Online learning of control policies for dynamical systems based on input/output data recorded on the fly

Roberto A. Bunge

***Abstract*—A computationally efficient learning algorithm is presented which learns an adequate control policy to drive a dynamical system from an intial state to a final desired state. It does this by taking actions, storing the reward received, and using the bank of recorded data to estimate the best action in future, possibly unvisited, states. A natural threshold separating positive and negative actions is set, indicating if random exploration of the action space or if exploitation of the accumulated data should be undertaken, thus ensuring that the goal is always approached. The algorithm is limited to problems where maximizing the reward at each time step leads to a good long term performance. The designer is required to set a number of parameters which depend on the scale of the different variables involved, as well as providing a reasonable path planning function. Results on three control problems are presented showing that the algorithm reaches near to optimal control policies, and that it can handle undamped inertial transient dynamics, as well as cross coupling of control inputs.**

I. Introduction

In this project we explore the problem of learning an adequate control policy so as to drive a dynamical system from an initial state to a final desired state, using only input/output data recorded on line, and without estimating an explicit model for the system, be it a deterministic linear representation or a transition probability model. The motivation for this goal has different sources. In the first place, estimating a linear representation assumes that the system, to some extent, has close to linear dynamics. This may not be the case in many problems, and if it were, we would still rely heavily on the designers knowledge about specific aspects of the system to be effective in generating a good control algorithm. Secondly, we want to generate a framework that is based on generic concepts, such that the architecture can be scaled and expanded to different cases, without changing the general setup too much. Thirdly, trying to estimate a transition probability model, requires that we be operating in a specific region of the state space, so that the system is going over the same states quite often. Since we are envisioning applying this algorithm to navigation problems, rather than strictly regulation problems, by definition we expect to traverse the state space making this approach ill conditioned.

In addition, this work is an attempt to resemble certain aspects of how humans learn to control systems "on the fly", this is: by testing controls, memorizing good ones and using these to infer good controls for new situations. A very important question arises about human control: Is there an explicit function minimization or function approximation instance in the human brain that is central to our effectiveness to drive systems? Our intuition is that no, humans don't rely on a mathematical representation, but instead a very malleable and gapless system of guessing and checking, mixing and matching, and an ability to recall from memory the previously generated mappings between situations, actions and rewards. We are going to try to follow these overarching concepts in the present work, striving to use mathematical tools that resemble these features.

One of the restrictions for the approach here presented, is that it is limited to problems where the optimal route to the goal is that of the "shortest" route. Since the agent is going to be designed so as to maximize a given reward at every time step, then, problems where an initial period of low rewards is required in order to ever reach the goal, would not be effectively solved by this approach. Still, the subset of problems for which it could work is representative of many interesting control problems.

The proposed solution has been applied to three different control and navigation problems: positioning a mass-spring-damper in 1D controlling the force applied; positioning a free mass in 2D controlling two orthogonal forces; driving a 2D tank from a given position to a final one.

A. An overview of the architecture

The agent will be fedback a scalar reward after every action it takes from every state it is in. This State-Action-Reward data will be stored in a "bank of previous experiences". At each new state, the agent can try to estimate the reward for all possible actions from the data it has previously collected. If the new state isn't too "different" from the previous states it has been in, then it might be able to infer quite well the reward for all the actions. Doing this estimate, it can pick the one that, surpassing a given threshold, has the highest estimated reward. If there is none that surpass the "goodness" threshold, the agent decides to pick an action randomly with the intention of exploring the State-Action-Reward space, thus expanding its bank of knowledge about the problem. Thus, we have instances of: reward estimation, reward evaluation, taking an action, receiving a true reward, and finally storing this information in the data bank for future use. Below is a diagram which shows this.

In the coming sections we will discuss with more detail our proposed reward function, our reward estimation strategy and how we will deal with the expanding size of the data bank.

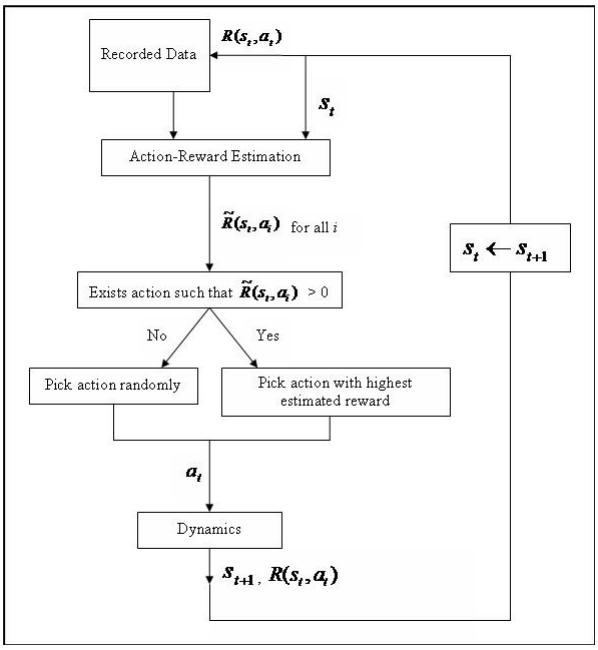

Fig. 1. Overview of algorithm

II. Designing the reward function

As already mentioned, the agent has to be provided with a reward function, which when maximized at each time step gives a good long term performance. To fulfill this objective, the reward function has a path planning component embedded in it. So, for each state, we have a path planning function which indicates what is the desired next state:

$$s_{t+1}^{desired} = f_{plan}(s_t) \tag{1}$$

Hence, at every time step there is an error with respect to the future desired state:

$$\epsilon_t = s_t - s_{t+1}^{desired} \tag{2}$$

After an action has been taken the system transitions to a new state, and we can asses the effectiveness of the action by comparing the magnitude of the error before and after the action. We add a diagonal matrix with positive entries so as to compensate for scale in different error terms and also to incorporate some priority as to which state variable we want to follow more. Following this, we build the reward function:

$$R(s_t, a_t) = \epsilon_t^T D \epsilon_t - \epsilon_{t+1}^T D \epsilon_{t+1} \tag{3}$$

A very natural threshold, helpful in discriminating good and bad actions, arises from this reward function: if the reward is positive, then the error has been reduced, if it is negative the error has grown. Following this, the threshold is set equal to zero. This structure for the reward function is general in the sense that it can be applied directly to any system. One draw back of this reward function and the threshold chosen, is that it works very well when we are far away from the desired final state, but has a singularity when we are exactly on desirde final state. Lets suppose, we somehow reached the final desired state, then the first term is zero, and the optimal action would be to take no action (or take the neutral action), so as to remain in the same state, but this makes the reward be exactly zero, so given the threshold chosen, this optimal action will never be taken. On the flip side, if we allow zero to be part of the acceptable actions, there is a high chance of ending up in a situation where the neutral action is taken forever, since the neutral action by definition gives a zero reward, and the bank will never be expanded so as to find an action with a better estimated reward. This could be dealt with in different ways. One is to include a non-zero probability of taken a random action regardless of the estimated rewards, thus keeping the door open for new information coming in, which will hopefully indicate towards an action that is better than the neutral one. This option is not favored because it adds an instance of unnecessary randomness, and the agent may still be slow in getting out of this singularity. Another option is to redefine the reward function as:

$$R(s_t, a_t) = \begin{cases} \epsilon_t^T D \epsilon_t - \epsilon_{t+1}^T D \epsilon_{t+1} + 1 & \text{if } \epsilon_t = 0 \\ \epsilon_t^T D \epsilon_t - \epsilon_{t+1}^T D \epsilon_{t+1} & \text{otherwise} \end{cases} \tag{4}$$

In addition, since its almost impossible that the final desired state will be exactly reached, then we will set a sphere about the desired state, which will be our goal. Thus, the path planning function is modified to:

$$s_{t+1}^{desired} = \begin{cases} s_t & \text{if } \parallel s_t - s^{desired} \parallel \leq r \\ f_{plan}(s_t) & \text{otherwise} \end{cases} \tag{5}$$

This modification in the reward and path planning function should prompt the agent to finally rest at some point which we consider near enough the final desired state.

III. Estimating the reward function

There are different ways of estimating some value from collected data. The one that seemed more general, robust and malleable was using a weighted linear regression approach. This method manages in a very natural way interpolation and extrapolation, allowing the agent to take large adventurous steps from regions that have been explored, to possibly unexplored ones. As the agent acts, the region is populated, so WLR gradually disregards the "far away" data and pays attention to the data nearby. Another advantage is that it doesn't put too much pressure on the specific features we pick to make a precise estimation, only that these features be indicative in some way of the estimated variable. In this architecture we pick state errors and actions as a basis functions, thus releasing pressure from de designer to pick "the correct" features for this estimation. Obviously, if the designer provides better features than these, the estimation will be more precise and the learning faster.

A covariance matrix is used in the WLR algorithm, accounting for scale and variability in each of the dimensions. This matrix in a way indicates how fast the "paying attention" factor decays with distance to the current point. It also gives the shape of the decay function. We will pick a diagonal matrix, since we won't require the designer to have a priori knowledge of which directions should be treated specially. The higher the value of the diagonal entries, the farther away we look for information in that dimension. The smaller the value, the more we disregard far away information. So, if we have nearby data, setting a small value will lead to more precise estimations. On the flip side, if we reach a region for which we've got no nearby data, a small value will mean, that we won't be able to get anything out of the far away data we have collected. So, a compromise has to be reached for these values. Setting them to anywhere between 100% and 25% of the expected range for each of the variables involved has shown to work well.

IV. Managing the growing size of the data bank

The problem with the WLR approach is that being non-parametric, it carries and goes over every data point for each estimation. As time goes by, the size of the recorded data grows exponentially, so each estimation gets more and more expensive. This problem will be alleviated by implementing the following reasoning. First, for every new data that comes in, we'll check what is the discrepancy between the estimated reward and the actual true reward. If the estimated reward is very similar to the true reward (which is something that happens often), then it means that adding this data would be in a way redundant, so we don't need to keep it. Secondly, after some indication of learning performance has been reached, for every step we make (and consequently every new data we record) we will delete one of the previously recorded data points, thus stabilizing the size from that moment on. Before deleting, we will first do the redundancy check just mentioned. If the data isnt redundant, then we delete a point randomly and keep the knew one. This scheme turns out to be efficient in the long run at keeping the data size small while preserving en even distribution of data points across all the traversed State-Action space, thus ensuring a good learning perforamce.

V. Applied problems

A. Positioning a Mass with spring and damper in 1D

We presente here results on the classical mass-spring-dampere system. The general set up for this case is given by:

$$s_t = \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix}$$

$$D = \text{diag}([1\ 10])$$

This prioritizes velocity tracking over position tracking.

$$f_{plan}(s_t) = s_{t+1}^{desired} = \begin{bmatrix} 0 \\ -x_t \end{bmatrix}$$

This plannig function makes the rewards function comply with the general requirement we stated above. The features and the covariance used in the WLR were:

$$\phi(s) = \begin{bmatrix} x - x^{desired} \\ \dot{x} - \dot{x}^{desired} \end{bmatrix}$$

$$\Sigma = \text{diag}([2\ 5\ 2])$$

The entries in Σ correspond to: error in position, error in velocity and force applied, respectively. The action space has been discretized to {-2, -0.2, - 0.1, 0, 0.1, 0.2, 2}. This allows for powerful actions as well as some fine regulation.

We approached this problem for two choices of ζ and w_n, the damping ratio and natural frequency respectively. The first is $\zeta = 0$, $w_n = 0$, while the second one is $\zeta = 0.5$, $w_n = 1$ For both we have allowed for reinitialization of the trial once the goal has been reached, or after some maximum time has passed, allowing for learning to occur.

Below is shown the trayectory evolution for the first case. It celarly shows how the agent learns from previous trials, making its control policy better.

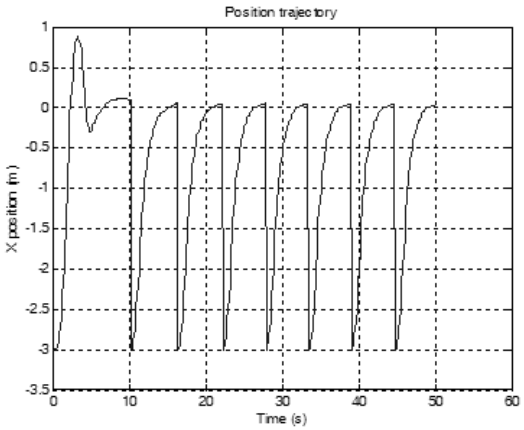

Fig. 2. Subsequent trials for the free mass case ($\zeta = 0$, $w_n = 0$)

Below we have plotted for both cases the learnt trajectory, with the controls policy overlayed on each of the plots. In order to benchmark the learning capabilities, we have calculated separately the optimal trajectories and control sequences from the knowledge of the linear system using the same metric: trying to maximize the reward function at every step. So, for each of the cases we show alongside the optimal trayectory and optimal controls.

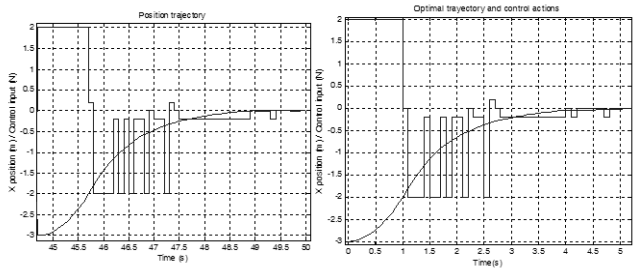

Fig. 3. Learnt and optimal trajectory and control policy for ($\zeta = 0$, $w_n = 0$) case. Learnt left, optimal right.

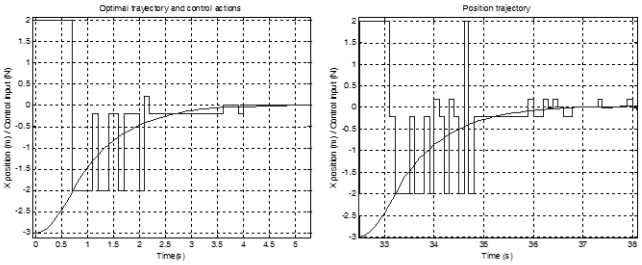

Fig. 4. Learnt and optimal trajectory and control policy for ($\zeta = 0.5$, $w_n = 1$) case. Learnt left, optimal right.

For the free mass case, its outstanding how similar the agent's learnt trayectory and control sequence is compared to the optimal. It can be seen how it first accelerates the mass, and as the objective is approached then controls are reverted to bring it smoothly to a stop. Except for some really minor differences we could say they are exactly the same. This example shows how the architecture proposed is indeed capable of finding the optimal policy, with respect to the defined reward function. In the damped-oscillatory case we can also see an outstanding similarity between the learnt and the optimal trajectory and control sequences. Overall, these reults show that the architecture is capable of dealing effectively with damped inertial transient dynamics, obtaining a near to optimal policy. The algorithm has also been tested for the csae of pure osillatory dynamics ($\zeta = 0.5$, $w_n = 1$), showing similar results, wich are not included here for brevity.

We also note here that the strategy to stabilize the size of the data bank has been successfull, since it allowed us to simulate many trials rapidly (less than 10 seconds, which is bearable for the designer), while preseserving enough information for learning to occur.

B. Positioning a free mass in 2D

To test how the approach reacts to scaling, we show the same problem with an extra dimension (i.e. we added a force in the y direction). The parameters were expanded accordingly to accomodate the change. The part that will be suffering most is the WLR, since adding another 3 dimensions will put more stress on it. Below we show a run with a few trials.

In its first attempt, the mass is wiped around the objective. As trials advance, the route taken converges on the optimal direct route.

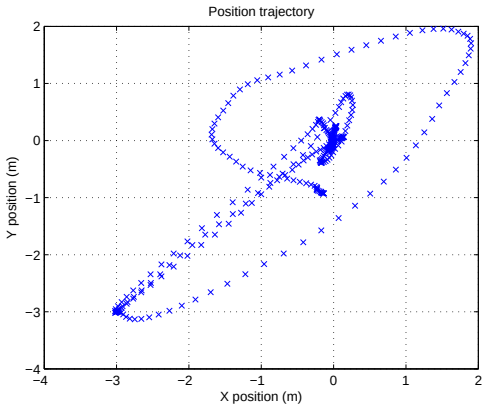

Fig. 5. Different trials of 2D positioning of a free mass.

C. Driving a tank in 2D

Lastly, we present a case in 2 dimensions where we have a strong cross coupling of control inputs, requiring coordination to achieve the goal. This is the case of the 2D tank. The equations of motion for this system are:

$$\begin{cases} V = \omega_1 + \omega_2 \\ \dot{x} = V sin(\theta) \\ \dot{y} = V cos(\theta) \\ \dot{\theta} = \omega_1 - \omega_2 \end{cases}$$

Where V and θ are the forward speed and heading angle, respectively, and ω_1 and ω_2 are the left and right track speeds, which are the control inputs. By setting these equal, we move in a straight line, by setting them exactly opposite we turn without changing position. The state and path planning function for this problem were:

$$s_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix}$$

$$f_{plan}(s_t) = s_{t+1}^{desired} = \begin{bmatrix} 0 \\ 0 \\ atan(x_t/y_t) \end{bmatrix}$$

The rest of the parameters were set to:

$$D = \text{ diag}[1\ 1\ 25]$$

This choice of D accomodates for the fact that the angle is in radians and the positions are in the order of 8 meters.

$$\Sigma = \text{diag}([2\ 2\ 3\ 0.5\ 0.5])$$

For the WLR, the base variables were: error in x, error in y, error in θ, and the two control inputs ω_1 and ω_2. Below is shown a run with successive trials.

In the first attempt, the agent takes a zigzagy route, but eventually seems to be heading towards the origin. Already

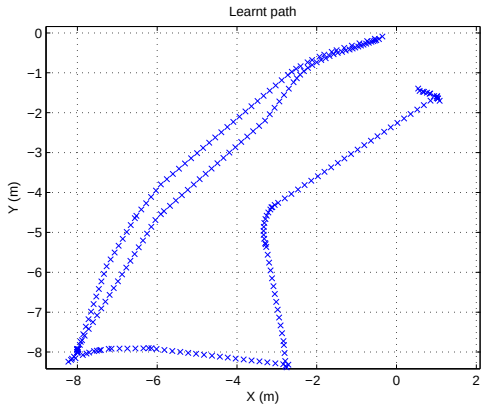

Fig. 6. Different trials for driving a 2D tank to the origin.

in the second and third trials the agent has learnt a route close to the optimal direct route. This case shows that the architecture handles dynamical systems with cross coupled control inputs.

VI. Conclusion

The learning architecture here presented has proven effective in learning adequate control policies for a variety of problems, by simply interacting with the environment and exploiting past experiences in a natural manner. The specifics of the dynamical system are coped with in a generic and robust way, dealing effectively with cases of undamped inertial transient dynamics, as is the case of the 1D and 2D mass, and cross coupled controls, as is the 2D tank. In many cases the algorithm converges to a control policy that is close to the optimal policy, reflecting a good learning quality. Although for every problem a number of parameters have to be set by the designer, these only require intuition of scale for the given problem and not hard technical knowledge. Issues of computational efficiency have been dealt with effectively by the scheme presented of checking for redundancy and deleting data points randomly for each new one that is added, after a certain level of learning has been reached. This has reduced notably the computational cost, withou affecting the learning capabilities of the algorithm. The 2D mass problem is an indication that scalability to higher dimensional problems might be possible.

VII. Future work

Future work on the topic will involve extending the architecture to higher dimensional problems as is the control of a fixed wing aircraft. Adding a "soft" action picking, where there is a probability of picking one of the candidate actions (those that surpass the threshold), might make the agent converge faster to the optimal policy, and would also protect it from singular bad sequences it can't get out of. Also, adding longer time horizons and optimizing with respect to a sum of rewards should definitely be the next step, since this will allow for problems where initial negative rewards are actually optimal, and will also alleviate the necessity of a reasonable path planning function that has to be provided by the designer.

VIII. Acknowledgements

I would like to thank Prof. Ng for his always kind attention to my inquiries and for setting up this outstanding course on Machine Learning. This is my first formal exposure to it and it has left me wanting more. I'd also like to thank the staff for their predisposition to make these projects a success, especially Quoc Le, for always being attent to my questions and needs. I would finally like to thank Adam Coates for his advice, and also for pointing me in different interesting directions I could take this research effort in the future.