Trajectory tracking algorithm for autonomous vehicles using adaptive reinforcement learning

Mariano De Paula
Facultad de Ingeniería, Universidad Nacional del Centro de la Provincia de Buenos Aires
INTELYMEC - CIFICEN – UNICEN - CONICET
Olavarría, Argentina
mariano.depaula@fio.unicen.edu.ar

Gerardo G. Acosta
Facultad de Ingeniería, Universidad Nacional del Centro de la Provincia de Buenos Aires
INTELYMEC - CIFICEN – UNICEN - CONICET
Olavarría, Argentina
ggacosta@fio.unicen.edu.ar

***Abstract*— The off-shore industry requires periodic monitoring of underwater infrastructure for preventive maintenance and security reasons. The tasks in hostile environments can be achieved automatically through autonomous robots like UUV, AUV and ASV. When the robot controllers are based on prespecified conditions they could not function properly in these hostile changing environments. It is beneficial to count with adaptive control strategies that are capable to readapt the control policies when deviations, from the desired behavior, are detected. In this paper, we present an on-line selective reinforcement learning approach for learning reference tracking control policies given no prior knowledge of the dynamical system. The proposed approach enables real-time adaptation of the control policies, executed by the adaptive controller, based on ongoing interactions with the non-stationary environments. Applications on surface vehicles under non-stationary and perturbed environments are simulated. The presented simulation results demonstrate the performance of this novel algorithm to find optimal control policies that solve the trajectory tracking control task in unmanned vehicles.**

Keywords*— Autonomous vehicles; trajectory tracking, cognitive control; reinforcement learning; adaptive control.

I. INTRODUCTION

Autonomous vehicles have become very attractive for various tasks where the human operation is dangerous, or almost impossible. In these cases the autonomy is one of the most critical issues. Nowadays, autonomous vehicles are required to operate under so complex and uncertain environments as in the offshore industry. The complexity of these dynamical systems and uncertainty about the operating conditions make the design of control strategies so complex. Significant efforts have been made in order to provide autonomy to many autonomous vehicles (AV) such as autonomous surface vehicles (ASV), unmanned underwater vehicle (UUV), underwater autonomous vehicle (AUV), and others. Many works have been developed about tracking control strategies based on classical control techniques [1], [2], such as feedback- [3], [4], optimal-, robust- [5], [6] and model predictive- control [7]. However the major weakness of these approaches lies in the fact that they require a known model of the system dynamics. Moreover, structured and static environments are required.

When the system of interest operates under so complex and uncertain environments the dynamics modeling becomes very cumbersome or even impossible. Many times, in this type of environments the presence of human operators in the control loops becomes essential. For this reason other branches of research have been focused on artificial intelligent techniques to develop adaptive control strategies for trajectory tracking of AVs [8]. Mainly, the modern paradigm of cognitive control [9] has gained prominence into different fields of engineering as the navigation of mobile robots. The backbone of closed-loop feedback cognitive control system is based on reinforcement learning (RL) [10] paradigm to formulate adaptive control strategies for robot motion [11]–[16]. As it was recognized in [15] it is mandatory real-time strategies for robot control.

In this paper, we present an on-line selective reinforcement learning approach combined with Gaussian Process (GP) regression for learning reference tracking control policies given no prior knowledge of the dynamical system, conformed by the robot and its surroundings. The proposed approach enables real-time adaptation of the control policies, executed by the adaptive controller, based on ongoing interactions with the non-stationary environments. To cope with the size and dimension of the state space an active learning mechanism [17], [18] is incorporated by using an utility function that allows balancing the relevant information for actions selection. Based on the data obtained by the growing influence of operation conditions, approximation models are obtained by GPs. These models represent the dynamics of transition states [19]. Also, GPs models are used to describe the value functions used by the learning algorithms in continuous domains. Applications on surface vehicles under non-stationary and perturbed environments are simulated.

The organization of the paper is as follow. In Section II we present the Reinforcement learning (RL) paradigm for adaptive control. In Section III we develop our approach and we specify the proposed trajectory tracking algorithm. In Section IV we present the problem statement and the results achieved with our proposals. Finally, in Section V we present the conclusions of the present work.

978-0-933957-43-5 ©2015 MTS

II. Reinforcement Learning for Adaptive Control

Reinforcement Learning is a paradigm for learning decision-making tasks that enables systems to on-line learn and adapt to their operation environments. The general concept of RL does not require expert knowledge, that is task-specific prior knowledge or an intricate understanding of world before interacting with it. Solving a RL problem consists in learning iteratively a task from interactions to achieve a goal [10]. During learning, an agent (or a robot controller) interacts with the target system by taking an action $\mathbf{u}_t \in \mathbf{U} \subseteq \mathbf{R}$ and, after that, the system evolves from the state $\mathbf{x}_t \in \mathbf{U} \subseteq \mathbf{R}^{n_x}$ to $\mathbf{x}_{t+1}$ and the agent receives a numerical signal r_t called *reward* (or cost) which provides a measure of how good (or bad) is the action taken at $\mathbf{x}_t$ in terms of the observed state transition. Rewards are given as hints regarding goal achievement or optimal behavior.

In applying RL to the task of trajectory tracking control, the main objective of the agent is to learn the optimal policy, π^*, which defines the optimal action for different system states, bearing in mind both short and long term rewards. Let's assume that under a given policy π, the expected cumulative reward $V^{\pi}(\mathbf{x})$, or value function over a certain time interval, is a function of $\mathbf{x}^{\pi}$, where $\mathbf{x}^{\pi} = \{\mathbf{x}_t\}\ \forall\ t = 1, ..., N$ are the corresponding state values and $\mathbf{u}^{\pi} = \{\mathbf{u}_t\}\ \forall\ t = 1, ..., N$ defines the policy-specific sequence of control actions. The sequence $\mathbf{x}^{\pi}$ of state transitions gives rise to rewards $\{r_t\}\ \forall\ t = 1, ..., N$.

The trajectory tracking control is a continuous task without a single final state, hence N could be a large number. Therefore, the discounted sum of future rewards $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...$ is used to define the (discounted) expected state-value function for a policy π from the state $\mathbf{x}$:

$$V^{\pi}(\mathbf{x}) = \mathrm{E}_{\pi}\{R_t \mid \mathbf{x}_t = \mathbf{x}\} = \mathrm{E}_{\pi}\{\Sigma^{N}_{k=0}\, \gamma^k\, r_{t+k+1} \mid \mathbf{x}_t = \mathbf{x}\} \qquad (1)$$

where $\gamma \in (0,1]$ is the discount factor which weights future rewards. $V^*(\mathbf{x})$ is used to denote the maximum discounted reward obtained when the agent starts in state $\mathbf{x}$ and executes the optimal policy π^*. Thus, the associated optimal state-value function satisfies the Bellman's equation for all state $\mathbf{x}$ is:

$$V^*(\mathbf{x}_t) = \mathrm{argmax}_{\mathrm{u}}\{r_t + \gamma\, \mathrm{E}_{\mathbf{x}t+1}\,[V^*(\mathbf{x}_{t+1}) \mid \mathbf{x}_t, \mathrm{u}_t]\} \qquad (2)$$

where $\mathrm{u}_t = \pi^*(\mathbf{x}_t)$. Similarly, the state–action value function Q^* is defined by:

$$Q^*(\mathbf{x}_t, \mathrm{u}_t) = r_t + \gamma\, \mathrm{E}_{\mathbf{x}t+1}\,[V^*(\mathbf{x}_{t+1}) \mid \mathbf{x}_t, \mathrm{u}_t] \qquad (3)$$

such that $V^*(\mathbf{x}_t) = \mathrm{argmax}_{\mathrm{u}}\, Q^*(\mathbf{x}, \mathrm{u})$ for all $\mathbf{x}$. Once Q^* is known through interactions, then the optimal policy can be obtained directly through:

$$\pi^*(\mathbf{x}) = \mathrm{argmax}_{\mathrm{u}}\, Q^*(\mathbf{x}, \mathrm{u}) \qquad (4)$$

III. Trajectory Tracking Algorithm

When the state space and the action space are discrete and finite, the optimal policy can be obtained directly from a tabular representation of the value function. However, in our problem the state and actions are taken from a continuum. Hence a function approximation technique is required for value function approximation. Moreover, inductive modeling of the optimal policy is mandatory to allow selecting optimal actions in the continuous space. Gaussian Process models are a powerful alternative for generalization and on-line learning in RL algorithms.

One way to increase the learning efficiency is to extract more useful information from available data. Thus, for learning and adaptation of a robot control policy, the robot must gather information about the world exploring it. Thus, an efficient RL algorithm for trajectory tracking control must learn in few samples [20]. During the on-going interactions between the robot control policy and the environment, a model of the robot dynamics must be learned on-line, allowing the policy to adapt itself to changes in the system response to control actions. GPs models are used to represent the dynamics of transition states. For this reason, the on-line learning algorithm must be capable of dealing with continuous data stream to update the support data set. To this aim, incremental on-line sparsification suitable to work with non-parametric Gaussian process regression is required [21], [22]. The incremental sparsification technique makes possible to determine whether arriving data provide valuable information regarding the current support sets for the dynamics, state values and the control policy.

A. Gaussian Process

Gaussian Process (GP) is a relatively new kernel method popularized within the machine learning community by Rasmussen and Williams [23]. It is a powerful, non-parametric tool for regression in high dimensional spaces returning a non-parametric probabilistic model. A GP is a generalization of a Gaussian probability distribution where the distribution is over functions instead of assuming a model with a given structure before data is considered.

A key advantage of the GP is the ability to provide uncertainty estimates and learning the noise and smoothness parameters from training data. In solving RL problems through on-going interactions, the experience can be collected in data sets as $\{\mathbf{X},\mathbf{Y}\}$, where $\mathbf{X}$:$\{\mathbf{x}_i \in \mathrm{R}^d / i = 1, ..., n\}$ is the set of input vectors, $\mathbf{Y}$:$\{y_i \in \mathrm{R} / i = 1, ..., n\}$ is the set of the corresponding observations and n is the number of samples. Assume that between pairs of input-output data there is a functional relationship h: $\mathrm{R}^d \rightarrow \mathrm{R}$, such that $y_i = h(\mathbf{x}_i) + \varepsilon$, $\varepsilon \sim N(0, \sigma_\varepsilon^2)$. Then, in order to make inference about a function value $h(\mathbf{x}^*)$ for an unknown input $\mathbf{x}^*$, an inferred model for underlying function h is needed. Within a Bayesian framework, the inference of the underlying function h is described by the posterior probability:

$$p(h|\mathbf{X},\mathbf{Y}) = p(\mathbf{Y}|h,\mathbf{X})\, p(h) / p(\mathbf{Y}|\mathbf{X}) \qquad (5)$$

where $p(\mathbf{Y}|h,\mathbf{X})$ is the likelihood and $p(h)$ is a prior distribution on plausible value functions assumed by the GP model. The term $p(\mathbf{Y}|\mathbf{X})$ is called the *evidence* or *the marginal likelihood*. When modeling with GPs, a GP prior $p(h)$ is placed directly in the space of functions without the necessity to consider an explicit parameterization of the approximating function h. This prior typically reflects assumptions on the, at least locally, smoothness of h.

Similar to a Gaussian distribution, which is fully specified by a mean vector and a covariance matrix, a GP is specified by a *mean* function $m(\cdot)$ and a *covariance* function $Cov(\cdot,\cdot)$, also known as a *kernel*. A GP can be considered as a distribution over functions. However, considering a function as an infinitely long vector, all necessary computations for inference and prediction of value functions can be broken down to manipulate well-known Gaussian distributions. The fact that function $h(\cdot)$ is GP distributed is indicated by $h(\cdot) \sim GP_h(m, Cov)$ hereafter.

Given a GP model of the function h, we are interested in predicting the value function for an arbitrary input $\mathbf{x}_t^*$. The predictive (marginal) distribution of the function value $h_*=h(\mathbf{x}_t^*) \sim GP_h(\mathbf{x}_t^*)$ for a test input $\mathbf{x}_t^*$ is Gaussian distributed with mean and variance given by:

$$\mathrm{E}[h(\mathbf{x}_t^*)] = Cov(\mathbf{x}_t^*, \mathbf{X}) + (\mathbf{K}+\sigma_\varepsilon^2\mathbf{I})^{-1}\,\mathbf{Y} \tag{6}$$

$$Var[h(\mathbf{x}_t^*)] = Cov(\mathbf{x}_t^*, \mathbf{x}_t^*) + Cov(\mathbf{x}_t^*,\mathbf{X})\,(\mathbf{K}+\sigma_\varepsilon^2\mathbf{I})^{-1} Cov(\mathbf{X}, \mathbf{x}_t^*) \tag{7}$$

where $\mathbf{K}$ is the kernel matrix with $K_{ij}= Cov(\mathbf{x}_t^i, \mathbf{x}_t^j) \;\forall\; \mathbf{x}_t \in \mathbf{X}$. A common covariance function is the squared exponential (SE):

$$Cov_{\mathrm{SE}}(\mathbf{x}_t^i, \mathbf{x}_t^j) = \zeta^2 \exp[-1/2\,(\mathbf{x}_t^i - \mathbf{x}_t^j)^{\mathrm{T}}\, \boldsymbol{\Lambda}\, (\mathbf{x}_t^i - \mathbf{x}_t^j)] \tag{8}$$

Where matrix $\boldsymbol{\Lambda}=diag([l_1^2,l_2^2,...,l_d^2])$ and l_r, $r = 1,...,d$, being the characteristic length scales. The parameter ζ^2 describes the variability of the inductive model h. The parameters of the covariance function are the *hyperparameters* of the GP_h and are collected within the vector $\boldsymbol{\psi}$. To fit parameters to value function data the evidence maximization or *marginal likelihood optimization* approach is recommended (see [23]). The log-evidence is given by:

$$\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\psi}) = \int (\mathbf{Y}|h(\mathbf{X}),\mathbf{X}, \boldsymbol{\psi})\; p(h(\mathbf{x})|\mathbf{X}, \boldsymbol{\psi})\mathrm{d}h$$

$$= -1/2\;\mathbf{Y}^{\mathrm{T}}\,(\mathbf{K}_{\boldsymbol{\psi}} + \sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{Y} \;-\; 1/2 \log |\mathbf{K}_{\boldsymbol{\psi}} + \sigma_\varepsilon^2\mathbf{I}| \;-\; d/2\log(2\pi) \tag{9}$$

In (9), $h(\mathbf{X}) = [h(\mathbf{x}_1), ..., h(\mathbf{x}_n)]$ where n is the number of training points and the dependency of $\mathbf{K}$ on the hyper-parameters $\boldsymbol{\psi}$ is made explicit by writing $\mathbf{K}_{\boldsymbol{\psi}}$. Further details on GP regression can be found in the excellent books of Rasmussen and Williams [23] and MacKay [24], and references therein.

B. *Active Learning*

Active learning can be seen as a strategy for introducing data bias to make probabilistic optimal control more efficient. A priori it is unclear, which parts of the observable state space are relevant. Hence, ‘‘relevance’’ is rated by a utility function within a Bayesian active learning framework in which the posterior distributions of the value function model GP_v will play a central role in designing policy evaluation experiments [18]. This idea is represented in Fig. 1.

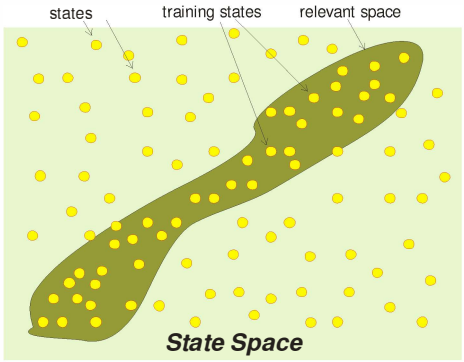


Fig. 1. State Space exploration by active learning.

In our case, training data for GPs models (used to approximate the dynamics and the value functions) are selected according to an utility function. The utility function often rates outcomes or information gain of an experiment. Before running an (simulated or real) experiment, these quantities are uncertain. Hence, into a Bayesian framework to formulate an active learning mechanism, the expected utility is considered by averaging over possible outcomes in GP models for the state transition dynamics and the value functions which are built on the fly. Online biasing data gathering largely exploits information which is already computed in the learning algorithm and make rooms for a two-stage approach in probabilistic control: simulation-based development of an optimal control policy π^* followed by policy adaptation using experimental data. Accordingly, the most promising state $\underline{\mathbf{x}}^*$ is the one that maximizes the expected utility function from a given set $\underline{\mathbf{X}}$ of possible input locations, i.e. ($\underline{\mathbf{x}}^* \in \underline{\mathbf{X}}$), which could be added to the support sets. For efficiency reasons, only the best candidates shall be added to the support sets for dynamics and values GPs models. In probabilistic optimal control, we naturally expect from a ‘‘good’’ state $\underline{\mathbf{x}}^* \in \underline{\mathbf{X}}$ to gain both information about the latent value function and to reduce uncertainty about the optimal control action at each state. Hence, we choose an utility function $U(\cdot)$ as in (10) which captures both objectives to rate the quality of candidate states used to infer GPs models. We aim to find the most promising state $\underline{\mathbf{x}}^*$ that maximizes the expected utility function based on Bayesian averaging:

$$U(\underline{\mathbf{x}}) = \rho\, \mathrm{E}_v(V^*(\underline{\mathbf{x}})|\underline{\mathbf{X}}) + \beta/2 \log (\mathrm{var}_v(V^*(\underline{\mathbf{x}})|\underline{\mathbf{X}})) \tag{10}$$

where ρ and β are used to control the exploration–exploitation tradeoff, and the predictive mean and variance of the value function are $E_v(V^*(\underline{\mathbf{x}})|\underline{\mathbf{X}})$ and $var_v(V^*(\underline{\mathbf{x}})|\underline{\mathbf{X}})$, respectively. For further calculations refer to [18].

C. On-line sparsification

Sparsification techniques have been successfully employed in robotics for online identification of data-driven models. Additionally, sparsification techniques have been combined with regression techniques [25] in order to limit the computational complexity. Particularly, in the work of Nguyen-Tuong and Peters [22] an incremental online sparsification (IOS) method has been proposed which is mainly inspired in the Kernel Recursive Least-Squares algorithm proposed by Engel et al. in [21]. To prevent expensive computational costs in on-line learning, once a new datum ($\boldsymbol{d}_{new}$) carrying novel information is detected, the point that brings the least information from a support set ($\mathbf{D}$) should be removed. By incorporating incremental online sparsification into the approach employed by the Algorithm 1, an incremental selection of samples that are used to update the support set. Accordingly, the GP models of the dynamics (GP_f) and the value function (GP_v) are kept updated by re-estimating their hyper-parameters. On-line learning of the models used to approximate GP_f and GP_v are critical for continuous policy adaptation to a given dynamic system characteristics. Following, an overview about the incremental online sparsification technique is given. Further details on IOS can be found in the works of Nguyen-Tuong and Peters [22] and Engel et al. [21], and references therein. Also, the reader is referred to work of Chen et al. [26] for a novel integration of on-line sparsification with temporal difference learning.

The general notion is that the kernel matrix $\boldsymbol{K}_v$ should be fully ranked. That is to ensure that any sample in the support set cannot be linearly represented by the other samples in $\mathbf{D}$. Then, for online sparsification, the basic idea is that if an arriving datum can be linearly represented by the data in the support set $\mathbf{D}$, it should not be added. Assuming that at time t, the support set has m data points such that $\mathbf{D}=\{\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_m\}$, where, by construction, $\{\phi(\boldsymbol{d}_1), \phi(\boldsymbol{d}_2), ..., \phi(\boldsymbol{d}_m)\}$ are linearly independent feature vectors. When a new datum, $\boldsymbol{d}_{new}$, arrives, it is crucial to decide whether it can be linearly represented by the existing data in $\mathbf{D}$. So, testing whether $\phi(\boldsymbol{d}_{new})$ is linearly dependent on data points is required. The linear dependence [27], [28] can be measured by

$$\delta = \| \{ \Sigma^{m}_{i=1} a_i \phi(\boldsymbol{d}_i) - \phi(\boldsymbol{d}_{new}) \| ^2 \tag{11}$$

Therefore, when $\boldsymbol{d}_{new}$ brings some novelty since δ exceedes a threshold η, then it must be added to $\mathbf{D}$. Accordingly, this procedure aims to cover only the relevant region of state space with a limited support set. In (11) we have a vector $\boldsymbol{a}=(a_i,...,a_m)^T$ of coefficients a_i of linear dependence which can be determined by minimizing δ as follows:

$$\boldsymbol{a} = \min_a [\boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a} - 2\, \boldsymbol{a}^T \boldsymbol{k} + k] \tag{12}$$

where $\boldsymbol{K}=k(\mathbf{D},\mathbf{D})$, $\boldsymbol{k}=k(\mathbf{D},\boldsymbol{d}_{new})$, $k=k(\boldsymbol{d}_{new},\boldsymbol{d}_{new})$ and $k(\cdot)$ is assumed to be a Gaussian kernel function defined as in (8). Solving the unconstrained optimization problem of (12) yields the optimal $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$. Replacing this result in Eq. (11) gives rise to

$$\delta = k(\boldsymbol{d}_{new},\boldsymbol{d}_{new}) - \boldsymbol{k}^T \boldsymbol{a} \tag{13}$$

Once δ is computed by (13) for a new data point, its value is compared with the threshold η and a decision to incorporate $\boldsymbol{d}_{new}$ into the set $\mathbf{D}$ is taken. Fig. 2 shows the pseudocode for the sparsification procedure.

Algorithm 1. Sparsification

Input: $\mathbf{D}$, $\boldsymbol{d}_{new}$, η, N_{max}
Compute $\boldsymbol{k} = k(\mathbf{D}, \boldsymbol{d}_{new})$, $\boldsymbol{a} = \boldsymbol{K}^{-1}\boldsymbol{k}$
Compute $\delta = k(\boldsymbol{d}_{new}, \boldsymbol{d}_{new}) - \boldsymbol{k}^T \boldsymbol{a}$
If $\delta > \eta$ **then**
  If $\|\boldsymbol{D}\| < N_{max}$ **then**
    $\boldsymbol{D} = \{\boldsymbol{d}_i\}_{i=1}^{m+1}$
    For $i = 1$ **to** m
      $\boldsymbol{k}_{m+1} = k(\mathbf{D}^i, \boldsymbol{d}_{new})$
      $k_{m+1} = k(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{new})$
      $k_{i,m+1} = k(\boldsymbol{d}_i, \boldsymbol{d}_{new})$
      $\boldsymbol{\alpha}_i = (\boldsymbol{K}^i_{old})^{-1} \boldsymbol{k}_{m+1}$
      $\gamma_i = k_{m+1} - \boldsymbol{k}^T_{m+1} \boldsymbol{\alpha}_i$
      Update δ^i
      Update $(\boldsymbol{K}^i_{new})^{-1}$
    End For
  else
    Compute $\lambda^i \; \forall \; \boldsymbol{d}_i \in \mathbf{D}$
    Compute $j = \min_i (\lambda^i \delta^i)$
    Update $\mathbf{D}$ overwriting $\boldsymbol{d}_j = \boldsymbol{d}_{new}$
    For $i = 1$ **to** m
      $\boldsymbol{k}_{m+1} = k(\mathbf{D}^i, \boldsymbol{d}_{new})$
      $k_{m+1} = k(\boldsymbol{d}_{m+1}, \boldsymbol{d}_{new})$
      $k_{i,m+1} = k(\boldsymbol{d}_i, \boldsymbol{d}_{new})$
      $\boldsymbol{r} = \boldsymbol{k}_{m+1} - \text{row}_j [\boldsymbol{K}^i_{old}]^T$
      Update δ^i
      Update $(\boldsymbol{K}^i_{new})^{-1}$
    End For
  End If
End If

Fig. 2. Sparsification Algorithm.

In the sparsification algorithm when $\delta > \eta$ and the support set which has not been reached its maximum size (N_{max}), $\mathbf{D}$ must be updated using the Algorithm 1 in Fig. 2. Whenever the number of data in the set $\mathbf{D}$ is N_{max}, $\mathbf{D}$ must be updated by replacing an old point $\boldsymbol{d}_j \in \mathbf{D}$ by the new one $\boldsymbol{d}_{new}$. In the latter case the spatial and temporal allocation of the data is taken into account through a forgetting rate $\lambda^i \in [0,1]$ [22]. The forgetting rate is defined as:

$$\lambda^i = \exp(-(t - t_i)^2/(2h)) \tag{14}$$

where t_i is the time when the point i was included into **D** and the parameter h controls the trade-off between spatial and temporal coverage.

D. Adaptive tracking algorithm

Based on the RL framework and with the ideas presented above our proposed approach is presented and summarized by the Algorithm 2, outlined in Fig. 3. This algorithm is able to adapt on-line a policy with the robot-environment interactions. Initially, it is assumed a random control policy π. This policy can also be modeled by a GP model, i.e. $\pi_0 \sim GP_\pi$. As indicated in line 2, this initial policy π_0 is used to interact with the system during τ sampling times. In this way, τ state-action transitions are observed and used to define the support data set **D**. Then, with the collected data a dynamic GP model, GP_f, is defined. Next, the initial function values for each state-action transitions are directly computed using the reward function $r(\cdot)$ and a value function GP model (GP_v) is defined.

On-line policy learning in Algorithm 2 takes place in the loop defined from line 8 to 25. Through the active learning mechanism, safe exploration while interacting with the environment is made by choosing the best action for the current vehicle state. Once the control action is applied, the data point for observed state transition is available and recorded in $\boldsymbol{d}_{new}$ (line 10). By Algorithm 1 the support set **D** is updated iff $\boldsymbol{d}_{new}$ provides valuable information (line 11). Following, the dynamics model GP_f is updated based on the latter version of **D** (line 12). In this way, an updated version of the model used to describe the robot response to control actions is available. Also, an improved version of the control policy adapted to the system operating conditions is obtained.

In line 14, the Q-learning rule is implemented to update the Q-values regarding an immediate reward $r(\mathbf{x}_i,\mathrm{u}_i)$ and a discount factor γ. As was said, the value function and the dynamics are approximated by GP models, therefore the uncertainties about value function and dynamics have been taken into account. Due to the generalization property of GP_f and GP_v when the expected value function of $\mathbf{x}_i'$ ($\mathrm{E}[V(\mathbf{x}_i')|\mathbf{x}_i,\mathrm{u}_i,GP_f]$) is computed there is no need to restrict to a finite set of successor states $\mathbf{x}_i'$. Thus, in the computation of these expected values, the expected value of V^* at a successor uncertain state $\mathbf{x}_i'$ is estimated (for calculation details refer to [18]). Later on, in the next *for loop* (line 16 to 20), with the computed Q-values and all actions $\mathrm{u}_i \in \pi^*(\mathbf{X}) \subset \mathbf{D}$, a model of state-action value function, GP_q, is learned. Then, the optimal control $\mathrm{u}_i^* = \pi^*(\mathbf{x}_i)$ is the maximizing argument of the Q-values for a certain state $\mathbf{x}_i$, and the value function $V^*(\mathbf{x}_i)$ at $\mathbf{x}_i$ is the corresponding maximum value. To maximize Q for a certain state $\mathbf{x}_i$, standard optimization methods such as Golden section or Fibonacci Search are used. Note that GP_q models a function of u only since $\mathbf{x}_i$ is fixed. In this way, the optimal action is obtained through $\max_\mathrm{u} Q(\mathbf{x}_i,\mathrm{u}) \approx \max_\mathrm{u} m_q(\mathrm{u})$, the maximum of the mean function $m_q(\mathrm{u})$ of GP_q.

Once the optimal values $V^*(\mathbf{x}_i)\ \forall\ \mathbf{x}_i \in \mathbf{X} \subset \mathbf{D}$ are computed, a GP model, GP_v, to approximate the state-value function is learned. Moreover, with all $\mathbf{x}_i \in \mathbf{X} \subset \mathbf{D}$ and the optimal controls computed in line 18 for all $\mathbf{x}_i \in \mathbf{X}$ an updated version of the control policy can be approximated by a new GP model, which is referred as GP_π, in the recursion p of the Algorithm 2.

As iterations of Algorithm 2 are made, an adaptation process of the tracking control policy takes place. We say that there is an "adaptation" since the dynamics GP model, GP_f, is learned with data coming from direct interactions. The changes in system's dynamic behavior will be reflected in the underlying structure of the collected data. Therefore, the dynamic model must be updated in each recursion of Algorithm 2. Hence, the flexible features of nonparametric approximators plays a central issue. Modeling system response by GP_f allows a nonparametric representation that can be improved continuously to the real system behavior based on data from on-going interactions. Accordingly, a policy adaptation is made since the state-value function depends on the dynamic model accuracy and the policy model, GP_π. Note that GP_π is learnt based on optimal controls ($\pi^*(\mathbf{X})$), which depends on the GP_q model which, in turn depends on state-actions values (computed in line 14) are estimated using the dynamics model GP_f.

Algorithm 2. Pseudocode of trajectory tracking algorithm

1: Assume π_0 as a random control policy.
2: Interact with the system during τ samples applying π_0.
3: Define the support set **D** with the collected data in step 2.
4: Initialize the dynamic model GP_f with **D**
5: Compute $V_0(\mathbf{X}) = r(\mathbf{X}, \pi^*(\mathbf{X}))$ such that $\mathbf{X} \subset \mathbf{D}$, $\pi^*(\mathbf{X}) \subset \mathbf{D}$
6: Initialize the value function model GP_v with $V_0(\mathbf{X})$ and $\mathbf{X} \subset \mathbf{D}$
7: $p = 1$
8: **Loop**
9: Determine an action by active learning mechanism
10: Interact with the system ⇒ collect the new data $\boldsymbol{d}_{new}$
11: Update the support set **D** with the new data ⇒ Sparsification Algorithm
12: Update the dynamic model GP_f with the updated **D**
13: **For all** pair $(\mathbf{x}_i, \mathrm{u}_i)$ such that $\mathbf{x}_i \in \mathbf{X} \subset \mathbf{D}$, $\mathrm{u}_i \in \pi^*(\mathbf{X}) \subset \mathbf{D}$; $i = 1, \ldots, \|\mathbf{D}\|$ **do**
14: $Q(\mathbf{x}_i, \mathrm{u}_i) = r(\mathbf{x}_i, \mathrm{u}_i) + \gamma\, \mathrm{E}[V(\mathbf{x}_i') \mid \mathbf{x}_i, \mathrm{u}_i, GP_f]$
15: **End For**
16: **For all** $\mathbf{x}_i$ such that $\mathbf{x}_i \in \mathbf{X} \subset \mathbf{D}$
17: Approximate a Q-value model GP_q as $Q(\mathbf{x}_i, \cdot) \sim GP_q$
18: Obtain an optimal control $\mathbf{u}^* = \pi^*(\mathbf{x}_i)$ for $\mathbf{x}_i$
$\Rightarrow \pi^*(\mathbf{x}_i) \in \arg\max_\mathbf{u} Q(\mathbf{x}_i, \mathbf{u}) \approx \max_\mathbf{u} m_q(\mathbf{u})$
19: Obtain the optimal state-value for $\mathbf{x}_i \Rightarrow V^*(\mathbf{x}_i) = Q(\mathbf{x}_i, \pi^*(\mathbf{x}_i))$
20: **End For**
21: Determine the value function model GP_v with $V^*(\mathbf{X})$ such that $\mathbf{X} \subset \mathbf{D}$
22: Determine the current GP_{π^*} policy model with all $\mathbf{x}_i \in \mathbf{X} \subset \mathbf{D}$ and $\pi^*(\mathbf{X})$
23: $\pi_p^* := GP_{\pi^*}$ ⇒ *Improved tracking control policy*
24: $p = p + 1$
25: **End Loop**

Fig. 3. Trajectory tracking algorithm.

IV. Experiment Results

A. Problem statement

To test and demonstrate the benefits of the proposed approach we will use a modified example based on the studied by Bryson and Ho [29] under optimal control framework. The original case consists in guiding a boat navigating with constant velocity on a river with a certain water current profile.

The dynamics of the system is defined by the following set of differential equations [30]:

$$\begin{aligned} dx/dt &= C\cos(\phi) - y.C \\ dy/dt &= C\sin(\phi) \end{aligned} \quad (15)$$

where the term $y.C$ represents the effect of the water current profile for the boat displacement. However, for our trials we will use a modified version in which the dynamics of the water profile does not follow a deterministic pattern as in (15). Thus, our dynamic system is described by the following set of differential equations:

$$\begin{aligned} dx/dt &= C\cos(\phi) - y.C\,(1+N_x) \\ dy/dt &= C\sin(\phi) + C\ N_y \end{aligned} \quad (16)$$

where N_x is a value defined according to a normal distribution with mean 0 and standard deviation that depends on the "y" coordinate; i.e. $N_x = N(0,|y|/10)$. Then, N_y is defined according to a normal distribution with mean 0 and standard deviation C; i.e.: $N_y = N(0,C)$. We assume that the boat is moving at constant velocity C = 0.1 km/h. Moreover, it is assumed that the control action (rudder direction, ϕ) is applied instantaneously.

Note that the change in the dynamic system was performed with a dual purpose. On the one hand, it is to model a situation closer to the reality to which the trajectory control techniques must face. On the other hand, it is to make clear that the algorithm proposed in this paper is able to address these situations, closer to engineering application.

B. *Trajectory tracking experiment*

The experimental tests of Algorithm 2 consist in guiding the boat along a straight trajectory under an unstructured and perturbed environment as in (16). To solve this trajectory tracking control task we have to carefully define the reward function .Then, the reward function was defined by a gaussian function that saturates for great deviations from a chosen reference target trajectory, This target trajectory is indicated by the black line in Fig. 4. Thus, the used reward function is:

$$r = -1 + \exp(-d_\perp^2/(2a^2)) \quad (17)$$

where $d_\perp$ is the perpendicular distance between the boat position and the target line. Naturally, the highest values of rewards will be over the target trajectory where $d_\perp = 0$. In this way, with the defined reward function, we seek to generate a "funnel effect" aiming to obtain a control policy able to keep the boat along the reference trajectory minimizing the distance to it. This effect can be observed in Fig. 5. Fig. 5b shows the projections of reward function values in the x,y plane, around the reference trajectory. In this figure the "funnel effect" can be seen directly, i.e. reward values decrease as we move away from the target line.

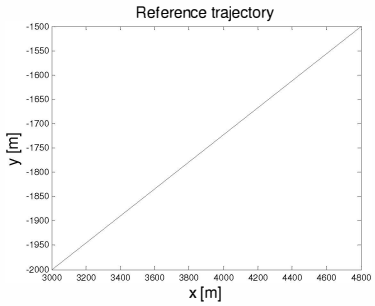
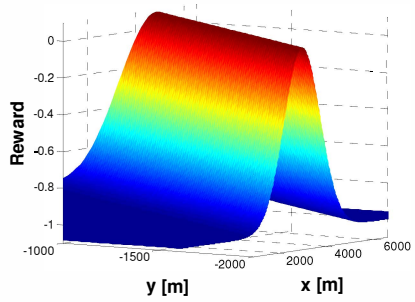
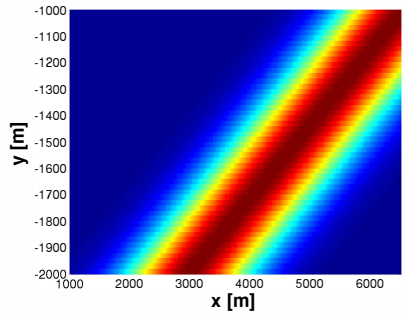


Fig. 4. Reference trajectory.



(a)



(b)

Fig. 5. Reward function. a) Reward function values for the distance to the target line in the plane coordinates x,y. b) Projections of rewards function values in the x,y plane.

To solve the tracking control task using the Algorithm 2, an initial random policy is regarded. The discount rate γ is set to 0.95. The a parameter of the reward function in (17) is set to 1. As the algorithm executions progress, the dynamic model (GP_f) and the value function model (GP_v) are improved. Accordingly, with the algorithm progresses the tracking control policy is improved.

Fig. 6 shows a progress of value function improvement by the application of Algorithm 2. As it can be seen, as the algorithm recursion progresses it learns a better value function, which finally gives rise to the trajectory tracking control policy. Moreover, Fig. 6 reveals the real active learning mechanism which leads the system to the desired target zone. Clearly, when the algorithm iterations progress, it can be seen

that the projections of state-values into the plane *x,y* reveal the target zone, i.e. the reference trajectory.

Figure 7a shows the function values for the obtained optimal trajectory tracking policy. Similarly to Fig. 6, Fig 7b shows the projections of state-values into the plane *x,y*. As expected, the highest values correspond to the states nearest to the reference trajectory.

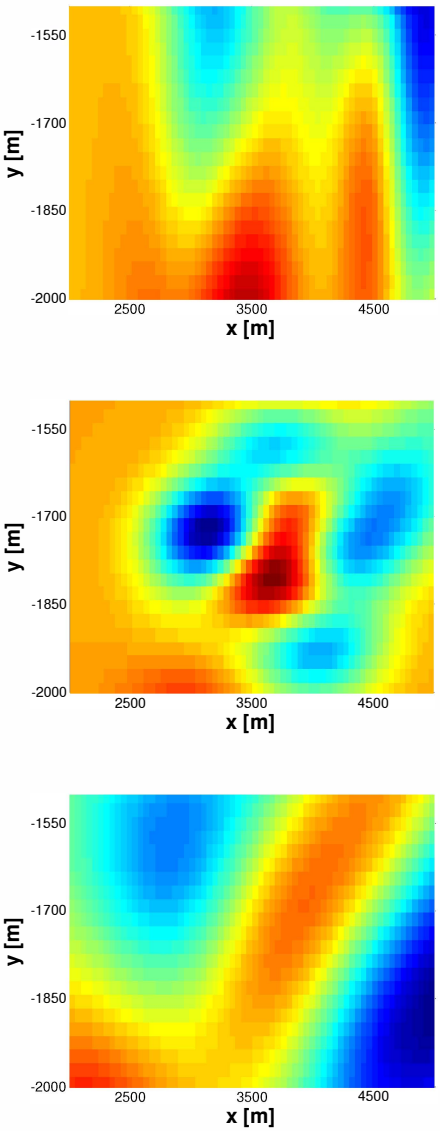
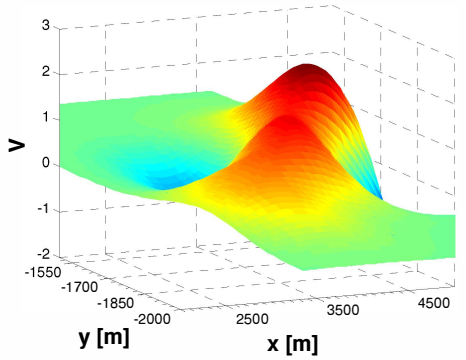
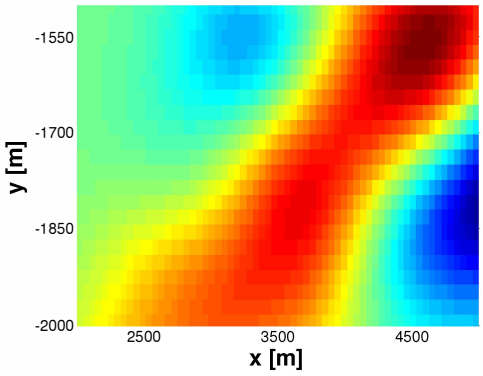


Fig. 6. Progress of values function.



(a)



(b)

Fig. 7. Optimal value function. a) Function values for the states x,y. b) Projections of the function values into the plane x,y.

In Fig. 8 it is depicted a result of applying the optimal policy for the boat trajectory tracking under the described stochastic environment. The green line is the resulting trajectory and the black line is the reference trajectory. The blue arrows represent the stochastic components of the watercourse. As can be seen, despite the severely perturbed environment, the optimal policy tends to maintain the boat as close as possible along the reference trajectory. It is noticeable that the effect of unpredictable components of the water flow made it very difficult to find an optimal policy that can guide the boat over the reference trajectory.

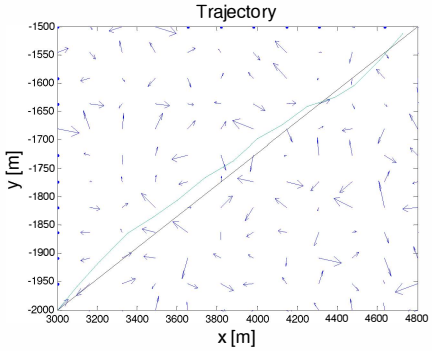


Fig. 8. Trajectory tracking result under the perturbated environment.

V. Final Remarks

The main motivation of the present work is to propose a tracking control strategy capable to adapt itself to the unstructured and perturbed environments. Precisely, the development aims to generate a proposal for trajectory tracking of autonomous vehicles, such as ASVs and AUVs, for potential applications in off-shore industries. Then, this previous computational experiments were required before downloading the control strategy to a real surface vehicle under non-stationary and perturbed environment.

The proposed RL algorithm guarantees finding at least one optimal control policy. These simulation results demonstrate the performance of our proposed algorithm to find an optimal control policy for solving the trajectory tracking control task in unmanned vehicles.

These preliminary results are a starting point for future developments in autonomous robotics. Particularly, we are currently developing the ICTIOBOT AUV as a test platform for these control algorithms, in environments that are more hazardous and complex for the autonomous decision making process. This will imply further research on the reward function definition and the adaptive behavior of the control algorithms with the input of real data from wet tests.

References

[1] H. Ashrafiuon, K. R. Muske, y L. C. McNinch, "Review of nonlinear tracking and setpoint control approaches for autonomous underactuated marine vehicles", en American Control Conference (ACC), 2010, 2010, pp. 5203-5211.

[2] T. I. Fossen, Marine control systems: guidance, navigation and control of ships, rigs and underwater vehicles. Marine Cybernetics, 2002.

[3] I. Kaminer, A. Pascoal, E. Hallberg, y C. Silvestre, "Trajectory Tracking for Autonomous Vehicles: An Integrated Approach to Guidance and Control", J. Guid. Control Dyn., vol. 21, n.º 1, pp. 29-38, 1998.

[4] B. Madhevan y M. Sreekumar, "Tracking algorithm using leader follower approach for multi robots", Procedia Eng., vol. 64, pp. 1426-1435, 2013.

[5] H. Tan, S. Shu, y F. Lin, "An optimal control approach to robust tracking of linear systems", Int. J. Control, vol. 82, n.º 3, pp. 525-540, mar. 2009.

[6] L. Vandenberghe, "Robust linear programming and optimal control", 2002, pp. 293-293.

[7] I. S. Gregor Klancar, "Tracking-error model-based predictive control for mobile robots in real time", Robot. Auton. Syst., vol. 55, pp. 460-469, 2007.

[8] M. Knudson y K. Tumer, "Adaptive navigation for autonomous robots", Robot. Auton. Syst., vol. 59, n.º 6, pp. 410-420, jun. 2011.

[9] S. Haykin, M. Fatemi, P. Setoodeh, y Y. Xue, "Cognitive Control", Proc. IEEE, vol. 100, n.º 12, pp. 3156-3169, dic. 2012.

[10] R. S. Sutton y A. G. Barto, Reinforcement learning: An introduction. MIT Press, 1998.

[11] A. L. da Costa, A. G. S. Ceonceicao, R. G. Cerqueira, y T. T. Ribeiro, "Omnidirectional mobile robots navigation: A joint approach combining reinforcement learning and knowledge-based systems", en Biosignals and Biorobotics Conference (BRC), 2013 ISSNIP, 2013, pp. 1-6.

[12] M. A.-R. Mohammad Abdel Kareem Jaradat, "Reinforcement based mobile robot navigation in dynamic environment. Robotics Comput-Integr Manuf", Robot. Comput.-Integr. Manuf., vol. 27, n.º 1, pp. 135-149, 2011.

[13] M. Carreras, J. Yuh, J. Batlle, y P. Ridao, "A behavior-based scheme using reinforcement learning for autonomous underwater vehicles", IEEE J. Ocean. Eng., vol. 30, n.º 2, pp. 416-427, abr. 2005.

[14] C. Gaskett, D. Wettergreen, y A. Zelinsky, "Reinforcement Learning applied to the control of an Autonomous Underwater Vehicle", en In Proc. of the Australian Conference on Robotics and Automation (AUCRA'99, 1999, pp. 125–131.

[15] T. Hester, M. Quinlan, y P. Stone, "RTMBA: A real-time model-based reinforcement learning architecture for robot control", en 2012 IEEE International Conference on Robotics and Automation (ICRA), 2012, pp. 85-90.

[16] J. Kober, J. A. Bagnell, y J. Peters, "Reinforcement Learning in Robotics: A Survey", Int. J. Robot. Res., pp. 579-610, 2013.

[17] A. Baranes y P.-Y. Oudeyer, "Active learning of inverse models with intrinsically motivated goal exploration in robots", Robot. Auton. Syst., vol. 61, n.º 1, pp. 49-73, ene. 2013.

[18] M. P. Deisenroth, C. E. Rasmussen, y J. Peters, "Gaussian process dynamic programming", Neurocomputing, vol. 72, n.º 7-9, pp. 1508-1524, mar. 2009.

[19] M. P. Deisenroth, Efficient Reinforcement Learning Using Gaussian Processes. KIT Scientific Publishing, 2010.

[20] M. Deisenroth y C. E. Rasmussen, "Efficient Reinforcement Learning for Motor Control", 2009.

[21] Y. Engel, S. Mannor, y R. Meir, "The kernel recursive least-squares algorithm", IEEE Trans. Signal Process., vol. 52, n.º 8, pp. 2275 - 2285, 2004.

[22] D. Nguyen-Tuong y J. Peters, "Incremental online sparsification for model learning in real-time robot control", Neurocomputing, vol. 74, n.º 11, pp. 1859-1867, may 2011.

[23] C. E. Rasmussen y C. K. I. Williams, Gaussian processes for machine learning. MIT Press, 2006.

[24] D. J. C. MacKay, Information theory, inference, and learning algorithms. Cambridge, UK; New York: Cambridge University Press, 2003.

[25] O. Sigaud, C. Salaün, y V. Padois, "On-line regression algorithms for learning mechanical models of robots: A survey", Robot. Auton. Syst., vol. 59, n.º 12, pp. 1115-1129, dic. 2011.

[26] X. Chen, Y. Gao, y R. Wang, "Online selective Kernel-based temporal difference learning", IEEE Trans. Neural Netw. Learn. Syst., vol. 24, n.º 12, pp. 1944-1956, dic. 2013.

[27] B. Schölkopf y A. J. Smola, Learning with kernels: support vector machines, regularization, optimization, and beyond, 1st ed. The MIT Press, 2001.

[28] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Muller, G. Ratsch, y A. J. Smola, "Input space versus feature space in kernel-based methods", IEEE Trans. Neural Netw., vol. 10, n.º 5, pp. 1000 -1017, sep. 1999.

[29] J. A. E. Bryson y Y.-C. Ho, Applied optimal control: optimization, estimation and control, Revised. Taylor & Francis, 1975.

[30] M. T. Rosenstein y A. G. Barto, "Supervised actor-critic reinforcement learning", en Handbook of Learning and Approximate Dynamic Programming, John Wiley & Sons, Inc., 2004, pp. 359–380.