2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

RoMPLA: An Efficient Robot Motion and Planning Learning Architecture

Javier Gonzalez-Quijano, Mohamed Abderrahim, Choukri Bensalah, Silvia Rodríguez-Jiménez
RoboticsLab, University Carlos III of Madrid, Spain
jgonza1,mohamed,cbensala,srjimene@ing.uc3m.es

Abstract— **Robot motor skill learning is currently one of the most active research areas in robotics. Many learning techniques have been developed for relatively simple problems. However, very few of them have direct applicability in complex robotics systems without assuming prior knowledge about the task due to two facts. On one hand, they scale badly to continues and high dimensional problems. On the other hand, they require too many real learning episodes. In this sense, this paper provides a detailed description of an original approach capable of learning from scratch suboptimal solutions and of providing closed-loop motor control policies in the proximity of such solutions. The developed architecture manages the solution in two consecutive phases. The first phase provides an initial open-loop solution state-action trajectory by mixing kinodynamic planning with model learning. In the second phase, the initial state trajectory solution is first smoothed and then, a closed-loop controller with active learning capabilities is learned in its proximity. We will demonstrate the efficiency of this two phases approach in the Cart-Pole Swing-Up Task problem.**

I. Introduction

Motor skill development in humans being is a very complex process involving many cognitive skills which takes many years to improve [1]. Neuroscience has taken care of this concrete aspect for a long time. A very important issue is about the importance of planning and model learning as a synergy that guides the exploration process when learning new motor skills. Planning employs internal models, even when they are immature. Then, the plan is executed and the model gets improved using the obtained experience. Later, a new planning process starts and so on. This planning-learning phase finishes when the model is good enough to generate an appropriate action sequence to reach the goal. During the learning phase, exploration usually does not take place explicitly. Generally, it is indeed the consequence of executing actions which are partially invalid due to the imprecision of the models that where used in the motor planning process. In addition, it is necessary to realize that the capability of planning in humans being is not accurate enough to generate smooth and exact plans to carry out a certain task precisely. It only provides an initial plan which must be later refined. In turn, the planning process is carried out fast enough to allow humans to learn new motor skills in a short time period. When this raw solution has already been obtained, some local motor control learning mechanisms manage to refine the solution, and to obtain robust and accurate closed-loop controllers capable of tracking such solution accurately. Some authors claim that these controllers work in a model-predictive way [2].

The objective of this work is to translate the ideas described above into a motor learning architecture capable of learning new motor learning tasks without assuming prior knowledge, proving robust closed-loop controllers in the surroundings of the final solution. The Robot Motion Planning Architecture (RoMPLA) also follows a two phases approach. In the first phase, a mixed scheme based on kinodynamic planning and model learning is proposed. In the second phase, the solution is first refined using polynomial fitting and, later, a model predictive controller with active learning capabilities manages to learn a closed-loop policy in the proximity of the solution.

II. Related Work

Several different approaches may be applied for learning motor control tasks. A formidable overview of the whole problem has been recently performed by Schaal et al. in [3]. Basically, there exist three main learning schemes: model-free learning, model-based learning and mixed learning approaches. The most famous ones are related to optimal control theory, concretely to the dynamic programming framework [4]. Dynamic Programming has been described as the most general of the optimization approaches because conceivably it can solve the broadest problems classes. Dynamic Programming assumes a perfect knowledge about the model dynamics, thus being indeed considered as a probabilistic planning technique. Nevertheless, its combination either with model learning techniques or other types of learning techniques has led to many different learning approaches which are generally grouped by the term of reinforcement learning [5]. On one hand, model-based approaches require often less real interaction with the environment than model-free approaches, as the behaviour of the system can be simulated with such models. Furthermore, this learned models may be reused for learning other different tasks. On the other hand, there exist a different set of techniques, called model-free learning, which do not make use of the dynamical model of the system. Here, a value or an action-value function may be mainly computed directly using temporal-difference, Montecarlo or gradient-descend methods. The most well-known one, which has inspired many other later methods, is the Q-learning algorithm [6]. Furthermore, it is possible to find mixed approaches that try to get the best advan-

tages from model-based and from model-free techniques by integrated learning, planning and reacting. As an example, we can mention the Dyna architecture [7]. Nonetheless, the scalability of this class of algorithms to high dimensional state and action spaces is still a matter of study, more in the case of continues state-action spaces. To overcome this problem, the authors of this publication proposed in previous work a new model-based learning algorithm, called KiPLA [8], which handles much more efficiently the curse of dimensionality problem. This algorithm mixes kinodynamic planning and model learning for the purpose of finding suboptimal open-loop policies for achieving a certain task. The fact that KiPLA does not offer closed-loop policies solutions, motivates the design of the proposed RoMPLA architecture, which takes care of this issue. Other types of methods are not discussed here as they usually operate locally and can not handle general motor learning problems without assuming prior knowledge about the task.

Other motor learning techniques are focused on learning tracking tasks. These methods usually operate locally. Some important methods, in relation to this work, are based on the Active Learning paradigm. These techniques aim at maximizing the information during the exploration process while reducing the number of real robot-environment interactions. This relatively new field is helping to improve not only the general approaches described above but also providing smart learning capabilities to many classical control techniques. Approaches to active data selection may be based on heuristics or the optimization of an objective function [9]. Several mechanisms, which balance the exploration-exploitation trade-off in Reinforcement Learning, can also be understood as active learning strategies, such as the Boltzmann exploration rule. A complete overview on this problem was done by Robbel [10]. He also presents a new active learning approach based on the Locally Weighted Projection Regression (LWPR) algorithm for motor control. This regression method [11], which is also used in our proposed architecture, will be discussed in Section IV.

III. Blind-RRTs

Blind-RRTs are a new modification of classical RRTs in order to adequate them for their employment in motor learning problems. This modification is needed as neither environment constraints nor kinodynamic constraints are known a priori. Our new kinodynamic planner uses forward dynamics models which links the knowledge in the following way: $\{s_t, a_t\} \longrightarrow \{s_{t+1}\}$. When learning the model, real experimental data are used. This fact makes that not possible states, being part of the constrained state space, are not incorporated in the regression training process. Due to this reason, if we apply a certain action in a certain valid state s_t, then the future state s_{t+1} will also be valid. As the tree is created incrementally, if we can ensure that the starting state s_0 is a valid state, then we can also ensure that the rest of the nodes of the tree represent indeed valid states which are not contained in the constrained state space. Inverse model can not be applied directly to the construction of the tree due to two reasons. The first one is that they are, generally, not proper functions and therefore, it is not possible to learn the model via regression. The second one is that they do not encode the constraints by themselves. Thus, the employment of forward models is justified in the context of applying kinodynamic planners to learning problems.

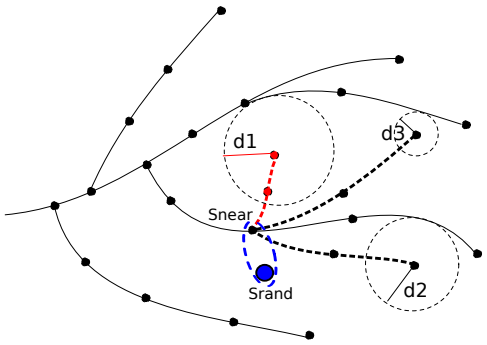

Fig. 1. Blind-RRT planning expansion.

The way the Blind-RRT expands its nodes is very similar to the RRT. Both algorithms select the node to expand in the same way. A random state is generated, then it finds the nearest node to this random state in the tree and selects it. However, there is still a big difference when creating a new node. Blind-RRTs do not expand in the direction of the random state used to select the expanding node. This process is described in Fig. 1. A fixed number of candidate actions, accomplishing the action-space constraints, is gathered randomly. Then, a forward model is used to create new candidate branches which are represented with a discontinuous line. In the direct extension of RRTs to kinodynamic planning, the branch that most approximate to the random state should be chosen. However, due to strong biases in the candidate branches, this way of selecting the best branch does not ensure a good state-space coverage. Our modification, instead, evaluates the final state of each of the branches. The best branch is the one which maximizes the distance of its final state to the nearest node in the tree. In Fig. 1, the best branch was represented by a red discontinuous line and the distances of the final state points of each of the branches to the nearest neighbour ($d1$, $d2$ and $d3$) is represented by the radius of the drawn circumferences.

IV. Model Learning Through Local Weighted Projection Regression

Function approximation techniques can handle the identification of complex dynamical systems. Machine learning community has largely contributed to this issue in the past few years by providing a large variety of regression techniques. In the context of robotics, such techniques should in many cases fulfill some conditions, mainly related to the scalability to high dimensional state-action spaces and, also, to the ability of handling efficiently incremental learning. In this sense, local approximation techniques have gained popularity. In this work, we employ the well known Local Weighted Projection Regression technique (LWPR) to learn

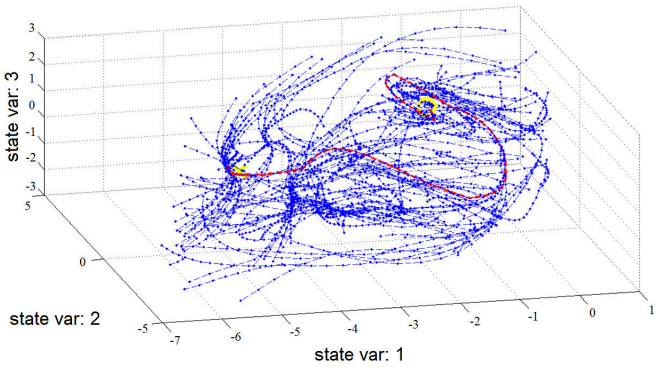

Fig. 2. A generic planning solution is illustrated. The blue lines represent the planning tree whereas the red discontinues line represent the final solution.

the robot system dynamics in a forward way. It uses experience data, denoted by the tuple $\{x_t, a_t, x_{t+1}\}$, as training data for the regression algorithm. The input is formed by the state and action vectors in an instant time t and the output, what we want to to generalize, represents the state change rate. Discrete computation requires the following approximation:

$$\dot{x}(t) \simeq \frac{x_{t+1} - x_t}{\delta t}$$

The LWPR algorithm assumes that the objective function can be approximated using a weighted average of different linear models:

$$\hat{y}(x) = \sum_{k=1}^{K} w_k(x)\hat{y_k}(x) / \sum_{k=1}^{K} w_k(x), \quad (1)$$

where y_k are the hyperplanes representing the linear models:

$$y_k(x) = b_k^0 + b_k^T(x - c_k), \quad (2)$$

and w_k represents the kernel function which weights the influence of each of the linear models:

$$w_k(x) = exp(-\frac{1}{2}(x - c_k)^T D_k(x - c_k)) \quad (3)$$

Given a query point x, every linear model calculates a prediction $y_k(x)$. For nonlinear function approximation, the core concept of the learning system is to find approximations by means of piecewise linear models. Learning involves automatically determining the appropriate number of local models K, the parameters b_k of the hyperplane in each model, and also the region of validity, called receptive field (RF), parametrized as the distance metric D_k in the Gaussian kernel. Local models are then created when needed. A deeper explanation is found in [11]. The main difference between this algorithm and other similar ones is that it first searches for lower dimensional distributions of the training data and then performs the regression. Thus the number of local models needed to approximate the whole function are reduced.

V. Description of RoMPLA Architecture

Mixing planning and learning is not a new idea, as it is the principle of model-based learning approaches. A vast majority of model-based learning approaches, which can handle the learning of motor complex tasks without assuming prior knowledge, employ methods derived from the Dynamic Programming framework as the core of the planning process. Dynamic Programming and its related methods are not focused on planning open-loop state-action trajectories. They recursively construct, at the same time, an optimal closed-loop policy for the entire state-space. Nonetheless, the computational cost becomes prohibitively huge, even more in the case of continuous and high-dimensional state-action spaces. Dynamic Programming is unlikely to represent really the planning process occurring in human brains. Some studies support the idea that humans first learn valid open-loop policies and, then, search for a closed-loop policy around this solution [12]. Our proposed architecture, RoMPLA (Robot Motion Planning Learning Architecture), is inspired in this idea. Two main concepts make the difference with existing other model-based learning approaches. The first one is related to the mixed planning and model learning process occurring in human beings, which provides only open-loop control policies. In turn, it is capable of offering very quickly suboptimal solutions. It is necessary to highlight that this mixed planning-learning concept greatly reduces the number of real robot-environment interactions. The second one is related to the motor babbling phase, which is indeed a post process which refines the trajectory and provides closed-loop control policies only in the proximity of the state-action trajectory solution. Existing model-based learning approaches related to techniques based on dynamic programming do not need this phase as the planning process already builds this closed-loop control policy. However, the proposed two-phases learning scheme, which is shown in Fig. 3, results in a much faster approach that tends to center its computational resources in the proximity of the state-action trajectory solution, just like in human beings learning procedure.

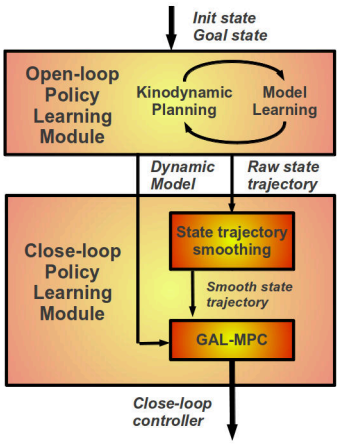

Fig. 3. Detail of the RoMPLA architecture.

The input is the initial state and the goal state. The output of the global architecture is a closed-loop controller capable of achieving the task even in the case of noisy environments causing perturbations to state transitions.

A. Open-loop Policy Learning Module

The Open-loop Policy Learning Module is focused in implementing the first main idea of the architecture, the mixed planning and model learning scheme that provides an initial raw open-loop policy solutions to the learning problem. The Kinodynamic Planning and Model Learning Algorithm (KiPLA) has been developed to face this problem. The planning process inside KiPLA is carried out by the Blind-RRT kinodynamic planner. As mentioned before, it is capable of quickly obtaining suboptimal plans even in high dimensional state-action spaces and under strong robot and environment kinodynamic constraints. Most remarkable innovation behind KiPLA is that it does not make use of an already known model for the planning process. It begins planning with an unknown model (randomly initialized) which result ends of invalid plans. Applying these plans permits to gather more data which is employed to update the forward dynamic model. For learning the model, the LWPR algorithm, already described in Section IV has been employed. Then, this planning-execution and model learning cycle is executed several times.

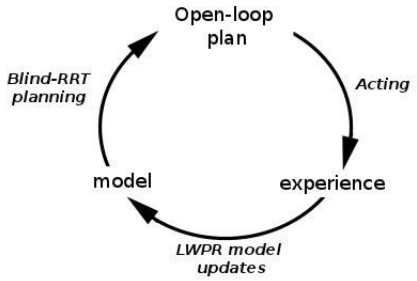

Fig. 4. Overview of the Blind-RRT learning algorithm.

Each time the planner is queried after the model has been updated with experimental data, the obtained plans make more sense as such model has been improved. Fig. 5 shows information of a certain episode during the learning process in the mountain car problem. In this episode, the forward dynamical model is still not mature enough to provide valid solutions. It is possible to appreciate this fact by looking at the differences between the planned state trajectory (yellow line), which ends at the goal state, with respect to the state trajectory followed by the car during the real execution (red line).

After some more episodes, the planner begins to provide valid plans which execution is capable of leading to congruent solutions. Then, both lines, the planned state trajectory and the real execution state trajectory, will match. This fact can be appreciated in Fig. 9, Fig. 10 and Fig. 11 of Section VI where results after the learning process has concluded are shown. This is due to the fact that the learned model is then

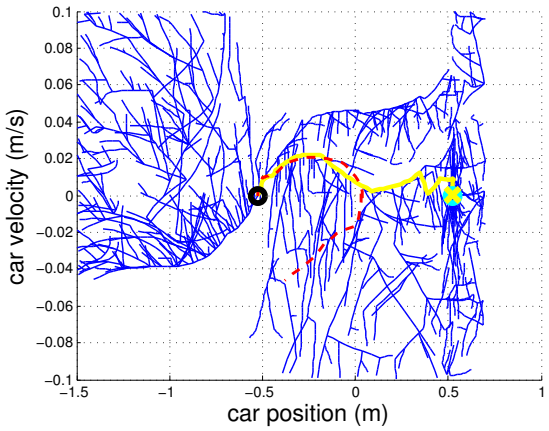

Fig. 5. Before KiPLA ends the learning process, there exist a divergence between the theoretical state trajectory and the real execution of the trajectory.

mature enough, at least for the given solutions. The most interesting feature regarding the learning process is that the exploration is indeed the result of exploiting the immature model. The main advantage here is that the trade-off between exploring and exploiting is implicitly guided.

B. Closed-loop Policy Learning Module

The second main idea of this architecture is embedded in the Closed-loop Policy Learning Module. After the mixed planning-learning phase that has taken place in Open-loop Policy Learning Module has provided an initial raw solution to the problem, this module will be in charge of refining it. In order to do so, such state trajectory solution is first smoothed using a minimum least square polynomial fitting procedure. Then, the Genetic Active Learning Model Predictive Controller (GALMPC) learns a closed-loop control policy in its proximity. Learning the closed-loop control policy implies that the controller explores the state-action space area in the proximity of the solution using an active learning motor babbling strategy which will be embedded in the fitness function.

1) State Space Trajectory Smoothing: After an initial solution has been obtained with the open-loop policy learning module, it is desired to smooth the state trajectory solution. An example is shown in Fig. 6. The initial trajectory solution is formed by a set of K state reference points:

$$traj = \{S_0, S_1, ..., S_K\},$$

where each of the state reference points are denoted as:

$$S_i = \{S_{i1}, S_{i2}, ..., S_{iN}\},$$

and i is the index representing the temporal order of these trajectory points and N is the number of dimensions of the state space. For achieving this trajectory smoothing, least squares polynomial regression is employed. The regression is applied over each of the dimensions independently, having each of the polynomial regression functions the following

form:

$$S_n(t) = a_0 + a_1 t + a_2 t^2 + ... + a_m t^m,$$

where n denotes the dimension number, t represents the instant where each of the state points are achieved and m the polynomial order.

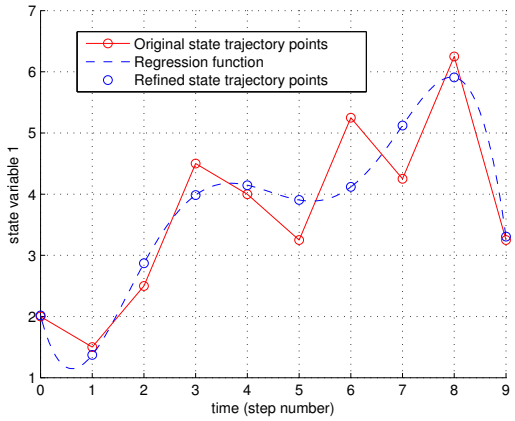

Fig. 6. The smoothing of the state reference trajectory solution is performed with a least squares minimization applied to a polynomial fitting.

A least square minimization procedure is then performed to calculate the monomials coefficients. Then, a new state trajectory may be obtained by resampling this function at the original instant points. The original initial and final points should be kept as they represent the starting and end conditions. By adjusting the polynomial order, it is possible to manage a less or more smooth state trajectory. The polynomial order, should always be kept below the number of trajectory points K. As a general rule, good results may be obtained with a polynomial order equal to a positive integer number between a 75% and 90% of the total number of trajectory points. Nevertheless, the best coefficient value should be related to the degree of smoothness of the state trajectory solution before the refinement process. If such trajectory is very rough, the degree should be tuned with a low value and vice versa.

2) GAL-MPC Controller: After the initial trajectory solution has been smoothed, it is desired to provide a closed-loop controller capable of tracking this trajectory even in the presence of stochastic behaviours that could deviate the robot from its reference. In order to do so, a motor control architecture combining ideas from active learning and model predictive control has been designed. The architecture is shown in Fig. 7.

In the first place, there is a state reference selection mechanism which chooses the current state reference nearest to the current state. Afterwards, an optimization engine employing a genetic algorithm and the learned forward model calculates the needed action with the ultimate goal of tracking the desired state reference. Due to this reason, the fitness in the optimization module is mainly responsible for describing the error between the state reference and the

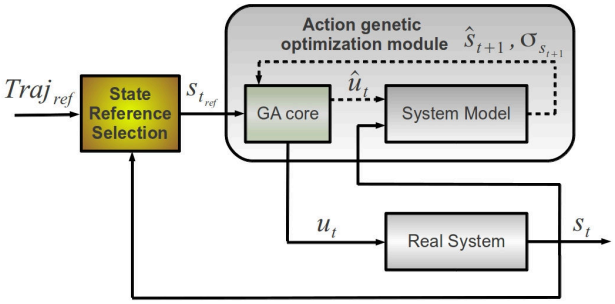

Fig. 7. Detail of the GAL-MPC block shown in Fig. 3.

prediction of the forward model. As mentioned previously, the model is not necessarily mature in the surrounding of the reference state trajectory. This may cause the controller not to work properly. Thus, it is desired to improve the model by exploring in such surrounding areas. An active learning approach will do this work. It has been encoded in the fitness function (second term of (4). This approach forces the system to choose actions that help in reducing the uncertainty in the model in the proximity of the smoothed trajectory solution. As long as the model is mature, the active learning approach will not affect the fitness value, thus the optimization takes into account only the tracking error. It is necessary to let this module work for some episodes before it is possible to track properly the refined state reference trajectory.

The objective of the optimization problem is to find the action u which minimizes the following cost function:

$$J = \| S_{t+1}^{ref} - \hat{S}_{t+1} \| - K\sigma(\hat{S}_{t+1}) D(\hat{S}_{t+1} + 2\sigma(\hat{S}_{t+1}), S_{t+1}^{ref}) \quad (4)$$

where $\hat{S}_{t+1}$ is the prediction of the model when being in a certain state S_t and applying an action u:

$$\hat{S}_{t+1} = f(S_t, u)$$

The term $\| S_{t+1}^{ref} - \hat{S}_{t+1} \|$ is the expected error when applying the action u. The term $\sigma(\hat{S}_{t+1})$ represents the confidence in the prediction. If the model is mature, then the standard deviation of the prediction will tend to be zero, making the whole term null. Therefore, the controller will be focused in minimizing the tracking error. If the model is immature, that means the standard deviation of the prediction is greater than zero, the term will tend to minimize the total cost, thus making the controller explore to reduce the uncertainty. However, the function D will balance the importance of exploring the evaluated action, discouraging the exploration process if such action could bring the robot far way from the reference state trajectory, even if the uncertainty in the prediction is high. The K parameter may be tuned to decide how far from the state reference trajectory should the algorithm explore. This function follows a multivariate normal distribution:

$$D \sim \mathcal{N}(S_{t+1}^{ref}, diag(d_1, ..., d_n)),$$

which is centred in the the current state reference point. The covariance matrix is diagonal and has the parameters d_1, ..., d_n which weights the importance of being deviated from each of the n state variable values.

VI. Experimental Results

The Cart-Pole Swing-Up Task problem is very similar to the Inverted Pendulum Swing-up Task. Both systems are classical problems in the motor control research field. The cart pendulum is an inverted pendulum over a cart system with an horizontal degree of freedom. Unlike the problem of the Cart-Pole Balancing Task problem, the Swing-up Task problem is much more difficult as it involves searching for a complex policy in a large state-space area. One of its main advantages is that it is widely used as a benchmark for testing control algorithms. In the original problem, the joint that connects the pendulum with the cart is passive and it is only possible to exert an horizontal force on the cart to move it left or right.

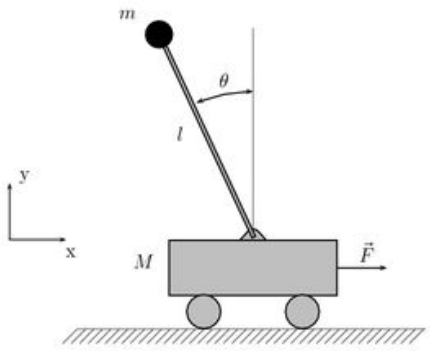

Fig. 8. Cart pendulum system.

Relevant system variables are contained in a four dimensional state space and one action space dimension. The simulation of the dynamics has been done using (5) and (6). These equations and their complete development can be found in [13], where Coulomb friction and viscous friction are also taken into account. We highlight that these equations are a correction of other models that have been previously used in literature. The state vector $X(\theta, \dot{\theta}, \dot{x}, x)$ represents the angular position of the pendulum, its angular velocity, its linear velocity and the linear position of the cart. The action vector is only formed by the linear force, F, exerted over the cart.

$$\ddot{\theta} = \frac{gsin(\theta) + cos(\theta)[\frac{-F - m_p l\dot{\theta}^2 sin(\theta) + \mu_c sgn(\dot{x})}{m_c + m_p}] - \frac{\mu_p \dot{\theta}}{m_p l}}{l[\frac{4}{3} - \frac{m_p cos^2(\theta)}{m_c + m_p}]} \tag{5}$$

$$\ddot{x} = \frac{F + m_p l[\dot{\theta}^2 sin(\theta) - \ddot{\theta} cos(\theta)] - \mu_c sgn(\dot{x})}{m_c + m_p} \tag{6}$$

The cart has mass m_c, and the pendulum has mass m_p and length l. Coulomb friction with constant value μ_c and viscous friction with constant value μ_p have also been taken into account. Gravity is represented by g. The initial state, in this case, corresponds to the pendulum downwards, at $-\pi$ rad, and the cart at linear position $x = 2$ m. Both the angular velocity of the pendulum and the linear velocity of the cart are initially null. The objective is to swing-up the pendulum vertically, at 0 rad, with null angular velocity. In addition, it is desired the linear velocity of the cart and the horizontal position to be zero.

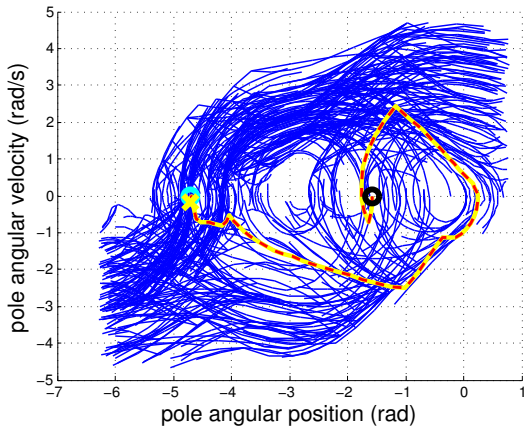

Fig. 9. Projection of the planning tree in the cart-pole system onto the angular velocity of the pole and the linear position of the cart variables.

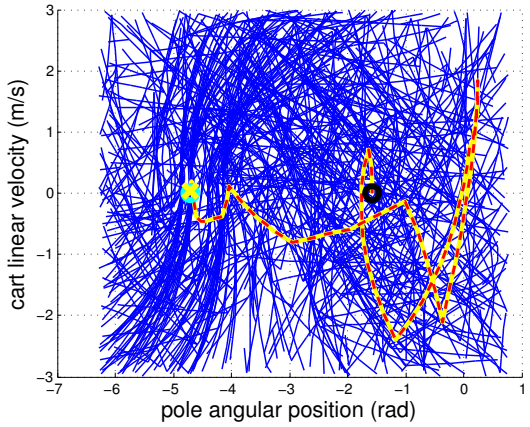

Fig. 10. Projection of the planning tree in the cart-pole system onto the angular velocity of the pole and the linear position of the cart variables.

After some iterations, the learning phase concludes, which means that the execution of the chosen plan leads the system along a state trajectory that matches the reference trajectory (the theorical trajectory associated with the chosen plan).

Fig. 9, Fig. 10 and Fig. 11 show the state-space planning tree and the state-state trajectories associated with the execution of the plan. Here, the continuous yellow line represents the reference trajectory (using the theoretical prediction model) and the red discontinuous line the followed trajectory (real execution). In addition, Fig. 12 shows the evolution of each of the state variables over time. It is possible to

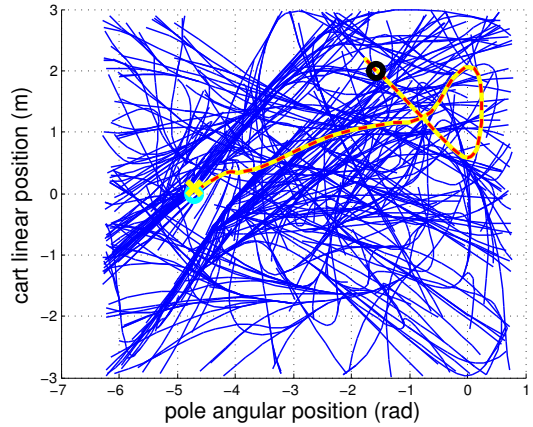

Fig. 11. Projection of the planning tree in the cart-pole system onto the angular velocity of the pole and the linear position of the cart variables.

appreciate that both lines coincide and end in the goal state position. This fact illustrates that the learning process has concluded.

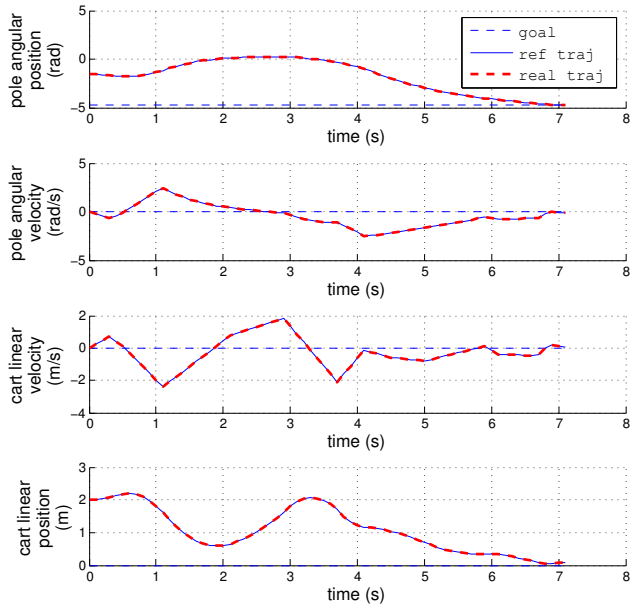

Fig. 12. Temporal response of the cart-pole system.

A graphical sequence of the cart-pole system achieving the goal has been represented in Fig. 13. Due to the imposed constraints in the maximum torque that could be employed, the policy conducts the pole to balance first to the right side to accumulate potential energy. Then, the system takes advance of this extra energy to manage reaching the goal by balancing to the left side. This behaviour is due to the constraints of the problem, (i.e. the limitation in the applied force and the limits in the cart displacements).

A summary of the learning process corresponding to the first phase, carried out by the KiPLA algorithm, is illustrated in Fig. 14. The horizontal axis represents the number of iterations along the learning process. The vertical axis represents the module of the final state error vector. Data have been gathered every 25 iterations. The graph represents not only the average error in each of the 25 episodes period but also the minimum and maximum error.

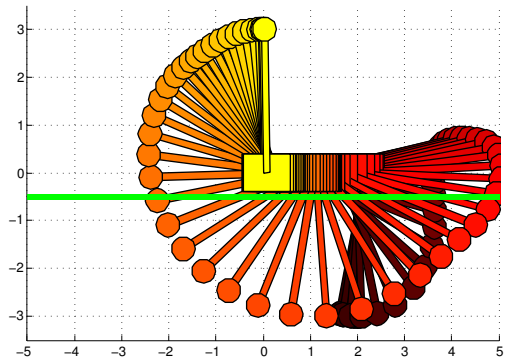

Fig. 13. Graphical sequence of the cart-pole system achieving the goal.

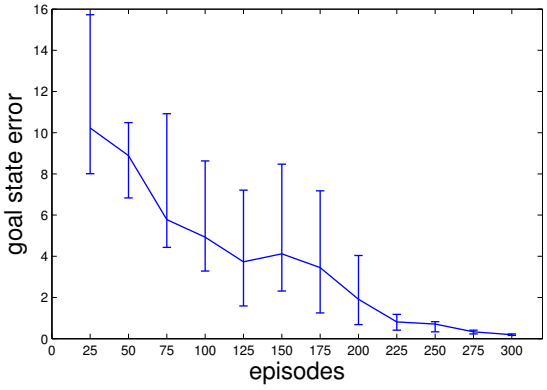

Fig. 14. Final state error in each iteration along the learning process performed by the Open-loop policy learning module.

After the initial solution has been obtained within the first phase, the Close-Loop Policy Learning Module refines the solution and then a new learning process using the GAL-MPC controller begins. The tracking error, which is computed as the mean square error between the current state and its current reference, is shown in Fig 15. At the beginning of such process the error is again large, due to the fact that the smoothing process is forcing the controller to search in slightly different state space areas where the model is not mature enough. Nevertheless, this error rapidly decreases.

The large maximum errors do not correspond to tracking errors, but to complete failures in the tracking tasks. This is due to the fact that exploring becomes very risky in some situations, as the systems are not completely controllable, mainly due to the passive joint and maximum force torque constraint.

VII. CONCLUSIONS AND FUTURE WORK

This work has proposed a general and new original architecture to allow complex robotic systems to learn closed-

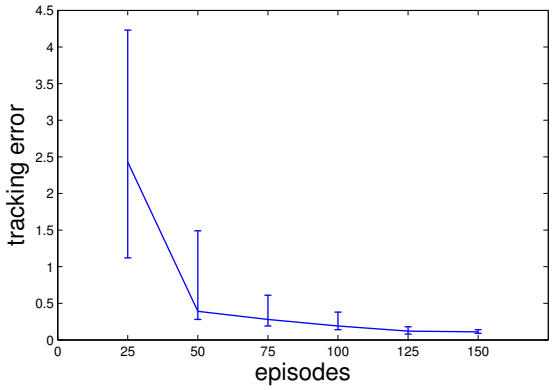

Fig. 15. Tracking error error in each iteration along the learning process performed by the Open-loop policy learning module.

loop motor control policies for accomplishing new tasks without assuming prior knowledge. The approach, which is inspired by motor learning in humans, manages this learning process in two different phases. The first phase allows to quickly search for valid open-loop policies. While this phase is achieved with the KiPLA algorithm, the most remarkable feature is that mixing kinodynamic planning and model learning aids at handling the curse of dimensionality problem better than Dynamic Programming based learning techniques. The second module, which core concept is implemented with the GAL-MPC controller, is focused on refining the solution and learning a closed-loop control policy robust to disturbances. The new proposed active learning strategy, embedded in the cost function of such controller, seems to be simple and capable of balancing the exploration-exploitation dilemma.

While preliminary comparisons with other methods hint that the architecture seems to be promising, an exact comparative study with state-of-art methods needs still to be performed. There are several other issues related to the implementation that can be improved. For instance, by providing KiPLA with methods to use probabilistic information in the criteria for choosing the policies to be executed in each of the planning-learning cycles. This information would help to decide which action sequence should be chosen in order to improve the model in the regions of interest using less trials. Finally, an evaluation of RoMPLA on a real experimental platform is required for the purpose of demonstrating its efficiency and its potential.

VIII. ACKNOWLEDGMENTS

The research leading to these results has been partially supported by the HANDLE project, which has received funding from the European Community's Seventh Framework Programme (FP7-231640)

REFERENCES

[1] *Handbook of child psychology*. Wiley, 2007, ch. Motor Development.
[2] D. M. Wolpert, "Probabilistic models in human sensorimotor control." *Human movement science*, vol. 26, no. 4, pp. 511–24, Aug. 2007.
[3] S. Schaal and C. G. Atkeson, "Learning control in robotics," *IEEE Robotics and Automation Magazine*, 2012.
[4] *Dynamic programming*. Princeton University Press, 1957.
[5] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
[6] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, England, 1989.
[7] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *SIGART Bull.*, vol. 2, pp. 160–163, July 1991.
[8] J. Gonzalez-Quijano, M. Abderrahim, F. Fernandez, and C. Bensalah, "A kinodynamic planning-learning algorithm for complex robot motor control," in *Evolving and Adaptive Intelligent Systems (EAIS), 2012 IEEE Conference on*, 2012, pp. 80–83.
[9] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space," in *In Proceedings of the Seventeenth International Conference on Machine Learning (ICML*, 2000, pp. 1079–1086.
[10] P. Robbel, "Active learning in motor control," Master's thesis, School of Informatics (University of Edinburgh), 2005.
[11] S. Vijayakumar, D. A., and S. Schaal, "Lwpr: A scalable method for incremental online learning in high dimensions," Edinburgh University Press, Tech. Rep., 2005.
[12] D. W. Franklin and D. M. Wolpert, "Computational mechanisms of sensorimotor control," *Neuron*, vol. 72, no. 3, pp. 425–442, Nov. 2011.
[13] R. V. Florian, "Correct equations for the dynamics of the cart-pole system," Center for Cognitive and Neural Studies, Tech. Rep., 2007.