Task-Relevant Roadmaps: A Framework for Humanoid Motion Planning

Marijn Stollenga[†*], Leo Pape[*], Mikhail Frank[*], Jürgen Leitner[*], Alexander Förster[*], Jürgen Schmidhuber[*]

***Abstract*— To plan complex motions of robots with many degrees of freedom, our novel, very flexible framework builds task-relevant roadmaps (TRMs), using a new sampling-based optimizer called Natural Gradient Inverse Kinematics (NGIK) based on natural evolution strategies (NES).**

To build TRMs, NGIK iteratively optimizes postures covering task-spaces expressed by arbitrary task-functions, subject to constraints expressed by arbitrary cost-functions, transparently dealing with both hard and soft constraints. TRMs are grown to maximally cover the task-space while minimizing costs. Unlike Jacobian methods, our algorithm does not rely on calculation of gradients, making application of the algorithm much simpler. We show how NGIK outperforms recent related sampling algorithms. A video demo (`http://youtu.be/N6x2e1Zf_yg`) successfully applies TRMs to an iCub humanoid robot with 41 DOF in its upper body, arms, hands, head, and eyes. To our knowledge, no similar methods exhibit such a degree of flexibility in defining movements.

I. Introduction

Humanoid robots are designed to have the same degrees-of-freedom (DOF) as humans which quickly add up to more than 40. In contrast: a typical industrial robot arm has 6 or 7 DOF. Planning coordinated movements with such high DOF is still a major challenge in robotics, and there is a need for frameworks that can achieve this.

Most humanoid control frameworks take a dynamical approach [1], [2], [3]. These frameworks can combine several dynamical constraints related to a task, and add them together as forces to create combined movements. However, the forces are computed locally and require intricate knowledge of the robot's dynamics, making them unsuited for employing planning algorithms.

We introduce a novel framework that takes the more traditional approach of inverse kinematics, but combines it with recent inverse kinematics and planning approaches to succesfully plan complex movements on a humanoid robot. By using sampling algorithms, complex postures can be optimized using only the forward model [4], [5], [6]. To this end we create a new inverse kinematics solver called Natural Gradient Inverse Kinematics (NGIK) that uses the Natural Evolution Strategies (NES) algorithm [7] to optimize the posture of the robot. We show NGIK outperforms similar approaches in this domain, and can successfully optimize postures on a humanoid robot.

NGIK is combined with a novel roadmap construction algorithm, that combines recent ideas from planning literature to focus the search of the configuration space to subspaces that are relevant to a task [8], [9], [10]. Our algorithm iteratively constructs a roadmap to maximally cover a task-space, freely defined by the user, resulting in task-relevant roadmaps (TRM) that can be (re)used to *plan* motion trajectories in task or configuration-space. Our framework puts as little requirements on the constraints as possible, deals with *hard* and *soft* constraints transparently, and allows us to freely define the task-space. We show the effectiveness of our approach on the iCub humanoid robot [11] using the 41 DOF of its upper body in different manipulation tasks.

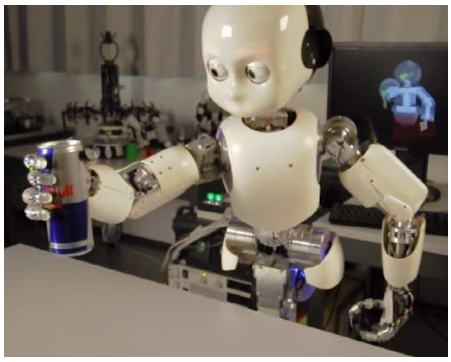

Fig. 1. The framework is applied to the iCub humanoid [11], resulting in smooth, natural motions.

II. Background

The state of a robot is given by its joint angles $q \in \mathcal{Q} \subseteq \mathbb{R}^n$ where n is the number of joints and $\mathcal{Q}$ is the configuration space consisting of all valid robot states. A typical approach to control a robot consists of the following steps:

1) *Inverse Kinematics:* Define a desired posture of the robot and find a corresponding robot state q_{goal} using an optimization algorithm.
2) *Planning:* Find a path from the current joint-state $q_{current}$ to the goal-state q_{goal}.
3) *Control:* Use a controller to control the robot to its goal over this path.

Our framework focuses on *inverse kinematics* and *planning*, and delegates *control* to a relavely simple controller.

Inverse Kinematics

Using the forward kinematics function f we can calculate the pose of every body part, given the robot state. Calculating the inverse f^{-1} is known as the *Inverse Kinematics* problem (IK), which would map our desired pose expressed in operational space, to a configuration-state. Although f is

* The authors are with the Dalle Molle Institute for Artificial Intelligence (IDSIA) / SUPSI / Universitá della Svizzera Italiana (USI), Lugano Switzerland. This research was supported by the *IM-CLeVeR* EU project, contract no. *FP7-ICT-IP-231722*.

† Corresponding author: `marijn@idsia.ch`

trivial to compute, even for complex robots, f^{-1} is non-linear and has in general infinite solutions and does not exist mathematically.

The traditional approach to IK is to explicitly find f^{-1} by constraining the problem until a *closed form* solution is obtained [12], [13]. This approach is very fast, but requires careful engineering and is restrictive in the constraints that can be used. Numerical approaches indirectly find f^{-1} by iteratively optimizing using the gradient ∇f, often by calculating the pseudo-inverse or transpose of the Jacobian [14], [15]. Recent work applies such methods to humanoid robots [16] and adds an efficient way to handle prioritized hard constraints. However, the constaints have to be setup carefully and the dimensionality of the system has to be controlled to compute the inverse efficiently. Although these approaches are much more flexible than *closed form* solutions, they are sensitive to singularities and require the gradient/Jacobian to be known, which restricts the set of constraints and kinematic chains that can be represented. It also can not take the curvature and non-linearity of f into account, and only looks locally at the *slope*.

Recently sampling methods, have tried to circumvent there problems. Such methods never explicitly calculate f^{-1} or the gradient ∇f, but estimate it by *sampling* from f, so they can deal with arbitrary cost-functions, making them much more flexible and robust than traditional IK algorithms. Also, higher order interactions between dimensions that wouldn't be detected by a Jacobian approach, will be noticed by a samping based algorithm.

Several sampling optimizers have been used; such as Simulated Annealing [4], which is very flexible but has only been used for small kinematic chains. Sequential Monte Carlo (SMC) [5] uses a non-parametric distribution of particles, but relies on good proposal distributions that puts constraints on the kinematic chain that can be used. A particle filter method [6] has been used in the computer game "Spore" that allows the player to create their own widely different creatures, dealing with unpredictable and complex kinematic chains.

Our method is based on these insights, but uses Natural Evolution Strategies [7] (NES) as an optimizer, as will be explained in Section III. NES has state of the art performance, beating its closest competitor CMA-ES on some tasks, and is robust and easy to use. We show in experiments that NES outperforms the other optimization approaches and can solve the IK problem for complex poses on our iCub humanoid with 41 DOF.

Flexible Inverse Kinematics with Planning in mind

Planning methods have a rich history and are an active field of research [17]. Rapidly Exploring Random Trees [18] randomly grow trees in configuration space $\mathcal{Q}$ that quickly cover the search-space until a non-colliding path is found. Probabilistic roadmap planning [19] is a popular method where a traversable graph of robot states is constructed by randomly sampling from $\mathcal{Q}$ creating a graph with non-colliding edges in the process. By planning and moving through this graph, the robot moves from one position to the other while avoiding obstacles and self-collisions.

These planning methods have the disadvantage that there is little control over how the configuration-space is searched. Recent work has acknowledged the lack of control over the search space: The STOMP algorithm [8] allows for flexible arbitrary cost-functions and plans a path that minimizes these costs. However, it plans directly in the configuration-space, leaving the mapping between a task-space and configuration-space open.

CBiRRT [9] uses a rapidly exploring random tree on a constrained manifold; a subset of the configuration-space defined by constraints, and can create impressive movements. CBiRRT was augmented with the concept of task-space regions [10] to focus the search regions of the configuration space. However, it can only use a restricted set of projectable constraints and task-space regions. Berenson et al. do mention 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 can't handle such flexibility, use it only in a part of their algorithm, or find only *one* posture and not a full movement. Recently several frameworks have approached both IK and planning, aiming to be generic and flexible to use [1], [2], [3], [20]. These frameworks have impressive results, but always put certain restrictions on constraints that can be used and often have difficulty with high degrees of freedom.

Our framework tackles complex IK and planning at the same time by combining our novel *sampling based* inverse kinematics solver NGIK with an iterative construction strategy. It finds a family of *reusable* postures that are optimized under constraints defined by arbitrary cost-functions, and at the same time maximally covers a user-defined task-space. Connecting these postures creates a traversable graph, called a task-relevant roadmap (TRM). In other words, the task-relevant constraints are built directly into the TRM. As shown in Section V, it allows us to build TRMs that can perform useful tasks in the 41-dimensional configuration space of the upper body of the iCub humanoid.

III. Natural Gradient Inverse Kinematics

We want to find a robot posture with desired properties. Therefore we define cost-functions that calculate a cost from robot state and a world state, represented by the poses of the bodyparts and joint angles, and poses of objects in the world respectively.

Let $f : \mathcal{Q} \rightarrow \mathcal{P}$ be the *forward kinematics* function that maps the configuration-space to the operational space. The set of poses of all l body parts $p = \{p_1, p_2..p_l\} \in \mathcal{P}$ form the *operational space*, where every body part is indexed by a number[1]. Here $p_i \in SE(3)$, where $SE(3)$ is the special

[1]Instead of indexing a body part by a number we can also use the name of a body part, e.g. p_{right_hand}, directly.

Type	Purpose	Formula
Home Posture	Bias body posture search to a stable posture. In case multiple solutions exist for a given point in task-space, the home posture allows to find a unique solution. It also increases similarity in configuration space between nearby points in task-space.	$h_{home} = \|q - q^*\|$, where q^* is a home posture and $\|\|\dot{\|}$ is the $L2$-norm.
Collision	Penalize collisions. xNES is able to estimate a gradient over collisions using the number of collisions. This number usually increases with deeper penetration.	$h_{collision} = \|collisions\|$.
Position	Attract a body part to a pixed position or to another body part. The target can be specified as an area or volume in task-space, rather than a point.	$h_{position} = \|v_{bodypart} - v^*\|$, where v^* is the desired position.
Orientation	Control the orientation of a body part. The desired orientation can be either a fixed orientation in task-space, or the orientation of another body part.	$h_{orientation} = (\vec{u}_{base}^T R_{bodypart} \vec{u}_{desired} + 1)/2$. Here $\vec{u}_{base}$ is the unit base vector that has to point to the desired direction $\vec{u}_{desired}$ after being rotated over the transformation matrix $R_{bodypart}$ of the body part.
Pointing	Point a body part toward another body part or a certain position in task-space. Optionally it also keeps the end effector at a fixed distance to this point or body part.	$h_{pointing} = (\vec{u}_{base}^T R_{bodypart} \frac{v_{target} - v_{bodypart}}{\|v_{target} - v_{bodypart}\|} + 1)/2$.
Repelling	Repels two body parts away from each other. Repulsion can optionally start after a minimum distance and is helpful for finding non-colliding postures and to increase mobility in the map.	$h_{repel} = min(0, d - \|v_{bodypart1} - v_{bodypart2}\|)/d$.

TABLE I

A VARIETY OF COST-FUNCTIONS ARE SHOWN, DESCRIBING THEIR PURPOSE AND CORRESPONDING FORMULA.

Euclidean group. $SE(3)$ expresses both the position $v_i \in \mathbb{R}^3$ and the orientation of a body part, represented by $R_i \in SO(3)$, where $SO(3)$ is the special orthogonal group.

We sum all cost-functions toghether in *one* cost-function:

$$h(p, q, w) = \sum_i a_i h_i(p, q, w)$$

where $h_i : \mathcal{P} \times \mathcal{Q} \times \mathcal{W} \to \mathbb{R}_{\geq 0}$ is the i-th cost-function with as input the poses of all body parts p, the joint-state vector q, and world state w, and output a non-negative cost. Each function is weighted by a_i to control its strength.

Because we use a sampling-based evolutionary algorithm, the functions h_i don't have to be differentiable. This allows us to transparently handle *hard constraints* and *soft constraints* with discrete and continuous functions respectively. This setup gives us maximum flexibility in the expression of constraints, leaving room for creative cost-functions.

We show several cost-functions in Table I. All cost-functions have internal parameters to give finer control their behaviour. For example, the home posture function can penalize the torso more than its arms to prefer more efficient movements. And the position and repel functions can ignore certain dimensions, resulting in attraction toward a plane instead of a point.

Natural Evolution Strategies (NES)

Finding robot postures under general constraints often involves a non-convex and non-smooth search space, which makes it particularly relevant to use the right search algorithm. Comparing a range of optimization methods, Rios et. al [21] found that evolutionary strategies perform exceptionally well in non-convex non-smooth optimization problems.

We use the NES, a principled method for real-valued evolutionary optimization [7], to minimize the cost-function. It is shown to be beat its closest competitor CMA-ES on some benchmarks while being based on a more principled approach. It represents its current population as a Gaussian distribution embedded in the search space, the configuration space in our case, and samples from it. From these samples it calculates the natural gradient to minimize the function. We use the improved variant of NES called Exponential NES (xNES), that is invariant to linear transformations to the search space.

To create robust movements, the optimizer needs find postures that are removed from sudden increases in the cost-function (e.g. a millimeter away from a collision). We can either create cost-functions that explicitly calculate and penalize being close to collisions, but this can be computationally prohibitive, puts constraints on the possible colliding shapes, and doesn't take other cost-functions into account. Instead we add a (small) Gaussian noise to the robot state. The result is that NES optimizes the *expected value* of h under noise on the robot state, implicitly finding robust poses taking *all* constraints into account.

IV. CONSTRUCTING A TASK-RELEVANT ROADMAP

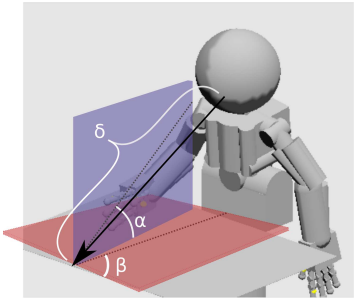

Fig. 2. This example shows a task-space designed to inspect an object from different angles and distances. The task-space is formed as $\{\alpha, \beta, \delta\} = t \in \mathcal{T}$, where δ is the distance to the point, and α and β are angles that the head makes with respect to the object.

The forward kinematics function maps the configuration-

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.	$g_{rotation} = \vec{u}_{bodypart}$
Distance	The distance between a body parts 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 $\vec{u}_{dim1}$ and $\vec{u}_{dim2}$.	$g_{angle} = \arctan(\vec{u}_{dim1}^T(v_1 - v_2), \vec{u}_{dim2}^T(v_1 - v_2))$

TABLE II

SEVERAL EXAMPLES OF TASK-FUNCTIONS.

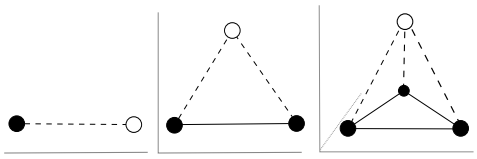

Fig. 3. The map building constraint is minimized when a predefined distance d to n nearest graph nodes is achieved. The number of nearest neighbours corresponds to the dimensionality of the task-space as shown in the figures.

space to the operational space $f : \mathcal{Q} \to \mathcal{P}$. Often, it is not adequate to express a task directly in the operational space. Therefore we introduce a task-space $\mathcal{T}$ that is calculated by a freely chosen task-function:

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

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. The task-function g can be any real-valued function. An example of a task-space is shown in Figure 2, and task-functions are shown in Table II.

TRM Construction Algorithm

Now we can introduce the TRM-constructing algorithm, by augmenting NGIK with a map-constructing strategy. The goal is to build a map of robot-states with corresponding task-states $\{(q_1, t_1), (q_2, t_2), .., (q_k, t_k)\} \subset \mathbf{M}$ that have a maximum coverage of the task-space $\mathcal{T}$, while minimizing the cost-function h for all states. To build the TRM we add a special map-constructing *cost-function* to h which is defined as follows:

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

where t is the task-vector calculated by the task-function $t = g(p, q, w)$, q is the robot state, and $\mathrm{NN}(m, t, \mathbf{M})$ calculates the m nearest neighbors to t in map $\mathbf{M}$.

The first part *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 part accounts for *smoothness* by minimizing the change in joint angles over neighbouring states, where c is a weighting constant. Figure 3 shows how the map is grown, in task-spaces of different dimensions. Note that the number of nearest neighbours is tied to the dimensionality of the task-space. Depending on the sampled posture, different nearest neighbours are found, resulting in an adaptive growing behavior.

Figure 4 shows a schematic of the TRM-building algorithm, which works as follows: We are given a cost-function $h = \sum_i a_i h_i$ and an empty map $\mathbf{M} = \varnothing$. We create an augmented cost-function by adding the map building constraint $h^* = h + a_{construct} h_{construct}$. Then we repeat the following until satisfied with the map:

1) **Selection** Select a point $(t', q') \subset \mathbf{M}$ to initialize NGIK based on previous success rate. This favours parts of the map that can easily expand, over parts of the map that have, for example, reached physical limits of the robot. If the map is empty, we use the standard home posture.
2) **Optimization** Initialize NGIK with q' and minimize the augmented cost-function h^* resulting in a new configuration- and task-vector q_{new} and t_{new}.
3) **Post-selection** We check if the resulting point t_{new} resides outside the current map, and check if q_{new} is not a colliding posture. If this is true, add the point to the map: $(q_{new}, t_{new}) \xrightarrow{add} \mathbf{M}$ and increase the success rate of (t', q').

After the algorithm is finished, a post-process step adds edges to the map $\mathbf{M}$ using an n-nearest-neighbour connection strategy in the configuration space, resulting in a TRM. As all maps are defined in the same configuration space, we can connect several maps built for different tasks. This allows us to plan movements through different maps for combined tasks. To create *adaptive planning*, we use an A^* planner which evaluates the traversability of edges online. The algorithm heuristically searches for a free path to a goal that is not obstructed under the current world-state.

V. EXPERIMENTS

For the forward kinematics we use the Modular Behavioral Environment (MoBeE) [22], which calculates the forward kinematics, and performs collision detection at a high frequency. We specify a model of the iCub humanoid in XML. The same model is used throughout the whole process, from building TRMs to controlling the iCub. This allows us to apply the framework easily to other robots. We use the upper body of the iCub, comprising the torso (3 DOF), the arms (2 $\times$ 7 DOF), the hands (2 $\times$ 9 DOF), the neck (3 DOF), and the eyes (3 DOF) giving a total of 41 DOF.

We compared the performance of NGIK against two versions of Sequential Importance Resampling (SIR), Metropo-

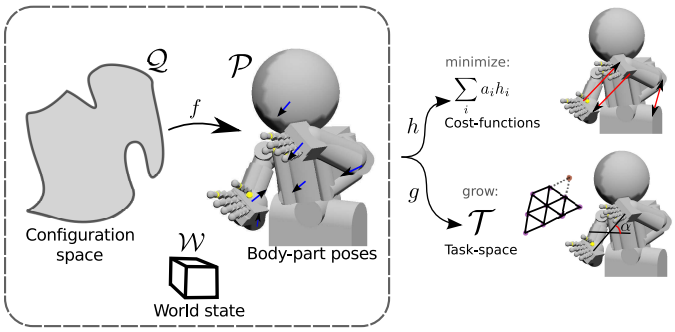

Fig. 4. *TRM-building algorithm:* The postures of body parts $\mathcal{P}$ are calculated from the configuration space $\mathcal{Q}$ using the forward function f. The information of the world state is represented in $\mathcal{W}$. The TRM-building algorithm will iteratively find postures that incrementally cover the task-space $\mathcal{T}$, while minimizing the cost-functions. Both the task-function g defining the task-space, and the cost-function h defining the constraints, can use all information from $\mathcal{Q}, \mathcal{P}, \mathcal{W}$ allowing flexibility in defining TRMs.

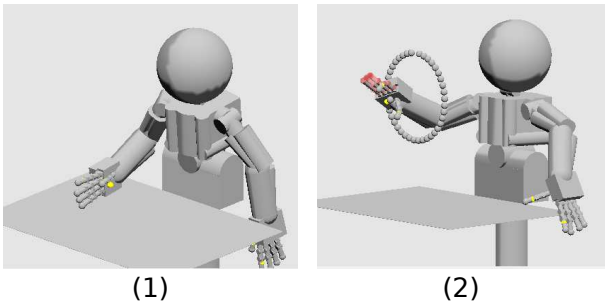

Fig. 5. The two postures used to compare the different optimization algorithms. Left posture is constrained to hold the left hand behind the table and the right hand above it. The right posture is a challenging grabbing posture through a loop using the thumb and index finger. The results are shown in Table III

lis Hastings (MH) sampling and the simplex optimization algorithm [23]. To make the comparison fair, we use simple Gaussian proposal distributions for SIR and MC instead of tailored distributions that use information about the robot [5], as NGIK also doesn't assume extra knowledge on the task at hand. We don't compare NGIK against CMA-ES as it is already shown that performance differences are small [7] with NES often beating CMA-ES on complex problems. We also don't compare to Jacobian inverse/transpose based methods, as these can't handle the cost-functions we use and thus simply are not applicable in this framework.

SIR is the method closest to the capabilities of NGIK, as it uses a population of samples for optimization. Every sample has a corresponding weight, which is updated as new samples are drawn from previous samples. SIR resamples when the effective weights are below a threshold, in which case a new population is sampled according to the weights, and the weights are reset. We found that using a threshold of 75% of the number of particles works well. We also compare a direct version of SIR that resamples on every iteration, which we call SIR direct (SIRD). The optimal standard deviation of the initial search distributions were determined experimentally.

The Simplex optimization algorithm is meant for convex optimization problems, which we compare to to show that such algorithms are not suited for the optimization problem at hand.

The algorithms are used to optimize two challenging postures shown in Figure 5. The left posture (Fig. 5-1) requires the left hand to be behind the table and the right hand to be in front of the robot. The right posture (Fig. 5-2) is more difficult and requires a reaching posture through a loop.

NGIK evaluation

The results are shown in Table III, where we calculated the average best fitness value after 30 runs, and the corresponding standard deviation of the mean. The population size for SIR, SIRD and XNES was 300; a relatively large population size to prevent local optima. The standard deviation of the proposal distributions was $\sigma = 0.005$ for MC, SIR and SIRD. NES has an initial standard deviation of $\sigma = .4$, which is higher than for the other methods as it can adjust the distribution dynamically. The other algorithms don't perform well unless the deviations are relatively small. All algorithms keep track of the best encountered fitness and the time it was encountered. If the best fitness doesn't improve for a large number of evaluations, the algorithm is stopped. We store the number of evaluations at the time the best fitness was encountered as the running time.

NGIK outperforms the other algorithms. The SIMPLEX algorithm has the lowest performance as it assumes a convex problem and gets stuck early in the optimization. SIR and SIRD have even performance, although their performance is significantly lower than NES, as it can not adjust the proposal distribution. MC is similar to the SIR algorithm with a population of 1, which accounts for its bad performance in such a high-dimensional search space. For posture (2), both SIR and MC find their best result after relatively few iterations and have trouble improving it afterwards, resulting in the low reported running times.

Performance and Build Time: The time it takes to build a map depends mainly on two factors: the desired density of postures and the dimensionality of the task-space. The number of postures needed to cover the map, and thus the number of times NGIK is run, scales approximately proportional to $\frac{l}{d}^n$, with l the approximate length to cover in a dimension, d the distance between points and n the dimensionality. For most maps a distance of a few centimeters between postures is enough (if the task-space is defined in cartesian space). In practice it takes a few minutes to build a 2d map, and tens of minutes for a 3d map. For higher dimensionality the search-time becomes prohibitive because of exponential scaling. We stress that, once created, the maps can be re-used in several tasks and building time is not an issue anymore.

TRM examples

Maps **a**-**j** in Figure 6 show several examples of TRMs that are tailored to certain tasks (a video-demonstration is shown at: `http://youtu.be/N6x2e1Zf_yg`). All maps use the collision cost-function and home-posture cost-function

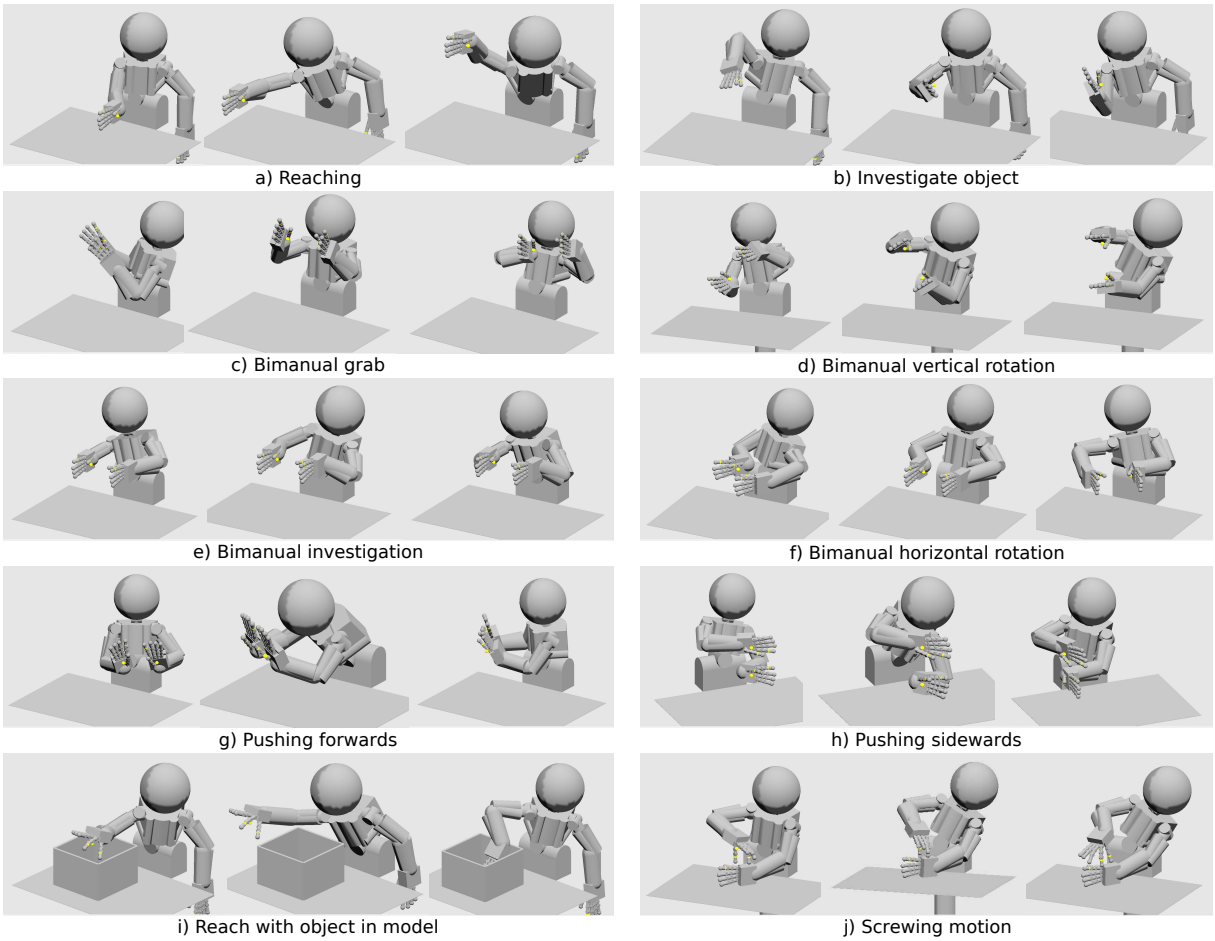

Fig. 6. Examples of movements made using task-relevant roadmaps.

TABLE III
COMPARISON OF NES VERSUS DIFFERENT OPTIMIZATION ALGORITHMS. THE POSTURES ARE SHOWN IN FIGURE 5.

Algorithm	Final Cost Posture (1)	Nr. Evaluations
XNES	**0.1122** ±0.0269	199290 ±11444
SIR	0.5068 ±0.0342	144920 ±16172
SIRD	0.4492 ±0.0356	158190 ±24613
MH	13.5507 ±5.7343	80432 ±17200
SIMPLEX	111.113 ±3.8616	420090 ±0
	Final Cost Posture (2)	Nr. Evaluations
XNES	**0.1765** ±0.0207	239040 ±3824
SIR	1.3167 ±0.0278	9260 ±2181
SIRD	1.2835 ±0.0272	15330 ±3950
MH	1.4934 ±0.0352	11320 ±3009
SIMPLEX	1.661 ±0.019	420090 ±0

to create collision free postures that look natural.

Basic Manipulation: Map–**a** was built for a grasping task. As a task-space we use the position of the left hand. The hand is free to move, but its orientation is kept upright by an orientation cost-function. Finally a pointing constraint is added from the head to the hand to keep the eyes fixed on the moving hand, which would be required for a typical grasping task. The table is added to the world model as a static object. Map–**b** restricts the position of the right hand, but allows it to rotate freely. By using the angles of the hand relative to the head, the TRM allows the humanoid to investigate an object in its hand from different angles.

Bimanual Manipulation: We created several maps that can manipulate a box. Because the box is big, we use bimanual manipulation. The left hand is constrained to point to the right hand, and vice versa, using pointing cost-functions. Map–**c** uses the point between the left and right hand as the task-space, resulting in bimanual reaches. Map–**e** fixes the position of the hands, and uses the position of the head as a task-space, creating a TRM that can investigate the box between the hands.

If we use the angle between a virtual 'rod', between the left and right hand, and the z- or y-axis as a task-space, we get vertical and horizontal rotating motions respectively (maps **d** and **f**). By constraining the hands to be parallel and fix their relative position, we can get pushing motions. We can create forwards and sidewards pushing motions by controlling their orientation, shown in maps **g** and **h**.

Obstacles: To avoid obstacles we can either use our planning algorithm on a general TRM to plan around it, or put the object in the model while building the TRM. The latter approach is shown in map–**i**, where a box is added,

while using the same constraints as the simple reach TRM of map–**a**. This allows us to find difficult solutions that cannot easily be found or specified without putting an object in the model.

To show the complexity of movements that TRMs can express, we also build a map to perform a unscrewing movement. By putting pointing cost-functions on the thumb and index finger we can create a grasping posture. By fixing the point between the two fingers and setting the angle of one of the fingers as the task-space we can build a TRM that creates a unscrewing motion, shown in map–**j**.

VI. Conclusion and Discussion

We presented a new framework to create task-relevant roadmaps (TRMs) for planning complex movements on a 41 DOF humanoid. We introduced a new inverse kinematics algorithm called NGIK that uses NES to optimize a posture. Its sampling-based nature allows us to use arbitrary cost-functions and to transparently incorporate hard and soft constraints.

We used NGIK to construct TRMs through iterative search for new postures that maximally cover a user-defined task-space. The method is more flexible than related methods and we showed its advantages by applying it to the full 41-DOF of the upper body of the iCub humanoid robot.

Currently roadmaps are represented by discrete points in configuration and task-space, densely packed to create smooth trajectories. The number of points, however, scales exponentially with task-space dimensionality. To efficiently deal with higher dimensions, a single graph node could hold a linear manifold (instead of discrete points) representing a large part of the configuration space.

Roadmaps are currently built offline, making the framework less adaptive than dynamic full-body control frameworks [1], [2], [3]. But the algorithm is easily parallelized in on-line fashion, since samples within a NES iteration are independent. Also, the (natural) gradient calculation of NES is suitable for direct control, yielding a method as dynamic as previous ones, while retaining the flexibility of the new framework.

Ongoing work will extend TRMs to incorporate reinforcement learning algorithms. Since TRMs encode families of discrete task-relevant postures, they lend themselves to repeatedly testing similar sequences of interactions with an environment, to learn the order of actions needed to achieve a given goal.

References

[1] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 2641–2648.

[2] J. M. Badger, S. W. Hart, and J. Yamokoski, "Towards autonomous operation of robonaut 2," 2011.

[3] K. Hauser, V. Ng-Thow-Hing, and H. Gonzalez-Baños, "Multi-modal motion planning for a humanoid robot manipulation task," *Robotics Research*, pp. 307–317, 2011.

[4] M. Dutra, I. Salcedo, and L. Diaz, "New technique for inverse kinematics problems using simulated annealing," in *Int. Conf. on Engineering Optimization*, 2008, pp. 01–05.

[5] N. Courty and E. Arnaud, "Inverse kinematics using sequential monte carlo methods," *Articulated Motion and Deformable Objects*, pp. 1–10, 2008.

[6] C. Hecker, B. Raabe, R. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, "Real-time motion retargeting to highly varied user-created morphologies," in *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3. ACM, 2008, p. 27.

[7] T. Glasmachers, T. Schaul, S. Yi, D. Wierstra, and J. Schmidhuber, "Exponential natural evolution strategies," in *12th annual conference on Genetic and Evolutionary Computation*. ACM, 2010, pp. 393–400.

[8] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 4569–4574.

[9] D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*. IEEE, 2009, pp. 625–632.

[10] D. Berenson, S. Srinivasa, and J. Kuffner, "Task space regions a framework for pose-constrained manipulation planning," *The Int. Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, 2011.

[11] N. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. Ijspeert, M. Carrozza, *et al.*, "icub: the design and realization of an open humanoid platform for cognitive and neuroscience research," *Advanced Robotics*, vol. 21, no. 10, pp. 1151–1175, 2007.

[12] A. Hemami, "A more general closed-form solution to the inverse kinematics of mechanical arms," *Advanced robotics*, vol. 2, no. 4, pp. 315–325, 1987.

[13] M. Kauschke, "Closed form solutions applied to redundant serial link manipulators," *Mathematics and Computers in Simulation*, vol. 41, no. 5, pp. 509–516, 1996.

[14] W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *The 23rd IEEE Conference on Decision and Control*, vol. 23. IEEE, 1984, pp. 1359–1363.

[15] M. Zohdy, M. Fadali, and N. Loh, "Robust control of robotic manipulators," in *American Control Conference*. IEEE, 1989, pp. 999–1004.

[16] P. Baerlocher and R. Boulic, "An inverse kinematics architecture enforcing an arbitrary number of strict priority levels," *The visual computer*, vol. 20, no. 6, pp. 402–417, 2004.

[17] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[18] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 3. IEEE, 1997, pp. 2719–2726.

[19] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[20] M. Kallmann, Y. Huang, and R. Backman, "A skill-based motion planning framework for humanoids," in *Robotics and Automation (ICRA), 2010 IEEE Int. Conf. on*. IEEE, 2010, pp. 2507–2514.

[21] L. M. Rios and N. V. Sahinidis, "Derivative-free optimization: A review of algorithms and comparison of software implementations," *Journal of Global Optimization*, pp. 1–47, 2011.

[22] M. Frank, J. Leitner, M. Stollenga, S. Harding, A. Förster, and J. Schmidhuber, "The modular behavioral environment for humanoids and other robots (mobee)," in *ICINCO*. SciTePress, 2012, pp. 304–313.

[23] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the nelder–mead simplex method in low dimensions," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 112–147, 1998.