Rapid Humanoid Motion Learning through Coordinated, Parallel Evolution

Marijn Stollenga, Jürgen Schmidhuber, and Faustino Gomez

IDSIA, USI-SUPSI
Galleria 2
Manno-Lugano, CH 6928
{marijn, juergen, tino}@idsia.ch

Abstract. Planning movements for humanoid robots is still a major challenge due to the very high degrees-of-freedom involved. Most humanoid control frameworks incorporate dynamical constraints related to a task that require detailed knowledge of the robot's dynamics, making them impractical as efficient planning. In previous work, we introduced a novel planning method that uses an inverse kinematics solver called Natural Gradient Inverse Kinematics (NGIK) to build task-relevant *roadmaps* (graphs in task space representing robot configurations that satisfy task constraints) by searching the configuration space via the Natural Evolution Strategies (NES) algorithm. The approach places minimal requirements on the constraints, allowing for complex planning in the task space. However, building a roadmap via NGIK is too slow for dynamic environments. In this paper, the approach is scaled-up to a fully-parallelized implementation where additional constraints coordinate the interaction between independent NES searches running on separate threads. Parallelization yields a 12× speedup that moves this promising planning method a major step closer to working in dynamic environments.

Keywords: robotics, planning, parallel search, NES

1 Introduction

The difficulty in planning coordinated motion for high-DOF robots, like the iCub humanoid (see figure 1), is that while the trajectory, of say the right hand, connecting point a to point b in the 3D *operational* workspace, $\mathcal{P}$, may be easy to compute, the trajectory in n-dimensional configuration space, $\mathcal{Q}$, (41-dimensional in the case of the iCub upper-body) that produces the hand movement by controlling the joints is generally not known. Determining the *configuration* $q \in \mathcal{Q}$ for each *pose* $p \in \mathcal{P}$ along the trajectory requires solving the Inverse Kinematics (IK) problem which is ill-posed due to the many-to-one nature of the forward kinematics, $f : Q \rightarrow P$, i.e. there are an infinite number of joint configurations that produce the same pose. Therefore, in order to solve the IK problem, the space of functions $f' : P \rightarrow Q$ must be constrained so that each pose maps to a unique configuration.

In previous work [17], a method called Natural Gradient IK (NGIK) was introduced that solves the IK problem by using Natural Evolutionary Strategies (NES; [7]) to search for configurations that minimize arbitrary cost-functions. NGIK is applied repeatedly to incrementally build a Task-Relevant Roadmap (TRM): a collection of configurations whose corresponding poses uniformly fill a user-defined *task space*, where poses related to a particular task are easily represented. Starting with a single configuration (point), new points are added by searching in the neighborhood of points already in the map.

While this approach yielded TRMs for the 41-DOF iCub (upper-body) that could then be used to plan sophisticated motion, the amount of time it takes to build a map limits its use to static environments. If the state of the world changes, for example because an object in the workspace has moved, some points may now violate hard constraints (they now collide the with object), or soft constraints (the repositioning of the object has altered the task). If the map cannot be rebuilt or repaired efficiently before the world undergoes more change, the planner will not be able to keep up.

In this paper, TRM construction is sped up by implementing it in parallel so that the map is grown from many points simultaneously (one for each processing core). Parallelization requires more than just a multi-threaded re-implementation; the independent NES searches must be coordinated by incorporating additional constraints (fitness terms) that prevent them from interfering with each other in task space if their search distributions overlap.

The ultimate goal is to accelerate the map building process to the point where the map can be reconstructed fast enough to cope with dynamic environments. Our preliminary experiments show that a significant speedup can be achieved that, while not yet at the level necessary to deal with dynamic environments, is still encouraging given the potential to increase the level of parallelization in the future.

The next section discusses the concept of a task space and TRMs. Sections 3 and 4 describe how TRMs are evolved using NGIK, and how coordinated parallel version is realized, respectively. Finally, section 5 presents our preliminary results using the simulated iCub robot.

2 Task-Relevant Roadmaps

Well known robot planning methods [14], such as Rapidly Exploring Random Trees [9] and Probabilistic roadmap planning [12] are able to build non-colliding motion plans by randomly sampling configuration space. However, the plans they produce often generate unnatural movement, especially with high-DOF robots, because they are not constrained.

Recent work has acknowledged the lack of control over how the configuration space is searched, and that to gain control the notion of a task space is required: a specialized coordinate system that often reduces the dimensionality of how poses are defined and provides a way of easily comparing poses in terms of the features (angles, distances) that are most relevant to the task (see figure 1).

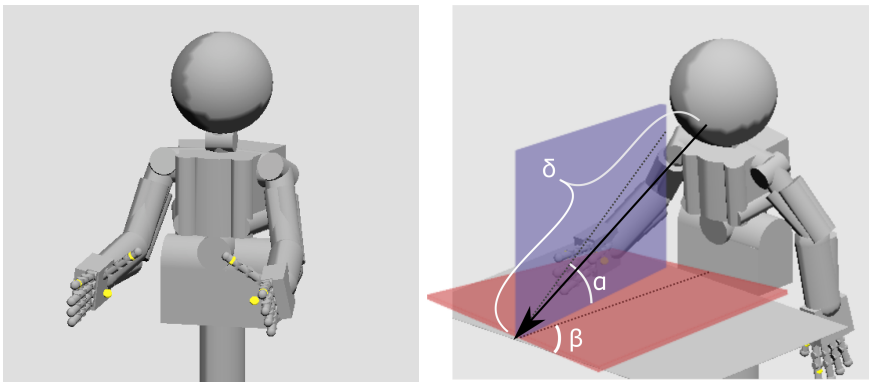

Fig. 1. The iCub Humanoid Robot. (left) the iCub, simulated in MoBeE, shown in its*home pose*. (right) an example task space designed to inspect an object from different angles and distances, formed as $\{\alpha, \beta, \delta\} = t \in \mathcal{T}$, where δ is the distance to the point, and α and β are angles that the head makes with respect to the object.

Formally a task space is defined by a real-valued map of the form:

$$g : \mathcal{P} \times \mathcal{Q} \times \mathcal{W} \rightarrow \mathcal{T} \subseteq \mathbb{R}^m,$$

where $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{W}$ are the poses of the body parts, the configuration space, and the world state respectively. $\mathcal{T}$ is a task space of m dimensions. Examples of task-maps are shown in Table 1. The task space guides the search to a sub-space of configurations that correspond to poses which relate to a given task. For example if the task is to hold an object at a fixed distance from the head.

The STOMP algorithm [10] allows for flexible, arbitrary cost-functions and plans a path that minimizes these costs. However, it plans directly in the configuration space, without considering how a selected path maps to task space. CBiRRT [4] uses a rapidly exploring random tree on a constrained manifold; a subset of the configuration space defined by constraints, and can create impressive movements. While CBiRRT has been augmented with the concept of task space regions [5] to focus the search to feasible regions of the configuration space, it can only use a restricted set of constraints that are projectable to task space. Berenson et al. [5] mention that they use a direct sampling algorithm that allows for "arbitrarily complex" constraint parameterization, but only use it to sample goals and *not* to plan paths as it "can be difficult to generate samples in a desired region".

Clearly it is desirable to have a maximum flexibility in the defining constraints and task spaces, but current approaches either cannot handle such flexibility, use it only in a part of their algorithm, or find only *one* posture and not a full movement. Recently several methods have approached both IK and planning, aiming to be generic and flexible to use [3, 8, 11, 16]. These methods have impressive results, but always put certain restrictions on constraints that can be used and often have difficulty with high degrees of freedom.

Type	Task Space Dimension(s)	Formula
Position	The position of a body part. Can be masked to select only a certain dimension of the position.	$g_{position} = v_{bodypart}$
Rotation	The rotation of a body part. Can be masked to select only a certain rotation.	$g_{rotation} = \boldsymbol{u}_{bodypart}$
Distance	The distance between a body part v_1 and another body part or object v_2.	$g_{distance} = \|v_1 - v_2\|$
Angle	The angle of the vector from a body part v_1 to another body part or object v_2, projected on a plane defined by $\boldsymbol{u}_{dim1}$ and $\boldsymbol{u}_{dim2}$.	$g_{angle} = \arctan(\boldsymbol{u}_{dim1}^T(v_1 - v_2), \boldsymbol{u}_{dim2}^T(v_1 - v_2))$

Table 1. Several examples of task-maps.

In [17], we tackled complex IK and planning at the same time, by combining a sample-based inverse kinematics solver with an iterative roadmap construction strategy. The resulting Task-Relevant Roadmaps (TRMs) provide a graph of poses that are evenly spread over the task space, and can be used to plan natural, constrained motions. The next section describes how TRMs are evolved using NGIK.

3 Evolving TRMs

The goal of the method is to build a map of between configurations and points in task space: $\{(q_1, t_1), (q_2, t_2), .., (q_k, t_k)\} \subset \mathbf{M}$ that have a maximum coverage of the task space $\mathcal{T}$. TRMs are constructed incrementally by repeatedly applying Natural Gradient Inverse Kinematics (NGIK) to discover configurations which satisfy task constraints, and adding them to an initially empty map. NGIK searches the configuration space, using Natural Evolution Strategies (NES;[7]), in a neighborhood around points already in the map to minimize a cost or fitness function:

$$h(p, q, w) = \sum_i \alpha_i h_i(p, q, w), \tag{1}$$

where $h_i : \mathcal{P} \times \mathcal{Q} \times \mathcal{W} \rightarrow \mathbb{R}^+$ is the i-th cost-function which has as input the poses of all body parts p, the joint-state vector q, and world state w, and outputs a non-negative cost. Each function is weighted by α_i.

The NES family of black-box optimization algorithms use parameterized probability distributions over the search space, instead of an explicit population (i.e. a conventional ES). Typically the distribution is a multivariate Gaussian parameterized by $\theta = (\mu, \Sigma)$, where μ is the mean vector, and Σ is the covariance matrix. Each generation, a set of samples is taken from the distribution and

evaluated. The distribution is then updated the direction of the natural gradient, in order to minimize/maximize the expected fitness of the distribution.

Algorithm 1 describes the map building procedure in pseudo-code. First, the empty map is initialized and the map-building constraint,

$$h_{map} = \sum_{\{q',t'\}\in \mathrm{NN}(m,t,\mathbf{M})} \underbrace{|d - \|t - t'\||}_{construction} + \underbrace{c\|q - q'\|}_{smoothness}, \tag{2}$$

is added to the given constraints h which describe the desired task for the particular robot in question, where t is the task-vector calculated by the task-map $t = g(p, q, w)$, and $\mathrm{NN}(m, t, \mathbf{M})$ calculates the m nearest neighbors to t in map $\mathbf{M}$. The first term *constructs* the map by pulling the solution close to the previous points in task space, but keeps it at a certain distance d, growing the map. The second term enforces *smoothness* by minimizing the change in joint angles over neighboring points in the map, where c is a weighting constant.

Each iteration through the while loop attempts to search for a new configuration to add to the map, starting from an existing point that is selected by SelectProportional$(\cdot)$. If the map is empty, the configuration of the home posture (see figure 1) is used. The returned configuration in the map, q_{start}, becomes the starting point for the evolutionary search conducted by NES.

At each generation, NES takes λ samples from the current search distribution over the configuration space (the 41-dimensional joint-space, in the case of the iCub). Each sample (candidate configuration), $q_i, i = 1..\lambda$, is evaluated by first computing its corresponding pose, p, in operational space, using the forward kinematics, f, and then plugging x, p, and the state of the world, w into fitness function $h()$ (equation (1)). After all samples are evaluated, the distribution parameters, θ, are updated.

The cycle terminates, returning a proposed configuration, q', when the search distribution has converged according to some pre-defined criteria. A converged distribution means that the search is in a local minima. q' is then mapped to task space by $g(\cdot)$ and added to the map if t' is outside the current map and is a non-colliding posture. These hard constraints must be checked because, even though they form part of the fitness function, they only penalize violations numerically, but do not prevent NES for accidentally converging to an unfeasible configuration. The process of adding points continues until the algorithm fails to generate a new, acceptable configuration after a predefined number of attempts, k.

Once the map is constructed, edges are added between elements using an n-nearest-neighbor connection strategy in the configuration space, which then can be used to plan motion using, e.g. an A^* planner (see [17] for further details).

4 Parallelizing E-TRM

Parallelizing E-TRM is not a simple matter of distributing Algorithm 1 over multiple cores. The independent searches in configuration space must be coordinated in order ensure that they do not duplicate work or interfere with each

Algorithm 1: E-TRM(h, k)

```
i ← 0
M ← {}                                          // initialize an empty map
h* ← h + α_map h_map                            // add the map-build cost function
while i < k do
    q_start ← SelectProportional(M)             // choose element from map
    q' ← NES(h*, q_start)                       // minimize h*, return optimized config
    t' ← g(q')                                  // compute corresponding point in task space
    if Check(q', t') then                       // check hard constraints and if t' not in map
        M ← M ∪ (q', t')
    else
        i++
    end
end
```

Procedure selectProportional($\mathbf{M}$)

```
if Empty?(M) then
    return q_home
end
scores ← {}
for p ∈ M do
    scores ← scores ∪ CountNeighbours(M, p) + fails[p]
end
selection ← {}
for p ∈ M do
    if scores[p] = min_p scores then
        selection ← selection ∪ p
    end
end
return SelectRandom(selection)
```

other in building the map. This coordination is implemented by introducing an additional *repulsion* constraint for each of the active NES searches:

$$h^i_{repel} = \sum_{i \neq j} \max[0, d - \|g(\mu_j) - g(\mu_i)\|], \tag{3}$$

which penalizes the fitness of an individual from the i-th NES if the center of the distribution from which it was drawn, μ_i, is close to other centers $\mu_j, i \neq j$. This constraint pushes the separate NES distributions away from each other so that they move into uncharted parts of task space rather than search in the same region. Algorithm 2 shows Parallelized E-TRM in pseudocode. The most important differences from the non-parallel version (Algorithm 1) are lines 4, where the repulsion constraint is added, 6-10, where the separate NES algorithms are initialized, and 11-12, where the NES searches are updated (one generation) in parallel.

Algorithm 2: Parallel E-TRM(h, k, P)

```
1  M ← {}                                          // initialize an empty map
2  S ← {}                          // initialize an empty set of NES algorithms
3  h* ← h + α_map h_map                    // add the map-build cost function
4  h* ← h* + α_repel h_repel                  // add repulsion cost function
5  while i < k do
6      while size(S) < min(P, size(M)) do
7          q_start ← SelectProportional(M)
8          Initialize (s,q_start)                                // start a new NES
9          S ← S ∪ s                          // add it to already active set
10     end
11     parallel_foreach s ∈ S do
12         Update(s)
13     endfor
14     foreach s ∈ S do
15         if Converged?(s) then
16             S ← S \ {s}                                  // remove s from the set
17             q' ← getMean(s)
18             if Check(q', t') then
19                 M ← M ∪ (q', t')
20             else
21                 i++
22             end
23         end
24     end
25 end
```

5 Experiments

5.1 Setup

The iCub robot [18], with the full 41 degrees-of-freedom, was simulated using MoBeE [6] which performs fast forward kinematics calculations and collision detection, using the SOLID 3.5.6 [2] collision detection library. Parallelization was implemented using the Threading Building Blocks library, which is responsible for spawning threads and assigning them to each core [15]. The constraints rely on fast nearest neighbours searches, thus we use the Approximate Nearest Neighbors library, which uses kd-trees for efficient search [1].

The method was evaluated on a reaching task, where the task space is formed by the (x, y, z) coordinates at the center of the right-hand palm of the iCub. A collision constraint and homepose constraint prevent the iCub from hitting itself and guide the robot into more natural postures. Two more constraints are added to keep the left hand oriented straight and close to a certain position, to keep it out of the way of the right hand (see [17] for complete details). The distance parameter d was set such that the points in the constructed map are spaced roughly 2.5cm from each other in task space.

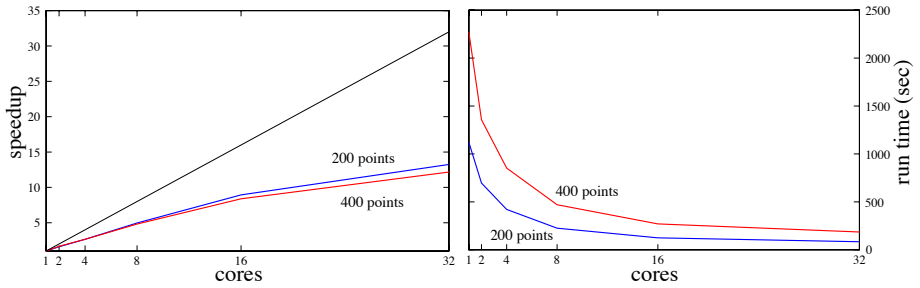

Fig. 2. Parallel speedup. The curves show the speedup achieved for each number of cores when building a map with 200 points (upper curve), and 400 points (lower curve).

The population size for every NES, λ, was set to 30 and the covariance matrices were initialized with values of 0.03 on the diagonal. In other words, NES initially searches using a standard deviation of 0.03 for every joint. For the very first point of the map, different values are used as there is no other point to start from, and thus the search needs to be more thorough. We use a population of 150 and a standard deviation of 0.15 in this case.

The stopping criterion (the CONVERGED?() function in Algorithm 2) used to decide when an individual NES should stop searching and return a new point, works by maintaining two moving averages, one averaging the last 20 fitness values of an individual at the center, μ, of the distribution, and one averaging the last 40. If the average of the last 20 is higher than the that of the last 40, the search is considered to have stagnated, and is terminated. The parameters were determined experimentally to lead to a good tradeoff between quality of results and speed.

A set of 10 simulations was run for each of six levels of parallelization: $1, 2, 4, 8, 16$, and 32 cores. All simulations were run until the map contained 400 points, on a 64-core Dell PowerEdge C6145[1].

5.2 Results and Discussion

Figure 2 shows the average performance gain afforded by Parallel E-TRM over (serial) E-TRM . The graph on the left plots the speedup for each number of cores for reaching the first 200 and 400 points. The black diagonal line represents the ideal, where speedup equals the number of cores. The graph on the right shows the performance in terms of the amount of real time required to reach 200 and 400 points, for a given level of parallelization. Parallelization at 32 cores reduces the time to generate a 400-point graph from 2267 seconds ($\approx$ 37 minutes) to 186 seconds. While this is a far cry from what would be required to allow for planning in even very slowly changing dynamic environments, the speedup of

[1] with 4 AMD Opteron 6376 16-core CPUs running at 2.3 GHz, and 128 GB of RAM (16×8 GB modules).s

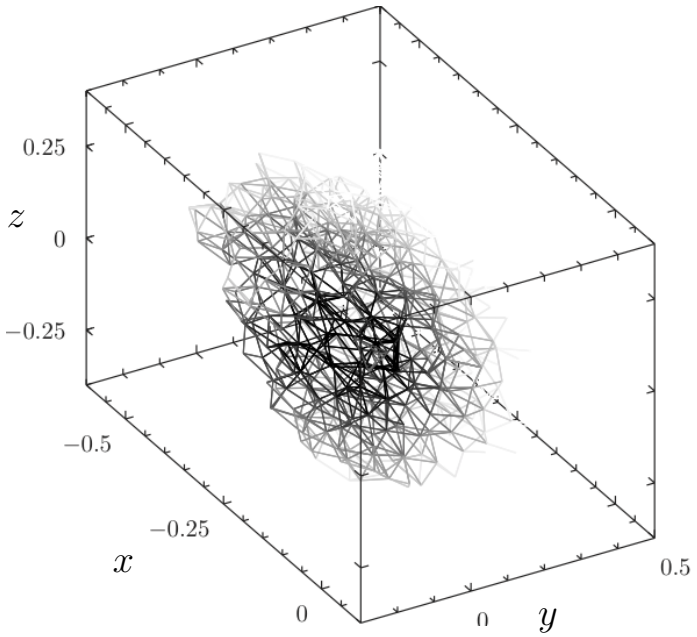

Fig. 3. Evolved TRM. The plot shows one for the evolved TRMs plots in 3D task space (see figure 1). Each point int the graph represents a pose in which the right hand of the robot is at location (x, y, x).

12× is significant. Moreover, the difference between the speedup for 200 and 400 was not found to be statistically significant ($\rho = 0.05$), which indicates that the rather modest parallel efficiency is not due to the increasing size of the map. Instead, the hardware architecture seems to suffer from a memory hierarchy that is not well suited to high-throughput parallel access [13] Future experiments will migrate the system to new hardware with substantially larger L3 caches.

References

[1] libann: A library for approximate nearest neighbor searching. `http://www.cs.umd.edu/~mount/ANN/`, .
[2] Solid: Collision detection library. `http://www.dtecta.com/`, .
[3] Julia M Badger, Stephen W Hart, and JD Yamokoski. Towards autonomous operation of robonaut 2. 2011.
[4] D. Berenson, S.S. Srinivasa, D. Ferguson, and J.J. Kuffner. Manipulation planning on constraint manifolds. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 625–632. IEEE, 2009.
[5] D. Berenson, S. Srinivasa, and J. Kuffner. Task space regions a framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*, 30(12):1435–1460, 2011.

[6] Mikhail Frank, Jürgen Leitner, Marijn Stollenga, Simon Harding, Alexander Förster, and Jürgen Schmidhuber. The modular behavioral environment for humanoids and other robots (MoBeE). In *ICINCO*, pages 304–313. SciTePress, 2012. ISBN 978-989-8565-22-8.

[7] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 393–400. ACM, 2010.

[8] Kris Hauser, Victor Ng-Thow-Hing, and Hector Gonzalez-Baños. Multimodal motion planning for a humanoid robot manipulation task. *Robotics Research*, pages 307–317, 2011.

[9] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2719–2726. IEEE, 1997.

[10] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 4569–4574. IEEE, 2011.

[11] Marcelo Kallmann, Yazhou Huang, and Robert Backman. A skill-based motion planning framework for humanoids. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2507–2514. IEEE, 2010.

[12] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[13] Desktop-HPC Lab. First results for swift on a 64-core amd opteron 6376. `https://community.dur.ac.uk/pedro.gonnet/?p=269`.

[14] Steven M LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[15] James Reinders. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism*. O'Reilly Media, Inc., 2010.

[16] Luis Sentis and Oussama Khatib. A whole-body control framework for humanoids operating in human environments. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2648. IEEE, 2006.

[17] Marijn Stollenga, Leo Pape, Mikhail Frank, Jürgen Leitner, Alexander Förster, and Jürgen Schmidhuber. Task-relevant roadmaps: A framework for humanoid motion planning. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 5772–5778, 2013.

[18] N.G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A.J. Ijspeert, M.C. Carrozza, et al. iCub: the design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10):1151–1175, 2007.