GPU-Enabled particle based optimization for robotic-hand pose estimation and self-calibration

Pedro Vicente, Ricardo Ferreira, Lorenzo Jamone, Alexandre Bernardino
Instituto de Sistemas e Robótica, Instituto Superior Técnico, Lisboa, Portugal
Email: {pvicente,ricardo,ljamone,alex}@isr.ist.utl.pt

***Abstract*—Humanoid robots have complex kinematic chains that are difficult to model with the precision required to reach and/or grasp objects properly. In this paper we propose a GPU-enabled vision based 3D hand pose estimation method that runs during robotic reaching tasks to calibrate in real time the kinematic chain of the robot arm. This is achieved by combining: i) proprioceptive and visual sensing; and ii) a kinematic and computer graphics model of the system. We use proprioceptive input to create visual hypotheses about the hand appearance in the image using a 3D CAD model inside the game engine from Unity Technologies. These hypotheses are compared with the actual visual input using particle filter techniques. The outcome of this processing is the best hypothesis for the hand pose and a set of joint offsets to calibrate the arm. We tested our approach in a simulation environment and verified that the angular error is reduced 3 times and the position error about 12 times comparing with the non-calibrated case (proprioception only). The used GPU implementation techniques ensures a performance 2.5 times faster than performing the computations on the CPU.**

***Index Terms*—humanoid robot, robotic-hand pose estimation, robot self-calibration, 3D model based tracking, GPU, reaching.**

I. Introduction

Everyday tasks like grasping an object with precision or push an electrical plug into a socket are quite simple for humans that make these activities in an daily basis but complex and challenging for autonomous robots to perform. A correct kinematic model of the limb structure is of paramount importance when performing such complex tasks, particularly of the arms and hands in order to properly grasp and manipulate objects. Although an accurate analytical model of the kinematic chain of a humanoid arm with several degrees of freedom can be difficult to obtain due to hard-to-model aspects (e.g. elasticity) and changes that might occur over time (e.g. unalignment of a joint rotation axis), estimating the hand pose error and re-calibrating the model is fundamental in every robotic platform.

From studies with humans we know that five-months-old children use vision to correct the hand position and orientation during the movement [1], with performance that improves during development [2]. With the intention of speeding up the reaching process, real time reaching and grasping tasks in robots are performed, normally, without any visual feedback control approach [3] [4]. Nevertheless, if the robot's internal model is not accurate enough, grasping will not be successful.

The computational effort needed to estimate the hand pose using the visual sensors is the bottleneck for the systems: the usage of the GPU for these computer vision purposes can be one of the solutions.

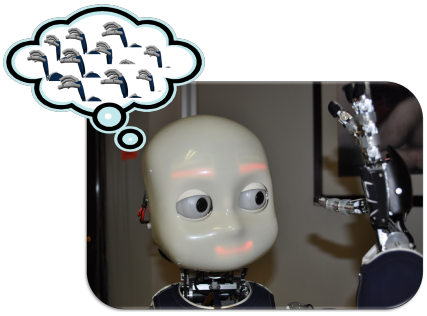


Fig. 1: The iCub humanoid robot imagining hypotheses of his hand pose.

In the last years these techniques have been used to speed-up well-known computer vision algorithms, as Canny-Edge detection [5] or image segmentation procedures [6]. Some authors state that it is possible to achieve a speed-up of almost 10 times [7] in high level computer vision algorithms, when more conservative ones, in a wider perspective, claim for 2.5 times in average [8].

In this paper our goal is to focus on the implementations procedures on the GPU and use its capabilities to improve the robotic-hand pose estimation in real-time and reduce the error on the end-effector near the vicinity of the object. The calibration procedure will improve the actual internal model of the robot, the *iKin* kinematic model [9] provided within the YARP/iCub software framework, adapting it during reaching movements in order to cope with the modeling inaccuracies.

The rest of the paper is organized as follows. In Section II we report the related work in robotics and we highlight our contribution more specifically. Then in Section III we formalize the problem, while in Section IV we provide the details of our proposed solution. In Section V we account the implementation details as well as the robotic platform used. Finally, in Section VI we present the experimental results, and in Section VII we draw our conclusions and sketch future work.

II. Related Work

The detection and tracking of the robot hand in its visual system are fundamental components of our method. Diversified approaches for vision-based hand pose estimation have

been proposed by a few authors [10]. The following two cited authors aim to estimate the hand pose in an arbitrary configuration. In [11] they propose to track the human hand using a colored glove to simplify the hand detection and them search efficiently for a correspondence pose in a database of examples. The human hand is divided into different segments in [12]. They estimate the 26-DOF of the human hand model using a GPU implementation to speed-up the algorithm and run it in quasi-real time using a skin color segmentation and edge maps within a Particle Swarm Optimization. Unlike these approaches, at each time step we have an initial estimation of the hand pose based in the robot forward kinematics which helps to reduce the problem in a local search method. In [13] they propose the estimation of the hand in a predefined pose, thus reducing the problem to a 6D search. The usage of image moments allows them to compute analytically the translation therefore seeking the hand orientation in a 3D database. Machine learning techniques were also applied to this estimation problem. The Cartesian Genetic Programming method was used by [14] to learn from visual examples how to detect the robot hand inside an image. Moreover, another machine learning method (Online Multiple Instance Learning) was used in [15] for this hand detection, thus exploiting the incoming information from the encoders and visual optic flow labelling automatically the training images. The previous methods only estimate the hand position inside the image discarding its angular orientation. The 26-DOF of the human hand model was tracked by [16] utilizing real 3D sensory data. The second purpose of our method is to calibrate the eye-head-arm-hand kinematic chain of a humanoid robot. Several authors suggested solutions for this problem, as reviewed in [17] including offline and online learning. The method used by [18] requires 5 minutes of data acquired during specific robotic movements with a special marker in the robot wrist. It optimizes offline some parameters of the kinematic chain (angle offsets and elasticity) of a upper humanoid torso using non-linear least squares. An online solution has been studied by [19], [20] which adapted the kinematic chain using markers as well. In [21] they decomposed the kinematic chain into smaller segments. They proposed an offline and online learning solution without using markers. The research in [22] is also based on a marker-free correction of the robot kinematics using RGB-D data. The eye-hand coordination problem was studied in [23] and [24]. In [23] they use a particle filter framework and the visual and proprioceptive sensors to improve an *a priori* knowledge of the kinematic chain. In [24] the calibration is performed with several ellipsoid movements in a predefined hand-pose. They track the tip of the index finger using the 3D calibrated stereo system and employ optimization techniques to learn the mapping function between the two reference frames.

A. Our contribution

Automatic calibration of the robot kinematic chain and hand pose estimation are a fundamental topic for autonomous robots. In this paper we propose a GPU implementation of a robotic hand pose estimation method within a particle filter framework. Our objective is to achieve a faster and real time implementation of the method for the iCub arm. We upload the incoming proprioceptive information from the robot to a computer graphics visual simulator. The rendering engine generates a series of images with possible hypothesis of the hand appearance around the nominal pose indicated by the proprioceptive information. These hypothetical images are then compared with the real image coming from the robot cameras to approximate the real pose. Both the hypotheses rendering and image comparisons are done in the GPU using a combination of the Unity® Game engine and OpenCV GPU functions, to decrease the latency between the CPU/GPU data transfer.

III. Problem statement

Our goal is to estimate the hand pose in the eye reference frame in order to do an online calibration of the kinematic chain of a humanoid robotic arm. We follow the theoretical approach presented in [23].

A. State Model

The internal model's learning consists in estimating the joint offsets (β) of the angles in the kinematic chain defined as:

$$\beta = \theta - \theta_r + \eta \tag{1}$$

where θ is the value read by the encoders, θ_r the real value of the joint position, β is a systematic offset, and η a zero mean Gaussian noise with covariance $\mathbf{Q}$, $\eta \sim \mathcal{N}(0, \mathbf{Q})$.

The state vector for the system is composed of the offsets in the arm joints. We do not include offset errors in the eye-head kinematic chain to reduce the complexity of the problem. Instead, we have a calibrated head chain using the procedure defined in [25].

The offsets in Equation (1) define the state vector of an unobserved Markov process as $\boldsymbol{\beta} = [\beta_1\ \beta_2\ \beta_3\ \beta_4\ \beta_5\ \beta_6\ \beta_7]^T$ 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 allow for small changes in $\boldsymbol{\beta}$ we introduce a state transition noise $\mathbf{w}$ and model the system state transition model as: $\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + \mathbf{w}$, here $\mathbf{w} \sim \mathcal{N}(0, \mathbf{K})$ is a zero mean Gaussian noise with a given covariance $\mathbf{K} = \sigma_s^2 \mathbf{I}_7$, where σ_s is the standard deviation.

B. Observation Model

Our humanoid robot has two sources of information, the head and arm's encoders and the cameras on the eyeballs. We use images from the cameras as the state's observation ($\mathbf{y}$) and define the measurement probability as:

$$p(\mathbf{y}_t|\boldsymbol{\beta}_t, \boldsymbol{\theta}_t) \tag{2}$$

where $\mathbf{y}_t$ is the concatenation of the values from the two images, left and right and $\boldsymbol{\theta}_t$ the encoders values at time t.

From our Unity® iCub simulator, as depicted in Figure 2, we can obtain two predicted images, $\hat{\mathbf{y}}$, feeding the visual simulator with the encoder readings and the state vector $\boldsymbol{\beta}$.

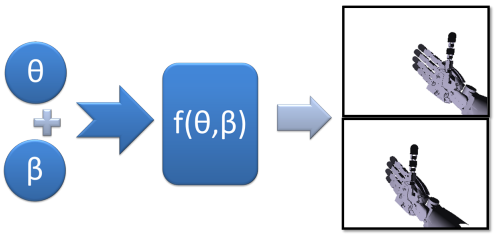


Fig. 2: Image generation at Unity® iCub simulator.

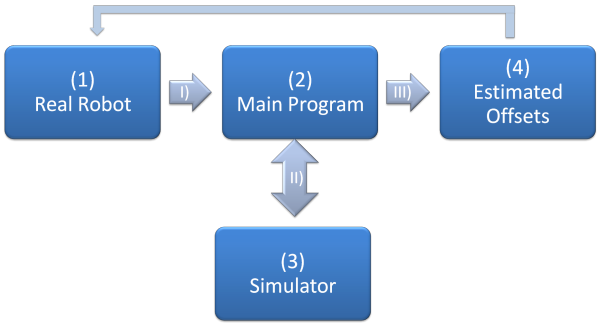


Fig. 3: General Work Flow of the our approach using a simulator to generate hypotheses.

We compute the probability in Eq.(2) using the Hammoude metric (d_{HMD}) [26] as a distance metric between the predicted and the real images silhouettes. Since the Hammoude distance has a range between [0 1], we assume a likelihood model as:

$$p(\mathbf{y}_t|\boldsymbol{\beta}_t, \boldsymbol{\theta}_t) \propto 1 - d_{\text{HMD}}(\mathbf{y}_t, f(\boldsymbol{\theta}, \boldsymbol{\beta})) \tag{3}$$

We need to underline that redundancy in the joint angles may lead to different states ($\boldsymbol{\beta}$) with the same final pose. Therefore, the estimated $\boldsymbol{\beta}$ will be just one set of offsets that can explain our pose in the images.

IV. Our approach

In our approach we will use a Particle Filter as the one defined in [27] and [23]. Under the Markov assumption (See Sec. III-A) we can compute recursively the *a posteriori* distribution $p(\boldsymbol{\beta}_t|\mathbf{y}_{1:t}, \boldsymbol{\theta}_{1:t})$ using the previous estimation $p(\boldsymbol{\beta}_{t-1}|\mathbf{y}_{1:t-1}, \boldsymbol{\theta}_{1:t-1})$ and the observation model in Eq.(2).

The four stages of our particle filter are:

1) Prediction - we use the state transition equation defined before
2) Observation - the predicted images $\hat{\mathbf{y}}$ are generated using the visual simulator and compared with the real ones
3) Update - the likelihood (See Eq. (3)) is used to re-weight the particles.
4) Re-sampling - the particles are sampled according to their weight. We use the systematic re-sampling method [28] to guarantee that a particle with a weight greater than $1/n$ is always re-sampled, where n is the number of particles.

Although the state is represented at each time step as a distribution approximated by the particles, for evaluation purposes we must compute our best state's value guess. For this purpose, we use a kernel density estimation to smooth the weight of the particles according to the information of neighbor particles and choose the particle with the highest weight as defined in [23].

V. Implementation

A. The robotic platform

The iCub (see Figure 1) is a humanoid robot for research in artificial intelligence and human cognition. It has 53 motors that move the legs, waist, head, arms and hands and it has an average size of a 3 years 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 to be able to study autonomy in humanoid robots.

B. General flowchart

The proposed algorithm uses a new developed iCub simulator within Unity® which works with CPU and GPU computation. In Figure 3 we can see the work flow of one iteration of our method from a wide perspective. Four entities are present: Real Robot (1), Main Program (2), Simulator (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 Simulator (3).

The Main Program is responsible for the generation of the particles that sends to the Simulator through communication II) via the YARP middleware: the offsets ($\boldsymbol{\beta}$), the encoders readings ($\boldsymbol{\theta}$) and the segmented images from the Real Robot ($\mathbf{y}$). The predicted images ($\hat{\mathbf{y}}$) are generated inside the Simulator using the iCub Kinematics and appearance model. Then the predicted images are compared to the real image and the likelihood computed via our observation model (See Sec. III-B).

The simulator generates images on the GPU memory and in order to compare the images in the main program we have two possibilities: "download" the generated images from the GPU to CPU memory and compare them in CPU, or compare them inside the GPU uploading the real image. In this work we choose to compare the images in GPU, thereby decreasing the computational effort of the algorithm. In Figure 4 we see in more detail the CPU/GPU inter-operation and the data structures that are shared and sent between the two processing units.

The CPU is used to communicate with the robot receiving the data from the cameras and encoders the particles' generation 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 Sec. V-D.

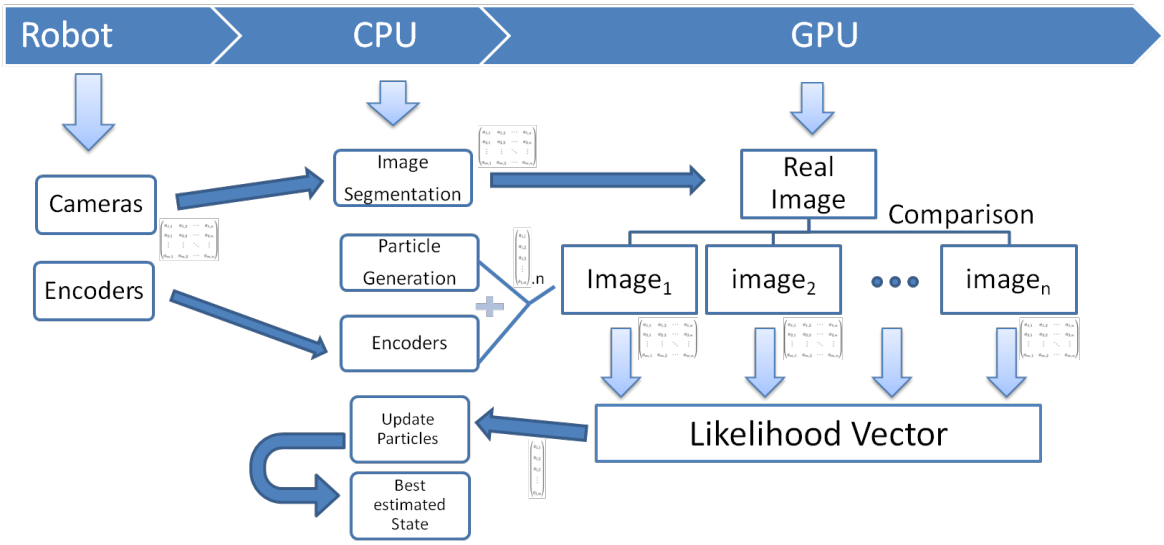


Fig. 4: Scheme showing the operations made in CPU, GPU and at the robot level where n is the number of particles.

In section VI-C we compare the possible ways to branch the algorithm. We will benchmark the time needed to generate the images (frame rate) and how many particles we can evaluate in one second (particle rate).

C. Details on the Unity® iCub Simulator

Unity® has a built-in physics engine and we can use it to affect objects with gravity, forces and torques. We decide to turn physics off to increase the speed of image generation. Our intention was to develop a geometric simulator capable to generate several images per second in a desired pose.

Unity® software is able to generate approximately 1000 frames per second (FPS). In spite of the high theoretical frame rate, the actual one will depend on the rendered scene (number of objects) and the GPU's specifications (number of cores).

The iCub CAD model was imported inside Unity® using a hierarchical tree. This tree consist in the relationship between the several objects of the robot. For instance, the fingers are coupled with the robot's 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.

Another aspect used to increase the generation's speed was the batching method within the Unity® software. 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. On average, and due to the high number of different objects of the iCub body, we reduce in half the number of draw calls, from 500 to 250.

The cameras in the eyeballs of our platform were defined using the intrinsic parameters of the real cameras. In computer graphics, normally, we change the Field Of View (FOV) of the camera instead of the focal length. Using some trigonometry we calculate the horizontal field of view (FOV_h) as:

$$\text{FOV}_h = 2 \cdot \tan^{-1}\left(\frac{W}{2f}\right) \tag{4}$$

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

D. Interoperability between Libraries

The observation model computation in a GPU fashion way was possible due to the integration between Unity®, OpenGL, OpenCV and CUDA libraries. We use the Unity® software in OpenGL mode to render the scene into render texture structures and get their native/hardware pointer within the GPU memory. We initialized n render textures and generated n different scenes according to the n particles received from the particle filter.

The OpenCV library, in particularly, the GPU-accelerated Computer Vision module has some image processing functions as well as matrices and per-element operations. Using some of these functions it was possible to implement our likelihood metric on the GPU, for instance the `gpu::subtract` function and the bitwise operator.

The CUDA programming language of NVIDIA was also used to convert the OpenGL render texture with the scenes from Unity® to the GpuMat structure of the OpenCV library.

VI. Results

In this section we will present our experimental results showing: i) automatic calibration ii) generalization of the method iii) fast computation. In the first section we add artificial joint offsets in the arm kinematic chain and compare the results with the non-calibrated case. We use our filter with 200 particles initially sampled using a Normal distribution with $\sigma = 5.0°$. In the second point, we show how our method generalizes in different arm poses. We compare to other solutions in which the kinematic error is encapsulated as a roto-translational matrix (as proposed in other works) instead of joint offsets. Finally, we stress the benefits of using the GPU for the implementation of our method.

	Frames				
	1	50	100	150	200
Angular Error [°]	12.05	5.616	4.65	2.02	2.58
Position Error [mm]	7.88	7.26	4.16	2.61	1.19

TABLE I: Mean and Variance of the final orientation and position errors over several experiments (different initial and final position).

	With Filter		Without filter	
	Mean	Std Dev	Mean	Std Dev
Angular Error [°]	4.5495	2.032	14.01	1.65
Position Error [mm]	3.3357	1.5163	42.67	5.47

TABLE II: Mean and Variance of the final orientation and position errors over several experiments (different initial and final position).

A. Simulation experiments

In the simulation experiments we add offsets in the arm kinematic chain of the iCub simulator. The joints offsets have the same order of magnitude of the real robot. In order to illustrate the convergence of the filter in Table I we show the position and angular errors at same steps of the algorithm.

We perform several reaching tasks with different initial and final hand positions. In Table II it can be seen the mean and standard deviation of the results and we compare them with the non-calibrated case non calibrated ($\beta_i = 0$). The errors are in the final pose after the reaching movement was performed.

B. 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:

$$T_{\text{err}} = \hat{T_r} \cdot T_{\text{enc}}^{-1} \tag{5}$$

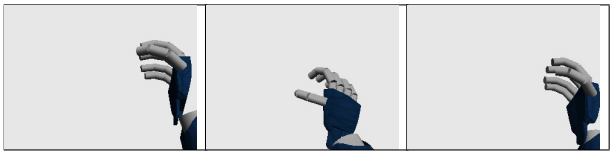
where, $\hat{T_r}$ is the estimation of the real transformation and T_{enc} the transformation defined by the encoders joints. The T_{err} will be the transformation to apply in the end-effector to correct the final pose of the hand.

In [29] 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 given set of artificial joint offsets to a final position. In this final position, the estimated matrix $\hat{T_r}$ using the joint offsets (our method) or the matrix T_{err} is the same, therefore the errors are equal.

We tested the two approaches in five different positions (See Figure 5) to argue about the generalization after learning the errors. For this purpose, we maintain the initial offsets in the joints and: i) apply the transformation T_{err} ii) add the estimated joint offsets.

In the Table III we can see the final errors in these new positions between $\hat{T_r}$ 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 joint offsets.

(a) Final training Pose (b) Testing pose 1 (c) Testing pose 2



(d) Testing pose 3 (e) Testing pose 4 (f) Testing pose 5

Fig. 5: Training in a trajectory with a final pose and testing in different poses. The position and orientation of the hand are very distinct.

	Angular Error [°]		Position Error [mm]	
	Joint offset	*End-effector*	*Joint offset*	*End-effector*
Position 1	2.5602	3.8892	2.2221	4.9162
Position 2	4.7247	3.9889	9.6751	13.3802
Position 3	2.3720	4.3778	4.1777	10.4285
Position 4	4.0699	8.6078	6.4442	7.6157
Position 5	3.8914	0.8117	8.5436	16.1736

TABLE III: Differences between the end-effector Transformation and joint offset methods. The position error is better in all the cases with the joint offsets method.

C. Comparison between CPU and GPU methodologies

The algorithm's implementation on the GPU was one of the main focus of this work. In order to evaluate our effort we compare three different methodologies: CPU, CPU/GPU and GPU. In Table IV we can see the computational efficiency of the different options in terms of particle rate (particles evaluated per second) and Frame rate (frames generated per second on the visual simulator). The first methodology uses the previous iCub simulator to generate the hypotheses and compare them in the CPU using C++ language. In the CPU/GPU we use the Unity® iCub simulator to generate images and "download" them into the CPU to perform the comparison. The GPU method is our proposed approach; we generate and compare the images inside the GPU.

In the last two methods (CPU/GPU and GPU), despite te fact that the same visual simulator is used, 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 don't have to copy all the n generated images (n - particles) into the CPU. The only memory copied from the GPU to the CPU, as can be seen in Figure 4, 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 division into CPU and GPU presented before is the best way to branch the algorithm in order to increase the computational speed.

	CPU	CPU/GPU	GPU
Particle Rate	~ 30	~ 100	~ 250
Frame Rate	~ 50	~ 200	~ 600

TABLE IV: Comparison between the three different possible methodologies. We can see the better performance using only the GPU.

VII. Conclusions and future work

We presented a method to estimate the pose of a robotic hand, based on a particle filter and GPU framework. We obtained good results comparing with the non calibrated case, decreasing 3 times the angular error and 12 times the position error. Our approach of using offsets was compared, for a generalization purpose, with a matrix transformation error method. In the five trial positions, the position error of the hand was always better using offsets in the joints.

The GPU implementation increased the speed of the algorithm 2.5 times making possible to evaluate more incoming images per second.

As future work we 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 Unity®. If we could generate more hypotheses in real time we could evaluate more images from our robotic platform.

The segmentation procedure can be also improved to cope with more complex backgrounds. On one hand, this aspect could be achieved using a 3D vision system based on the stereo calibrated cameras in the iCub eyeballs. On the other hand, an edge-based comparison could circumvent some of the problems of segmentation in cluttered backgrounds.

Acknowledgment

This work was partially supported by FCT [UID/EEA/50009/2013], by the FCT project BIOMORPH-EXPL/EEI AUT/2175/2013 and the EU Projects POETICON++ [FP7-ICT-288382] and LIMOMAN [PIEF-GA-2013-628315].

References

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

[2] 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.

[3] 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.

[4] 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.

[5] S. I. Park, S. Ponce, J. Huang, Y. Cao, and F. Quek, "Low-cost, high-speed computer vision using nvidia's cuda architecture," in *Applied Imagery Pattern Recognition Workshop, 2008. AIPR '08. 37th IEEE*, Oct 2008, pp. 1–7.

[6] B. Fulkerson and S. Soatto, "Really quick shift: Image segmentation on a gpu," in *Workshop on Computer Vision using GPUs, held with the European Conference on Computer Vision*, September 2010.

[7] S. N. Sinha, J.-m. Frahm, M. Pollefeys, and Y. Genc, "GPU-based Video Feature Tracking and Matching," Tech. Rep., 2006.

[8] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 451–460, Jun. 2010.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] N. K. Iason Oikonomidis and A. Argyros, "Efficient model-based 3d tracking of hand articulations using kinect," in *Proceedings of the British Machine Vision Conference*. BMVA Press, 2011, pp. 101.1–101.11.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] S. Ulbrich, V. R. de Angulo, T. Asfour, C. Torras, and R. Dillmann, "General robot kinematics decomposition without intermediate markers." *IEEE Trans. Neural Netw. Learning Syst.*, vol. 23, pp. 620–630, 2012.

[22] 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.

[23] P. Vicente, R. Ferreira, L. Jamone, and A. Bernardino, "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)*, Oct 2014, pp. 175–180.

[24] S. R. Fanello, U. Pattacini, I. Gori, V. Tikhanoff, M. Randazzo, A. Roncone, F. Odone, and G. Metta, "3d stereo estimation and fully automated learning of eye-hand coordination in humanoid robots," in *IEEE-RAS International Conference on Humanoid Robots*, 2014.

[25] N. Moutinho, M. Brandao, R. Ferreira, J. Gaspar, A. Bernardino, A. Takanishi, and J. Santos-Victor, "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),*, Oct 2012, pp. 2070–2075.

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

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

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

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