# Task-Relevant Roadmaps: A Framework for Humanoid Motion Planning

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

***Abstract*— We introduce a novel framework that builds task-relevant roadmaps (TRMs). TRMs can be used to plan complex task-relevant motions on robots with many degrees of freedom. To this end we create a new sampling based inverse kinematics optimizer called Natural Gradient Inverse Kinematics (NGIK), based on the principled heuristic solver called natural evolution strategies (NES). We build TRMs using a construction algorithm that iteratively optimizes postures using NGIK that cover an arbitrary task-space.**

**NGIK can express constraints with arbitrary cost-functions and thus can transparently deal with *hard* and *soft* constraints. The construction algorithm can express the task-space by arbitrary task-functions and grows the TRM to maximally cover the task-space while minimizing the constraints. These properties make the framework very flexible while still handling complex movements in robots with high degrees of freedom.**

**We show NGIK outperforms recent algorithms in this domain and show the effectiveness of our method on the iCub robot, using the 41 DOF of its full upper body, arms, hands, head, and eyes. A video-demo shows TRMs applied on real robotic tasks on the iCub humanoid `http://vps9114.xlshosting.net/trm.mp4`. To our knowledge there are no similar methods that allow this degree of flexibility in defining the 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 for different humanoids 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.

To incorporate planning, 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], [8] to optimize the posture of the robot. We show NGIK outperforms similar approaches in this domain, and can successfully optimize poses on a humanoid robot.

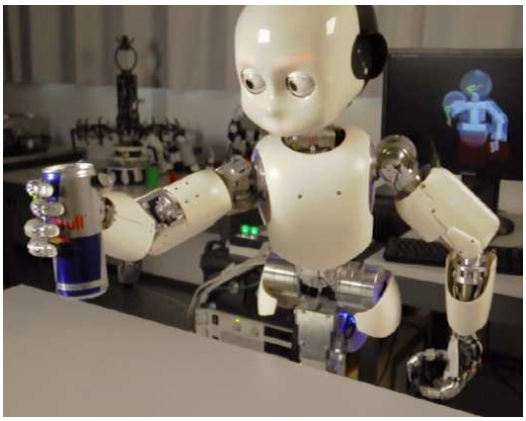

Fig. 1. The framework is applied to the iCub humanoid [12], resulting in smooth, natural motions.

We combine this optimizer with a construction algorithm, that combines recent ideas from planning literature that allow to focus the search of the configuration space to subspaces that are relevant to a task [9], [10], [11]. Our algorithm iteratively constructs a roadmap to maximally covers a task-space, freely defined by the user, resulting in task-relevant roadmaps (TRM) that can be used to *plan* motion trajectories in task or configuration-space. The resulting trajectories can then be followed to *control* the humanoid robot. Our framework puts as little requirements on the constraints as possible and allows us to freely define the task-space. We show the effectiveness of our approach on the iCub humanoid robot [12] using the 41 DOF of its upper body in different manipulation tasks.

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

[*] The authors are with the Dalle Molle Institute for Artificial Intelligence (IDSIA) / SUPSI / Universitá della Svizzera Italiana (USI), Lugano Switzerland

[†] Corresponding author: `marijn@idsia.ch`

Our framework focuses on step (1) and (2) and we use a relatively simple controller to deal with the dynamics of the humanoid robot for step (3).

*Step 1: 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). The latter function would map our desired pose, expressed in a operational space, to a configuration-state. Although $f$ is 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 finding $f^{-1}$ by constraining the problem until you get a *closed form* solution [13], [14]. Closed form approaches are very fast, but require careful engineering and are restrictive in the constraints that can be used. Numerical approaches indirectly find $f^{-1}$ by iteratively optimizing using the gradient $\nabla f$ [15], [16], [17], [18]. Such methods are much more flexible but are sensitive to singularities and require the gradient to be known, which again restricts the set of constraints and kinematic chains that can be represented.

Recent approaches have tried to ease these restrictions by solving IK through sampling methods. Such methods never explicitly calculate $f^{-1}$ or the gradient $\nabla f$, but iteratively estimate it by sampling from $f$. This optimization approach can deal with arbitrary cost-functions, making them much more flexible than traditional IK algorithms.

Several sampling optimizers have been used; such as Simulated Annealing [5]. This approach is very flexible, but has only been used for small kinematic chains. Sequential Monte Carlo (SMC) [4] uses a non-parametric distribution of particles to iteratively guess better solutions. The method however 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 creatures with widely different morphology's. The authors note that the sampling method allows them to deal with unpredictable and complex kinematic chains that can be created.

Our method is based on these insights, but uses natural evolution stategies [7], [8] (NES) as an optimizer, as will be explained in section III. NES has state of the art performance 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.

*Step 2: Planning*

Once the goal state in configuration-space is known, the robot has to find a way to go there, without running into obstacles. Planning methods have a rich history and are an active field of research [19], [20]. Rapidly exploring random trees [21] randomly grow trees in configuration space $\mathcal{Q}$ that quickly cover the search-space until a non-colliding path is found. Probabilistic roadmap planning [22], [23] 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 subspace searched in planning algorithms: The STOMP algorithm [9] 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 [10] uses a rapidly exploring random tree on a constrained manifold; a subset of the configuration-space defined by constraints. The method can control the searched configuration-space, but can only use a restricted set of constraints and isn't applicable to high-dimensional configuration-spaces. CBiRRT was also augmented with the concept of a task-space regions [11], which focus the search to preferred regions of the configuration space. The method is however limited to restricted types of task-spaces.

*Combining step 1 and 2*

Our framework tackles step 1 and 2 at the same time by finding a family of postures that are optimized using an inverse kinematics solver, and at the same time maximally cover a user-defined task-space. By connecting these postures we create a traversable graph, called a task-relevant roadmap (TRM).

By combining our new inverse kinematics solver NGIK with a iterative construction strategy, these TRMs both allow arbitrary cost-functions to control the constraints on the map, and arbitrary task-functions to flexibly control the task-space that the map represents. In other words, the task-relevant constraints are built directly into the roadmap. The algorithm gives us the flexibility to define any task-function that defines the dimensions of our task-space, and automatically maximizes the reach of the roadmap. As shown in section V, it allows us to build roadmaps 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 control the robot to a posture with desired properties. Let the set of poses of all body-parts $p = \{p_1, p_2..p_n\} \in \mathcal{P}$ form the *operational space*. Here $p_i \in SE(3)$[1], where $SE(3)$ is the special 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. In section IV we will augment the operational space into a more flexible task-space $\mathcal{T}$.

Let $f : \mathcal{Q} \rightarrow \mathcal{P}$ be the *forward kinematics* (FK) function that maps the configuration-space to the operational space.

[1]Instead of the number $p_1$ we can also use the name of a body part $p_{right\_hand}$ directly.

TABLE I

SEVERAL EXAMPLES OF COST-FUNCTIONS.

| Type | Description |
|---|---|
| Home Posture | A full body posture in configuration space to which the robot is attracted: $h_{home} = \|q - q^*\|$, where $q^*$ is a home posture and $\|\|$ is the $L2$-norm. In case multiple solutions exist for a given point in task-space, the home posture allows to find a unique solution. It also enforces similarly in configuration space between nearby points in task-space. |
| Collision | Penalizes collisions: $h_{collision} = \|collisions\|$. xNES is able to estimate a gradient over collisions using the number of collisions. This number usually increases with deeper penetration. |
| Position | Attracts a bodypart to a pixed position or another bodypart: $h_{position} = \|v_{bodypart} - v^*\|$, where $v^*$ is the desired position. In case of a fixed position, the target can be specified as an area rather than a point. |
| Orientation | This constraint prefers a certain orientation of a bodypart: $h_{orientation} = \vec{u}_{base}^T \mathbf{R}_{bodypart} \vec{u}_{desired}$. Here $\vec{u}_{base}$ is the base vector that has to point to the desired $\vec{u}_{desired}$, and $\mathbf{R}_{bodypart}$ is the transformation matrix of the bodypart. The desired orientation can be either a fixed orientation in task-space, or the orientation of another bodypart. |
| Pointing | Points the orientation of a bodypart toward a fixed position in task-space or to the position of another bodypart: $h_{pointing} = \vec{u}_{base}^T \mathbf{R}_{bodypart} \frac{v_{target} - v_{bodypart}}{\|v_{target} - v_{bodypart}\|}$. Optionally it also keeps the end effector at a fixed distance to this point or bodypart. |
| Repel | The repel constraint repels two body-parts away from each other: $h_{repel} = min(0, d - \|v_{bodypart1} - v_{bodypart2}\|)/d$. Repulsion can optionally start after a minimum distance. The repel constraint is helpful for finding non-colliding postures and to increase mobility in the map. |

To represent arbitrary cost-functions, we simply combine all cost-functions we are interested in into a sum:

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

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

The cost-functions can use any information encoded in the arguments. The only requirement for function $h_i$ is that its minimum is equal to 0. Because we use an evolution-based algorithm, the functions 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 allows for maximum flexibility in the expression of constraints, leaving room for creative cost-functions. In section V we show implementations of constraints as cost-functions.

### *Natural Evolution Strategies*

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 on combinations of (non)convex (non)smooth optimization problems Rios et. al [24] found that evolutionary strategies perform exceptionally well in non-convex non-smooth optimization problems.

We chosex the Natural Evolution Strategies (NES) algorithm [7] to minimize the cost-function $h$. NES is a principled method for real-valued evolutionary optimization. It represents its current state 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) [8], that is invariant to linear transformations to the search space.

To achieve invariance to monotonically growing (i.e., rank preserving) transformations, xNES uses *fitness shaping*: It sorts the sampled population in order of decreasing fitness, and then assigns new fitness-values according to their position. The effect of fitness shaping is that the best ranked individuals get a significantly higher fitness than lower ranked ones, regardless of the actual difference in fitness. This prevents diluting the selection pressure when the difference between fitness values gets small towards the optimum. This is especially important for joints such as the fingers, that have smaller effects on the cost-function due to smaller movements, but are equally important.

We experimentally show the effectiveness of NES compared to similar methods (Section V). We found that NES can find minima of the function it minimizes that are on the edge of drastic increases in the function. Thus we would find postures that are less than a milimeter away from a collision. To find more robust postures, we add a gaussian noise to the robot state before $h$ is evaluated: $q' = q + n, n \sim \mathcal{N}(0, \sigma^2 I)$, where $\sigma$ is the standard deviation of the noise, and $I$ the identity matrix. The result is that NES optimizes the *expected value* of $h$ under noise on the robot state, which increases the distance to colliding postures.

### *Designing Cost-Functions*

One of the main design principles we used for cost-functions is to prefer linear growing functions over higher order functions. For example, we often use the regular $L2$ norm when computing distances, and not the squared distance. The result are functions that are easy to manage as they don't suddenly overpower each other, and more intuitive for a human designer. We show several cost-functions in table I.

Each dimension of a constraints can be modified by a weight, which allows for setting the importance of the dimension, or disable it altogether. For example, the home posture constraint can give more weight to keeping the torso close to home, and decrease the weights further down the kinematic chain. In case of the position and repel constraints,

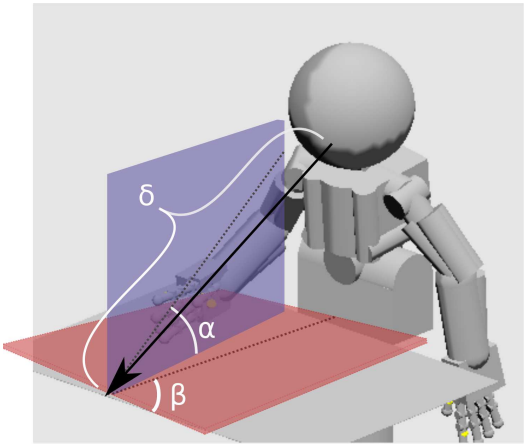


Fig. 2. An example of a task-space. In this example the task is to look inspect an certain point from different angles and distances. The task-space is this case is $\{\alpha, \beta, \delta\} = t \in \mathcal{T} \subseteq \mathrm{R}^3$, where $\delta$ is the distance to the point, and $\alpha$ and $\beta$ are angles that the head makes with respect to the object, projected on two planes. The corresponding task-functions can be implemented using simple geometric functions (see Fig II).

disabling dimensions results in attraction toward (repulsion from) a point, a line or a plane, while for the pointing and orientation constraints only certain rotation angles can be stressed or fixed.

## IV. Constructing a Task-Relevant Roadmap

The forward kinematics function maps the configuration-space to the operational space $f : \mathcal{Q} \rightarrow \mathcal{P}$. Often, it is not adequate to express a task directly in the operational space. Therefore we introduce a task-space $\mathcal{T}$ that is more flexible than operational space $\mathcal{P}$. It is created by the task-function:

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

Where $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{W}$ are the configuration space, poses of the body-parts, and world state respectively. $\mathcal{T}$ is a task-space of $m$ dimensions. The task-function $g$ can be any real-valued function, thus the task-space is more flexible than the operational space $\mathcal{P}$. An example of a task-space is shown in Figure 2.

We can now augment NGIK with a map-constructing strategy to create the TRM-building algorithm. The goal of the algorithm is to build a map of robot-states with corresponding task-states $\{(q_1, t_1), (q_2, t_2), .., (q_n, t_n)\} \subset \mathsf{M}$ that have a maximum covering in the task-space $\mathcal{T}$, while minimizing the cost-function $h$ for all states.

To build the map we add a special map-construction cost-function to $h$ which is defined as follows:

$$h_{map} = \sum_{\{q^0, t^0\} \in \mathrm{NN}(t, M, n)} \underbrace{|d - \|t - t'\|_2|}_{construction} + c \underbrace{\|q - q'\|}_{smoothness}$$

where $t$ is the current task-vector calculated by the task-function $t = h(p, q, w)$, $q$ is the current robot state, $\mathrm{NN}(t, \mathsf{M}, n)$ calculates the $n$ nearest neighbors to $t$ in map $\mathsf{M}$. The first part of the function accounts for the *construction* of the map using $d$ as the desired distance in task-space from existing points in the map. The second part accounts for *smoothness* by minimizing the change in joint angles over neighbouring states, where $c$ is a weighting constant.

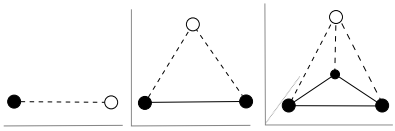

Fig. 3. The map building constraint is minimized when a predefined distance to $n$ nearest graph nodes is achieved. Using $n = 1$ (left) results in a 1-simplex (line), $n = 2$ (middle) results in a 2-simplex (triangle) and $n = 3$ results in a 3-simplex (tetrahedron). They are used for building 1-d, 2-d and 3-d maps respectively.

Figure 3 illustrates the effect of this function. It pulls the solution close to the previous points in task-space, but keeps it at a certain distance, growing the map by adding simplexes. Depending on the posture, different nearest neighbours are found, resulting in an adaptive growing behavior. The dimensionality can be controlled by setting the number of nearest neighbors; using 2 nearest neighbors would add 2-simplexes, or triangles, to build a 2-d map. If we use 3 nearest neighbors, we add 3-simplexes, or tetrahedrons, to build 3d maps[2].

### *TRM-construction Algorithm*

The TRM-building algorithm is shown in Figure 4 and works as follows: We are given a cost-function $h = \sum_i a_i h_i$ and an empty map $\mathsf{M} = ?$. 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:

- **Selection** Select a point $(t', q') \subset \mathsf{M}$ to initialize NGIK based on previous success rate. This favours parts of the

[2]For two dimensions the resulting graph is regular in task-space. However in three dimensions task-spaces are irregular because tetrahedrons do not pack well. This however does not affect the usability of the map

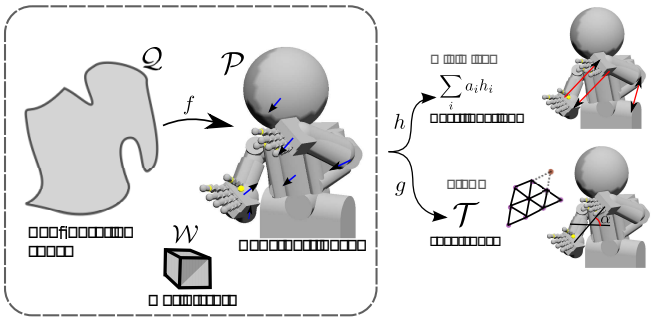


Fig. 4. *TRM-building algorithm:* The postures of bodyparts $\mathcal{P}$ are calculated from the configuration space $\mathcal{Q}$ using the forward function $f$. The information of the world state is represented in $\mathcal{W}$. The TRM-building algorithm will iteratively find postures that incrementally 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.

TABLE II

SEVERAL EXAMPLES OF TASK-FUNCTIONS.

| Type | Task Space | Calculation |
|---|---|---|
| Position | The position of a bodypart. Can be masked to select only a certain dimension of the position. | $g_{position} = v_{bodypart}$ |
| Rotation | The rotation of a bodypart. Can be masked to select only a certain rotation. | $g_{rotation} = \vec{u}_{bodypart}$ |
| Distance | Calculates the distance between a bodyparts $v_1$ and another bodypart or object $v_2$. | $g_{distance} = \|v_1 - v_2\|$ |
| Angle | Calculates the angle of the vector from a bodypart $v_1$ to another bodypart or object $v_2$, projected on a plane defined by $\vec{u}_{dim1}$ and $\vec{u}_{dim2}$. | $g_{angle} = \arctan 2((\vec{u}_{dim1}^T v_1 - v_2), \vec{u}_{dim2}^T (v_1 - v_2))$ |

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.

- **Optimization** Initialize NGIK with $q'$ and minimize the augmented cost-function $h^*$ using NES resulting in a configuration- and task-vector $q_{new}$ and $t_{new}$.
- **Post-selection** The optimizer might not be able to find a robot state from the chosen starting point. Thus 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: $(q_{new}, t_{new}) \rightarrow M$ and increase the success rate of $(t', q')$.

After the map is built, a post-process step adds edges to the map using an $n$-nearest-neighbour connection strategy in the configuration space. 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 in task-space that is not obstructed under the current world-state.

## V. EXPERIMENTS

*Software Architecture:* For the forward kinematics we use the Modular Behavioral Environment (MoBeE) [25], 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 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 * 7 DOF), the hands (2 * 9 DOF) , the neck (3 DOF), and the eyes (3 DOF) giving a total of 41 DOF.

### *NGIK evaluation*

We compared the performance of NGIK against two versions of sequential importance resampling (SIR), metropolis hastings (MH) sampling and the SIMPLEX algorithm. 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 [4]. This is fair as NGIK also doesnt assume extra knowledge on the optimization problem or the task at hand.

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.

reference simplex The SIMPLEX 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.

The results are shown in table III, where we calculated the average best fitness value and the corresponding standard deviation of this mean. All algorithms keep track of the best encountered fitness and the time it was encountered. If the best fitness doesnt 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 perform similarly, although their performance is 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. In summary, the advantages of NGIK is that it adjusts its search distribution dynamically and has several invariances to the structure of the cost-function, making it efficient in optimizing high-dimensional non-convex problems.

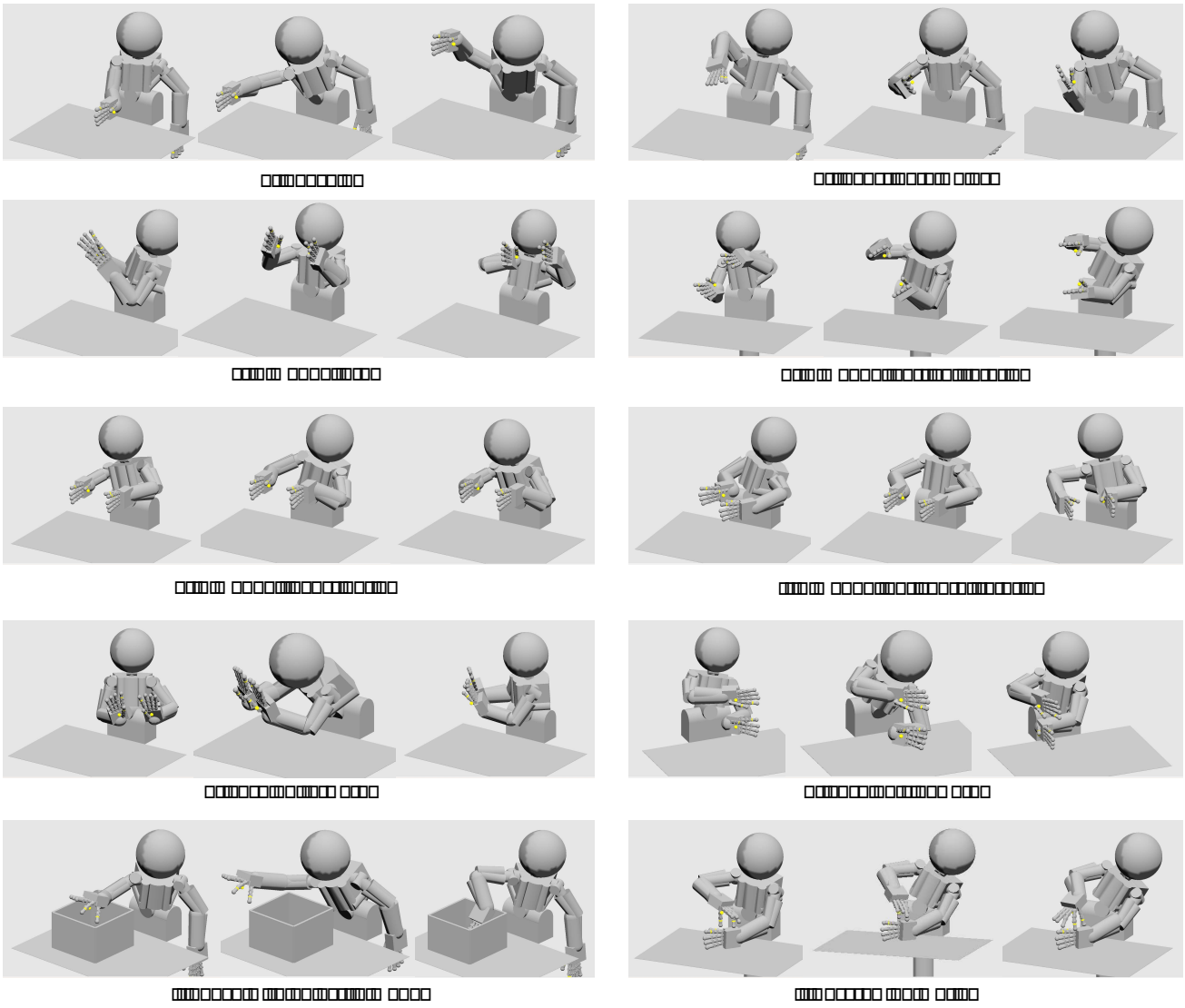

Fig. 6. Examples of movements made using task-relevant roadmaps.

TABLE III

COMPARISON OF NES VERSUS SIR. MEAN SCORES (AND STANDARD ERROR OF THE MEAN) OF THE COST-FUNCTION AFTER 10 RUNS ARE SHOWN. THE POSTURES ARE SHOWN IN FIGURE 5.

| Algorithm | Posture (1) | Nr. Evaluations |
|---|---|---|
| NES | 0.1562 (+-0.0416) | |
| SIR | 0.3717 (+-0.0467) | |
| SIRD | 0.4787 (+-0.0631) | |
| SIMPLEX | 111.113 (+-3.8616) | 420090 (0) |
| MH | 17.3045 (+-6.39837) | 6630.23 (+-692.022) |
| | Posture (2) | Nr. Evaluations |
| NES | 0.4034 (+-0.1596) | |
| SIR | 3.1429 (+-0.1011) | |
| SIRD | 3.1292 (+-0.0913) | |
| SIMPLEX | 1.661 (+-0.019) | 420090 (+-0) |
| MH | 1.434 (+-0.039) | 1969.37 (+-750.10) |

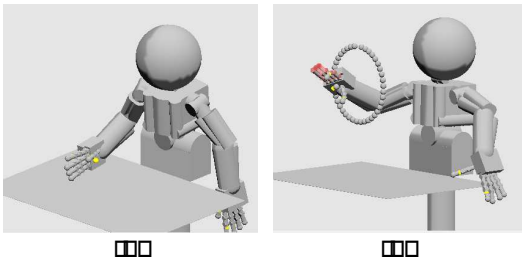

Fig. 5. The two postures used to compare NES and SIR. Left posture is constrained to hold the left hand behind the table and the right hand above it. The right posture is a challenging grabbing posture through a loop using the thumb and index finger. The results are shown in table III

*TRM examples*

Figure 6 shows several TRMs (a video-demonstration is shown at:

`http://vps9114.xlshosting.net/trm.mp4`).
All maps use the collision cost-function and home-posture cost-function to create collision free postures that look natural.

*Basic Manipulation:* Map-**a** (Fig. 6-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** (Fig. 6-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** (Fig. 6-c) uses the point between the left and right hand as the task-space, resulting in bimanual reaches. Map-**e** (Fig. 6-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, we get vertical and horizontal rotating motions respectively (maps **d** (Fig. 6-d) and **f** (Fig. 6-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** (Fig. 6-g) and **h** (Fig. 6-h).

*Dealing with obstacles:* We can use our planning algorithm to deal with obstacles. However if we put the object in the model while building the TRM, we can create more smooth motions. We can for example put a box in the model and use the same constraints as the simple reach TRM of map-**a** (Fig. 6-a). This create a TRM that can reach into the box, shown in map-**i** (Fig. 6-i).

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** (Fig. 6-j).

## VI. Discussion

There are certain disadvantages to the framework: Currently the roadmaps are represented by *discrete points* in configuration and task-space and are densely packed to create smooth trajectories. However, the number of points scales exponentially with the dimensionality of the task-space. To alleviate this, the nodes of the graph could represent manifolds to represent large parts of the configuration space with a single node, efficiently dealing with higher dimensions.

Another disadvantage is that roadmaps are currently build *offline*, making the framework less adaptive than the dynamic full-body control frameworks [1], [2], [3]. However the algorithm can largely be parallelized as the samples within a iteration of NES are independent, such that the algorithm could run on-line. Also, the (natural) gradient calculation in the NES algorithm can be used for direct control resulting in a dynamic algorithm close to the other frameworks, but with the flexibilities of the presented framework.

TRMs can also be extended to incorporate reinforcement learning algorithms. Because the TRMs represent a family of discrete task-relevant postures, they are very suited to repeatedly performing similar actions that interact with an environment and learn the order of actions needed to achieve a given goal. We leave this for future work.

## VII. Conclusion

We presented a new framework to create TRMs to plan complex movements on a 41 DOF humanoid. To this end, 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, allowing us to transparently incorporate *hard* and *soft* constraints. We used NGIK to create an constructing algorithm that iteratively finds new postures that maximally cover a user-defined task-space, resulting in task-relevant roadmaps. The method improves on other methods in its flexibility of constraint and task definitions. We showed the flexibility and effectiveness of the algorithm by applying it on the full 41-DOF of the upper body of the iCub humanoid robot.

## VIII. Acknowledgments

This research was supported by the *IM-CLEVER* EU project, contract no. *FP7-ICT-IP-231722*,

## References

[1] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments," in *IEEE International Conference 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] N. Courty and E. Arnaud, "Inverse kinematics using sequential monte carlo methods," *Articulated Motion and Deformable Objects*, pp. 1–10, 2008.

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

[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] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber, "Natural evolution strategies," in *IEEE Congress on Evolutionary Computation*. IEEE, 2008, pp. 3381–3387.

[8] 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.

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

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

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

[12] 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.

[13] 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.

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

[15] 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.

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

[17] C. W. Wampler, "Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.

[18] S. R. Buss and J.-S. Kim, "Selectively damped least squares for inverse kinematics," *journal of graphics, gpu, and game tools*, vol. 10, no. 3, pp. 37–49, 2005.

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

[20] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

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

[22] 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.

[23] D. Hsu, J. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *The International Journal of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.

[24] 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.

[25] 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*, J.-L. Ferrier, A. Bernard, O. Y. Gusikhin, and K. Madani, Eds. SciTePress, 2012, pp. 304–313.