the hand pose (position and orientation) (See Fig. 6) obtained during the experiments. We compare both methods to the non-Calibrated case (where we use only the proprioception, i.e., $\boldsymbol{\beta} = 0$). The convergence of both methods and the reduced error compared to the Non-Calibrated case show that both metrics can be applied to this problem. Notably, the orientation error using the edge extraction approach is lower compared with the silhouette segmentation. The position error of the hand has similar magnitudes in both cases.

	Frames				
Errors	0	15	30	60	90
Orient. Error [°]	9.00	8.06	6.84	6.07	5.40
Pos. Error [mm]	35.87	5.02	3.04	1.87	0.83

Table 2: The convergence of the pose estimation error during one reaching movement.

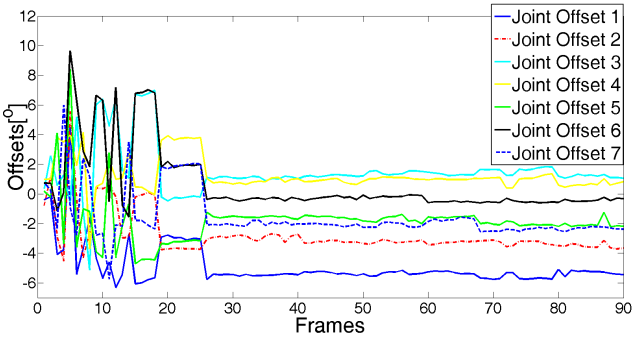
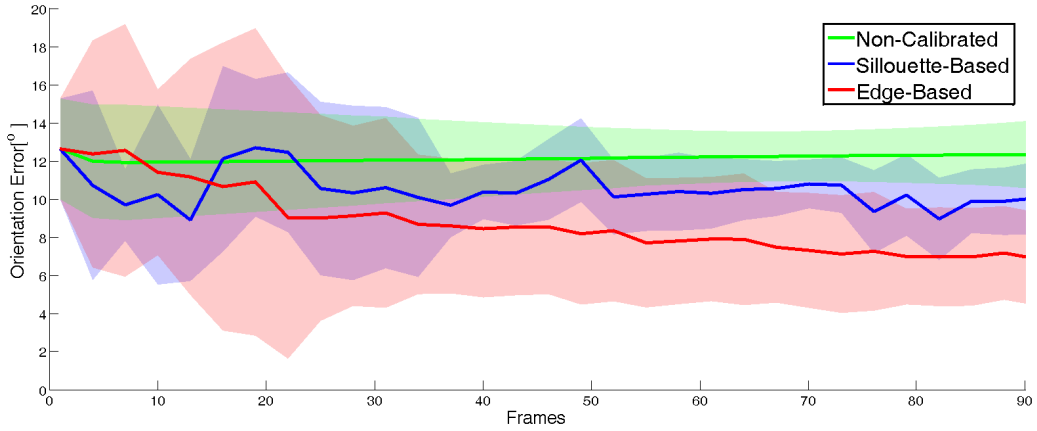


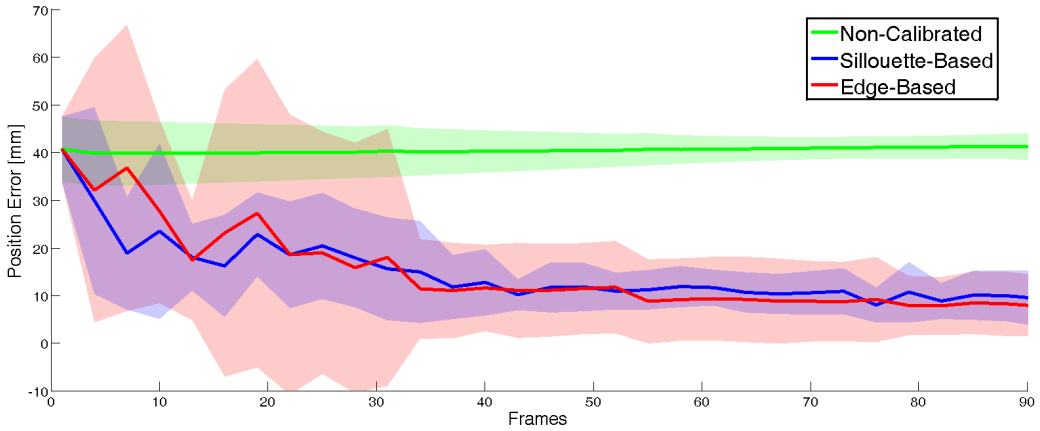
Fig. 12: The estimated offsets reach a steady state after about 30 frames (i.e. 30 iterations of the algorithm), showing the online calibration of the robot model during one reaching movement. (Best viewed in color)

6.3 Simultaneous pose estimation and robot calibration

In this section we illustrate the convergence properties of our algorithm using the GPU methodology (see Section 6.1) with the edge extraction approach (see Section 6.2). In Table 2 we show the estimation error of the hand, position and orientation, at different frames (i.e. iterations of the algorithm) in one reaching movement. We can see that the orientation and position errors significantly decrease during the motion. Moreover, in Fig. 12 we see the online offsets estimation during the movement. The estimation reaches a steady state after 30 frames. The calibration parameters in this figure are the joint offsets that calibrate the robot model, producing the hand pose estimation error that is shown in Table 2. This is just one of the possible sets of joint offsets that calibrate the model: as we explained in Section 3.3, due to the redundancy of the robot arm kinematics, other sets of offsets might be equally effective. Then, to better characterize the convergence properties of our estimation approach, we perform 10 different reaching movements with different initial and final hand poses. We use the same artificial offsets ($\boldsymbol{\beta}$) in all the movements. In Table 3 we show the mean and standard deviation (Std Dev) of the hand pose estimation error at the end of the reaching movements (Frame 90), comparing the online calibration and estimation with the non-calibrated (without filter) case ($\boldsymbol{\beta} = 0$).



(a) Orientation Error



(b) Position Error

Fig. 11: Position and orientation errors during reaching movements in a uniformly colored background. Comparison between the silhouette approach, the edges approach, and non-calibrated case. Thick lines represent the mean over 10 movements; the envelopes indicate the standard deviation. (Best viewed in color)

	With Filter		Without filter	
Errors	Mean	Std Dev	Mean	Std Dev
Orient. Error [°]	6.87	2.44	12.34	1.73
Pos. Error [mm]	7.81	6.37	41.27	2.74

Table 3: Mean and standard deviation of the final orientation and position errors over several experiments (different initial and final position).

6.3.1 Generalization and comparison with Cartesian parametrization

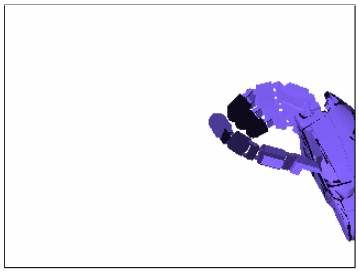
In this section we compare the generalization of the learned calibration parameters using two different types of parametrization methods: one in the joint space (proposed in this work) and other in the Cartesian space (proposed in [12,14,15]). Our approach uses offsets in the joints to compensate for calibration errors and estimate the hand pose (Eq. (3)). Another alternative, as mentioned in Section 3, is to use a Cartesian error transformation matrix (Eq. (5)). We can compute $\hat{\mathbf{T}}_{\boldsymbol{\beta}}$ in Eq. (5) as the error transformation matrix to be applied to the nominal transformation (see Fig. 13).

In order to compare both methods, we introduce a given set of artificial joint offsets in the model, and then perform a reaching movement from an initial pose toward a final pose; we call this the training movement. During the movement, we estimate both the joint offsets $\hat{\boldsymbol{\beta}}$ and the Cartesian error transformation matrix according to:

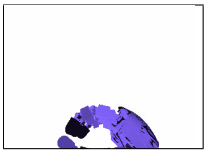
$$\hat{\mathbf{T}}_{\boldsymbol{\beta}} = \mathscr{K}(\boldsymbol{\theta})^{-1}\mathscr{K}(\boldsymbol{\theta}+\hat{\boldsymbol{\beta}}) \tag{21}$$

We retain the values of $\hat{\boldsymbol{\beta}}$ and $\hat{\mathbf{T}}_{\boldsymbol{\beta}}$ at the final (training) pose. Then, we move the hand to six different testing poses (see Fig. 14). At each pose we estimate the hand pose using both the joint error parametrization $\hat{\boldsymbol{\beta}}$, and

Fig. 13: Cartesian error transformation matrix ($T_{\boldsymbol{\beta}}$) between the real hand pose and the hand pose estimated by using the robot model.



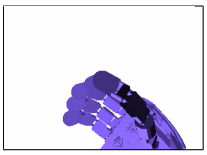
(a) Final training Pose



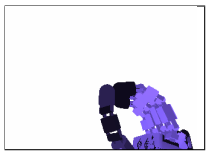
(b) Testing pose 1

(c) Testing pose 2

(d) Testing pose 3



(e) Testing pose 4



(f) Testing pose 5

(g) Testing pose 6

Fig. 14: Training in a trajectory with a final pose and testing in different poses. The position and orientation of the hand are very distinct. Here we show the left images.

the Cartesian error parametrization $\hat{\mathbf{T}}_{\boldsymbol{\beta}}$, obtained during the training movement, and compare them to ground truth. In Table 4 we show the position and orientation errors both in the training pose and in the different testing poses. We compare the joint offsets parametrization with the Cartesian parametrization; the nominal (non-calibrated) case is also shown for reference. It can be seen that the joint offsets parametrization is better in all the cases.

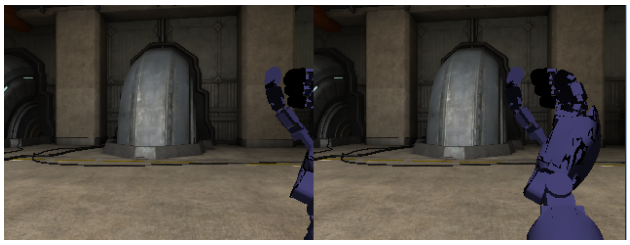


Fig. 15: Simulation of a non-uniform environment into the left and right eyes of the iCub robot.

6.4 Simultaneous pose estimation and robot calibration with non-uniform background

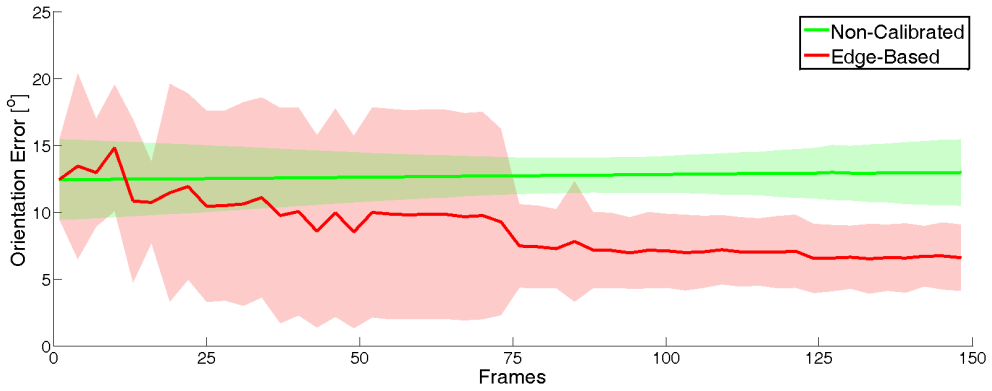
We simulated a realistic virtual environment which results in a non-uniform visual background in the robot images (see Fig. 15), to evaluate the robustness of our method to more complex visual scenes. We use the *robot lab project* from Unity Technologies that emulates a laboratory environment which is similar to a realistic background. With such non-uniform visual background, we exploit the edge extraction approach, since the silhouette approach is not very effective in this kind of scenarios, where it is difficult to obtain a good segmentation of the hand. We perform reaching movements, similar to the ones presented in Section 6.3, with the new background environment. In Fig. 16, we present the performance of our approach comparing it to the non-calibrated case. The computed errors at the end of the movement (6.61° orientation error and 8.69 mm position error) have the same magnitude as in the uniform background case (6.87° orientation error and 7.81 mm position error, see Fig. 11) which indicates that our approach is robust to non-uniform environments. Moreover, after 70 frames the filter converges to a good solution with small variance.

7 Conclusions and future work

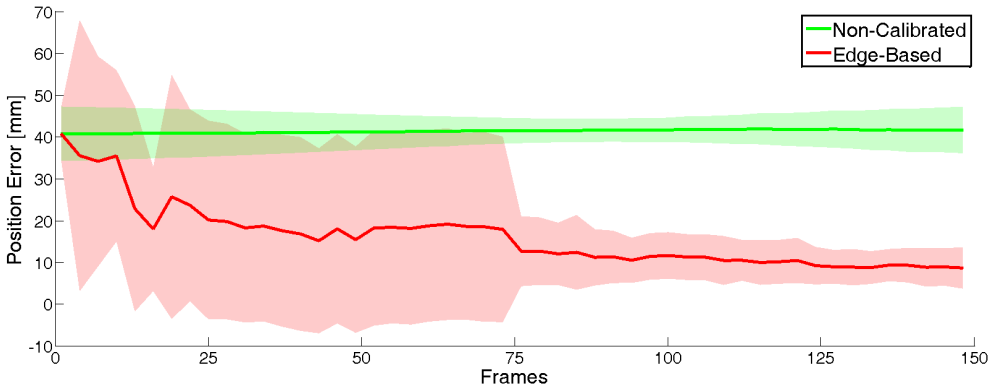
We presented a method to simultaneously estimate the pose of the robot hand and to calibrate the robot model using a GPU-enabled particle filter approach. Our solution is marker-free and solely based on the robot embedded sensors (stereo vision and proprioception). We use an online estimation technique, and the GPU implementation allows us to perform the computation in real time. Indeed, we show that we can increase the speed of computation almost of a factor of three, with respect to our previous hybrid CPU/GPU implementation.
We introduce a new approach for the visual estimation of the hand pose, based on edges extraction, that provides advantages with respect to silhouette segmentation. In-

	Orientation Error [°]			**Position Error [mm]**		
	Joint offsets	*Cartesian*	*Nominal*	*Joint offsets*	*Cartesian*	*Nominal*
Final training pose	5.42	5.42	12.28	4.16	4.16	42.05
Testing pose 1	6.55	7.2	13.84	9.17	9.30	46.06
Testing pose 2	5.53	11.66	13.210	8.14	31.69	41.71
Testing pose 3	8.10	14.23	16.50	12.56	35.37	50.15
Testing pose 4	7.70	14.3	16.32	13.31	30.97	44.34
Testing pose 5	3.44	6.65	9.87	5.60	17.14	33.06
Testing pose 6	5.90	6.45	12.30	3.82	4.33	34.96

Table 4: Estimation errors with the joint offsets parametrization, Cartesian parametrization and nominal model (i.e. non-calibrated). The orientation and position errors are always smaller with the joint offsets parametrization.



(a) Orientation Error



(b) Position Error

Fig. 16: Position and orientations errors during reaching movement with a non-uniform background. Comparison between edge-based approach and the uncalibrated case.

deed, we show the robustness of the edge extraction approach to a non-uniform visual background, which is similar to the environment present in a realistic scenario.

Overall, we show that we can reduce the hand pose estimation error considerably with respect to using the nominal (non-calibrated) robot model (of about 2 times in the hand orientation and 6 times in the hand position). Moreover, we achieve good generalization when calibrating the robot on a single reaching movement and then testing the calibration in different areas of the workspace. Therefore, the proposed method can be used to correct the kinematic model of a robot during reaching to improve, for instance, the task of grasping an object. Despite no real world experiments are shown in this manuscript, the proposed method and the developed software can be applied immediately to the real robot.

Indeed, we plan to make real world experiments using external sensing to compute the ground-truth. Setting-up

a 3D sensor, for instance, the optitrack motion capture [31], could be a solution to compute the hand pose in a real scenario. We also plan to continue improving the speed of the algorithm implementing the Particle Filter framework at the GPU architecture and using more than one iCub model inside the game engine. If we could generate more hypotheses in real time we could evaluate more images from our robotic platform. The segmentation procedure for the silhouette-based approach can be also improved to cope with more complex backgrounds. This aspect could be achieved using a 3D vision system based on the stereo calibrated cameras in the iCub eyeballs.

Acknowledgements This work was partially supported by Fundação para a Ciência e Tecnologia (FCT) PLURIANUAL budget [UID/EEA/50009/2013] and the EU Projects POETICON++ [FP7-ICT-288382] and LIMOMAN [PIEF-GA-2013-628315].

References

1. Ashmead, D., McCarty, M., Lucas, L., Belvedere, M.: Visual guidance in infants' reaching toward suddenly displaced targets. Child Development **64**, 1111–1127 (1993)
2. Birbach, O., Bäuml, B., Frese, U.: Automatic and self-contained calibration of a multi-sensorial humanoid's upper body. In: Intl. Conf. on Robotics and Automation. Saint Paul, Minnesota, USA (2012)
3. Borgefors, G.: Distance transformations in digital images. Comput. Vision Graph. Image Process. **34**(3), 344–371 (1986)
4. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
5. Canny, J.: A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence (1986)
6. Choi, C., Christensen, H.I.: 3d textureless object detection and tracking: An edge-based approach. In: IROS, pp. 3877–3884. IEEE (2012)
7. Ciliberto, C., Smeraldi, F., Natale, L., Metta, G.: Online multiple instance learning applied to hand detection in a humanoid robot. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1526–1532 (2011)
8. Ciocarlie, M., Hsiao, K., Jones, E.G., Chitta, S., Rusu, R.B., Sucan, I.A.: Towards Reliable Grasping and Manipulation in Household Environments. In: Intl. Symposium on Experimental Robotics (ISER). New Delhi, India (2010)
9. Comport, A., Marchand, E., Pressigout, M., Chaumette, F.: Real-time markerless tracking for augmented reality: the virtual visual servoing framework. IEEE Transactions on Visualization and Computer Graphics **12**(04), 615–628 (2006)
10. Cox, T.F., Cox, M.: Multidimensional Scaling, Second Edition, 2 edn. Chapman and Hall/CRC (2000)
11. Erol, A., Bebis, G., Nicolescu, M., Boyle, R.D., Twombly, X.: Vision-based hand pose estimation: A review. Computer Vision and Image Understanding **108**, 52–73 (2007)
12. Fanello, S.R., Pattacini, U., Gori, I., Tikhanoff, V., Randazzo, M., Roncone, A., Odone, F., Metta, G.: 3d stereo estimation and fully automated learning of eye-hand coordination in humanoid robots. In: IEEE-RAS International Conference on Humanoid Robots (2014)
13. Fulkerson, B., Soatto, S.: Really quick shift: Image segmentation on a gpu. In: Workshop on Computer Vision using GPUs, held with the European Conference on Computer Vision (2010)
14. Gratal, X., Romero, J., Bohg, J., Kragic, D.: Visual servoing on unknown objects. Mechatronics **22**(4), 423 – 435 (2012)
15. Gratal, X., Romero, J., Kragic, D.: Virtual visual servoing for real-time robot pose estimation. In: Proc. of the 18th IFAC World Congress, pp. 9017–9022 (2011)
16. Hoffmann, M., Marques, H., Hernandez Arieta, A., Sumioka, H., Lungarella, M., Pfeifer, R.: Body schema in robotics: A review. IEEE Transactions on Autonomous Mental Development **2**(4), 304–324 (2010)
17. Hol, J.D., Schon, T.B., Gustafsson, F.: On resampling algorithms for particle filters. In: IEEE Nonlinear Statistical Signal Processing Workshop, pp. 79–82 (2006)
18. Jamone, L., Brandao, M., Natale, L., Hashimoto, K., Sandini, G., Takanishi, A.: Autonomous online generation of a motor representation of the workspace for intelligent whole-body reaching. Robotics and Autonomous Systems **64**(4), 556–567 (2014)
19. Jamone, L., Damas, B., Endo, N., Santos-Victor, J., Takanishi, A.: Incremental development of multiple tool models for robotic reaching through autonomous exploration. PALADYN Journal of Behavioral Robotics **03**(03), 113–127 (2013)
20. Jamone, L., Damas, B., Santos-Victor, J., Takanishi, A.: Online learning of humanoid robot kinematics under switching tools contexts. In: IEEE-RAS International Conference on Robotics and Automation (ICRA), pp. 4811–4817 (2013)
21. Jamone, L., Natale, L., Nori, F., Metta, G., Sandini, G.: Autonomous online learning of reaching behavior in a humanoid robot. International Journal of Humanoid Robotics **09**(03), 1250,017 (2012)
22. Klingensmith, M., Galluzzo, T., Dellin, C., Kazemi, M., Bagnell, J.A., Pollard, N.: Closed-loop servoing using real-time markerless arm tracking. In: IEEE-RAS International Conference on Robotics and Automation (ICRA) - Humanoids Workshop (2013)
23. Kording, K.P., Wolpert, D.M.: Bayesian integration in sensorimotor learning. Nature **427**, 244–247 (2004)
24. Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P.: Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU. SIGARCH Comput. Archit. News **38**(3), 451–460 (2010)
25. Leitner, J., Harding, S., Frank, M., Forster, A., Schmidhuber, J.: Humanoid learns to detect its own hands. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1411–1418 (2013)
26. Lockman, J.J., Ashmead, D.H., Bushnell, E.W.: The development of anticipatory hand orientation during infancy. Journal of Experimental Child Psychology **37**, 176–186 (1984)
27. Mathew, A., Cook, M.: The control of reaching movements by young infants. Child Development **61**, 1238–1257 (1990)
28. Metta, G., Fitzpatrick, P., Natale, L.: YARP: Yet Another Robot Platform. International Journal on Advanced Robotics Systems (2006)

29. Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., Montesano, L.: The icub humanoid robot: an open-systems platform for research in cognitive development. Neural Networks **23** (2010)
30. Moutinho, N., Brandao, M., Ferreira, R., Gaspar, J., Bernardino, A., Takanishi, A., Santos-Victor, J.: Online calibration of a humanoid robot head from relative encoders, imu readings and visual data. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2070–2075 (2012)
31. NaturalPoint: Optitrack Motion Capture System. `http://www.optitrack.com/`. [Online; accessed 1-06-2015]
32. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable Parallel Programming with CUDA. Queue **6**(2), 40–53 (2008)
33. Oikonomidis, I., Kyriazis, N., Argyros, A.: Markerless and efficient 26-dof hand pose recovery. In: 10th Asian Conference on Computer Vision. Queenstown, New Zealand (2010)
34. Oikonomidis, I., Kyriazis, N., Argyros, A.: Efficient model-based 3d tracking of hand articulations using kinect. In: Proceedings of the British Machine Vision Conference, pp. 101.1–101.11. BMVA Press (2011)
35. Park, S.I., Ponce, S., Huang, J., Cao, Y., Quek, F.: Low-cost, high-speed computer vision using nvidia's cuda architecture. In: Applied Imagery Pattern Recognition Workshop, 2008. AIPR '08. 37th IEEE, pp. 1–7 (2008)
36. Pattacini, U.: Modular cartesian controllers for humanoid robots: Design and implementation on the icub. Ph.D. thesis, Italian Institute of Technology (2011)
37. Periquito, D., Nascimento, J., Bernardino, A., Sequeira, J.: 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 (2013)
38. Saxena, A., Driemeyer, J., Ng, A.Y.: Robotic grasping of novel objects using vision. Int. J. Rob. Res. **27**(2), 157–173 (2008)
39. Shreiner, D., Sellers, G., Kessenich, J.M., Licea-Kane, B.M.: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3, 8th edn. Addison-Wesley Professional (2013)
40. Sinha, S.N., Frahm, J.m., Pollefeys, M., Genc, Y.: GPU-based Video Feature Tracking and Matching. Tech. rep. (2006)
41. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press (2005)
42. Ulbrich, S., de Angulo, V., Asfour, T., Torras, C., Dillmann, R.: Rapid learning of humanoid body schemas with kinematic bézier maps. In: IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp. 431–438 (2009)
43. Ulbrich, S., de Angulo, V.R., Asfour, T., Torras, C., Dillmann, R.: General robot kinematics decomposition without intermediate markers. IEEE Trans. Neural Netw. Learning Syst. **23**, 620–630 (2012)
44. Vicente, P., Ferreira, R., Jamone, L., Bernardino, A.: Eye-hand online adaptation during reaching tasks in a humanoid robot. In: Joint IEEE International Conferences on Development and Learning and Epigenetic Robotics (ICDL-Epirob), pp. 175–180 (2014)
45. Vicente, P., Ferreira, R., Jamone, L., Bernardino, A.: Gpu-enabled particle based optimization for robotic-hand pose estimation and self-calibration. In: Robotica / IEEE International Conference on Autonomous Robot Systems and Competitions (Robotica/ICARSC) (2015)
46. Wang, R.Y., Popović, J.: Real-time hand-tracking with a color glove. ACM Transactions on Graphics **28**(3) (2009)