Model-based and Model-free Reinforcement Learning for Visual Servoing

Amir massoud Farahmand, Azad Shademan, Martin Jägersand, and Csaba Szepesvári

***Abstract*— To address the difficulty of designing a controller for complex visual-servoing tasks, two learning-based uncalibrated approaches are introduced. The first method starts by building an estimated model for the visual-motor forward kinematic of the vision-robot system by a locally linear regression method. Afterwards, it uses a reinforcement learning method named Regularized Fitted Q-Iteration to find a controller (i.e. policy) for the system (model-based RL). The second method directly uses samples coming from the robot without building any intermediate model (model-free RL). The simulation results show that both methods perform comparably well despite not having any a priori knowledge about the robot.**

I. INTRODUCTION

Visual-servoing is the task of minimizing a visually-specified objective by giving appropriate control commands to a robot (See [1], [2], and [3] for tutorials on visual servoing). The usual practice for visual-servoing is designing a controller that stabilizes a dynamical system defined by visual features. This is often done by a linear controller that uses the visual-motor Jacobian of the robot-camera system.

There are at least two important drawback to the traditional approach. First, the analytic Jacobian depends on the calibration of both the robot and the camera, and their relative pose, so it cannot be computed without calibration data. For example consider the case that the relative position of the camera to the robot changes, or the situation where the servoing objective is not defined by the end-effector frame but by some other part of the robot, e.g. the middle of a link between two joints, or a grasped object.

The second problem is that the usual locally-defined linear controllers do not benefit from global dynamical knowledge of the system because they only use local information summarized in the visual-motor Jacobian. Hence, it is quite possible that relying on the local information might not lead to a globally optimal performance.

These two problems have been partially addressed previously. Several researchers have developed methods for estimating a visual-motor model of the robot (i.e. pose-indexed varying Jacobian) online [4][5][6][7]. However, most of these research have been only focused on estimating locally-valid models, and keeping one such at a time. Although local models can be used to design locally stabilizing controllers, those models are not natural for global planning and controlling approaches that usually need the global knowledge of the system's model.

Visual servoing has been extensively studied with different control schemes such as position-based controllers [8], image-based controllers ([9], [10]), or hybrid controllers [11]. These schemes can be either calibrated, where the robot or the camera are calibrated, or uncalibrated, where the model of the system is a priori unknown. Uncalibrated visual servoing can be considered as a solved problem for local tasks with non-redundant manipulators. However, in uncalibrated systems there are still theoretical and practical implications for global visual servoing due to nonlinear model estimation that make visual servoing challenging.

In this paper, we address both problems in a unified approach. We use kernel-based locally linear regression for estimating the visual-motor forward kinematic model of the robot (Section III). This globally-valid model can be used for designing a local controller and/or global planner. Afterwards, Regularized Fitted Q-Iteration (RFQI) [12] will be introduced that finds a close to optimal solution for the learning/planning problem (Section IV). This recently proposed nonparametric reinforcement learning (RL) method uses joint values data and a reward signal to find a policy that maximizes an objective functional. By appropriately designing the reward signal, it can find an optimal policy (i.e. controller) for the robot. RQFI can be used in both model-based or model-free approaches.

RQFI can be used in both calibrated and uncalibrated scenarios. In the calibrated scenario, RQFI perform as a global planner. For uncalibrated systems, we can either use RFQI in a model-based or model-free approaches. In the model-based approach, we first estimate a model of the robotic system and then use it for planning. In the model-free approach, we directly use data to find a close to optimal policy (Section V).

II. PROBLEM SETUP

Consider a robotic manipulator that consists of a series of revolute and prismatic joints characterized by joint angles or displacement $S \in \mathbb{R}^d$ and a vision system that observes the environment.[1] The vision system can observe the robot from a fixed position (eye-to-hand), can be attached to the end-effector (eye-in-hand), or any other fixed position on the robot. By changing joint variables, the robot moves and the image projected on each camera changes. By defining some

This research was partially supported by iCORE, NSERC, and the Alberta Ingenuity Fund.

All authors are with the Department of Computing Science, University of Alberta, Canada. {amir, azad, jag, szepesva}@cs.ualberta.ca

Csaba Szepesvári is on leave from MTA SZTAKI.

[1] We use S instead of more conventional notation q for describing joint variables to avoid confusion with Q, which denotes the action-value function in reinforcement learning. See Section IV.

feature points on camera images, we can relate the position of the joint variables to the position of those feature points. If we denote $x_i \in \mathbb{R}$ as the value of one of the visual features, there is a relation between S and x such as $x_i = f_i(S)$. Depending on the way we define the feature, f_i would be related to the kinematic model of the robot and the camera model. Note that this function is different for image-based or position-based visual servoing, but the essence of it, which relates feature variables to the joint variables, is the same.

Generalizing the previous formalism to several, say m features, the feature vector $X \in \mathbb{R}^m$ and visual-motor function $F(S)$ are defined as follows

$$X := [x_1; x_2; ...; x_m]_{m\times 1}$$
$$F := [f_1(S); f_2(S); ...; f_m(S)]_{m\times 1}$$
$$X = F(S).$$

Let $S(t)$ be a function of time; the dynamics of feature points are

$$\frac{dX}{dt} = J(S)\frac{dS}{dt},$$

where $J(S) = \partial F(S)/\partial S$ is the visual-motor Jacobian. To perform image-based tasks like regulating feature points to a specified position, one can design a local controller, design a complicated nonlinear controller, or solve a global planning problem that explicitly optimizes a cost functional. In general, the visual-motor function F is not known for an arbitrary robot-camera system. One can try to estimate F itself or its Jacobian $J(S)$. Our model-based RL approach uses an estimate of F (and not F itself, which is assumed to be unavailable) to design an appropriate controller.

III. Forward Kinematic Estimation

A. Locally Linear Regression for Model Estimation

To estimate the visual-motor forward kinematic model of the robot, we use *locally linear regression* method. This nonparametric method fits the weighted best hyperplane in the neighborhood of the query point S and returns $\hat{F}(S)$ (See Section 5.4 of [13]). See [14] and [15] for similar approaches.

Suppose we have sets of $\{S_l\}$ and $\{x_i(l)\}$ $(i = 1, \cdots, m)$ for $l = 1, \cdots, t$ that are collected by the robot by time t. In the locally linear regression, we find a_0^i and a_1^i such that it minimizes the following cost function.

$$\sum_{l=1}^{t} w_l(S)(x_i(l) - (a_0^i + a_1^i(S_l - S))^2.$$

where $w_l(S)$ is the kernel used for local regression method and is defined $w_l(s) = K_m((S - S_l)/h)$ and K_m is a kernel such a Gaussian or Epanechnikov kernel and h is its bandwidth. Solving this optimization problem, which is defined for any S, the estimated regressor would be

$$\hat{x}_i(S) = \sum_{l=1}^{t} b_l(S)x_i(l)$$

where $b(S)^T = (b_1(S), \cdots, b_t(S))$, and $b(S)^T = e_1^T(\mathbf{S_s}^T W_s \mathbf{S_s})^{-1}\mathbf{S_s}^T W_S$, $e_1 = (1, 0, \cdots, 0)^T$,

$$\mathbf{S_s} = \begin{pmatrix} 1 & S_1 - S \\ 1 & S_2 - S \\ \vdots & \vdots \\ 1 & S_t - S \end{pmatrix}$$

and W_s is a $t \times t$ diagonal matrix with $K_m((S_i - S)/h)$ at its (i, i)th element. The estimated model is $\hat{F}(S) = [\hat{x}_1(S); \cdots; \hat{x}_m(S)]$.

B. Model Selection

Although the performance of locally linear regression is not very sensitive to the choice of kernel (K_m), the choice of bandwidth (h) has a significant effect on the convergence rate of the method [13]. In general, a larger number of data points necessitates small bandwidth and vice versa. Based on the convergence rate bounds, one may choose a fixed schedule for bandwidth depending on the number of samples and the dimension of the problem, so that the rate would be optimal. There are, however, some constants in the optimum bandwidth size that are not known a priori. Thus, following a fixed schedule does not lead to the best possible results. A more practical solution is using a *cross-validation* method to choose the right bandwidth.

In this paper, we use a modified k-fold cross-validation to handle dependent data efficiently. Note that in practice $\{S_l\}$ are not i.i.d. samples from the joint space, but are coming from a stochastic process with temporal dependence. Two random variables S_{t_1} and S_{t_2} have strong dependence when t_1 and t_2 are close, and are almost independent when they are far apart. Usually this dependence can be described by the mixing property of the stochastic process.

The modified cross-validation starts by partitioning the data to k disjoint set as is usual in k-fold cross-validation. Let us denote them as $\bar{S}_0, \bar{S}_1, \cdots, \bar{S}_{k-1}$ where $\bar{S}_i = \{s_{i\frac{t}{k}+1}, s_{i\frac{t}{k}+2}, \cdots, s_{i\frac{t}{k}+\frac{t}{k}}\}$. (for simplicity, assume that k divides t.) The usual k-fold cross-validation selects partition $\bar{S}_i$ $(i = 1, \cdots, k)$ and removes it from the data set, use the remaining partitions to train the model, and evaluate it on $\bar{S}_i$ and then repeat this procedure for all $\bar{S}_i$. The empirical performance would be the average performance on all blocks.

Our method tries to reduce the dependence between the training data and test data by removing adjacent partitions of the testing partition. In other words, when we are evaluating the performance by data from $\bar{S}_i$, the training set would be $\{\bar{S}_0, \cdots, \bar{S}_{i-2}, \bar{S}_{i+2}, \cdots, \bar{S}_k\}$. If the data stream is weakly coupled, this ensures that there is little dependence between the training set and the testing set, and therefore the performance estimation would be more precise.

IV. Regularized Fitted Q-Iteration

A key to success of any reinforcement learning method that deals with high-dimensional state spaces is having a flexible function approximator. Choosing a function approximator that cannot adapt to the difficulty of the problem, such

as a one that uses a fixed number of basis functions, leads to low-performance systems. This calls for methods that can change the complexity of their function approximators according to the number of data point and the intrinsic regularities of the problem such as its smoothness or sparsity. There are several suggested approach to address this problem such as basis adaptation [16], incremental basis construction [17], tree-based methods [18] and the GPTD algorithm which uses Gaussian Processes Regression [19].

Our approach of choice is regularization which has been proven an effective and powerful tool in machine learning and in particular in supervised learning. The main idea is to consider learning as an optimization problem where one minimizes the sum of an empirical error term and a complexity penalty, the regularizer, which penalizes more complex solutions. The tradeoff between the empirical error term and the penalty term is controlled by a single numerical value: the regularization coefficient. This way the problem of model-selection is reduced to the problem of choosing a single numerical value. When the parameter is chosen in an appropriate way (based on the data or by complexity regularization), the resulting procedure is known to adapt to the complexity of the target function automatically, converging almost as fast as if the model was known beforehand (e.g. See Chapter 21 of [20]). Regularization is an example of mathematical formalization of the Occam's razor principle.

In this paper, we consider the Regularized Fitted Q-Iteration (RFQI). It is an instance of Fitted Q-Iteration ([18]) that uses Reproducing Kernel Hilbert Space (RKHS)-based regularized regression in its inner loop. This way we hope to bring the strength of a powerful supervised learning algorithm to the planning problem. See [12] for more information about RFQI and more precise statements about its theoretical guarantees. It is noteworthy to mention that there have been a few attempts to use regularization in reinforcement learning such as [21] and [22]. However, the former works only if the environment is deterministic and neither of them analyzes the performance of their algorithm.

A. Reinforcement Learning Background and Notations

We briefly review a few concepts and notations from analysis and Markovian Decision Processes (MDP). We refer the reader to [23] and [12] for further details. Reader who is not interested in rigorous definitions or is already familiar with them may just skip to Section IV-B.

For a measurable space with domain $\mathcal{S}$, we let $\mathcal{M}(\mathcal{S})$ denote the set of probability measures over $\mathcal{S}$. For $p \geq 1$, a measure $\nu \in \mathcal{M}(\mathcal{S})$, and a measurable function $f : \mathcal{S} \to \mathbb{R}$, we let $\|f\|_{p,\nu}$ denote the $L^p(\nu)$-norm of f defined as $\|f\|_{p,\nu}^p = \int |f(s)|^p \nu(ds)$. We shall also write $\|f\|_\nu$ to denote the $L^2(\nu)$-norm of f. We denote the space of bounded measurable functions with domain $\mathcal{S}$ by $B(\mathcal{S})$, and the space of measurable functions with bound $0 < K < \infty$ by $B(\mathcal{S}; K)$.

A finite-action discounted MDP is defined by a quintuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is the (possibly infinite) *state space*, $\mathcal{A} = \{a_1, a_2, \ldots, a_M\}$ is the finite set of *actions*, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{M}(\mathcal{S})$ is the *transition probability kernel*, $P(\cdot|s,a)$ defining the next-state distribution upon taking action a in state s, $R(\cdot|s,a)$ gives the corresponding distribution of *immediate rewards*, and $\gamma \in (0,1)$ is the discount factor. We make the following assumptions on the MDP:

Assumption A1 (MDP Regularity) $\mathcal{S}$ is a compact subset of the d-dimensional Euclidean space. We assume the expected immediate rewards $r(s,a) = \int r R(dr|s,a)$ are bounded by $R_{\max}$: $\|r\|_\infty \leq R_{\max}$.

A stationary Markov policy $\pi : \mathcal{S} \to \mathcal{M}(\mathcal{A})$ is defined as a time-independent (measurable) mapping from the current state s to a distribution over the set of actions $\pi(\cdot|s)$. A policy is deterministic if the probability distribution concentrates on a single action for all states. Deterministic stationary Markov policies will be identified with mappings from states to actions $\pi : \mathcal{S} \to \mathcal{A}$. In the rest of this paper, we use the term policy to refer to stationary Markov policies.

The value of a policy π when it is started from a state s is defined as the total expected discounted reward that is encountered while the policy is executed, i.e. $V^\pi(s) = \mathbb{E}_\pi\left[\sum_{t=0}^\infty \gamma^t R_t | \, S_0 = s\right]$. Here R_t denotes the reward received at time step t; $R_t \sim R(\cdot|S_t, A_t)$ and S_t evolves according to $S_{t+1} \sim P(\cdot|S_t, A_t)$ where A_t is sampled from the distribution assigned to the past observations by π. For a policy π, $A_t \sim \pi(\cdot|S_t)$, while if π is deterministic then we write $A_t = \pi(S_t)$. The function V^π is also called the state-value function of policy π. Closely related to the state-value functions are the action-value functions, defined by $Q^\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^\infty \gamma^t R_t | \, S_0 = s, A_0 = a\right]$. It is easy to see that for any policy π, the functions V^π and Q^π are bounded by $R_{\max}/(1-\gamma)$.

Given an MDP, the goal is to find a policy that attains the best possible values, $V^*(s) = \sup_\pi V^\pi(s)$, for all states $s \in \mathcal{S}$. Function V^* is called the optimal value function. A policy is called optimal if it attains the optimal values $V^*(s)$ for *any* state $s \in \mathcal{S}$, i.e., if $V^\pi(s) = V^*(s)$ for all $s \in \mathcal{S}$. In order to characterize optimal policies it will be useful to define the optimal action-value function, $Q^*(s,a)$: $Q^*(s,a) = \sup_\pi Q^\pi(s,a)$. Further, we say that a deterministic policy π is *greedy* w.r.t. an action-value function $Q \in B(\mathcal{S} \times \mathcal{A})$ and write $\pi = \hat{\pi}(\cdot; Q)$, if, for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $\pi(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q(s,a)$. The Bellman optimality operator $T : B(\mathcal{S} \times \mathcal{A}) \to B(\mathcal{S} \times \mathcal{A})$ is defined by $(TQ)(s,a) = r(s,a) + \gamma \int \max_{a' \in \mathcal{A}} Q(s', a') P(ds'|s,a)$. As it is well known, this operator T is a contraction operator w.r.t. the supremum-norm with index γ. Moreover, the optimal action-value function is the unique fixed point of T: $TQ^* = Q^*$.

Throughout the paper $\mathcal{F} \subset \{ f : \mathcal{S} \to \mathbb{R} \}$ will denote some subset of real-valued functions over the state-space $\mathcal{S}$. For convenience, we will treat elements of $\mathcal{F}^M$ as real-valued functions f defined over $\mathcal{S} \times \mathcal{A}$ with the obvious identification $f \equiv (f_1, \ldots, f_M)$, $f(s, a_j) = f_j(s)$, $j = 1, \ldots, M$. The set $\mathcal{F}^M$ will denote the set of admissible functions used in the optimization step of our algorithm.

FittedQ(D,K,Q_0)
// D: samples
// K: number of iterations
// Q_0: Initial action-value function
for $k = 0$ to $K - 1$ **do**
  $Q_{k+1} \leftarrow \text{FitQ}(Q_k, D, k)$
end for
return Q_K

Fig. 1. Generic Fitted Q-Iteration

B. Algorithm

The algorithm studied in this paper is an instance of the generic fitted Q-iteration method, whose pseudo-code is shown in Fig. 1. The algorithm attempts to approximate the optimal action-value function Q^* and mimics value iteration. Since computing the Bellman operator applied to the last iterate at any point involves evaluating a high-dimensional integral we use a Monte-Carlo approximation together with a regression procedure. For this purpose a set of samples D is generated: $D = \{(S_1, A_1, R_1, S'_1), \ldots, (S_N, A_N, R_N, S'_N)\}$. In this paper for the sake of simplifying the analysis we assume that the actions and next states are generated by some fixed stochastic stationary policy π_b: $A_t \sim \pi_b(\cdot|S_t)$, $S'_t \sim P(\cdot|S_t, A_t)$, $R_t \sim R(\cdot|S_t, A_t)$.

The state-marginal of ν is denoted by $\nu_{\mathcal{S}}$. We assume that ν is a strictly positive measure, i.e., its support is $\mathcal{S} \times \mathcal{A}$. Intuitively, this ensures that the samples cover all state-action pairs. In particular for this we must have that $\pi_{b0} \stackrel{\text{def}}{=} \min_{a\in\mathcal{A}} \inf_{s\in\mathcal{S}} \pi_b(a|s) > 0$.

The fitting procedure that we study in this paper is penalized least-squares. Assuming that in the k^{th} iteration we use samples with index $N_k \le i < N_k + M_k = N_{k+1} - 1$, the $(k+1)^{\text{th}}$ iterate is obtained by

$$Q_{k+1} = \operatorname*{argmin}_{Q\in\mathcal{F}^M} \frac{1}{M_k} \sum_{i=N_k}^{N_k+M_k-1} \big[R_i + \gamma \max_{a'\in\mathcal{A}} Q_k(S'_i, a') - Q(S_i, A_i)\big]^2 + \lambda \text{Pen}(Q), \tag{1}$$

where $\text{Pen}(Q)$ is a penalty term and $\lambda > 0$ is the regularization coefficient.[2] The first term is the sample-based least-squares error of using $Q(S_i, A_i)$ to predict $R_i + \gamma \max_{a'\in\mathcal{A}} Q_k(S'_i, a')$ at (S_i, A_i). This term is the empirical counterpart to the loss $L_k(Q) = \mathbb{E}\big[(R_i + \gamma \max_{a'\in\mathcal{A}} Q_k(S'_i, a') - Q(S_i, A_i))^2\big]$. The minimizer of this loss function is the regression function $\mathbb{E}\left[R_i + \gamma \max_{a'\in\mathcal{A}} Q_k(S'_i, a') \,|\, S_i = s, A_i = a\right] = (TQ_k)(s, a)$. As the number of samples grows to infinity the empirical loss converges to L_k and we wold like the iterate Q_{k+1} to converge to TQ_k. To do so, one needs to prevent overfitting or over-smoothing. This is the job of the second term on the right hand side of (1). This term regulates how complex solutions are acceptable in an implicit manner. Choosing a larger λ means searching in a smaller space of functions and vice versa.

When $\mathcal{F}^M$ is a Sobolev-space[3] and $\text{Pen}(Q)$ is the corresponding Sobolev-space norm (the squared norm of the generalized partial derivatives of Q), this optimization leads to thin-plane spline estimates, popular in the non-parametric statistics literature [20].

When searching for a solution in general the order of smoothness is unknown. Further, the optimal choice of the regularization coefficient would depend on the target function. The approach taken in regression can be followed here, too: Try different smoothness orders (this corresponds to different penalty terms) with different regularization coefficients and choose between them using a hold-out set. This leads to estimates whose rate of convergence has the optimal order and scales with the actual roughness, $\text{Pen}(TQ_k)$.

Optimizing over a Sobolev-space is a particular case of optimization in a reproducing kernel Hilbert space (RKHS). Thus, more generally, we may start with a Mercer kernel function k, and set $\text{Pen}(Q)$ to be the norm of Q in $\mathcal{H}$, the RKHS underlying k [24]. This way we obtain

$$Q_{k+1} = \operatorname*{argmin}_{Q\in\mathcal{H}} \frac{1}{M_k} \sum_{i=N_k}^{N_k+M_k-1} \big[R_i + \gamma \max_{a'\in\mathcal{A}} Q_k(S'_i, a') - Q(S_i, A_i)\big]^2 + \lambda \|Q\|^2_{\mathcal{H}}. \tag{2}$$

According to the Representer Theorem (e.g., see [24]), every solution to Eq. (2) is the sum of kernels centered on the observed samples: i.e., $Q(s, a) = \sum_{i=N_K}^{N_k+M_k-1} \alpha_{i-N_k+1} \mathrm{k}\big((S_i, A_i), (s, a)\big)$, where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{M_k})^\top$ are the coefficient that must be determined. Let us assume that Q_k was obtained previously in a similar form: $Q_k(x, a) = \sum_{i=N_{k-1}}^{N_{k-1}+M_{k-1}} \alpha^{(k)}_{i-N_{k-1}+1} \mathrm{k}\big((S_i, A_i), (x, a)\big)$, and let us collect the coefficients into a vector $\boldsymbol{\alpha}^{(k)} \in \mathbb{R}^{M_{k-1}}$. Replacing Q in Eq. (2) by its expansion and using RKHS properties, we get

$$\boldsymbol{\alpha}^{(k+1)} = \operatorname*{argmin}_{\boldsymbol{\alpha}\in\mathbb{R}^{M_k}} \frac{1}{M_k} \left\| \boldsymbol{r} + \gamma \boldsymbol{K}^+ \boldsymbol{\alpha}^{(k)} - \boldsymbol{K}\boldsymbol{\alpha} \right\|^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}, \tag{3}$$

with $\boldsymbol{K} \in \mathbb{R}^{M_k \times M_k}$, $\boldsymbol{K}^+ \in \mathbb{R}^{M_k \times M_{k-1}}$,

$$[\boldsymbol{K}]_{ij} = \mathrm{k}\big((S_{i-1+N_k}, A_{i-1+N_k}), (S_{j-1+N_k}, A_{j-1+N_k})\big),$$
$$[\boldsymbol{K}^+]_{ij} = \mathrm{k}\big((S'_{i-1+N_k}, A^{(k)}_{i-1+N_k}), (S_{j-1+N_{k-1}}, A_{j-1+N_{k-1}})\big),$$

where $A^{(k)}_j = \operatorname{argmax}_{a\in\mathcal{A}} Q_k(S'_j, a)$, and $\boldsymbol{r} = (R_{N_k}, \ldots, R_{N_k+M_k-1})^\top$. Solving Eq. (3) for $\boldsymbol{\alpha}$ we obtain $\boldsymbol{\alpha}^{(k+1)} = (\boldsymbol{K} + M_k\lambda \boldsymbol{I})^{-1}(\boldsymbol{r} + \gamma \boldsymbol{K}^+ \boldsymbol{\alpha}^k)$. The computational

[2] Note that in practice one would generate samples whenever needed, i.e., there is no need to generate and store all the samples. However, it is also possible to reuse the samples if sample generation is expensive. In such a case the analysis needs to be changed slightly.

[3] Sobolev-spaces generalize Hölder spaces by allowing functions which are only almost everywhere differentiable. Thus, they can be useful for control problems where value-functions often have ridges.

complexity of iteration k with a straightforward implementation is $O(M_k^3)$ as it involves the inversion of a matrix. This method is not computationally cheap. The computational problem is a common problem to all RKHS-based methods. There are approaches to reduce the computational complexity of these methods. For example, one may use sparsification method of [25]. This is the method that we will use in our experiments.

C. Theoretical Guarantees

We briefly mention the theoretical results for the proposed RFQI. See [12] for more information.

In this section we assume that Q_{k+1} is obtained by solving the RKHS regularization problem of Eq. (2). The result is for the case when $\mathcal{S} = [0,1]^d$, but can be generalized to other compact spaces with "regular" boundaries relatively easily. In the following theorem, we assume that $S_t \sim \nu_{\mathcal{S}}$ is an i.i.d. sequence and $A_t \sim \pi_b(\cdot|S_t)$ for some π_b that selects all actions with non-zero probability. This assumption basically means that we have access to the generative model, and is the case of *planning*. However, this assumption is not essential, and we just use it to simplify the proof. We can extend this result to the case that the agent observes a single trajectory generated by a fixed policy by having appropriate mixing condition on the MDP, i.e. learning case (see [26]).

Theorem 1 (L^2-bound): Assume that $\mathcal{S} = [0,1]^d$, k $\in$ Lip$^*(\alpha, C(\mathcal{S},\mathcal{S}))$, $\alpha > d$, and Q_k is such that $TQ_k \in \mathcal{H}(= \mathcal{H}_{\mathrm{k}})$.[4] Furthermore, (for the sake of simplicity) assume that all functions involved in the regression problem (the reward function, Q_k, and the result of the optimization problem (Q_{k+1}) are bounded by some constant $L > 0$.[5] Let Q_{k+1} be the solution of (2) with some $\lambda > 0$. Furthermore, assume that we use the same number of samples in each iteration: $M_1 = M_2 = \ldots = M_K$. Let π_K be greedy w.r.t. the K^{th} iterate, Q_K. Define $E_0 = \|\varepsilon_{-1}\|_\infty$ and let $B = \max_{0\le k\le K} \left\|T^k Q_0\right\|_{\mathcal{H}}^2$. Then, for any $\delta > 0$ with probability at least $1-\delta$,

$$\|V^* - V^{\pi_K}\|_\rho \le$$
$$2\left[\frac{1}{1-\gamma} + \frac{\gamma}{(1-\gamma)^2}\right]\gamma^{K/2}E_0 +$$
$$2\left[\frac{(C^1_{\rho,\nu})^{1/2}}{1-\gamma} + \frac{\gamma(C^2_{\rho,\nu})^{1/2}}{(1-\gamma)^2}\right]\left[c_1\lambda B + \frac{c_2 L^4}{M_1\lambda^{d/\alpha}} + \frac{c_3\log(1/\delta)}{M_1 L^4}\right]^{1/2}$$

for some constants $C^1_{\rho,\nu}$ and $C^2_{\rho,\nu}$ that only depend on ρ, ν, γ and the MDP dynamics and for some universal constants $c_1, c_2, c_3 > 0$.

Note that by choosing $\lambda = cM_1^{-1/(1+d/\alpha)}$ the second term is made converging to zero with $M_1 \to \infty$ at a rate $O(M_1^{-1/(2(1+d/\alpha))})$, corresponding to the optimal regression rate for smoothness order α. On the other hand, by choosing K larger one can make the first term as small as desired. Note that the cost of executing the procedure is $O(KM_1^3)$. Then given a computational budget $\mathcal{B}$, one may optimize K and M_1 to get the best possible performance. Clearly, it suffices to choose $K = \log(\mathcal{B})$, hence given the budget $\mathcal{B}$ the performance will be $\tilde{O}(\mathcal{B}^{-1/(6(1+d/\alpha))})$.

[4]For the definition of the generalized Lipschitz space Lip*, see [27].

[5]When this does not hold, a truncation argument is needed, but the result would essentially be left unchanged.

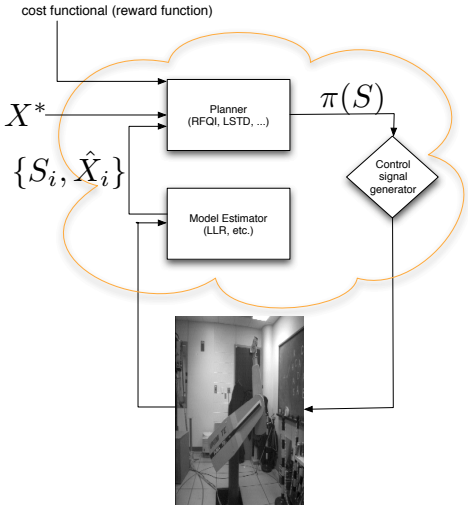

Fig. 2. Model-based Reinforcement Learning

V. Model-based and Model-free Reinforcement Learning for Visual-Servoing

In the previous two sections, we introduced a model estimation method (Section III) and RFQI (Section IV). Depending on whether we use the model estimation method or not, we will have two different RL approaches for designing a close to optimal policy (i.e. controller) for the robot. These modules and their relationship are depicted in Fig. 2.

The first approach, which we call model-free reinforcement learning, works by directly collecting $D = \{(S_1, A_1, R_1, S_2), \ldots, (S_n, A_n, R_n, S_{n+1})\}$ where S_i are joint parameters at time step i, A_i are the action commands given to the robot at the same time, and R_i are reward (or cost) function depending on the state transition and the observed visual features. In other words, R is a function of $X(= F(S))$. This reward function is determined by the designer. Afterwards, one applies RFQI described in Section IV to find an approximately optimal policy $\pi : \mathcal{S} \to \mathcal{A}$. In general, action space $\mathcal{A}$ can be low-level commands (e.g. joint velocity) or higher-level commands (e.g. execute grasp command for a hand, move toward the goal, etc.). In our experiments, we just focus on low-level commands.

The second approach is model-based reinforcement learning where we use $D_m = \{(S_1, X_1), \ldots, (S_n, X_n)\}$ to build an estimate $\hat{F}$ for visual-motor forward kinematic model of the robot as described in Section III. We use this estimated model $\hat{F}$ to generate $D = \{(S_1, A_1, \hat{R}_1, S'_1), \ldots, (S_n, A_n, \hat{R}_n, S'_n)\}$ set used for training in RFQI. Note that in this case, we do not directly gather data from the robot itself, but instead use the estimated model. Therefore, we can randomly select any S configuration and predict what would be observed $\hat{X}$, and based on that, estimate $\hat{R}$. Fig. 2 is a schematic for this method. It is important to mention that even though the name of the

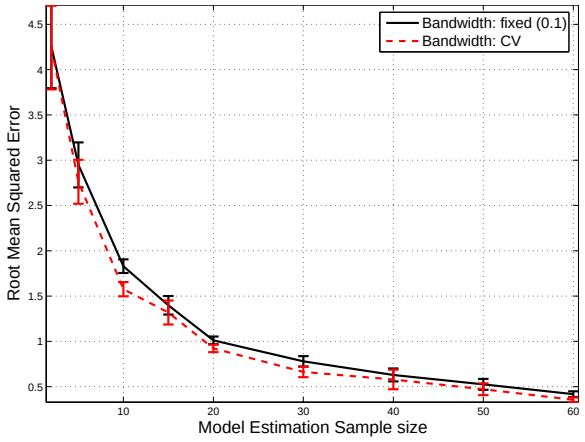

Fig. 3. Visual-motor kinematic model estimation

method is *model-based* RL, it does not use a priori model of the robot, but it estimates a model using data and then uses the estimated model for planning. This scheme is similar to Dyna-style planning methods common in reinforcement learning [28].

VI. Experiment

In this section, we first study the behavior of locally linear regression model described in Section III. After that, we study the performance of model-based and model-free RL, and study their behavior when we change regularization coefficient λ and kernel parameter σ. Our experiments are performed in MATLAB using Corke's Robotics Toolbox [29] and the Epipolar Geometry Toolbox [30].

For all these experiments, we use the *Puma 560* model with an eye-to-hand configuration. To define our visual features, we use a stationary stereo rig setup. The visual features are the projections of the end-effector's position into the image space of each of those cameras. Therefore, the feature space is four dimensional ($X \in \mathbb{R}^4$). The position of the end-effector has only three dimensions, therefore we used the first three joints of the *Puma* arm and $S \in \mathbb{R}^3$.

For the first experiment, we study the performance of visual-motor kinematic model estimation described in Section III. We sample data coming from a randomly generated smooth trajectory in joint space and use the suggested locally linear regression method with a Gaussian kernel to estimate the model. Afterwards, we compare the accuracy of estimation. (by evaluating the error at a 1000 randomly selected points in the joint space.) We repeat this procedure for different number of samples. The results is shown in Fig 3. As expected, the error decreases when the number of samples increases. Moreover, the effect of cross-validation is observable. It shows that adaptively choosing the bandwidth of kernels increases the model estimation accuracy.[6]

For the second experiment, we compare the performance of a model-based RL, model-free RL with a conventional linear controller. The problem is defined as visual set-point regulation. The visual features are the coordinate of end-effector in the image space of the two cameras ($X \in \mathbb{R}^4$) and the set-point is $X^* = [0\ 0\ 0\ 0]^T$. The discrete-time control signal is $u(t) \in \{-1, +1\}^3$ with time step of 0.02sec. We desire to minimize the number of steps to the goal from an arbitrary initial position. We formulate this problem as solving a discounted MDP with $\gamma = 0.95$ where we assign $R_t = -1$ whenever $||X_{t+1} - X^*||^2 > 10$ and $R_t = \frac{1}{1+||X_{t+1}-X^*||^2}$ otherwise. This encourages the robot to move toward the goal as soon as possible. For all experiments, we use i.i.d. samples from $S \in U((-1, 1)^3)$ for both training and policy evaluation.

In the case of model-free RL, the robot itself takes random (i.i.d.) samples from the environment (or likewise we assume that we have access to the *exact* generative model of the robot), and these data are used for RFQI.[7] In the case of model-based RL, the robot follows a smooth trajectory (as in the previous experiment) and build an estimated model $\hat{F}$ of visual-servoing forward kinematic. Then it gets i.i.d. *virtual* samples from this estimated model to provide data for RFQI. For our experiments, we generate twice the number of samples that is used to build $\hat{F}$, e.g. if we use 1000 samples to build the estimated visual-motor kinematic model, we generate 2000 samples for RFQI. Aside computational issues, the number of true samples would be the right measure of learning difficulty. Actions in the training set, for both methods, are generated uniformly. In our experiments, we re-use the same data in all iterations. The iterations are limited to $K = 2000$, but if the action-value function Q_{k+1} is very close to Q_k (empirical norm smaller than 10^{-5}), we stop the iteration. For this experiment, we use a Gaussian kernel $\mathrm{k}((q_1, u_1), (q_2, u_2)) = \exp(-\frac{||q_1-q_2||^2}{2\sigma^2})\mathbb{I}_{\{u_1=u_2\}}$ with kernel parameter $\sigma^2 = 0.3$. Also the regularization coefficient is fixed at $\lambda = 0.01$.

For the conventional controller, we use the exact local model of visual-motor kinematic to design the controller. Defining the visual-motor Jacobian as $J(S) = \frac{\partial f(S)}{\partial S}$, the control signal would be $u(t) = \dot{q}(t) = -J^\dagger(S)$ where $J^\dagger(S)$ is the pseudo-inverse of the Jacobian at $S(t)$. So here, the conventional controller does not use any estimated model. To make these two types of controller (RL-based and linear) more comparable, we need to make sure both use the same amount of control signal's power. The RL policy selects the control signal $u(t) \in \{-1, +1\}^3$. Therefore, the power of the signal is 3. For the conventional controller (which is a linear controller), we normalize its output so that its power become 3. When the error is large, this modification prevents the controller to spend so much power. When the error is very small, it makes the controller act like a switching controller.

Fig. 4 compares the performance of the model-based and model-free RL methods to the performance of a conventional linear controller when the number of *true samples* changes. For this experiment, the number of samples is changing from

[6]The result of this experiment is an average of 10 runs. We add a Gaussian noise with s.d. of 0.05 to the generated trajectories.

[7]In practice, we cannot make a robot to pick i.i.d. samples, but instead we can follow a smooth trajectory. The difference between i.i.d. samples and following a single trajectory depends on the mixing behavior of the corresponding stochastic process.

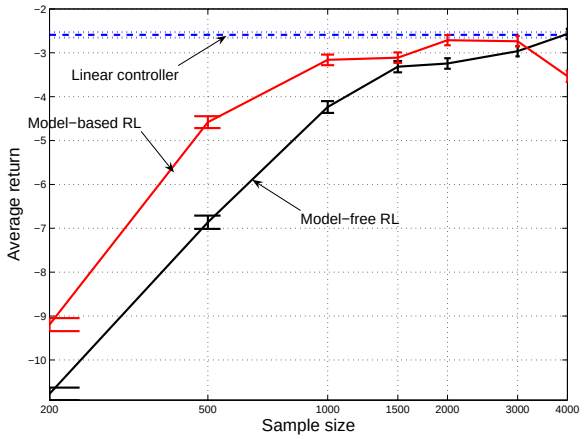

Fig. 4. Model-based RL, Model-free RL, and Linear controller without model selection

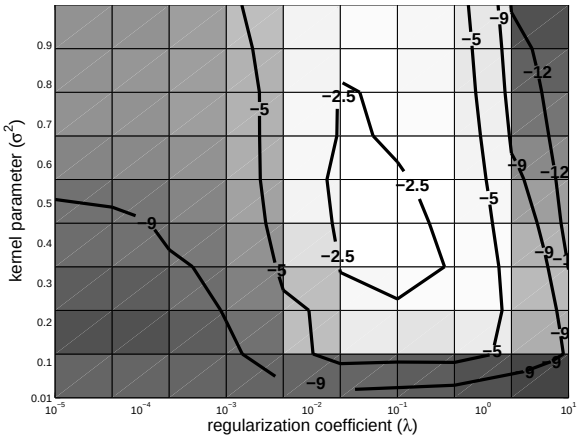

Fig. 5. The effect of changing regularization coefficient (λ) and kernel parameter (σ^2) on the policy's performance.

around 200 to 4000. As mentioned before, we apply RFQI with $\lambda = 0.01$ and $\sigma^2 = 0.3$. Ideally, these value should depend on the data and the difficulty of the problem (e.g. λ should decrease when the number of samples increases), but for now we fix them. To compare the results, we generate 1000 randomly generated initial robot's configurations and evaluate the performance of these controllers starting from each of them, and then take the average of all of them. This determines the performance of any controller. We run the experiment 10 times. The error bars (or dotted interval) shows the standard error around the empirical average

The result shown in Fig. 4 indicates that both RL methods perform well, though their performance is a bit lower than a linear controller that has access to the true Jacobian of the robot. One possible reason is that the action space of our robot is limited to 8 discrete actions. On the other hand, the action space of the linear controller is much richer (it is a point in S^2 - 2-dimensional sphere). The other reason is that we have not selected the best possible model, i.e. λ and σ are not selected optimally.

To study the effect of kernel parameter σ and regularization coefficient λ on the performance of the RFQI, we change these parameters and observe their effects on the performance. We evaluate the performance of $\hat{\pi}(\cdot; Q_K^{(\sigma,\lambda)})$, the greedy policy w.r.t. $Q_K^{(\sigma,\lambda)}$, for several values of σ and λ by a Monte Carlo method with 5000 trajectories starting from a uniformly chosen $S(0)$ initial positions and following the policy. Fig. 5 shows the performance of policies generated by RFQI with varying regularization coefficient and kernel parameter. Lighter regions show higher performance and darker regions show lower performance. The existence of a region where the performance is considerably better than other regions is evident. This region has a moderate values of λ and σ^2. The performance degradation for very small values of λ is an indication of over-fitting. Over-smoothing is also observable for large values of regularization coefficient. Although the performance is less sensitive to the kernel parameter than to the regularization coefficient, poor performance is visible for very small values of σ^2.

Based on the result of previous experiment, it is suggestive to generate several RFQI solutions with different σ and λ, and then select the best model among them. In this paper, our model selection method works by running several trajectories and evaluating the performance of the policy, and then selecting the statistically best one among them. (We use empirical Bernstein race model selection method described in [31].) This needs running the real robot or having an exact model of it. Here, we assumed that we had an exact model of it, but we conjecture that it is possible that we perform the model selection by evaluating policies with the help of the estimated model $\hat{F}$. For this reason, we just report the result of model-free RL with an assumption that the exact model of the robot is available.

In the next experiment, we apply RFQI for 20 models (different λ and σ), and use the model selection to choose one of them. For model selection, we put a maximum limit of 100 trajectory samples for each model and the confidence parameter $\delta = 0.3$. (Refer to [31] for details and definition.) After selecting the best model, we compare it with the conventional controller by evaluating the performance of 1000 paths with randomly generated initial states. We run this experiment 10 times. The results is shown in Fig. 6. For comparison, we also show the performance of model-free RFQI with the same fixed parameters as in the experiment of Fig. 4. It is evident that the model selection improves the performance of the robot. The resulted controller outperforms the linear controller after about 3000 samples. As before, the error bars (or dotted interval) shows the standard error around the empirical average.

VII. Conclusion

We introduced two uncalibrated methods for visual servoing. We observed that both model-based and model-free RL that uses samples in two different ways may perform quite well without having a priori knowledge about the robot. This is important in visual-servoing since in many cases we do not have access to visual-motor kinematic model, but we have access to the robot and can get samples from it. Also we

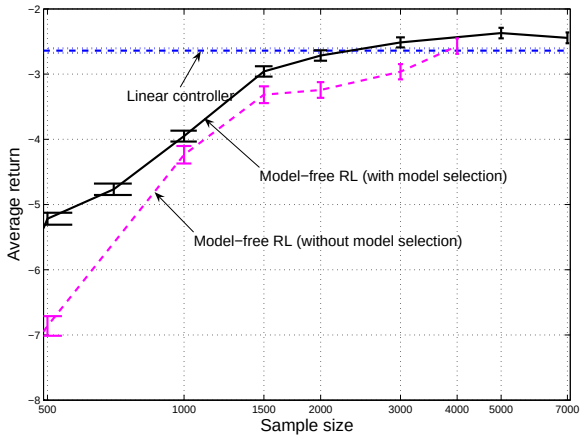

Fig. 6. Comparison of model-free RL with and without model selection with the linear controller

noticed that the performance of the resulted policy depends on selecting the right function space. A regularization-based method like RFQI lets us partially solve this problem by reducing the model selection problem to the selection of the regularization coefficient λ and the kernel parameter σ.

There are, however, many important problems that must be addressed. (1) The first is applying these ideas on a real robot. We have some partial, but promising, results for model estimation on a Whole Arm Manipulator (WAM), but we still need to do more experiments. (2) The second is finding an efficient way for model selecting in the RL setting. One possibility is using the estimated model $\hat{F}$ for evaluating the performance of a policy. This needs more theoretical investigation. (3) Another important issue is studying the conditions where the model-based RFQI may outperform the model-free RFQI. We believe that this would be the case when learning the dynamics of visual-motor system is easier than learning the value function, maybe because of different regularities in these problems. (4) Finally, extending the current RFQI to cope with continuous actions would be important for robotic applications.

REFERENCES

[1] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Trans. Robotics and Automation*, vol. 12, no. 5, pp. 651–670, Oct. 1996.

[2] F. Chaumette and S. Hutchinson, "Visual servo control, part I: Basic approaches," *IEEE Robotics and Automation Magazine*, vol. 13, no. 4, pp. 82–90, Dec. 2006.

[3] ——, "Visual servo control, part II: Advanced approaches," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 109–118, Mar. 2007.

[4] K. Hosoda and M. Asada, "Versatile visual servoing without knowledge of true Jacobian," in *IEEE/RSJ International Conf. Intelligent Robots and Systems (IROS)*, vol. 1, September 1994, pp. 186–193.

[5] M. Jägersand, O. Fuentes, and R. Nelson, "Experimental evaluation of uncalibrated visual servoing for precision manipulation," in *IEEE International Conf. Robotics and Automation (ICRA)*, vol. 4, April 1997, pp. 2874–2880.

[6] H. Sutanto, R. Sharma, and V. Varma, "The role of exploratory movement in visual servoing without calibration," *Robotics and Autonomous Systems*, vol. 23, pp. 153–169, 1998.

[7] J. A. Piepmeier, G. V. McMurray, and H. Lipkin, "Uncalibrated dynamic visual servoing," *IEEE Trans. Robotics and Automation*, vol. 20, no. 1, pp. 143–147, February 2004.

[8] W. J. Wilson, C. C. W. Hulls, and G. S. Bell, "Relative end-effector control using cartesian position based visual servoing," *IEEE Trans. Robotics and Automation*, vol. 12, no. 5, pp. 684–696, October 1996.

[9] L. Weiss, A. Sanderson, and C. Neuman, "Dynamic sensor-based control of robots with visual feedback," *IEEE Journal of Robotics and Automation*, vol. 3, no. 5, pp. 404–417, 1987.

[10] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Trans. Robotics and Automation*, vol. 8, no. 3, pp. 313–326, June 1992.

[11] E. Malis, F. Chaumette, and S. Boudet, "2-1/2-d visual servoing," *IEEE Trans. Robotics and Automation*, vol. 15, no. 2, pp. 238–250, April 1999.

[12] A. M. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor, "Regularized fitted Q-iteration: Application to bounded resource planning," in *Recent Advances in Reinforcement Learning: 8th European Workshop, EWRL 2008, Villeneuve d'Ascq, France, June 30-July 3, 2008, Revised and Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 55–68.

[13] L. Wasserman, *All of Nonparametric Statistics (Springer Texts in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[14] A. M. Farahmand, A. Shademan, and M. Jägersand, "Global visual-motor estimation for uncalibrated visual servoing," in *IEEE/RSJ International Conf. Intelligent Robots and Systems (IROS)*, 2007.

[15] S. Schaal, C. G. Atkeson, and S. Vijayakumar, "real-time robot learning with locally weighted statistical learning," in *international conference on robotics and automation (icra2000)*, 2000.

[16] S. Mannor, I. Menache, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, vol. 134, pp. 215–238, 2005.

[17] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman, "Analyzing feature generation for value-function approximation," in *ICML*, 2007, pp. 737–744.

[18] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[19] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with Gaussian processes," in *ICML '05: Proceedings of the 22nd international conference on Machine learning*. New York, NY, USA: ACM, 2005, pp. 201–208.

[20] L. Györfi, M. Kohler, A. Krzyżak, and H. Walk, *A distribution-free theory of nonparametric regression*. New York: Springer-Verlag, 2002.

[21] T. Jung and D. Polani, "Least squares SVM for least squares TD learning," in *ECAI*, 2006, pp. 499–503.

[22] M. Loth, M. Davy, and P. Preux, "Sparse temporal difference learning using LASSO," in *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

[23] D. P. Bertsekas and S. Shreve, *Stochastic Optimal Control (The Discrete Time Case)*. Academic Press, New York, 1978.

[24] B. Schölkopf and A. Smola, *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.

[25] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least squares algorithm," *IEEE Transaction on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[26] A. Antos, R. Munos, and C. Szepesvári, "Fitted Q-iteration in continuous action-space MDPs," in *Advances in Neural Information Processing Systems 20 (NIPS-2007)*, 2008, (in print).

[27] D.-X. Zhou, "Capacity of reproducing kernel spaces in learning theory," *IEEE Transactions on Information Theory*, vol. 49, pp. 1743–1752, 2003.

[28] R. S. Sutton, C. Szepesvári, A. Geramifard, and M. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," in *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

[29] P. Corke, "A robotics toolbox for MATLAB," *IEEE Robotics and Automation Magazine*, vol. 3, no. 1, pp. 24–32, Mar. 1996.

[30] G. Mariottini and D. Prattichizzo, "EGT: a toolbox for multiple view geometry and visual servoing," *IEEE Robotics and Automation Magazine*, vol. 3, no. 12, December 2005.

[31] V. Mnih, C. Szepesvári, and J.-Y. Audibert, "Empirical bernstein stopping," *Proceedings of the 25th Annual International Conference on Machine Learning (ICML 2008)*, pp. 672–679, 2008.