An Operational Method Toward Efficient Walk Control Policies for Humanoid Robots

Ludovic Hofer and **Quentin Rouxel**
LaBRI, 321 Cours de la Libération
33405 Talence CEDEX, France

Abstract

Optimizing policies for real-time control of humanoid robots is a difficult task due to the continuous and stochastic nature of the state and action spaces. In this paper, we propose a learning procedure to train a predictive motion model and RFPI, a solver for continuous-state and action MDP. We use the predictive model as a transition model to train policies for a robot soccer. Our method requires no external hardware, a small amount of human work and manages to outperform the expert policy used by our team Rhoban winning the last 2016 edition of the Robocup in kid-size soccer league. Moreover, the proposed method is able to adapt to non-holonomic robots more efficiently than the expert approach. Our results are confirmed by both simulations and real robot experiments.

1 Introduction

In this paper we propose a learning procedure to optimize the motion control of a small humanoid robot. The robot moves on a soccer field and tries to reach a desired kick position and orientation without colliding with the ball. This skill plays a major part in the RoboCup robotic soccer competition, since reaching the ball faster than the opponent is one of the key to victory. This problem can be seen as a Continuous State and Action Markov Decision Process (CSA-MDP for short). The computation of a good policy in this CSA-MDP is difficult for several reasons. First, the deterministic model of the walk motion does not fit reality due to the high amount of noise and the unpredictable contact with the ground. Second, both the state space (positions and speed of the robot) and the action space (acceleration) are continuous. Third, the computational resources available on-board are limited. Fourth, the reward perceived is discontinuous.

Despite the mechanical and control uncertainties of this kind of small low-cost humanoid robots, approaches have been proposed to learn the deterministic part of the walking model using motion capture system (Schmitz, Missura, and Behnke 2011) and (Rouxel et al. 2016). Building such a predictive model allows running realistic simulations which can then be used to evaluate the performance of different policies.

Several approaches based on policy gradient have been proposed in the last decade to learn policies for CSA-MDP on robots. Some of these algorithms have been able to learn control policies to hit a baseball with an anthropomorphic arm (Peters and Schaal 2008), while other approaches display an impressive sample efficiency (Deisenroth and Rasmussen 2011). Both approaches have been tested on high quality robots exhibiting repeatable dynamics. Moreover, they are based on an access to partial derivatives of the reward function with respect to the policy parameters. In one hand, These kind of constraints are difficult to satisfy, particularly for accessibility problems where the reward is binary. On the other hand, expert policies with hand-tuned parameters tend to achieve satisfying results on low-accuracy robots (Behnke 2006).

A few years ago, Symbolic Dynamic Programming has been proposed to find exact solutions to CSA-MDP (Sanner, Delgado, and de Barros 2012). This algorithm, based on eXtended Algebraic Decision Diagrams requires a symbolic model of both the transition function and the reward function. It also relies on several assumptions concerning the shape of these functions and is suited for a very short horizon. While this result is a major breakthrough it is not applicable for our target problem.

Algorithms based on local planning achieves outstanding control on high dimensional problems (Weinstein and Littman 2013). However, the computational cost of online planning is a burden for real-time applications. This issue is particularly crucial for embedded applications where the computational power is often constrained.

This paper presents our method to solve the motion control problem and focuses on three contributions.

1. We provide a method to train a predictive model of the motion of the robot which does not require external hardware nor vision processing.
2. Once the predictive model is trained, we compute motion control policy of the robot using our own generic purpose CSA-MDP solver.
3. We present very encouraging experimental results, based on simulation as well as real world experiments. The policy computed with our method strongly outperforms those of the expert policy we used to win the Robocup last year.

The training of the predictive model is based on param-

eters optimization in the same way as (Ivanjko, Komsic, and Petrovic 2007) did on wheeled robots. It is arguably more convenient to perform than the approach proposed in (Rouxel et al. 2016), because it does not require external hardware nor vision processing. We present a comparison of the motion prediction performance for different size of training sets using cross-validation.

The main benefit of the proposed approach compared to visual methods is the very low computing power requirement. The same simple correction model should be suitable for visual odometry even if the authors are unaware of typical errors (false positive) introduced by vision processing which may call for specific adaptations. Moreover, the playground lacks of visual features required for visual odometry.

The computation of the policy is performed offline and then played online by the robot, using very few computational resources. It is based on a generic purpose solver for CSA-MDPs. The solver optimizes the motion control through an access to a black-box model of the transition and the reward function. Both value functions and policies are approximated using regression forests (Breiman 1996). This type of representation allows retrieving values at a low-computational cost, making it well suited for robotic applications.

This paper is organized as follows: Section 2 defines the notations used. Section 3 presents the robotic platform and the navigation problem in details. Section 4 introduces our motion predictive model and the method used to learn its parameters. We propose our solver of CSA-MDP in Section 5 and present our results in Section 6. Finally, we summarize the contributions brought by our research in Section 7.

2 Background

2.1 Continuous State and Action Markov-Decision Process

A CSA-MDP is a 5-tuple $\langle S, A, R, T, \gamma \rangle$, where S is a space of states, A is a space of actions, R is a reward function where $R(s, a, s')$ denotes the expected reward when taking action a in state s and arriving in s', T is a transition function where $T(s, a, s')$ denotes the probability of reaching s' from s using a and $\gamma \in [0, 1[$ is a discount factor.

A *policy* is a mapping $\pi : S \to A$ where $\pi(s)$ denotes the action choice in state s. A *value function* $V : S \to \mathbb{R}$ is a function which maps states to expected rewards.

2.2 Regression Forests

A *regression tree* is an approximate representation of a function $f : X \to Y$ where $X \subseteq \mathbb{R}^k$ and $Y \subseteq \mathbb{R}$. It has a decision tree structure where every non-leaf node is a function mapping X to its children and every leaf is a basic function $\phi : X \to Y$. Several algorithms exist to extract regression trees from training set, for a complete introduction, refer to (Loh 2011). Predicting the output $y \in Y$ from an entry $x \in X$ requires finding the leaf corresponding to x and then computing $\phi(x)$, with ϕ the basic function found at the leaf corresponding to x. While some algorithms use oblique split (Li, Lue, and Chen 2000), we only used orthogonal splits (splits of the form $x_i \leq v$). A *regression forest* is a set of regression trees. It has been exhibited in (Breiman 1996) that averaging multiple trees to represent the function leads to a more accurate prediction. The algorithm used to grow the regression forests in our algorithm is based on the Extremely Randomized Trees, introduced in (Ernst, Geurts, and Wehenkel 2005). More details on the use of regression forests for CSA-MDP can be found in (Hofer and Gimbert 2016).

2.3 Robot Egocentric Frame

In this paper, the position and orientation of the robot is defined by the robot egocentric frame. This frame is centered on the robot's trunk (see Fig. 2). The Z axis is vertical. The X axis is oriented toward the front of robot (forward direction). The Y axis is oriented to the left of the robot (lateral direction).

3 Problem Description

3.1 Humanoid Robotic Platform

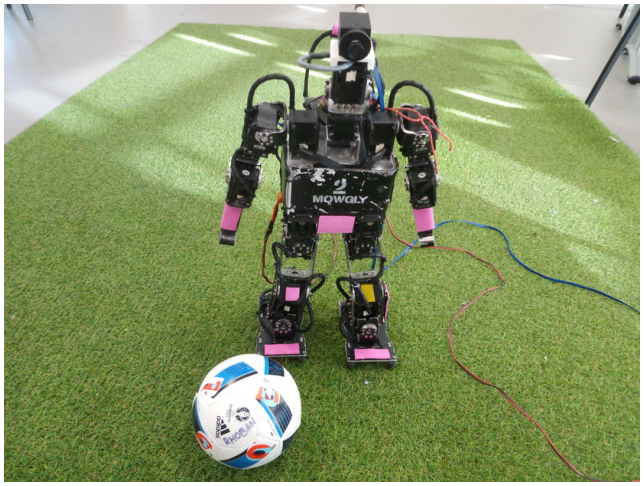

Figure 1: Sigmaban robotic platform.

All our experiments are run using Sigmaban, a small humanoid robot designed, built and developed by the Rhoban Team at the University of Bordeaux. The robot is 60 cm height, weights 4.2 Kg and has 20 degrees of freedom: 6 per leg, 3 per arm and two in the neck. Each joint is actuated by the commercial servo-motors Dynamixel[1] (MX-64, MX-28). Its conception is mainly directed toward the participation to the RoboCup competition in the Kid-Size Humanoid League[2].

The general morphology is constrained by humanoid-like ranges on some geometric features such as total height, leg-arm ratio, neck length and also the position of the center of mass. In addition, only "*human-like*" sensors are allowed. Infra-red cameras, laser sensors, lidars, ultrasonic sensors or

[1]Dynamixel servo-motors: `http://www.robotis.com/xe/dynamixel_en`

[2]RoboCup Humanoid League rules are available at: `https://www.robocuphumanoid.org/materials/rules/`

magnetometers are forbidden in order to put emphasis on the perception analysis and filtering. The robot only has a visible light camera, an accelerometer, a gyroscope, two foot pressure sensors and position encoders located at each joint measuring the absolute angular state.

The foot pressure sensors are made of strain gauges attached to each cleat under the robot's feet. They detect the contact with the ground but also measure the vertical center of pressure. These low-cost sensors are designed by the Rhoban team and have been presented in (Rouxel et al. 2015). Precisely detecting the actual supporting foot during the navigation is essential to compute an accurate odometry estimation of the robot's displacements.

During soccer games, the robot is fully autonomous. A lithium-polymer battery powers the direct current motors, the control electronics and the main processor. A small embedded x86 computer runs the robot's main program. It reads motors, sensors and camera inputs, analyses and computes the current internal and external state and issues to the servo-motors the next target position at about 100 Hz.

The main drawbacks of this type of small and *low-cost* humanoid robots are their mechanical and control inaccuracies. For example during a robotic soccer game, the robots often experience numerous falls and collisions. The mechanical parts get bended on a regular basis, the deformation reaching up to a few degrees. During the 2016 competition, a 5 degrees bending on the neck was measured, greatly affecting the camera-based distance computation accuracy.

Another source of dynamical uncertainty is the backlash added by each servo-motor and increasing with the gearbox wear. A typical backlash on a new gearbox is about 0.2 degree, whereas a 0.8 degree backlash was measured on the ankle, knee and hip of the leg of the robot. Also, during the robot's displacement on the artificial grass, a slide of the supporting foot can be observed. Finally, the default servo-motor controller is purely reactive and thus always exhibits an important dynamical tracking error depending on the target motion.

3.2 Walk Engine

The walk engine[3] used by the Sigmaban robot and presented in (Rouxel et al. 2015) is omni-directional and controls the target position of the 12 leg joints. The walk is externally driven by three values: the forward and lateral length of the next step and its angular rotation. These walk orders are issues at each complete walk cycle (two steps).

The engine is based on a set of parametrized polynomial splines in Cartesian space of the foot trajectories with respect to the robot's trunk. A stabilisation procedure detects strong perturbations by looking for unexpected weight measures from the foot pressure sensors. In case of perturbations, the walk timing is altered in order to keep the robot balanced.

Even with bounds presented at Table 1 and Table 2, some combinations of large speed and acceleration of walk orders can make the robot unstable. These events trigger the stabilization procedure which tries to recover from the perturbations but at the expense of high noise in the robot's displacement.

[3]Rhoban IKWalk implementation: `https://github.com/Rhoban/IKWalk`

3.3 Holonomic Approach

During robotic soccer games, several skills such as recovering from falls or kicking the ball are required, but most of the time is spent in the *approach* phase. During the approach, the robot has an estimate of both the ball position and its own position on the field. It is then able to choose a desired aim for its next kick. The main problem is to control the orders sent to the walk engine in order to reach the desired position as fast as possible.

The walk predictive model and the observations gathered by the robots are stochastic. Due to the lack of predictability of the robot's motions, it is impossible to use standard methods for planning in continuous domains. Moreover, since the decisions have to be taken online, the choice of the orders should not require heavy computations.

Sudden variations of the parameters provided to the walk engine can result in instability and fall of the robot. Therefore, we decided to control the acceleration of the robot and not the walk orders directly. The name, limits and units of the state space can be found at Table 1 and those of the action space at Table 2. We do not consider the ball speed in the state space for two reasons. First, it is very hard to get an appropriate estimate of the ball speed because the camera is moving with the robot and the position observations are already very noisy. Second, the ball is static most of the time, because robots spend a large proportion of their time trying to reach the ball.

Table 1: State space of Approach

Name	Units	min	max
Ball distance	m	0	1
Ball direction	rad	$-\pi$	π
Kick direction	rad	$-\pi$	π
Forward speed	$\frac{m}{\text{step}}$	-0.02	0.04
Lateral speed	$\frac{m}{\text{step}}$	-0.02	0.02
Angular speed	$\frac{\text{rad}}{\text{step}}$	-0.2	0.2

Table 2: Action space of Approach

Name	Units	min	max
Forward acceleration	$\frac{m}{\text{step}^2}$	-0.02	0.02
Lateral acceleration	$\frac{m}{\text{step}^2}$	-0.01	0.01
Angular acceleration	$\frac{\text{rad}}{\text{step}^2}$	-0.15	0.15

At every step, the robot is given a reward which depends on the area it is located in:

Kick : The distance to the ball along the X-axis has to be between 0.15 m and 0.3 m. The absolute position of the ball along the Y-axis has to be lower than 0.06 m. The absolute angle between the robot direction and the kick

target has to be lower than 10 degrees. Inside this area, the robot receives a reward of 0.

Collision : The position of the ball along the X-axis has to be between -0.20 m and 0.15 m. The absolute position of the ball along Y-axis has to be lower than 0.25 m. Inside this area, the robot receives a reward of -3.

Failure : The distance to the ball has to be higher than 1 m. In this case, we consider that the robot has failed the experiment, the state is terminal and the reward is -100.

Normal : For every state that does not belong to any of the previous cases. Inside this area, the robot receives a reward of -1 (i.e. unit cost).

At every step, the next state is computed using the motion predictive model described in Section 4. To reflect the stochastic aspect of the walk we also introduce an uniform noise on the step displacement after each cycle. Noise along X-axis and Y-axis is drawn between -0.02 m and 0.02 m. The noise on orientation is drawn between -5 degrees and 5 degrees. We will further refer to the problem of Holonomic Approach by the abbreviation HA.

3.4 Expert Approach

The expert approach depicted at Algorithm 1 has 15 parameters originally tuned by hand. They configure the several state transition thresholds and the orders sent to the walk engine controlled by simple proportional controllers. This policy was designed by the Rhoban team and used to won the 2016 RoboCup Kid-Size soccer competition. We will further refer to this policy as *Winner2016*.

Algorithm 1 Overview of the expert navigation algorithm

```
state = Far
while not in kick range do
    ball = getRelativeBallPosition()
    orientation = getRelativeTargetOrientation()
    if state == Far then
        Make forward step to go to the ball
        Make turn step to align with the ball
        if ball.distance is close enough then
            state = Rotate
        end if
    else if state == Rotate then
        Make forward step to stay at fixed ball distance
        Make lateral step to turn around the ball
        Make turn step to stay aligned with the ball
        if ball.angle and orientation are aligned then
            state = Near
        end if
    else if state == Near then
        Make small forward step to reach kick distance
        Make lateral step to keep the ball centered ahead
        Make turn step to keep the ball aligned
        if |ball.y| lateral position is too far then
            state = Rotate
        end if
    end if
    if ball.distance is too far then
        state = Far
    end if
end while
```

3.5 Almost Non-Holonomic Approach

While some humanoid robots are able to move very quickly laterally, other robots are mainly capable of moving forward, backward and rotate. For these robots, policies that rely on lateral motion are particularly slow. In order to test the flexibility of the policies, we designed a second version of the problem in which we divided the limits for the lateral speed and acceleration by 5, since those robots are not moving laterally, they also exhibit less lateral noise, therefore we divided by two the lateral noise with respect to HA. We will further refer to this problem as the *Almost Non-Holonomic Approach*, ANHA for short.

3.6 Initial State

For HA and ANHA, the initial distance to the ball was sampled from an uniform distribution between 0.4 m and 0.95 m. The initial direction of the ball and the kick direction were sampled from an uniform distribution in $[-\pi, \pi]$. The initial velocity of the robot is always 0 (for both, Cartesian and angular velocities).

4 Motion Predictive Model Calibration

As previously mentioned, one of the major disadvantages of this type of robot is their important mechanical and control inaccuracies. A large discrepancy can be observed between the ideal – integrated walk control inputs – and actual physical displacement of the robot. Despite this error, the deterministic part of the robot real behaviour can still be captured.

The online odometry and offline motion model have been previously studied in (Rouxel et al. 2016). The odometry evaluates during the robot's motion its own relative displacement by analyzing the sensors readings. On the contrary, the motion model tries to predict the future robot self displacements given the sequence of orders possibly sent to the walk engine. In this past work, an external motion capture setup was used to learn both the odometry estimation (without vision) and the motion prediction using a non linear, non parametric regression method (Locally Weighted Projection Regression).

However in the context of robotic competitions, deploying a full motion capture setup on the soccer field is too cumbersome. In the following section, a calibration method based on black-box optimization is presented. The same approach has been introduced by (Ivanjko, Komsic, and Petrovic 2007) on wheeled mobile robots. A simple motion prediction model is computed on a humanoid robot without any external hardware neither vision processing but a tape measure. Although this procedure is less accurate, the resulting precision still fits our needs and can be more easily applied on the field.

4.1 Calibration Through Parameters Optimization

Without a proper motion capture setup or other external device, the actual robot's displacement at each step can not be recorded. To generate learning data, the following procedure is used:

- Minimum and maximum bounds are set according with the problem on both raw walk orders (velocity) and delta walk orders (acceleration).
- The robot is placed at a known starting position and orientation.
- During 15 seconds, random orders are sent at each step to the walk engine. The orders are drawn from a random walk (Brownian motion) where accelerations are uniformly distributed inside allowed ranges.
- Final position is manually measured with respect to starting position. Since final orientation is difficult to accurately measure, it is only recorded for 12 possible orientations (more or less 30 degrees).

This process is then repeated to gather enough data.

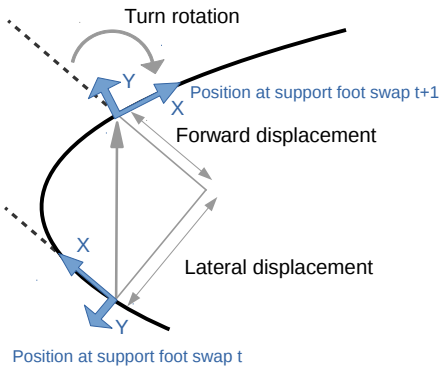

Figure 2: Robot displacement calculations at each walk cycle

The goal is to build the following *corrective* function computing a more accurate self relative displacement according to Fig. 2 between two walk cycles:

$$\Delta(x, y, \theta)_{walk\ orders,\ k} \longmapsto \Delta(x, y, \theta)_{corrected,\ k}$$

In this work, only simple linear models are considered and compared:

$$\begin{bmatrix} \Delta x_{corrected} \\ \Delta y_{corrected} \\ \Delta \theta_{corrected} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \begin{bmatrix} 1 \\ \Delta x_k \\ \Delta y_k \\ \Delta \theta_k \end{bmatrix} \quad (1)$$

The model parameters are the $a_{0,0}...a_{2,3}$ coefficients. To calibrate the model, the problem is treated as a black-box optimization problem. Parameters are optimized in order to minimize a fitness function scoring the error between actual final position and predicted position.

More precisely, the fitness function takes as input a set of parameters to be scored and a set of learning walking sequences. For each sequence, the robot's position and orientation is integrated at each step based on logged walk orders *ideal* displacements but corrected through the linear transformation of Equation 1. The resulting score is a weighted average between the Root Mean Squared Error (RMSE) on the final position and the RMSE on the final orientation – with a tolerance of more or less 30 degrees to account for the measurement uncertainty.

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm originally presented in (Hansen and Ostermeier 2001) and implemented in C++[4] is used to find the model parameters minimizing the fitness function. Since the model and the fitness function are simple, the optimization process is completed in a few seconds.

4.2 Calibration Results

Three different linear models are compared:

- Proportional model with 3 parameters.
  In Equation 1, only $a_{0,1}, a_{1,2}, a_{2,3}$ are non zero.
- Simple linear model with 6 parameters.
  In Equation 1, only $a_{0,0}, a_{1,0}, a_{2,0}, a_{0,1}, a_{1,2}, a_{2,3}$ are non zero.
- Full linear model with 12 parameters.
  In Equation 1, all $a_{0,0}...a_{2,3}$are non zero.

The Figure 3 displays the results of the motion prediction models calibration process. A total of 25 random walk sequences were recorded. Cross-validation using 5 validation sequences each were used to compare the linear models and the number of training sequences used by the optimization algorithm. The Y axis shows the mean validation RMSE on the final position and the confidence bounds.

Here, the proportional and simple linear model are performing quite equally. But since the former has more parameters, its convergence is slower. The found affine offsets $a_{0,0}, a_{1,0}, a_{2,0}$ are actually rather small. Indeed, the robot remains well steady when sending zero walk orders. As expected, the best fitting model but also the slower to converge is the full linear model. After 20 learning sequences, the full corrective model allows to reduce the final mean distance error from 0.39 m to 0.26 m. To compare, (Rouxel et al. 2016) goes from 0.68 m distance prediction error without correction to 0.14 m after learning, for 10 second of walking. Since the proposed approach does not measure individual step displacements with motion capture, neither use complex regression method, the improved accuracy is still interesting for operational use.

Experiments in Section 6 are run with the full linear model. The 12 parameters are chosen as the average of optimized parameters using the 20 sequences learning set. The Figure 4 is showing the convergence of the parameters with respect to the size of the learning set.

[4]The C++ CMA-ES Library: `https://github.com/beniz/libcmaes`

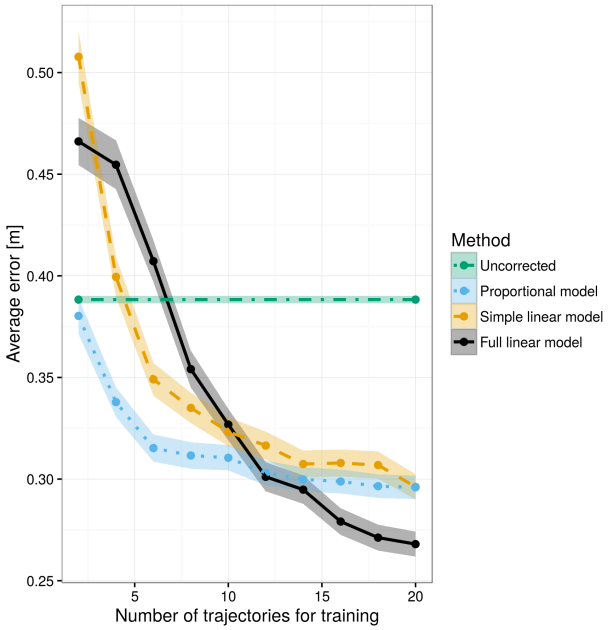

Figure 3: Performance of motion predictive models on cross-validation set

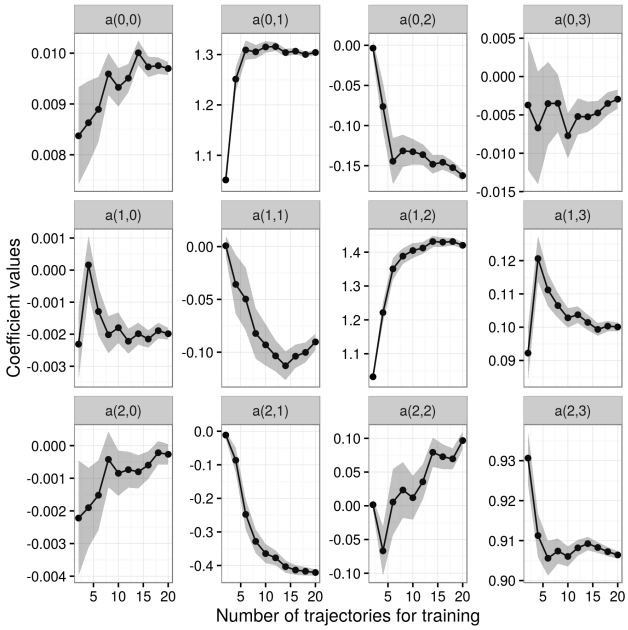

Figure 4: Optimized parameters average of the full linear model and their confidence interval

5 Regression Forests Policy Iteration

Regression Forests Policy Iteration, or RFPI for short, is inspired by the discrete version of *policy iteration*. Both the value function and the policies are represented as *regression forests*. A generic and abstract version of the algorithm is presented at Algorithm 2.

The learning algorithm uses visited states as basis to train the value functions and the policies. This choice of design ensures that more data is collected inside the areas of the state space which are frequently visited. The number of required runs to update the policy grows linearly along the number of policy updates. Therefore, the number of policy and value updates grow proportionally to $\sqrt{n}$, with n the number of visited states. This property is crucial to assure a reasonable complexity of the algorithm, even for problems that require a large set of samples. The value function V is initialized as a constant approximator with a value of 0. The initial policy samples actions from an uniform distribution among the whole action space. The list of visited states can be empty at the beginning or it can be fed with an external seed under the form of a list of visited states.

Both value functions and policies are built from samples through calls to the function *FitForest*. While details concerning this procedure are outside of the scope of this paper, an exhaustive review is provided in (Loh 2011). This function can be used to create piecewise constant models (PWC) or piecewise linear models (PWL).

The rules used to update the value function are provided at Algorithm 3. First, the value of every state is approximated independently, using multi-step simulations with the current policy π. Then, a regression forest is built based on visited states and their estimated reward. Although simple, this type of procedure can only be used with policies that have a very low complexity, because it requires a large number of calls to the policy. We use piecewise constant models to approximate the value function, because experiments and literature suggest that using linear models for the values are very likely to diverge from the real functions due to successive approximations.

Updating policies is performed through calls to the function defined at Algorithm 4. The main idea is to optimize actions with a single step horizon for each of the visited states independently. Then, the policy is obtained by building regression forests from the created matches from state to best action. We can safely use piecewise linear models for the policies since there is no problem of successive approximations for the policies.

6 Experiments

6.1 Tested Policies

Three different policies were experimented in both simulation and real world.

Winner2016: The policy used by the Rhoban team at RoboCup 2016 competition, see Section 3.4.

CMA-ES: This policy is the same as Winner2016, but all the parameters have been optimized using millions of simulations of the motion predictive model using the black-box optimization algorithm CMA-ES.

RFPI: This policy is represented as a regression forest. It is the result of a few hours of offline training on the problem. It has no information about the problem but a black-box access to the transition and the reward functions. In order to make sure that the samples were gathered inside the

Algorithm 2 The CSA-MDP learner algorithm

```
π = getRandomPolicy()
V = buildConstantApproximator(0)
visitedStates = seedStates
policyId = 1
runId = 0
while timeRemaining() do
    executeRun(π, visitedStates)
    runId++
    if runId == policyId then
        V = updateValue(π, V, visitedStates)
        π = updatePolicy(V, visitedStates)
        runId = 0
        policyId++
    end if
end while
```

Algorithm 3 The updateValue function

```
function UPDATEVALUE(π, V, visitedStates)
    values = { }
    for all s ∈ visitedStates do
        val = getAvgValue(s, π, maxSteps, nbRollouts)
        values.append(val)
    end for
    return FitForest(visitedStates, values, PWC)
end function
```

Algorithm 4 The updatePolicy function

```
function OPTIMIZEACTION(V, s)
    candidates = getRandom(actionSpace, nbActions)
    bestScore = −∞
    bestAction = null
    for all c ∈ candidates do
        score = 0
        for all i ∈ range(nbRollouts) do
            s' = sampleSuccessor(s, c)
            reward = getReward(s, c, s')
            score += reward + γV(s')
        end for
        if score > bestScore then
            bestScore = score
            bestAction = c
        end if
    end for
end function
function UPDATEPOLICY(V, visitedStates)
    bestActions = { }
    for all s ∈ visitedStates do
        action = optimizeAction(V,s)
        bestActions.append(action)
    end for
    return FitForest(visitedStates, bestActions, PWL)
end function
```

kick area, policies were trained using a seed of 25 trajectories generated by Winner2016.

CMA-ES and RFPI policies were trained through simulations, using a maximum length of 50 steps for all the roll outs. When required, the initial state was chosen according to the distribution described at section 3.6. Both were trained specifically for each problem, HA and ANHA.

6.2 Simulation

All the trained policies were evaluated on both problems, HA and ANHA. The evaluation is performed by measuring the average costs on 10'000 trials for each modality. The initial state is generated randomly according to the distribution described at section 3.6 and the maximal number of steps in a run was set to 50. The simulation results are presented in Table 3. All the source code used for simulation experiments is open-source and developed as a set of ROS packages freely available[5].

Table 3: Average costs for the different policies in simulation

	Winner2016	CMA-ES	RFPI
HA	31.84	14.90	11.88
ANHA	44.12	36.18	15.97

First of all, note that on the HA, CMA-ES strongly outperforms Winner2016. This highlights the interest of building a predictive model and using it to optimize the parameters of a policy. By using millions of simulations and advanced optimization technics, CMA-ES is able to divide by more than two the average time to reach the ball position. Our algorithm goes even further and reduces the required time by an additional 20 percent while it has no prior information about the shape of the policy, except a set of visited states by Winner2016.

As we expected, Winner2016 does not perform well at all on ANHA. It has a cost of 44.12 in average, while the maximal cost when avoiding collisions with the ball is 50. Automated tuning based on CMA-ES reduces the average cost by around 20 percent. RFPI exhibits a strong flexibility, since it divides by more than two the expected value with respect to the CMA-ES policy. Moreover, RFPI achieves similar performances on ANHA as CMA-ES policy on HA, while the task is much harder.

6.3 Real World

Experimental Conditions During this experiment, the robot is placed on artificial grass. An official white ball is used as navigation target and the target orientation is set toward the goal posts. A specific vision pipeline has been implemented to detect and track the ball at 25 Hz. The Cartesian position of the ball with respect to the robot location is obtained by using the model of both the camera and the robot's kinematic. Finally, the position of the ball is filtered through a low pass filter.

The recognition and the discrimination between the white ball and the white and round goal posts is a difficult task of-

[5]Source code (C++) available at: `https://bitbucket.org/account/user/rhoban/projects/ROS`

ten leading to false positives. To ease the experiment, the localisation module used during robotic competitions has been disabled. The initial kick target orientation is provided manually and is then tracked by integration of the inertial measurement unit.

Experimental Method Winner2016, CMA-ES policies and RFPI policies are all tested in real soccer conditions for both HA and ANHA. For each test, a total of 12 approaches are run, totalizing 72 different approaches. The Cartesian product of the following initial states is performed:

- The ball is put either at 1 m or 0.5 m.
- The ball is put either in front of the robot, on its left, on its right.
- The initial kick target is either 0°or 180°.

For each run, the time required for the robot to stabilize inside the kicking area is recorded.

Results Average time for all the trajectories depending on the problem type and the policy used are shown in Table 4. Even if the quantity of data collected is too small to have an accurate evaluation of the difference of performance among policies, the general trend is similar to the one obtained in simulation. The method tuned by CMA-ES outperforms Winner2016 and RFPI outperforms both.

Table 4: Average time in seconds before kicking the ball

	Winner2016	CMA-ES	RFPI
HA	19.98	13.72	11.45
ANHA	48.14	25.69	18.81

A representation of several trajectories perceived by the robot is given at Fig. 5. All these trajectories are directly extracted from the internal representation of the robot[6]. Here, the arrows represent the robot pose at each walk cycle. They all depict the same initial situation, solved for both HA and ANHA, with each of the proposed policies. It can be seen that although the robot started at a distance of 1.0 m of the ball, it initially believes that the distance is around 1.3 m. This type of error is the result of an accumulation of errors in the measurements of the joints, combined to some of the parts bending due to the frequent falls of the robot. The adaptability of the proposed method with respect to the robot constraints can easily be seen by comparing the two trajectories observed for the RFPI policy.

7 Conclusion

This article introduces an operational method to establish a motion predictive model and a solver for CSA-MDP named RFPI. The proposed approach has several advantages over existing methods: no external hardware is required, no expert knowledge on the problem is necessary and it adapts automatically to different types of locomotion. Unexpectedly, no adaptation was required to apply the policies learned in simulation on the robot, an unusual case in robotics.

[6]A video showing these trajectories on Sigmaban robot is available at: `https://youtu.be/PNA-rpNKfsY`

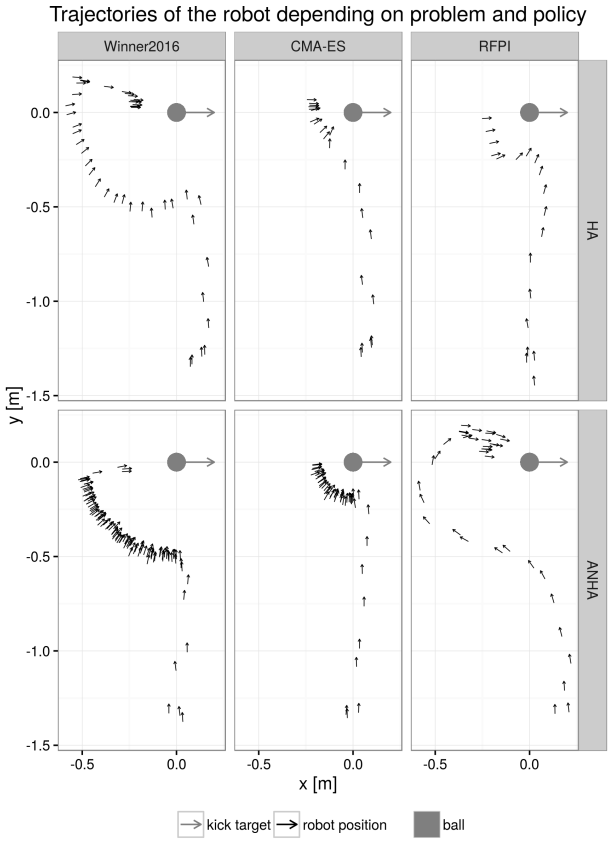

Figure 5: Examples of real world trajectories

When the robot detects a balance issue, it uses a stabilization procedure which costs a small amount of time and leads to additional noise. Sudden variations in the walk orders and combinations of large values for the orders are more likely to result in instability. One of the main sources of discrepancies between simulation and real world experiments is likely to be the simple noise model used which considers that noise is uncorrelated with speed and acceleration. Further refinements of our method should include a calibration procedure for the noise model. If required, additional state dimensions could be used to represent the ball speed or the position of obstacles and robots.

Acknowledgements

The authors acknowledge partial support from ANR project STOCH-MC (ANR-13-BS02-0011-01).

References

Behnke, S. 2006. Online trajectory generation for omnidirectional biped walking. *Proceedings - IEEE International Conference on Robotics and Automation* 2006(May):1597–1603.

Breiman, L. 1996. Bagging predictors. *Machine Learning* 24(2):123–140.

Deisenroth, M. P., and Rasmussen, C. E. 2011. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. *Icml* 465–472.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research* 6(1):503–556.

Hansen, N., and Ostermeier, A. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9(2):159–195.

Hofer, L., and Gimbert, H. 2016. Online reinforcement learning for real-time exploration in continuous state and action markov decision processes. In *PlanRob2016, Proceedings of the 4th Workshop on Planning and Robotics at ICAPS2016*. AAAI.

Ivanjko, E.; Komsic, I.; and Petrovic, I. 2007. Simple off-line odometry calibration of differential drive mobile robots. In *Proceedings of 16th Int. Workshop on Robotics in Alpe-Adria-Danube Region-RAAD*.

Li, K.-C.; Lue, H.-H.; and Chen, C.-H. 2000. Interactive Tree-Structured Regression via Principal Hessian Directions. *Journal of the American Statistical Association* 95(450):547–560.

Loh, W.-Y. 2011. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1(1):14–23.

Peters, J., and Schaal, S. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4):682–697.

Rouxel, Q.; Passault, G.; Hofer, L.; N'Guyen, S.; and Ly, O. 2015. Rhoban hardware and software open source contributions for robocup humanoids. In *Proceedings of 10th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conference on Humanoid Robots, Seoul, Korea*.

Rouxel, Q.; Passault, G.; Hofer, L.; N'Guyen, S.; and Ly, O. 2016. Learning the odometry on a small humanoid robot. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE.

Sanner, S.; Delgado, K. V.; and de Barros, L. N. 2012. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *Proceedings of the 26th Conference on Artificial Intelligence*, volume 2.

Schmitz, A.; Missura, M.; and Behnke, S. 2011. Learning footstep prediction from motion capture. In *RoboCup 2010: Robot Soccer World Cup XIV*. Springer. 97–108.

Weinstein, A., and Littman, M. 2013. Open-Loop Planning in Large-Scale Stochastic Domains. In *27th AAAI Conference on Artificial Intelligence*, volume 1, 1436–1442.