Approximation of Feasibility Tests for Reactive Walk on HRP-2

Nicolas Perrin, Olivier Stasse, Florent Lamiraux, Eiichi Yoshida

***Abstract*— We present here an original approach to test the feasibility of footsteps for a given walking pattern generator. It is based on a new approximation algorithm intended to cope with this specific problem. The result obtained is used on the robot HRP-2, and enables it to guess a step feasibility 40,000 times faster (in 9μs) than with the normal verification process. As a consequence some advance is made towards fast online motion (re)planning based on a continuous set of possible steps.**

I. INTRODUCTION

Planning and controlling motions of a legged humanoid robot is well known to be a difficult problem. First, these robots have a high number of degrees of freedom. This property has a strong impact on the computation cost of numerical methods used to produce motions on the one hand and on collision detection complexity on the other hand [9]. Second, they are subject to dynamic constraints that make most motions unfeasible.

One way to plan motions for a humanoid robot is to grow a tree of random steps with an A* algorithm. The growing tree of steps should obviously only consider feasible steps, but the current trajectory generation and verification techniques take at least a couple of hundreds of milliseconds for each step on recent computers. This fact drastically limits the size of the tree of steps, and makes this method unsuitable for reactive walk.

Yet, reactive walk is a major requirement for humanoid robots, because it is needed in any potentially changing environment, and *a fortiori* in any task involving cooperation with humans. The current state-of-the-art solution is to only allow a small set of steps for the robot. In that case the generation and verification phases are useless since all trajectories can be memorized and verified offline ([7], [8], [1]; for a recent extension, see [2]). This approach is not always satisfying for it leads to a gait which has no flexibility, and combined with planning it often results in the robot making a large number of steps to perform a task for which only one or two steps would have been arguably enough.

Instead of limiting the possible steps, we based our work on the following remark: even if a robot has a great number of degrees of freedom, the set of all possible steps only has 6 dimensions. So, if we add some restrictions ensuring that to one step corresponds only one unique trajectory, then the number of parameters used to describe a trajectory could be small enough to make machine learning techniques of practical interest. With such restrictions, we decided to attempt, in the set of all possible steps, to approximate the feasible region through extensive offline computations and then use the approximation online to guess extremely quickly whether a given step is feasible or not.

In this paper, we present an original approximation algorithm aimed at maximizing its efficiency by taking advantage of the specificities of our problem. We show how we used it and obtained an approximation which helped the robot HRP-2 to perform an experiment where online reactivity is constantly needed.

Nicolas Perrin (n.perrin@aist.go.jp) is with Université de Toulouse ; UPS, INSA, INP, ISAE ; CNRS ; LAAS ; F-31077 Toulouse, France, and CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT, AIST, Umezono 1-1-1, Tsukuba 305-8568 Japan

Florent Lamiraux (florent@laas.fr) is with CNRS ; LAAS ; 7 avenue du colonel Roche, F-31077 Toulouse, France

Olivier Stasse (olivier.stasse@aist.go.jp) and Eiichi Yoshida (e.yoshida@aist.go.jp) are with CNRS-AIST JRL (Joint Robotics Laboratory), UMI3218/CRT, AIST, Umezono 1-1-1, Tsukuba, Japan

II. PROBLEM STATEMENT

Let us denote by $\mathbf{x} = (\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ a state of the robot, i.e. a vector containing the configuration of the robot togeter with the first and second derivatives. In diverse applications, the trajectories of a robot are constrained by some real valued functions defined over the state space. In our work, we took into account four specific constraints:

1) The distance to self-collision: $SC(\mathbf{x})$, which is calculated using V-clip (see [11]), and which must stay greater than a margin σ_{SC} taking into account the possible tracking errors of the motor PID controllers.

2) The distance to joint limits: $JL(\mathbf{x})$ (negative when one joint is out of its range). A new margin σ_{JL} is added as a lower bound on the minimum value of $JL(\mathbf{x})$ along the trajectory.

3) The Zero Moment Point (ZMP) deviation: $ZD(\mathbf{x})$. This is the maximum distance between the ZMP reference used by the pattern generator (which is for example, in single support phase, the center of the support polygon), and the ZMP corresponding to the joint space trajectory it actually produces (we will call it the "multibody ZMP"). The robot HRP-2 uses a walking pattern generator (see [13]) which does not guarantee the exact tracking of the ZMP reference, and if the multibody ZMP goes out of the polygon of support, then the robot might fall (see [20]), therefore we add another threshold σ_{ZD}.

4) Through experiments, we noticed that in some cases if the multibody ZMP goes out of the polygon of support for a very short time and with a relatively small amplitude, the robot might not fall. Hence we took a relatively high value for σ_{ZD}, and instead decided to stress more on the case where the multibody ZMP is regularly far from the reference along the trajectory. Thus we decided to take into account the variance of the multibody ZMP and defined a

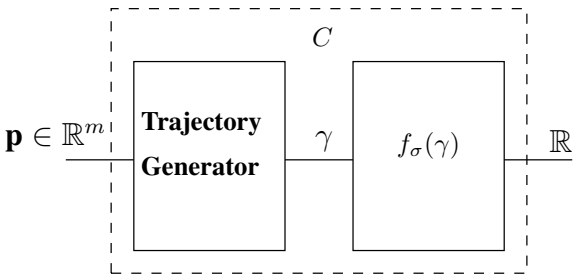

Fig. 1. The mapping to approximate. The Trajectory Generator takes in input a vector of parameters defining a step, and must return a *unique* trajectory. The value returned by the mapping is positive if the trajectory is acceptable (feasible), negative otherwise.

maximum value σ_{VZ} which should ensure that the multibody ZMP mostly stays near its reference.

We desire to characterize the actual feasibility of a given trajectory γ over a time interval $[0, T]$ with a unique real number, so finally, the formula we use returns the minimum of the four constraints considered:

$$\begin{aligned} f_\sigma(\gamma) = \\ min\Big(& -\sigma_{SC} + min_{t\in[0,T]}\big(SC(\gamma(t))\big), \\ & -\sigma_{JL} + min_{t\in[0,T]}\big(JL(\gamma(t))\big), \\ & \sigma_{ZD} - max_{t\in[0,T]}\big(ZD(\gamma(t))\big), \\ & \sigma_{VZ} - \frac{1}{T}\int_{t=0}^{T}\big(ZD(\gamma(t))\big)^2 dt\Big) \end{aligned} \quad (1)$$

where σ is the quadruple $(\sigma_{SC}, \sigma_{JL}, \sigma_{ZD}, \sigma_{VZ}) \in \mathbb{R}^4$, and γ a trajectory which assigns to any $t \in [0, T]$ a state of the robot. The way we obtained the four margins σ_{SC}, σ_{JL}, σ_{ZD} and σ_{VZ} is empirical. We chose a margin of 2.5 centimeters for the self-collisions, 3 degrees for the joint limits, and, as for the margins related to the ZMP trajectories, we calibrated them through tests.

It is straightforward to approach $f_\sigma(\gamma)$ when the trajectory is discretized. The computable function f_σ evaluates the feasibility of a trajectory corresponding to a step considered by the robot. As mentioned previously, it is unfortunately quite time consuming to generate a trajectory γ and then compute $f_\sigma(\gamma)$. So we would like to build offline an approximation helping us to guess the value and more importantly the sign of $f_\sigma(\gamma)$ without even having to generate the trajectory. To do so, we need to produce unique trajectories from vectors of parameters in $\mathbb{R}^m$. This is shown in section V, and it give us a mapping C as shown on Fig. 1, which can entirely be computed and approximated offline.

Before that, in section IV, we present the original approximation algorithm which is used to approximate C.

III. RELATED WORK

This idea of precomputing robot dependent data structures has been exploited in path planning for multibody robots in the past [10], [6], [14]. In these papers a roadmap is computed for a multibody robot without obstacles. Once the robot is placed in an environment with obstacles, the precomputed roadmap is pruned by removing edges in collision with the obstacles. The remaining roadmap is then used to plan paths.

In [19] a 2 dimensional map is built which returns the time necessary to change a HRP-2 step-length during the flying phase of the foot in order to realize an emergency stop. The keypoint of this work is to build a map which verifies that the ZMP realized by the robot stays in the support polygon for a given step-length modification done at a given time while walking. Indeed walking pattern generators such as the one proposed by Kajita et al. [5], or Morisawa [13], do not guarantee that the robot ZMP stays in the support polygon. The main difference between [19] and our approach is that we consider more constraints, and propose an adaptive partition of the input space well suited for higher dimensions. Indeed our work, taking into account free steps (their work only considers forward walking), has to aim at dealing with higher dimensions.

Concerning our approximation algorithm, we focused on techniques that reduce the required number of samples. The main specificity of our problem is that we are only interested in the sign of the function to approximate. Therefore, we naturally chose to use a method for which the sampling is adaptive and focuses on the regions where the mapping changes its sign. We adapted the concept of Recursive Stratified Sampling, which has been extensively used for Monte Carlo integration and image reconstruction (see [17], [3] and section 7.8 of [15]). Locally, we approximate by using a classic optimization problem. It is the same as the one which leads to Support Vector Regression techniques (see [18])). It minimizes under some constraints a distance between a polynomial and the samples (see (3)). We also use in our algorithm two techniques (one global, and one local) of approached Farthest Point Sampling, a method which has been shown to lead to high data acquisition rates (see [12] for recent developments on Farthest Point Sampling).

IV. MAPPING APPROXIMATION

A remark on notations: the cardinal of a finite set X is denoted by $|X|$.

Let C be a continuous or at least piecewise continuous mapping from a bounded hyper-rectangle $B_0 \subset \mathbb{R}^m$ to $\mathbb{R}$. We suppose that the set $\{x \in B_0 | C(x) = 0\}$ (the "frontier") is of Lebesgue measure zero. We also assume that C is, on its areas of continuity, $K-$Lipschitz for some $K \in \mathbb{R}$.

Our goal is to approximate C through sampling (since we know nothing about C we can see this task as nonparametric learning), focusing on the correctness of the sign of the result, and reducing as much as possible the number of samples needed for a satisfying approximation (because evaluating one sample is quite time consuming).

Our algorithm is articulated around two principles:

- Recursive Stratified Sampling: the initial input space B_0 is recursively partitioned into small boxes (a tree

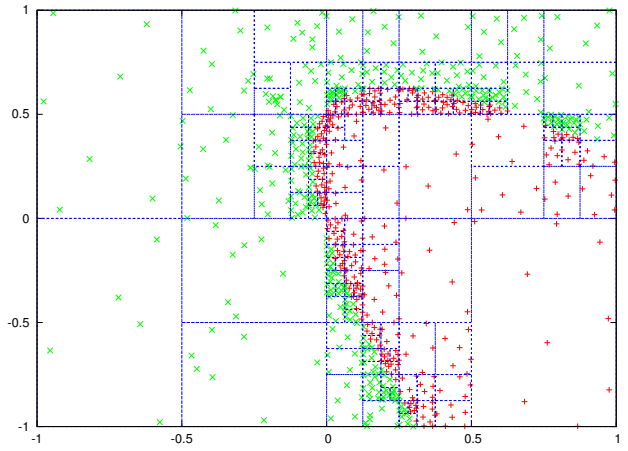

Fig. 2. An example run of our approximation algorithm on a continuous function from $\mathbb{R}^2$ to $\mathbb{R}$

structure keeps the trace of all splittings; the current partition is formed by the "leaf-boxes"). On all leaf-boxes, independant local approximations have been performed (through the resolution of optimization problems): their reunion gives the current global approximation.
We treat differently "frontier boxes" (typically containing samples on which the sign of the mapping differs – boxes with no samples are also presumed to be frontier boxes) and other, "regular boxes", in order to obtain an adaptive sampling focusing on the frontier.

- Farthest Point-like Sampling in two different scales:
  1) While selecting a box for sampling: we roughly estimate for each box our "confidence in the sign" of the local approximation, and select among boxes of lowest confidence.
  2) While sampling inside a box: since there will be a limited number of samples inside a box, we can use a naive technique for an approached farthest point-like sampling.

Algorithm 1 shows an overview of the approximation algorithm.

The while loop of the algorithm always terminates because if the number of samples in a box is 1 or zero, then the local approximation always succeeds. The algorithm implements a continuous learning: the approximation is endlessly updated, and the user decides when to stop. Various stop criteria could be considered; in our experiment we paused and tested the approximation several times, and stopped the learning when the percentage of false positive was below 2%.

Fig. 2 shows the result of our approximation algorithm on a function from $[-1,1]^2$ to $\mathbb{R}$. We can see that the boxes are split adaptively, and that it leads to an adaptive sampling focusing on the region where the function changes sign.

In the light of the algorithm structure, let us explain the two main features of our approximation method with more details:

A. Recursive Stratified Sampling

- *Step 4 of the algorithm:*

  As mentionned earlier the leaf-boxes are separated in two categories: the "frontier" boxes and "regular" boxes. Before selecting a new box to sample in, the realization of a random variable decides which category the selected box will belong to. The probability ρ_{boost} for the set of "frontier" boxes to be chosen is set by the user. Because the frontier is supposed of Lebesgue measure zero, and the size of boxes endlessly decreases as they are split, a fixed probability for sampling near the frontier inevitably leads to an adaptive sampling focusing mostly on the frontier. The probability set by the user impacts on the balance between known frontier approximation and the search for unseen frontier.
- *Steps 10 and 15 of the algorithm:*
  The successive splittings are made along successive dimensions, in a cyclic way.
  Therefore the structure of the splittings is fixed, as well as their position, but locally boxes are split with different speeds, a higher uncertainty in the mapping sign leading to a higher splitting rate.
- *Step 13 of the algorithm:*
  Let us show how we perform a local approximation on a box $B_{current}$. The samples in $B_{current}$ are the training data $\{(s_1,z_1),\dots(s_l,z_l)\} \subset \mathbb{R}^m \times \mathbb{R}$. Our goal is to find a function $f(x)$ that approaches this training data with correct sign.
  f will be searched among the elements of a finite dimension vector space chosen by the user (which must contain constant functions). Let us describe the basic case of affine functions f, taking the form

$$f(x_1,x_2,\dots,x_m) = \langle w,(x_1,x_2,\dots,x_m,1)\rangle, \quad (2)$$

  with $w \in \mathbb{R}^{m+1}$ and where $\langle\cdot,\cdot\rangle$ denotes the dot product on $\mathbb{R}^{m+1}$. The problem can be written as a convex optimization problem, similarly to the foundational idea of Support Vector Regression (see [18]), but with a different formulation since we don't oblige the result to stay in a fixed margin around the training data:

$$\begin{aligned} &\text{minimize} \sum_i (f(s_i)-z_i)^2 \\ &\qquad = \langle w,\mathbf{M}w\rangle + \langle d,w\rangle + \sum_i z_i^2 \\ &\text{subject to} \begin{cases} \forall i \mid z_i > 0, & f(s_i) > 0 \\ \forall i \mid z_i \le 0, & f(s_i) < 0 \end{cases} \end{aligned} \quad (3)$$

  where $\mathbf{M}$ is a symmetric positive $(m+1)\times(m+1)$ matrix built from $(s_i)_{1\le i\le l}$, and $d \in \mathbb{R}^{m+1}$ built from $(s_i)_{1\le i\le l}$ and $(z_i)_{1\le i\le l}$.
  We use the solver QL (see [16]) to solve this optimization problem, which generalizes well for other vector spaces than affine functions (in the case of our application we use the vector space of second order polynomials).

B. Farthest Point-like Sampling

- *Step 4 of the algorithm:*

Algorithm 1 Approximation Algorithm

Require: The Mapping $C : B_0 \subset \mathbb{R}^m \to \mathbb{R}$ that can be instanciated on its input space, B_0 being an hyper-rectangle.
$\rho_{boost} \in]0, 1[$
k_{MAX}: the maximum number of samples per leaf-box
$k < k_{MAX}$: the number of samples sampled at once.

1: $\mathbb{B}_{Frontier} \leftarrow \emptyset$; $\mathbb{B}_{Regular} \leftarrow \emptyset$
2: Push $(B_0, 0)$ in $\mathbb{B}_{Frontier}$ (0 is the height of B_0 in the tree of boxes).
3: Let $\mathbb{B}_{UnderProcess}$ be an empty stack (of type FIFO for example).
4: Randomly choose between $\mathbb{B}_{Frontier}$ and $\mathbb{B}_{Regular}$: with probability ρ_{boost} pick an element out of $\mathbb{B}_{Frontier}$; with probability $1 - \rho_{boost}$ pick an element out of $\mathbb{B}_{Regular}$ (unless one of the two sets is empty: in that case pick an element out of the other with probability 1). Once the set $\mathbb{B}_{Frontier}$ or $\mathbb{B}_{Regular}$ is chosen, we pick a box with a strategy described in section IV-B. Let us call the element picked (B_{pick}, h_{pick}).
5: In B_{pick}, sample k new samples.
6: Push B_{pick} in $\mathbb{B}_{UnderProcess}$.
7: **while** $\mathbb{B}_{UnderProcess}$ is not empty **do**
8: Pop an element $(B_{current}, h_{current})$ out of $\mathbb{B}_{UnderProcess}$. Let $S_{current}$ be the current set of samples in $B_{current}$.
9: **if** $|S_{current}| > k_{MAX}$ **then**
10: Split B_{pick} along dimension $(h_{current} \ mod \ m)$ into two son boxes of equal dimensions: B_{left} and B_{right}.
11: Push $(B_{left}, h_{current} + 1)$ and $(B_{right}, h_{current} + 1)$ in $\mathbb{B}_{UnderProcess}$.
12: **else**
13: Resolve an optimization problem on $B_{current}$ in order to perform a local approximation.
14: **if** the resolution fails (i.e. it is not possible to locally approximate with a correct sign on all samples of $B_{current}$) **then**
15: Split B_{pick} along dimension $(h_{current} \ mod \ m)$ into two son boxes of equal dimensions: B_{left} and B_{right}.
16: Push $(B_{left}, h_{current} + 1)$ and $(B_{right}, h_{current} + 1)$ in $\mathbb{B}_{UnderProcess}$.
17: **else**
18: Update the value of the approximation on $B_{current}$. Push back $B_{current}$ in $\mathbb{B}_{Frontier}$ or $\mathbb{B}_{Regular}$, depending on the current samples on $B_{current}$.
19: **end if**
20: **end if**
21: **end while**
22: Goto 4.

Once we know whether to pick a box (and its height) out of $\mathbb{B}_{Frontier}$ or out of $\mathbb{B}_{Regular}$, we need a strategy to decide which box to pick.

Since we are interested in the correctness of the approximation sign, it is logical to try to sample in the zone where our confidence in the current approximation sign is the lowest. Therefore, for each box B we give a rough estimation of this confidence (which we denote by $conf(B)$), and pick a box with lowest confidence.

If $B \in \mathbb{B}_{Regular}$, then the sign of the mapping is assumed to be constant on B. If the mapping C is K-Lipschitz, this assumption could be wrong, i.e. the sign could change somewhere on B if, roughly, the density of samples is small enough. If the mapping takes values around $z \in \mathbb{R}$ on the elements of B, and if we assume regular spacing between samples, it can be shown that the critical density is proportional to $\frac{K}{z}$. Therefore if we multiply the density of samples in B by z, the result is proportional to an estimation of the lowest K such that C can be both K-Lipschitz and change its sign on B. This can be taken as our "confidence in the sign". For z we take the minimum value seen on B, and we pose:

$$conf(B) = \frac{|S_B|}{\text{volume of } B} \times min_{s \in S_B} |C(s)|, \quad (4)$$

where S_B is the set of samples in B. B being an hyper-rectangle, its volume is simply the product of its m lengths.

If $B \in \mathbb{B}_{Frontier}$, we simply take the density of samples as an estimation of confidence:

$$conf(B) = \frac{|S_B|}{\text{volume of } B} \quad (5)$$

- *Step 5 of the algorithm:* We have to pick k samples in a box B_{pick} which contains already a set of samples S_{pick}, with $|S_{pick}| < k_{MAX}$.
For the first sample, we uniformly pick an arbitrary fixed number of candidates (we chose 40). We choose the candidate which maximizes its minimum distance to the samples in S_{pick} (the distance used depends also on the value of C on the samples when $B_{pick} \in \mathbb{B}_{Regular}$). Then we update S_{pick} with it and reiterate the process with the $k - 1$ remaining samples.
The upper bound k_{MAX} ensures us that the selection of the next k samples is made in constant time. This

approached farthest point sampling is interesting as long as its computation time remains negligible compared to the evaluation of C on one sample (about 0.4s in our case).

V. REDUCING THE DIMENSIONALITY OF THE PARAMETER SPACE

The only thing remaining before we can apply the algorithm to the robot HRP-2 is the definition of a parameter space from which unique trajectories can be generated. This is the purpose of this section.

A. Unique trajectories from 6 parameters

As mentioned in the introduction, 6 parameters can fully describe a step: 3 for the initial position and orientation of the swing foot (relatively to the stable foot), and 3 for its final position and orientation. Nevertheless, only the geometry of a step usually doesn't correspond to a unique trajectory: recent walking pattern generators produce fully dynamic walks, and thus take into account the initial speed of the robot's Center of Mass (CoM), as well as the next few steps that are going to be performed (see [4]).

In our work, we test isolated steps only: the initial and final speed of the CoM are zero. This corresponds to a conservative approach, since preliminary work showed that in most cases, a non-zero initial speed only expands the feasible set of steps.

Moreover, a fixed CoM height is given, as well as its initial and final position during any step: namely, at the barycenter of the feet positions. We also set the initial and final orientation of the robot (i.e. of its waist): its initial orientation is the one of the stable foot; its final orientation corresponds to the final orientation of the swing foot.

With all these restrictions, we obtain a unique trajectory from the 6 parameters describing a step geometry.

B. From 6 to 4 parameters

In a 6-dimensional space, 10 samples per dimension correspond to a total of 1 million samples.

Being able to treat 1 sample in about 0.4s, we were not able to obtain a satisfying approximation in a reasonable time with the computational power we disposed of.

Therefore, a further reduction of dimensionality was still necessary. We noticed that if we oblige the feet to stay at a reasonable distance from each other, and if the initial and final orientations of the swing foot remain low (in absolute value; e.g. between $-5\,^\circ$ and $+5\,^\circ$), then orientations don't have much impact on the feasibility region from a given initial position.

Fig. 3 shows the feasibility regions approximated for the same step with only initial and final orientations changing. The results are similar in shape, with differences when the lateral displacement is large. But we decided to set the upper bound limit for the lateral distance between the two feet of the robot to 29cm (partially due to the limited ankle flexibility of HRP-2). We also added a lower bound limit at 16cm, so that the feet stay relatively far from each other.

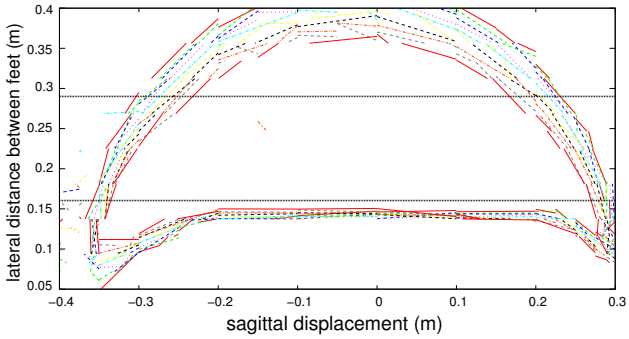

Fig. 3. Approximation of the feasibility region for next step (from a fixed initial position of the robot), for 11 very different values taken in $[-5\,^\circ, +5\,^\circ] \times [-5\,^\circ, +5\,^\circ]$ setting initial and final orientations of the swing foot (in this case the left foot). The line segments of a same color are the set of zero points of the approximation for one couple of orientations: they define the contour of the set of positions on which the swing foot can land after a valid step. We can see that the feasibility region is barely changed by changing slightly the initial and/or final orientation of the swing foot. The horizontal lines represent lower and upper bound limits that are explained in section V-B.

With these restrictions, we limit the feet orientation to the interval $[-5\,^\circ, +5\,^\circ]$ and ignore the orientation parameters when guessing the feasibility of a step. Fig. 3 motivates this heuristic.

VI. EXPERIMENTAL RESULTS

A. Analysis of our approximation

It took us 11 days with an Intel(R) Pentium(R) 4 CPU 3.40GHz to build the approximation we used, and the total number of points sampled is 2,672,928.

The resulting function is instanciated in 9μs: it means that with the same computer we divided the verification time for a step by about 40,000.

We tested the approximation on 349,559 points randomly drawn according to a uniform distribution over the input space. Among these, 8,607 were steps declared feasible by our verification process, 340,952 were declared not acceptable (which, because of the margins, doesn't exactly mean that the trajectory would actually fail). On the 8,607 positive points, the approximation was positive 8,278 times. On the 340,952 negative points, the approximation was negative 340,862 times.

With these random samples, if the approximation returns a positive result, it is correct (i.e. the trajectory generated would be considered feasible by our verification process) at 98.8%. If the returned result is negative, it is correct at 99.9%. Some security mecanism should be implemented to protect the robot if the result is a false positive.

B. Experiment

We conducted the following experiment: through a gamepad with 2 axis the user controls simple requests of steps that are repeatedly sent to the robot HRP-2. Each request has two components: a position of the objective footprint relative to the support foot, and an orientation

change. The mechanism of treatment of the requests of steps is described on Fig. 4.

Every 20ms, the step currently set by the gamepad is sent to our approximation, which guesses whether the step is feasible from the current posture of the robot or not. As long as the step is not feasible, it is discarded. But when a step is declared feasible, the software saves it and gets ready to send it to the robot. When another step is guessed feasible, the previous candidate step is overwritten: therefore, when the robot asks at regular intervals a new step to perform, it is always given the last feasible step.

Fig. 4. The experiment

With this simple approach, we have been able to teleoperate the robot several times for minutes without making it fall on the ground. It is very intuitive to guide it, and the user naturally doesn't keep axis positions corresponding to unfeasible steps. Nevertheless, in some cases, the robot, pushed to its limits, fell down. This is due to the discrepancy between the model on which we tested steps, and the real conditions (use of a stabilizer, smooth connections between steps in the actual experiment vs. zero speed CoM between steps in the model used for the approximation, robot compliance, inaccurate dynamics considered in simulation, etc.).

VII. CONCLUSION AND FURTHER WORK

We presented an original approximation algorithm and showed how, through some restrictions, we were able to limit dimensionality and perform the offline approximation of a walking pattern generator used on the robot HRP-2. Because HRP-2 was then able to guess the feasibility of a step 40,000 times faster than with the normal verification process, we successfully realized an experiment of reactive walk. Our further work will focus on three issues:

First, many suppositions that we made seem hard to justify theoretically. And even the full verification process of a trajectory does not ensure that the trajectory is feasible: for example we should also verify that the joint torques stay in the acceptable ranges. Even with a fully dynamic simulation, some uncertainty would still remain. Thus, it would be nice to give a solid theoretical foundation to our work, and if possible answer to that question: what can we ensure through offline approximation, and what will remain empirical?

Second, the question of connectivity in the approximated maps produced arises: what if the robot can reach a posture from which no step would be declared feasible? This issue naturally leads to the idea of post-treatment of the approximation: which smart and convenient data structure can we build from the approximation of a walking pattern generator?

Finally, we might try to take advantage of the fact that our algorithm is easily parallelizable in order to massively enhance our computational power, and attempt to successfully run our approximation in a high-dimensional space.

VIII. ACKNOWLEDGMENTS

This work was supported by a grant from the RBLINK Project, Contrat ANR-08-JCJC-0075-01.

REFERENCES

[1] J. Chestnutt, J. Kuffner, K. Nishiwaki, and S. Kagami. Planning biped navigation strategies in complex environments. In *IEEE Int. Conf. on Humanoid Robotics*, 2003.

[2] J. Chestnutt, K. Nishiwaki, J.J. Kuffner, and S. Kagami. An adaptive action model for legged navigation planning. In *IEEE Int. Conf. on Humanoid Robotics*, 2007.

[3] T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Wann Jensen. Multidimensional adaptive sampling and reconstruction for ray tracing. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, 2008.

[4] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, and K. Yokoi. Biped walking pattern generation by using preview control of zero-moment point. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1620–1626, 2003.

[5] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa. A realtime pattern generator for biped walking. In *IEEE Int. Conf. on Robotics and Automation*, pages 31–37, 2002.

[6] M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. In *IEEE Int. Conf. on Robotics and Automation*, April 2004.

[7] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'01)*, October 2001.

[8] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots. In *11th Int. Symp. of Robotics Research*, 2003.

[9] J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Self-collision detection and prevention for humanoid robots. In *IEEE Int. Conf. on Robotics and Automation*, pages 2265–2270, May 2002.

[10] P. Leven and S. Hutchinson. Toward Real-Time Path Planning in Changing Environments. *Algorithmic and Computational Robotics: New Directions: the Fourth Workshop on the Algorithmic Foundations of Robotics*, 2001.

[11] B. Mirtich. V-clip: fast and robust polyhedral collision detection. *ACM Transactions on Graphics (TOG)*, 17(3):177–208, 1998.

[12] C. Moenning and N. A. Dodgson. Fast marching farthest point sampling for point clouds and implicit surfaces. Technical report, 2003.

[13] M. Morisawa, K. Harada, S. Kajita, S. Nakaoka, K. Fujiwara, F. Kanehiro, K. Kaneko, and H. Hirukawa. Experimentation of humanoid walking allowing immediate modification of foot place based on analytical solution. In *IEEE Int. Conf. on Robotics and Automation*, pages 3989–3994, 2007.

[14] A. Nakhaei and F. Lamiraux. Motion planning for humanoid robots in environments modeled by vision. In *IEEE Int. Conf. on Humanoid Robotics*, December 2008.

[15] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. 1992.

[16] K. Schittkowski. Ql: A fortran code for convex quadratic programming - user's guide, version 2.11. Technical report, University of Bayreuth, 2005.

[17] R. Schürer. Adaptive quasi-monte carlo integration based on miser and vegas. In *5th Int. Conference on Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, 2002.

[18] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[19] T. Takubo, T. Tanaka, K. Inoue, and T. Arai. Emergent walking stop using 3-d zmp modification criteria map for humanoid robot. In *IEEE/RAS Int. Conf. on Robotics and Automation (ICRA)*, pages 2676–2681, 2007.

[20] M. Vukobratovic and B. Borovac. Zero-moment point – thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004.