Robot learning by Gaussian process regression

Denis Forte, Aleš Ude and Andrej Kos
"Jožef Stefan" Institut
Department of Automation, Biocybernetics and Robotics
Ljubljana, Slovenia
E-mail: denis.forte@ijs.si, ales.ude@ijs.si, andrej.kos@ijs.si

Abstract — **Intelligent robots cannot be programmed in advance for all possible situations, but they should be able to generalize based on the acquired knowledge. In robot learning based on imitation of human activity we often use statistical methods that generalize observed (learned) movements. The acquired data is used to generate useful robots responses in situations for which the robot has not been specifically instructed how to respond. The paper describes the robot learning with Gaussian process regression that creates the model and estimates the parameters for generalization of the acquired motor knowledge, which is accumulated as a database of example movements. New actions are synthesized by applying Gaussian process regression, where the goal and other characteristics of an action are utilized as queries to create an optimal control policy with respect to the previously acquired knowledge. The paper demonstrates that the proposed methodology can be integrated with an active vision system of a humanoid robot. 3D vision data is used to provide query points for statistical generalization.**

I. INTRODUCTION

Machine learning is an area of artificial intelligence that deals with the development of techniques that allow a computer or machines (robots) to obtain various skills. Machine learning heavily relies on statistics, because statistics also deals with processing and generalization of data.

Supervised learning is a principle of machine learning for function modeling based on the learning collections of samples. The learning system generalizes the knowledge acquired from learning examples in the right way. The result of supervised learning is in most cases a global function model that maps input data into some desired output. Often this mapping consists of local models such as the nearest neighbor algorithms.

Among various paradigms that could be used for this purpose, Gaussian process regression (GPR) has proved to be very effective. It is a Bayesian regression method that provides a predictive distribution. Therefore GPR can show good generalization performance and the predictive distribution can be used to measure the uncertainty of the estimated function. It has been demonstrated that this technique outperforms other regression methods on problems such as estimating inverse dynamics of a seven degrees of freedom robot arm [1].

The developed methods were implemented on a humanoid robot HOAP-3. The goal was to teach the robot a three-dimensional reaching motion, which could be used to grasp objects anywhere in front of the robot. This was accomplished by generalizing example reaching movements of the robot's right arm from its initial position to any point in its operating range using Gaussian process regression. We provided 140 previously learned trajectories to the algorithm with endpoints evenly distributed throughout the workspace of the right arm, which is approximately 30x30x10 cm. Endpoints were about 5 cm apart from each other. The top of the hand needed to be visible to the cameras on the robot's head and so the visible area limited the workspace of the robot. The trajectories were created by physically moving the robot's right arm and measuring the joint angles of the arm. Generalization enables the robot to move the right hand to the points inside the work space, which weren't learned previously, with the trajectory of similar shape as the example motions.

To encode the learned actions we use nonlinear dynamic systems (dynamic movement primitives – DMPs). Every generalized trajectory is represented by a DMP, which is guaranteed to converge towards the desired target point, even if the target moves while the arm is reaching towards the target [2]. The robot with two cameras can locate the 3D position of the object and thus reach towards it, provided the object is visible and in the robot's workspace. If the object changes position during the extension of the arm, the arm will modify the generalized trajectory so that it can follow the object.

II. GAUSSIAN PROCESS REGRESSION

Gaussian processes are an example of a probabilistic modeling with probabilistic forecasting outputs. The use of Gaussian processes comes from statistics and is based on Bayesian probability modeling. Models based on Gaussian processes have an interesting and useful feature that, besides output values, they also predict confidence in this value. This is important because it tells how much we can trust the prediction of the model for certain input signals.

Random variable is a variable whose value depends on chance. The random process (stochastic process) arises when we have more independent (temporal, spatial, etc.) realizations of the random variable. Random vector X is a vector whose components are random variables.

978-1-4244-6886-7/10/$26.00 ©2010 IEEE

Gaussian process is a random multi-dimensional process whose random variables x_i are set by the vector of the mean value $\mu(X)$ and covariance matrix K. Normal distribution is assumed. For the given learning data, which consists of pairs of system inputs and outputs, we assume that each output point $f(x_i)$ provides a single random variable x_i. The associated covariance matrix K of multidimensional normal distribution is modeled by covariance function $C(x_i,x_j)$, which is a function of input points of the system. Vector of the mean values of multidimensional distribution $\mu(X)$ and covariance function $C(x_i,x_j)$ completely define Gaussian process regression [1].

The value of the covariance function $C(x_i,x_j)$ expresses the correlation between the individual outputs $f(x_i)$ and $f(x_j)$ with respect to inputs x_i and x_j. Covariance function can be any function that generates a non-negative definite covariance matrix K for any set of input vectors X. We usually choose such covariance function that points, which are closer together in the input space, are more strongly correlated than more distant ones. By assumption of a stationary process, the most commonly used covariance function is Gaussian, where element K_{ij} of the covariance matrix K is calculated as [3]:

$$K_{ij} = C(x_i, x_j) = \sigma_f^2 \sum_{d=1}^{D} \exp\left(-\frac{1}{2l_d^2}\left(x_i - x_j\right)^2 \right) + \sigma_n^2 \delta_{ij} \tag{1}$$

Equivalent vector format is:

$$C(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{1}{2}(x_i - x_j)^T W^{-1} (x_i - x_j) \right) + \sigma_n^2 \delta_{ij}$$

where $$W^{-1} = \begin{bmatrix} l_1 \cdot \ddots & 0 \\ 0 & l_D \end{bmatrix} \tag{2}$$

Parameters of the covariance function (σ_f, σ_n, l_1, ... l_D) called hyperparameters are calculated based on the learning data. D is the length of the input vector x or its dimension. The covariance matrix is positive definite if, all hyperparameters are greater than zero. Parameter σ_f adjusts the size of variance, l_d parameters reflect the relative importance of individual component of the input signal, while the expression $\sigma_n^2 \delta_{ii}$ models the white noise of the output of the system and is different from zero only when $x_i = x_i$ [1].

The prediction of output at new entry point has normal probability distribution with mean value $\mu(X^*)$ and variance:

$$\begin{bmatrix} y \\ y^* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X^*) \\ K(X^*,X) & K(X^*,X^*) \end{bmatrix} \right) \tag{3}$$

$$\mu(X^*) = \bar{y}^* = K(X^*,X)\left[K(X,X) + \sigma_n^2 I \right]^{-1} y \tag{4}$$

$$\operatorname{cov}(y^*) = K(X^*,X^*) - K(X^*,X)\left[K(X,X) + \sigma_n^2 I \right]^{-1} K(X,X^*) \tag{5}$$

Size of the variance gives confidence in the predicted value of the output model. Vector X^* is the input test vector with dimension D by which we are interested in prediction of the output $\bar{y}^*$. The expected value of output $\bar{y}^*$ is equal to the mean value $\mu(X^*)$ of the distribution. We can look at the vector $K(X^*,X)[K(X,X) + \sigma_n^2]^{-1}$ as a vector of weights that estimates the importance of outputs. If the new input is very different from learning inputs, then the expression $K(X^*,X)[K(X,X) + \sigma_n^2]^{-1}K(X,X^*)$ is small and the variance $cov(y^*)$ becomes large [1].

III. Dynamic Movement Primitive - DMP

Dynamic movement primitives (DMPs) are a formal movement representation, based on non-linear dynamic systems. DMPs are based on a system of non-linear second order differential equations that describe the properties of a desired movement. One of the most important benefits of DMPs is that they aren't directly dependent on time. As explained in [4], the explicit dependency on time is annoying because it increases the complexity of the suspension, termination, or re-launch of time when unpredictable disturbances happen during the movement. To account for such perturbations we can analytically adjust the underlying system of differential equations. Such adjustments are very useful in case of various unpredictable disturbances during the execution that aren't part of learned data. Such an approach is suitable for integration with the visual system of a humanoid robot.

Let us present the theoretical background of DMPs. For one degree of freedom denoted by y, which can either be one of the internal joint angles or external task space coordinates, the following system of linear differential equations with constant coefficients has been proposed as a basis for motion specification [4]

$$\tau \dot{z} = \alpha_z \left(\beta_z (g - y) - z \right) \tag{6}$$

$$\tau \dot{y} = z \tag{7}$$

With properly selected parameters ($\alpha_z = 4\beta_z$ and $\tau > 0$) system (6)-(7) converges towards a unique attractor point $[y\ z]^T = [g\ 0]^T$. Differential equations (6) and (7) ensure that y converges towards g and can therefore be used for realization of discrete "point-to-point" movements. To increase the limited number of trajectories that can be described by equations (6) and (7) and thus enable the general approximation of "point-to-point" movements, we have to slightly modify equation (6). In case of discrete movements we can add a linear combination of radial basis functions to equation (6):

$$f(x) = \frac{\sum_{i=1}^{N} w_i \psi_i(x)}{\sum_{i=1}^{N} \psi_i(x)} x, \quad \psi_i(x) = \exp\left(-h_i (x - c_i)^2 \right), \tag{8}$$

where $h_i > 0$ and c_i are the centers of radial basis functions distributed along the trajectory. The phase variable x is used in (8) instead of time, to avoid direct dependency of f on time. The dynamics of the phase variable x is defined by:

$$\tau \dot{x} = -\alpha_x x, \tag{9}$$

with initial value $x(0) = 1$. Solution to (9) is given by $x = \exp(-\alpha_x t / \tau)$, thus x tends to 0 as time increases. The result is the following system of differential equations:

$$\tau \dot{z} = \alpha_z \left(\beta_z (g - y) - z \right) + f(x), \tag{10}$$

$$\tau \dot{y} = z, \tag{11}$$

which can be used for approximation of discrete movements of different forms. As x tends to zero, the influence of non-linear function $f(x)$ decreases with time and the system (10)-(11) converges to $[g\ 0]^T$ just like (6)-(7). The other role of x is to localize the radial basis functions along the trajectory that needs to be approximated. The resulting control policy associated with variable y defines what is called a dynamic movement primitive [5].

IV. Generalization with Gaussian Process Regression

To learn the grasping action we guided the right arm of the humanoid robot HOAP-3 along the desired reaching trajectory. We held the robot's arm near the right elbow joint and manually moved it from the initial position to the desired end-points in front of the robot.

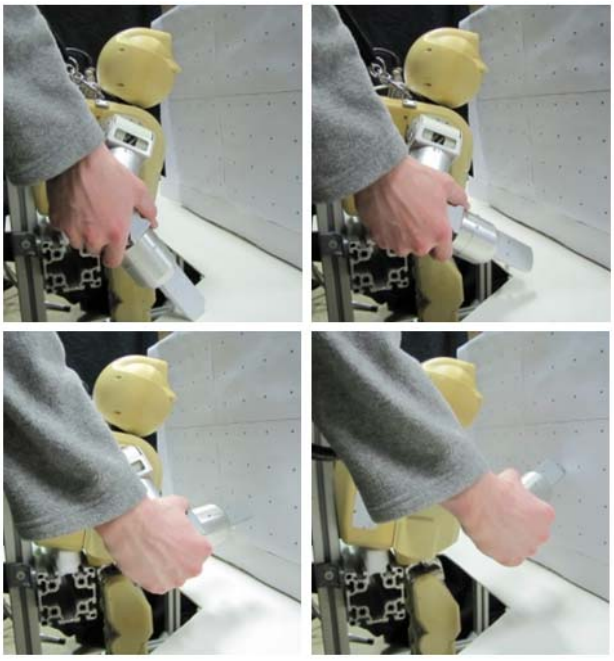

Figure 1. Image sequence shows the teaching of reaching movements to humanoid robot HOAP-3 with kinesthetic guiding.

All four joints of the right arm were collected along the demonstrated trajectory. We moved the arm towards the plate in front of the robot with plotted points with a distance of 5 cm between them. We gradually moved the plate away from the robot and made a series of motions at four different distances from the robot (Figure 2). Some movements were also made at the side of the robot without the platform. Altogether we performed 148 movements, but subsequently found that 8 of the final points can't be seen by the robot's cameras and have to be removed from the database of example trajectories.

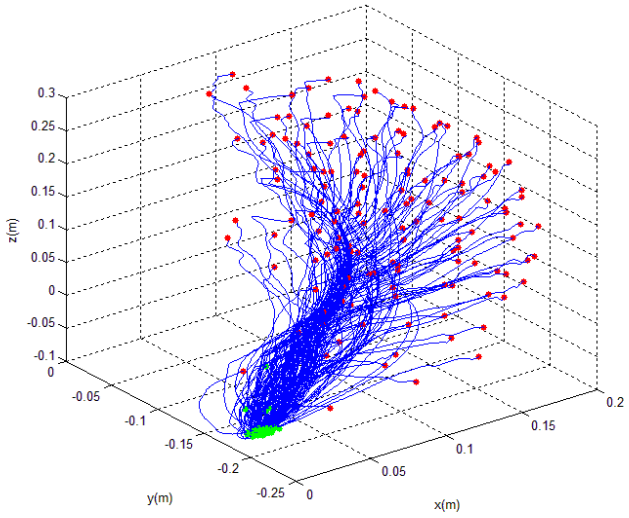

Figure 2. Query points and trajectories of reaching experiment. The goals for the demonstration movements are marked with red dots. The starting positions are marked with green dots and the trajectories with blue lines.

We used the obtained joint angles to calculate velocities and accelerations, and also to get the end-points of trajectories in external coordinates using the kinematics of the robot.

Our intention was that HOAP-3 locates the object with its cameras and reaches for it with his right hand by human like motion. The robot thus estimates the position of the object using stereo vision provided by its on-board cameras. Note that the head was moving during tracking to increase the visible area. After learning, if the object is anywhere within its operating range, the robot can reach for it with movement similar to the demonstrated ones, which were recorded in the training phase. When the robot hand comes to the final position, it grasps the object and performs the appropriate motion back to the initial position. If the object position changes while the arm is moving, the robot uses its vision to estimate the new object position and modify the DMP to reach the new goal position.

Robot was programmed to first turn its head and look for the desired object. After the cameras detect the position of the object, the head is moved using visual serving (Figure 3):

$$pitch_{new} = pitch_{old} - \frac{y}{z} \cdot k_{pitch}, \tag{12}$$

$$yaw_{new} = yaw_{old} - \frac{x}{z} \cdot k_{yaw}, \tag{13}$$

so that the object is in the center of the image of the right camera. The object is continuously tracked to estimate its position if it starts moving again.

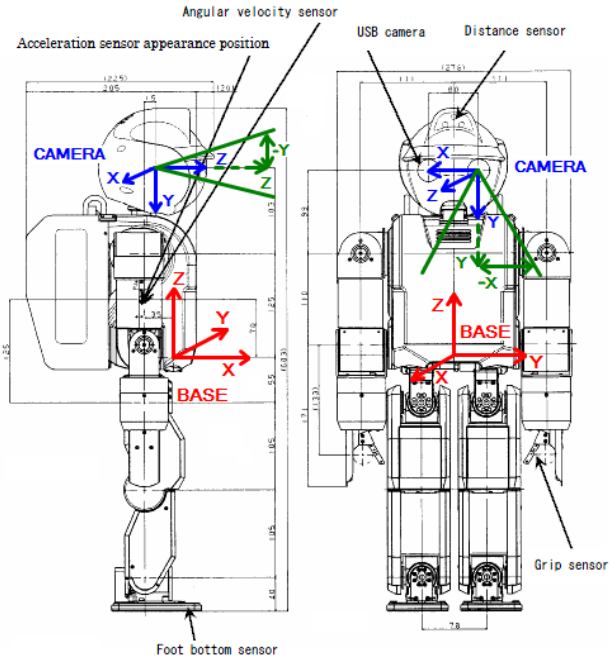

Figure 3. Base (red) and camera (blue) coordinate systems are (-2, 3, 22.4) cm apart. Visual serving moves the head and tracks the object regarding its distance proportion (green lines).

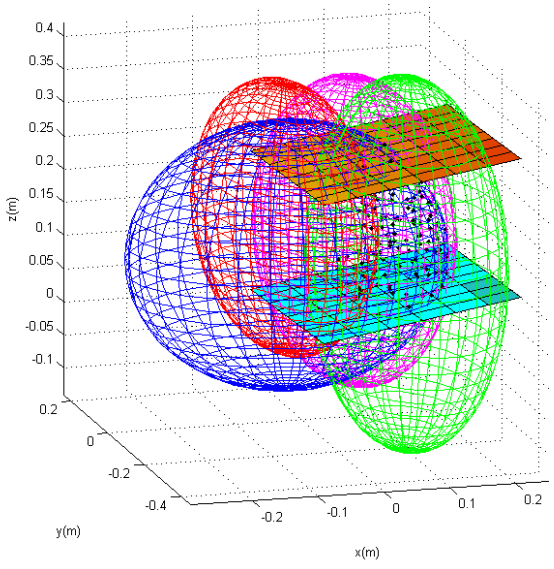

Figure 4. Black dots are the learned end-points, which are enclosed by 4 ellipsoids and 2 planes that analytically approximate the workspace of the robot's right hand.

We analytically described the operating range of the right hand with four ellipsoids and two planes (Figure 4) so that the surface area includes all of the endpoints of learned trajectories.

Based on the learned trajectories (that were shown to the robot previously) and nonlinear dynamic systems, a field of weights is created, where each trajectory is approximated by 25 basis functions. The acquired field of weights is used in the Gaussian process to calculate the weights of any intermediate trajectory in this area. It turns out that 25 is the optimal number of weights depending on the complexity and length of the training trajectories.

Gaussian process regression generalizes the function that maps the final object positions in the external coordinates to the field of final hand position in internal coordinates (equation (4)). Given a new goal in external coordinates (as estimated by the cameras), it calculates the final hand position in internal coordinates that represent the four final joint angles of the right arm, which serve as attractor points of the generalized DMP. In such a manner the Gaussian process regression approximates the partial inverse kinematics of the robot's right arm. We also created a field of movement durations and so with generalization determine the appropriate duration of each movement for any intermediate point. By means of Gaussian process regression we calculate the DMP parameters, which describe the desired trajectory (equations (10) and (11)), depending on the new target points. DMP is integrated using Euler integration and the output can be used to control robot arm.

The Euler integration calculates trajectories step by step (here x and z are internal system variables and y is one of the joint angles specifying the trajectory):

$$x_{new} = x_{old} + \frac{-\alpha_x \cdot x}{\tau}, \tag{14}$$

$$z_{new} = z_{old} + \frac{\alpha_z \cdot \left(\beta_z \cdot (g - y) - z\right) + f(x)}{\tau}, \tag{15}$$

$$y_{new} = y_{old} + \frac{z}{\tau}, \tag{16}$$

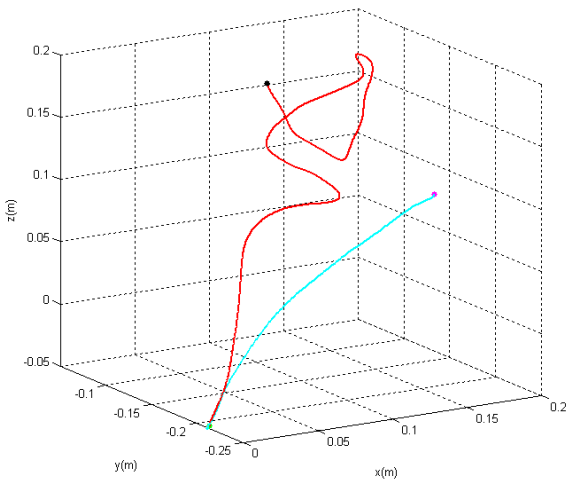

Figure 5. The light blue trajectory represents the right arm move if the object would not move. The red trajectory picture the right hand path tracking the moving object.

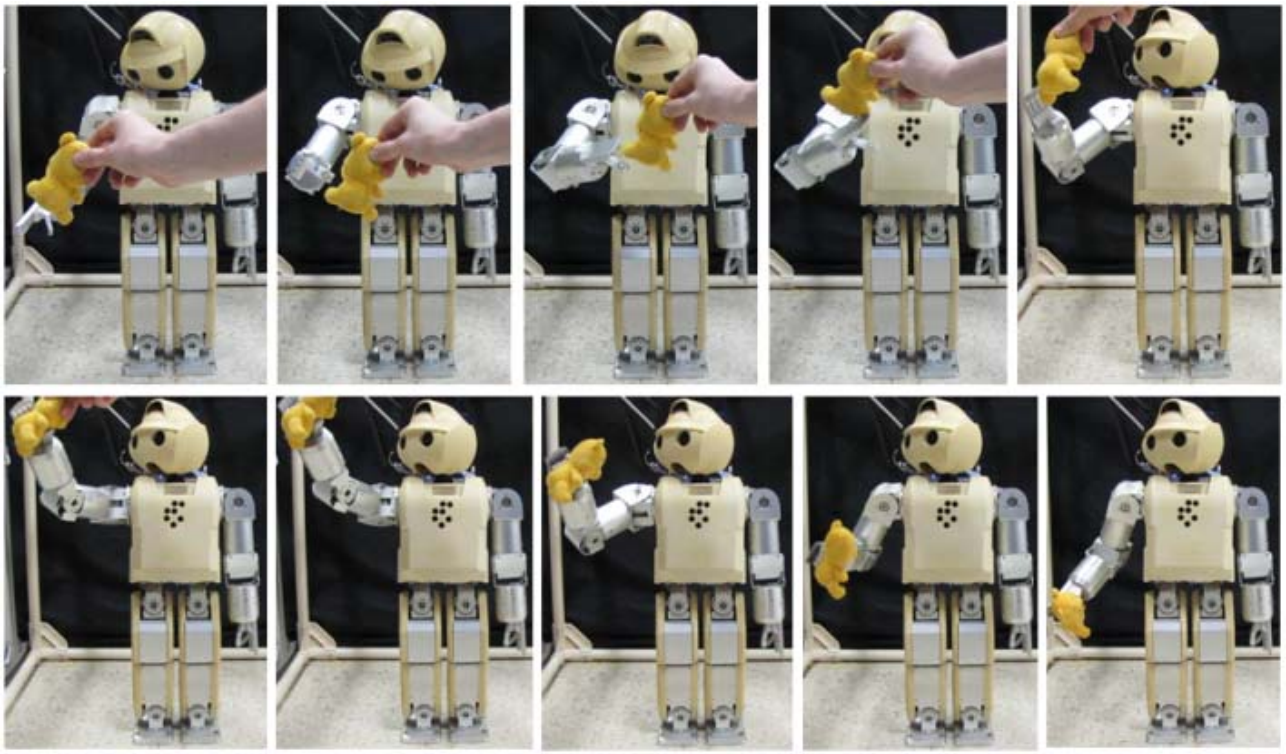

Figure 6. The robot HOAP-3 detects the object and tracks it. Once the object stops moving, the robot can grasp it and moves the arm back to its starting position.

The generated joint angles are used to control HOAP-3 at 120 Hz. The cameras continuously track the object and estimate its current position. If the object position changes, a new attractor point is given to the nonlinear dynamic system in the Euler integration, but the form of the movement trajectory will remain similar to the learning motion until the end of the estimated duration time. Afterwards it takes the form dictated by the critically damped system (Figure 5).

The target point cannot be reached perfectly because of two fundamental errors. The biggest error is caused by stereo vision, which sees the object on average of 63 mm away from the actual position in the robot's base coordinate system. This error is likely to occur due to inaccurately calibrated cameras and due to errors in the estimated joint angles, even though the calibration procedure was performed correctly. We have calibrated the cameras using the calibration chessboard, which was erected before the cameras at different positions with approximately 45 degrees tilt. The computer vision library OpenCV has then built-in support for a chessboard as a calibration pattern. A scene view is formed by projecting 3D points into the image plane using a perspective transformation:

$$m' = A \cdot \left[R \middle| t \right] \cdot M' \tag{17}$$

or

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{18}$$

where (x,y,z) are the coordinates of a 3D point in the world coordinate space. (u,v) are the coordinates of the projection point in pixels. A is called a camera matrix, or a matrix of intrinsic parameters. (c_x, c_y) is a principal point (that is usually at the image center), and f_x, f_y are the focal lengths expressed in pixel-related units. The matrix of intrinsic parameters does not depend on the scene viewed and, once estimated, can be re-used as long as the focal length is fixed. The joint rotation-translation matrix $[R|t]$ is called a matrix of extrinsic parameters. It is used to describe the camera motion around a static scene, or vice versa, rigid motion of an object in front of still camera. That is, $[R|t]$ translates coordinates of a point (x,y,z) to some coordinate system, fixed with respect to the camera [6].

The second error is from Gaussian regression that of course doesn't generalize absolutely accurately and so causes the average error of 8 mm, which is quite acceptable. Camera error was eliminated by directly using the estimated object positions to train the Gaussian processes. At the end of every movement we put a small spherical object into the robotic hand to determine the position of the object at each end point, as the cameras see it (Figure 7). Since stereo vision was used also to estimate the object position during movement execution, the errors in training object positions and the errors in the

current object position cancel each other out. This leaves only the noise of the cameras and Gaussian process regression error, which is small enough for the practical implementation of the grasping task.

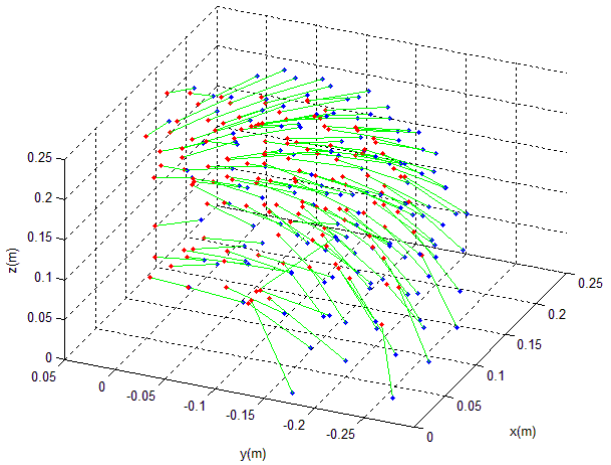

Figure 7. Blue dots are original end-points of all trajectories and red ones are the same end-points as stereo vision sees them. A green line illustrates the shift of end-points.

CONCLUSION

This paper describes the application of Gaussian process regression in combination with dynamic movement primitives to realize reaching and grasping using active vision on a humanoid robot. We demonstrated that good results can be achieved with the proposed approach as long as a sufficient amount of training data is available. To reduce the number of training trajectories, we could reduce the number of trajectories that were obtained by guiding. The robot could then gradually acquire more training data while reaching (or grasping) for objects at different positions. With this approach the robot immediately begins to execute the task and then improves with time, as more and more data become available.

REFERENCES

[1] C. E. Rasmussen and C. Williams, "Gaussian Processes for Machine Learning", Cambridge, MA: MIT Press, 2006.

[2] A. J. Ijspeert, J. Nakanishi, and S. Schaal, "Movement imitation with nonlinear dynamical systems in humanoid robots", in Proc. IEEE Int. Conf. Robotics and Automation, Washington, DC, 2002, pp. 1398–1403.

[3] M. N. Gibbs, "Bayesian Gaussian Processes for Regression and Classification", Ph.D. Thesis, Cambridge University, Cambridge, 1997.

[4] S. Schaal, P. Mohajerian, and A. Ijspeert, "Dynamics systems vs. optimal control – a unifying view", Progress in Brain Research, vol. 165, no. 6, pp. 425–445, 2007.

[5] A. Gams, A. Ude, "Generalization of Example Movements with Dynamic Systems", 9th IEEE-RAS International Conference on Humanoid Robots, 2009.

[6] M. Wilczkowiak, E. Boyer, P. Sturm "Camera Calibration and 3D Reconstruction from Single Images Using Parallelepipeds", Movi–Gravir–Inria Rhône-Alpes, 655 Avenue de l'Europe, 38330 Montbonnot, France