Robot learning for ball bouncing

Denny Dittmar
Denny.Dittmar@stud.tu-darmstadt.de

Bernhard Koch
Bernhard.Koch@stud.tu-darmstadt.de

Abstract

For robots automatically learning to solve a given task is still an hard job. Many real world challenges consist of complex movement, which has to be done in coordination with other objects or persons in order to achieve a required performance on a given task. In order for robots to be able to correctly behave in such environments it is important to use efficient controlling and learning algorithms that can cope with high dimensional action spaces and achieve good results after a relatively short time using only indirect feedback of the performance of the robot, a so called reward. In this paper we compare two controlling approaches with specific learning parameters in order to perform complex movements of an robot arm with seven DOFs and apply both of them to a learning algorithms. The task that has to be solved is to produce a stable bouncing of a ball with paddle as the end-effector of a manipulator. The first controlling approach tries to control the endeffector by defining and following a smooth polynomial trajectory in each step of the task. The second approach is to use Rhythmic Movement Primitives or RDMPs for short. For the learning algorithm we use the PoWER method.

1 Introduction

The task for the robot arm is to continuously bounce an free-floating ball on a racket attached to its endeffector for . As an additional difficulty there is a damping of the ball every time it hits the racket. To solve this task it is required to have a fast and precise controlling of the endeffector to an fast moving ball which can only be controlled at each hit with the rack. Thus this task can be considered as non-trivial. Solving this task efficiently is an interesting research topic as it requires fast learning in complex mostly uncontrolled environments. To calculate the movement of the arm so it can hit the ball with the best trajectory is done through two different approaches. The first approach is a handcrafted controller using more complex expert knowledge of the whole task to calculate desired states of the racket at certain times and cover these states by smooth polynomial trajectories. The second approach is to use RDMPs to create an rhythmic movement which is able to reliably hit the ball so that we are able to bounce it. The performance of both approaches is initially bad especially for the RDMP approach as they require parameters to be learning for each specific task. So as an learning algorithm we are using the PoWER method as it showed the best results in comparison to other methods such as eNAC or RWR [KP11].

Preliminary work. Under review by AISTATS 2012. Do not distribute.

2 Movement

For moving the racket mounted to the endeffector of the arm we have chosen to compare two completely different approaches. This is done to compare an general framework for controlling the movement of robots such as RDMPs with an handcrafted policy which is using expert knowledge of the task.

2.1 Controlling using smooth polynomial trajectories

In this controlling approach at the beginning of each new bouncing step, that is at the beginning of the task and every time the racket hits the ball, desired values for the position, velocity and orientation are chosen for the next moment, when the racket hits the ball, in order to get an optimal behavior of the ball, that is a desired ball velocity, after the racket has hit the ball. Since there are many possibilites for the next hit position, these are restricted to lie on a imaginary 2-d-plane, which is predefined by a height. Together with the current ball velocity it can be computed, where

the ball hits this plane next. This position then is desired to be the position, where the racket and the ball should collide.
As for the hit positions, there might also be many different velocities of the ball, that could result from the hit and so the velocities have to be restricted, too. The desired velocitiy of the ball is chosen, such that the next collision of the ball with the racket lies in an area, where the end-effector has much degrees of freedom with corresponding buffers for all these. Ideally this would be somewhere in the middle of the space that can be reached by the end-effector easily. This desired point was set by a learning learning parameter(Θ_1) that set the y-value of the point with z given by the plane height and $x = 0$. To achieve a good resulting ball velocity that should have a low deviation to the desired ball velocity after the collision it is necessary to make sure that the paddle has an appropriate velocity and orientation, when it collides with the ball. This values also depend on the velocity the ball hits the paddle with.
The orientation of the racket at the hit is chosen, such that both the velocity the ball hits the racket with and the desired velocity only possibly differ in normal direction of the paddle corresponding to reflection law assuming that the racket rests, i.e. has no zero velocity, and there is no damping etc... To compensate this velocity difference in normal direction and to take care of damping of unknown size the racket velocity in this direction is set to a value, such that the following chosen model equation holds for all these 1-dimensional velocities in racket normal direction:

$$v_{ballout} = v_{racket} + d(v_{racket} - v_{ballin}), \qquad (1)$$

with d as damping. Thus it follows for the desired setting of the racket velocity:

$$v_{racket} = \frac{v_{ballout} + dv_{ballin}}{1 + d} \qquad (2)$$

Since good values for d are a-priori not known, d here was choosen to be learning parameter(Θ_2) of this control method based on this given model.
After the time and desired racket state at the next collision with the ball have been calculated, there has to be a movement of the racket that makes sure, that the racket has this desired state, when the ball hits the plane again. In order to avoid controlling difficulties this movement is desired to have a smooth trajectory with steady velocity, and to try to return to an basis point under the plane in somewhere in the movement, such that the racket has the possibility to accelerate to a desired velocity before it hits the ball. That means there are 3 given points of the trajectory, namely starting state, some intermediate basis state and the end state, where the racket should hit the ball.
One possibility for such a smooth trajectory are linked polynomials, that are used in this approach. They were used by finding 2 polynomials, such that the first one goes from the current starting state through the intermediate basis state and the second from the intermediate basis state to the end state with the desired velocities. The polynomials also wer constraint to have equal velocities at the basis point, where they are connected. This desired velocity at the basis point was chosen such that it is directed to the desired collision point. The amount of the velocity was not fixed and instead used as settable learning parameter(Θ_3) together with the ratio of the time amounts of both trajectories(Θ_4), which have to sum up to the remaining time until the next collision.
Furthermore it is much more convenient to represent this desired states of the racket not in the task space, but in the joint space and to define the trajectory there instead. One heavy advantage of doing so is that for controlling it is not necessary to call an inverse kinematics method all time to find the corresponding joint configuration for current task state given by the trajectory at this moment. Instead the inverse kinematics method is only called three times per step in order to get the joint states for the desired start point, the basis point and end points of the resulting trajectory.
Since for both of the 2 linked trajectories it must hold that they start at some desired start point and end at some desired end point with desired velocities there are for each trajectory 4 constraints for each joint given by its angle and angle velocity at the beginning and at the end of the trajectory. Thus a polynomiala of degree 3 are sufficient to represent these one of both trajectories for each joint. Such polynomials have been used for this controlling method.

2.2 RDMPs - Rhythmic dynamic movement primitives

To create complex movements for robots dynamic movement primitives, DMPs for short, have been shown to be quite versatile [SMI07]. With them it is possible to create complex non-linear trajectories which can be easily adapted to different goal positions and can be scaled with the time to create slower or faster movements. As the movement we need to be able to bounce the ball is an rhythmic one we use the rhythmic variation of the DMPs [SPNI04]. Rhythmic dynamic movement primitives or RDMPs for short

consist of three equations. An transformation system,

$$\tau\dot{z} = \alpha_z(\beta_z(g-y)-z), \tau\dot{y} = z + f \tag{3}$$

where g is the setpoint of the oscillation of the rhythmic movement, α_z and β_z are time constants, τ is the temporal scaling factor to speed up or slow down the movement and $y, \dot{y}$ are the desired position and velocity. The second equation is the canonical system, in this case an phase-amplitude oscillator,

$$\tau\dot{r} = \alpha_r(A-r), \tau\dot{\phi} = 1 \tag{4}$$

where r is the amplitude, A the desired amplitude and ϕ the phase of the oscillator. The last equation needed is the non-linear function

$$f(r,\phi) = \frac{\sum_{i=1}^{N} \psi_i * w_i^T * v}{\sum_{i=1}^{N} \psi_i} \quad \text{with} \tag{5}$$

$$v = \begin{bmatrix} r*\cos\phi & r*\sin\phi \end{bmatrix}^T \quad \text{and}$$

$$\psi_i = \exp((-h_i(\phi \mod 2\pi) - c_i)^2)$$

where the w_i are the weights which essentially describe the trajectory. So the learning parameters for these equations are the τ for the time scaling and the weights w_i for the actual trajectory. For our setup we only use one RDMP and have therefore only one controllable DOF in the arm. The reason for this is so that we have fewer parameters which need to be learned and therefore an smaller learning space.

3 Learning methods

Both approaches described in this paper need to be adapted to specific movements of the ball. For example an slow moving ball requires slower movements of the arm in comparison to an fast moving ball. This is especially true when the initial position and initial velocity of the ball is changing with each iteration of the simulation. This adaption is done through learning parameters which need to be automatically fitted to the task. For this comparison we are using the PoWER learning algorithm.

3.1 PoWER - Policy learning by Weighting Exploration with the Returns

This learning algorithm is based on updating the learning parameters by doing multiple rollouts of the policy which should be learned with the same learning parameters each time disturbed by an small exploration value. The rewards for each of these rollouts will then be sorted and the X best rollouts will then be used to calculate the parameters for the next iteration of rollouts. This calculation is done by weighting the exploration values which were applied to the parameters for each rollout with the result of the Qsat and the Wst function and then multiplying it with its inverse but without the exploration values. The Qsat function takes the start state of the simulation, the action taken by the policy and the resulting ending state of the simulation as an input and uses this information to provide an evaluation on how good this action was given the states of the simulation. The Wst function is used to provide extra weight to certain states of the simulation making it possible to provide an boost for parameters of the policy which resulted in winning state or to punish when the last state of the simulation was especially bad. This algorithm is then run either until the parameters have converged to an optimum or until an maximum number of iterations is reached. We have chosen this method for learning the parameters as it performed best when compared to other learning methods such as eNAC and RWR [KP11]. It also treats the policy as an black box and does not need to know how it works or what it does at it operates entirely on the rewards and Qsat and Wst functions, making it perfectly suited to compare different policies against each other. But it cannot provide any guarantee that it will converge to the global maximum of the rewards and therefore not to the best possible values for the policy.

Algorithm 1 PoWER - Policy learning by Weighting Exploration with the Returns

Precondition: intial policy parameters Φ

function PoWER(Φ)
  repeat
    for $r \leq mR$ **do**
      $reward_r = policy(\Phi_k + exploration_r)$
    end for
    $\Phi_{k+1} = \Phi_k + \frac{\sum_{t=1}^{mR} Qsat_t * Wst_t * exploration_t}{\sum_{t=1}^{mR} Qsat_t * Wst_t}$
  until $\Phi converged$
  return $\Phi_{optimal}$
end function

4 Comparison

We compare both approaches by providing them with the same initial state of the simulation. In this initial state the ball is placed directly above racket with zero velocity. We are also using the previously described PoWER learning algorithm to learn both policies in order to give them the same conditions. Initially we wanted to makes them easily comparable by using the same reward function for both approaches but as we

cannot do the comparison cause of the poor performance achieved through the RDMP approach we decided to use different reward for each approach. Both rewards are described below. Figure 1 shows the learning curve of the RDMP approach over 100 iterations with 15 rollouts in each iteration. The blue line is the mean of the 10 best rollouts in that iteration while the shaded area shows the minimum/maximum reward. The reward is calculated by taking the difference of the ball velocity and an ideal velocity at each hit with the racket. The ideal velocity points straight up to make sure to keep the ball over the racket. So thee more the actual ball velocity differs from the ideal one at each hit the reward decreaes.

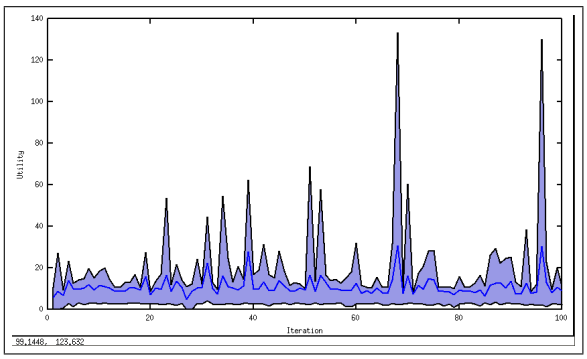

Figure 1: Learning curve for the RDMP approach of 100 iterations using 15 rollouts per iteration

As we had several problems with the RDMP approach an comparison with the other approach does not make much sense. As figure 1 shows the learning curve stays at an pretty low level for all iterations. We assume that this is caused by either just an bad reward function or by an incomplete or wrong understanding of the behavior of RDMPs.

In figure 2 you can see the learning curve for the approach using polynomial trajectories. As we cannot do an comparison cause of the bad performance of our RDMP approach we have used an different reward function just based on the number hits. This approach improves quite nicely until it reaches an local optima at nine hits per iteration.

5 Conclusion

In this paper we tried to compare two very different approaches for calculating the best movement for an racket attached to end of an seven DOF robot arm to continuously bounce an free-floating ball. The first method we used was to use our expert knowledge of

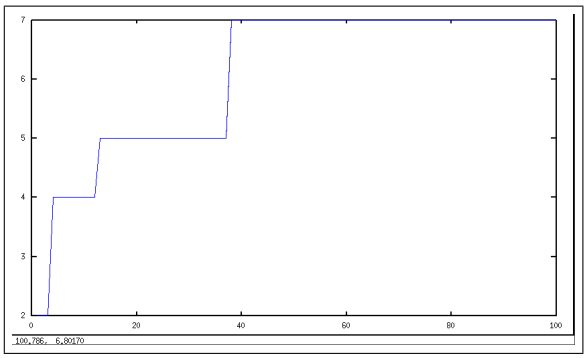

Figure 2: Learning curve for the polynomial trajectories approach over 100 iterations showing the highest number of hits achived until this iteration

the specific task to calculate the best trajectory to hit the ball. Such an approach is only feasible for smaller tasks as it is very complex to create such an policy. Another disadvantage of that approach is that it is limited to solving only that specific task. The second approach we used and compared to the first one was to use rhythmic dynamic movement primitives as they are an general framework for movement which can work in an multitude of different tasks. As such they are already used in various other tasks [Sch] and can be extended to also avoid obstacles [hPHPS]. For both algorithms we used the same reward function and the same learning algorithm, namely the PoWER algorithm. Because we had various problems with both approaches we failed to do an good comparison of them as both of their performances and their learning curves are suboptimal as shown in 1. The next steps would be to improve the learning of both approaches and to introduce multiple RDMPs to enable more than one controllable DOF.

References

[hPHPS] Dae hyung Park, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *in IEEE International Conference on Humanoid Robotics*.

[KP11] J. Kober and J. Peters. Policy search for motor primitives in robotics. (1-2):171–203, 2011.

[Sch] Stefan Schaal. Dynamic movement primitivesa framework for motor control in hu-

mans and humanoid robotics.

[SMI07] Stefan Schaal, Peyman Mohajerian, and Auke Ijspeert. A.j.: Dynamics systems vs. optimal control a unifying view. In *Progress in Brain Research*, pages 425–445, 2007.

[SPNI04] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In *International Symposium on Robotics Research (ISRR2003*. Springer, 2004.