Humanoid Posture Generation on non-Euclidean Manifolds

Stanislas Brossette[1,2], Adrien Escande[2], Grégoire Duchemin[2], Benjamin Chrétien[1,2], Abderrahmane Kheddar[2,1]

***Abstract*— We present a reformulation of the posture generation problem that encompasses non-Euclidean manifolds. Such a formulation allows a more elegant mathematical description of the constraints, which we exemplify through some scenarios in the simulation results section. In our previous work, the posture generation problem is formulated as a non-linear optimization program with constraints expressed only through Euclidean manifolds; we solve the latter problem using on-the-shelf solvers. Instead, we decided to implement a new SQP solver that is most suited to non-Euclidean manifolds structural objects. By doing so, we have a better mastering in the way to tune and specialize our SQP solver for robotic problems.**

I. Introduction

Computing robot configurations to meet the requirements of a given set of tasks, within a viable state, is a recurrent problem whose complexity grows with that of the robot. In this paper, we are interested in the following generalized inverse kinematics problem: we search a configuration for which the robot fulfills tasks under constraints of joint limits, auto-collision and non-desired collision avoidance, balance, torque limits, etc. We coined it posture generation. Such a problem is encountered in both planning and control. In both cases, computation time and robustness are critical issues.

We have already proposed various implementations of the humanoid posture generation problem. All of our implementations formulate the problem as a non-linear optimization program to address multi-contact planning. In [1], the multi-contact planner explores the contact space using thousands of HRP-2 humanoid posture generator (PG) queries; we used the FSQP solver [2]. In [3], the PG is extended to handle various humanoid robots and multiple agents, the solver used is IPOPT [4]. In [5] the PG is extended to various contact models and used to generate multiple related postures at once. The latter work and the DRC participation revealed that re-planning on the fly is necessary and having a robust PG is crucial in many situations. Other works also make use of PG, e.g. in [6][7].

Posture generation has been formulated as a problem over a Euclidean space. Robots variable may however be more naturally expressed over non-Euclidean manifolds. The archetypes for this are the rotation part of the root body for a humanoid robot, and ball joints, whose variables live in $SO(3)$. Some typical tasks are also naturally formulated on different manifolds. For example for making contact with any object that can be mapped on a sphere, the contact point position for this object can be parametrized in $S2$. Human shoulder can be elegantly parametrized on $S2 \times \mathbb{R}$, as proposed in [8].

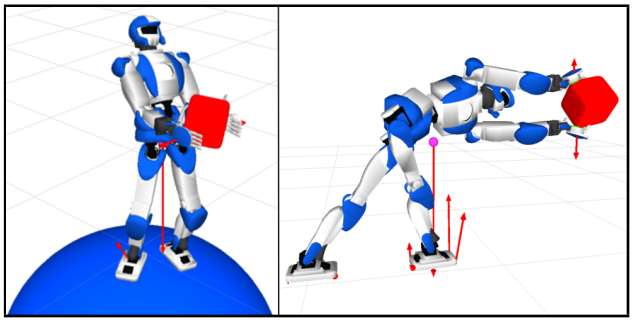

Fig. 1: HRP-4 carrying a 2-kg cube. Left: feet on a sphere, objective function is to maintain the cube at a given position. Right: right foot free to move on the floor, objective is to put the cube as far as possible in a given direction

Formulating the problem over $\mathbb{R}^n$ leads either to discontinuities that can prevent the convergence of the optimization solver, or to cumbersome writing to specify that the variable is actually living on a manifold (see [9]).

In this paper, we propose a new optimization solver able to work on generic smooth manifolds. We take inspiration from the approach used for unconstrained optimization on manifold [10] and adapt it to constrained optimization. To the best of our knowledge, constrained optimization on manifold has drawn few research for now. This is likely due to the fact that in most problems the only constraint is to be on the manifold. We are only aware of the work of Schulman *et al.* [11], where the authors explain the adaptation of their solver to work on $SE(3)$. This adaptation is however not valid for general manifolds without more care about hessian computation.

The second contribution of this paper is a Posture Generation framework developed to ease the writing of functions, so that the user can focus on the problem formulation without having to care about the tedious bookkeeping inherent to optimization problems of this size.

A background motivation for this work is to have our own optimization solver, instead of a black box. We will now be able to specialize the solver specifically to robotic problems, by leveraging modeling properties and approximations, for a gain in time and robustness. We also look forward to using this solver for problems with a varying number of constraints along the iterations (such as when complex collision constraints are considered).

[1]CNRS-UM2 LIRMM Interactive Digital Humans, UMR5600, Montpellier, France

[2]CNRS-AIST Joint Robotics Laboratory (JRL), UMI3218/RL, Tsukuba, Japan

This work was partially supported by the EU H2020 COMANOID project. A. Escande was funded by a grant from the Japan Society for Promotion of Science (JSPS; Grant-in-Aid for JSPS Fellows P13786)

The rest of the paper is organized in a classical way: we start with a bit of math to describe the foundations; then we introduce the PG *per se*, the problem formulation followed with illustration of successful generations.

II. Optimization on Manifolds

In this section, we describe a Sequential Quadratic Programming (SQP) approach [12] to solve the following non-linear constrained optimization program

$$\begin{aligned} \min_{x \in \mathcal{M}} \quad & f(x) \\ \text{subject to} \quad & l \leq c(x) \leq u \end{aligned} \tag{1}$$

where $\mathcal{M}$ is a n-dimensional smooth manifold and c is a m-dimensional real-valued function.

A. Representation problem

When $\mathcal{M} = \mathbb{R}^n$, the problem (1) is solved iteratively, starting from an initial guess x_0 and performing successive steps $x_{i+1} = x_i + \mathbf{p_i}$ where $\mathbf{p_i}$ is the increment found at the i-th iteration, until convergence is achieved. The strategy to compute $\mathbf{p_i}$ depends on the solver.

This classical scheme cannot be readily applied to optimization over non-Euclidean manifolds. First of all, only (a subset of) the real numbers can be stored in computers. To manipulate elements of $\mathcal{M}$ we need to choose a way to represent them in memory. This boils down to choosing a representation space $\mathbb{E} = \mathbb{R}^r$ (with $r \geq n$) and a map

$$\psi \; : \; \begin{matrix} x & \mapsto & \mathbf{x} \\ \mathcal{M} & \to & \mathbb{E} \end{matrix}$$

In the following, we identify $\mathcal{M}$ with the set $\psi(\mathcal{M}) \subseteq \mathbb{E}$.

With this representation, it is tempting to simply transform problem (1) as an optimization over $\mathbb{R}^r$ with objective $f \circ \psi^{-1}$ and constraint $c \circ \psi^{-1}$, and solve it with a usual solver. But depending on the representation choice, one of the two following problems arises:
(i) $r = n$, then it is not possible in the general non-Euclidean case to find ψ without derivative discontinuities. This can lead to critical convergence problems,
(ii) $r > n$, then most elements of $\mathbb{E}$ do not represent an element of $\mathcal{M}$ and ψ cannot be surjective. Constraints need to be added to force the solution on $\mathcal{M}$. As a result, the problem has more variables and constraints w.r.t (i). Moreover, the additional constraints are unlikely to be met along the iteration process (even if x_i is an element of $\mathcal{M}$, $x_i + \mathbf{p_i}$ is likely not, as nothing enforces it). This means that in order to evaluate $f \circ \psi^{-1}$ and $c \circ \psi^{-1}$ at a given x_i, one has to project it on $\psi(\mathcal{M})$ first, effectively computing $f \circ \psi^{-1} \circ \pi$ and $c \circ \psi^{-1} \circ \pi$, where π is the projection. The composition by π is an additional burden in programming (see e.g. in [13]).

As a simple example, the set of 3D-rotations $SO(3)$ is a manifold of dimension 3. The following (classical) choices can be made

- Rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3} \approx \mathbb{R}^9$, additional constraints: $\{\mathbf{R}^t\mathbf{R} = I \;,\; \det(\mathbf{R}) = 1\}$, projection by orthogonalization,
- Quaternion $\mathbf{q} \in \mathbb{R}^4$, additional constraints: $\{\|\mathbf{q}\| = 1\}$, projection $\pi(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|$,
- Euler angles ($\mathbb{E} = \mathbb{R}^3$), singularities when reaching gimbal lock.

B. Local parametrization

By definition, there is always, at a point x of a smooth n-dimensional manifold $\mathcal{M}$, a smooth map φ_x between an open set of $T_x\mathcal{M}$, the tangent space to $\mathcal{M}$ at x, and a neighborhood of x, with $\varphi_x(0) = x$. $T_x\mathcal{M}$ can be identified with $\mathbb{R}^n$. This gives us a local parametrization for $\mathcal{M}$. The driving idea of the optimization on manifolds is to change the parametrization at each iteration. Applying this idea, we can reformulate Problem (1) around x_i as

$$\begin{aligned} \min_{\mathbf{z} \in T_{x_i}\mathcal{M}} \quad & f \circ \varphi_{x_i}(\mathbf{z}) \\ \text{subject to} \quad & l \leq c \circ \varphi_{x_i}(\mathbf{z}) \leq b \end{aligned} \tag{2}$$

This is an optimization problem on $\mathbb{R}^n$. If we perform one iteration of a classical solver starting from $\mathbf{z_0} = 0$, we get an iterate $\mathbf{z_1}$, which corresponds to the iterate $x_{i+1} = \varphi_{x_i}(\mathbf{z_1})$. We can then reformulate Problem (1) around x_{i+1}, perform a new iteration and repeat the process until convergence.

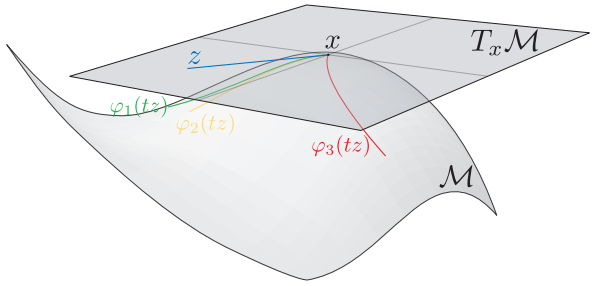

Fig. 2: There are many possible choices for φ_x but not all yield a curve $\varphi_x(t\mathbf{z})$ which is going in the same direction as $\mathbf{z}$: φ_1 and φ_2 are correct choices, φ_3 is not.

However, convergence cannot be achieved without care on the choice of φ_{x_i}: it must be such that for any $\mathbf{z}$, the curve $t \mapsto \varphi_{x_i}(tz)$ is tangent to $\mathbf{z}$, see Fig. 2, so that the update $x_{i+1} = \varphi_{x_i}(\mathbf{z_1})$ is made in the direction given by $\mathbf{z_1}$.

The exponential map is a good theoretical candidate, but it is often impractical or expensive to compute. Depending on the manifold, cheaper maps can be chosen.

With the iterative formulation approach described above, we do not have any parametrization issue, do not need additional constraints, and have the minimum number of optimization parameters. But we still need a map ψ and real space $\mathbb{E}$ to represent the x_i and keep track of them in a global way. The $\mathbf{x_i}$ are guaranteed to be on $\mathcal{M}$ so we can choose a representation with $r > n$ where ψ is singularity-free without any drawback. Also, the programmer can write the function $f' = f \circ \psi^{-1}$ as if it was a function from $\mathbb{E}$ to $\mathbb{R}$ without the need to project on $\psi(\mathcal{M})$ first (same goes for $c' = c \circ \psi^{-1}$). For example if $\mathcal{M} = SO(3)$ and $\mathbb{E} = \mathbb{R}^{3\times3}$, $\mathbf{x_i}$ is automatically a rotation matrix and can be used directly as such when writing the function.

C. Local SQP on manifolds

We choose to adopt an SQP approach to solve our problem. We first define the Lagrangian function

$$\mathcal{L}_x(\mathbf{z}, \lambda) = f \circ \varphi_x(\mathbf{z}) - \lambda^T c \circ \varphi_x(\mathbf{z}) \tag{3}$$

with $\lambda \in \mathbb{R}^m$ the vector of Lagrange multipliers, and note H_k the Hessian matrix $\nabla^2_{zz}\mathcal{L}_{x_k}$. Taking $\mathbf{z_0} = 0$, the first SQP step for Problem (2) is computed by solving the following quadratic program

$$\begin{aligned} \min_{\mathbf{z}\in\mathbb{R}^\mathbf{n}} \quad & \frac{\partial f \circ \varphi_{x_k}}{\partial \mathbf{z}}(0)^T \mathbf{z} + \frac{1}{2}\mathbf{z}^T H_k \mathbf{z} \\ \text{subject to } & 1 \leq c \circ \varphi_{x_k}(0) + \frac{\partial c \circ \varphi_{x_k}}{\partial \mathbf{z}}(0)\mathbf{z} \leq \mathrm{u} \end{aligned} \tag{4}$$

The basic SQP approach adapted to manifolds can be summarized as follows

1) set $k = 0$ and x_k to the initial value
2) compute $\mathbf{z}$ from Problem (4) for current x_k
3) set $x_k = \varphi_{x_k}(\mathbf{z})$
4) if convergence is not yet achieved go-to step 2

Computations of function values and derivatives are based on the fact that $f \circ \varphi = f' \circ \psi \circ \varphi$ (and same for c), and

$$\begin{aligned} f' &: \mathbb{E} \to \mathbb{R} \\ \psi \circ \varphi &: \mathbb{R}^n \to \mathbb{E} \end{aligned}$$

are representable functions. The gradient of $f \circ \varphi$ is

$$\frac{\partial f \circ \varphi_x}{\partial \mathbf{z}} = \frac{\partial f'}{\partial y}(\psi \circ \varphi_x) \times \frac{\partial (\psi \circ \varphi_x)}{\partial \mathbf{z}} \tag{5}$$

D. Practical implementation

The above SQP algorithm works locally, *i.e.* when starting close enough to the solution. In practice, various possible refinements are made to ensure convergence from any starting point. We detail hereafter our choices.

Maps φ_{x_i} are only valid locally, and we need to account for this: a step $\mathbf{z}$ found by Problem (4) should not be outside the validity region of the map. We could enforce this by adding a constraint $\mathbf{z}^-_{\text{map}} \leq \mathbf{z} \leq \mathbf{z}^+_{\text{map}}$ in (4). This leads naturally to trust region methods that we therefore favor over line-search approaches.

To know if a step $\mathbf{z}$ is acceptable or not, one usually uses a penalty-based merit function. In our early tests, the update of the penalty parameters proved to be difficult with our types of problems. We now use a filter instead.

Our algorithm is an adaptation of Fletcher's filter SQP [14] to the case of manifolds: we use an adaptive trust-region that is intersected with the validity region of φ_{x_i}, and a new iterate $x_{i+1} = \varphi_{x_i}(\mathbf{z})$ is accepted if either the cost function or the sum of constraint violations is made better than for any previous iterates.

Aside from the manifold adaptation, our main departure from Fletcher is in the Hessian computation where we used an approximation, since the exact one is too expensive to compute in our problems. After testing several possibilities, we settled for a self-scaling damped BFGS update [15], [12], adapted to the manifold framework. More precisely, given the Hessian approximation H_k at iteration k, we compute the approximation H_{k+1} as follows

$$\begin{aligned} s_k &= \mathcal{T}_z(z), \quad y_k = \nabla_z \mathcal{L}_{x_{k+1}}(0, \lambda_{k+1}) - \mathcal{T}_z(\mathcal{L}_{x_k}(0, \lambda_k)) \\ \theta_k &= \begin{cases} 1 & \text{if } s_k^T y_k \geq 0.2 s_k^T \tilde{H}_k s_k \\ \frac{0.8 s_k^T \tilde{H}_k s_k}{s_k^T \tilde{H}_k s_k - s_k^T y_k} & \text{otherwise} \end{cases} \\ r_k &= \theta_k y_k + (1 - \theta_k)\, \tilde{H}_k s_k \quad \text{(damped update)} \\ \tau_k &= \min\left(1, \frac{s_k^T r_k}{s_k^T \tilde{H}_k s_k}\right) \quad \text{(self-scaling)} \\ H_{k+1} &= \tau_k \left(\tilde{H}_k - \frac{\tilde{H}_k s_k s_k^T \tilde{H}_k}{s_k^T \tilde{H}_k s_k}\right) + \frac{r_k r_k^T}{s_k^T r_k} \end{aligned}$$

where $\mathcal{T}_\mathbf{z}$ is a vector transport along $\mathbf{z}$ (see [10]) and $\tilde{H}_k$ is such that for $\mathbf{u} \in T_{x_{k+1}}\mathcal{M}$, $\tilde{H}_k \mathbf{u} = \mathcal{T}_\mathbf{z}\left(H_k \mathcal{T}_\mathbf{z}^{-1}(\mathbf{u})\right)$.

Despite Powell's update, H_k might not be positive definite (but still symmetric). We regularize it as follows: we first perform a Bunch-Kaufman factorization $P_k H_k P_k^T = L_k B_k L_k^T$ where P_k is a permutation matrix, L_k is unit lower triangular and B_k is block diagonal with blocks of size 1×1 or 2×2 (obtaining B_k as a diagonal matrix is not numerically stable for Cholesky-like decomposition of indefinite matrices), see [16]. The eigenvalue decomposition $B_k = Q_k D_k Q_k^T$ is immediate and cheap to compute. From the diagonal matrix D_k we form D'_k such that $d'_{ii} = \max(d_{ii}, \mu_{\min})$ where $\mu_{\min} > 0$ is user-defined (we typically set it to 0.1). Defining $L'_k = L_k Q_k (D'_k)^{1/2}$, we get a regularized matrix $H'_k = P_k^T L_k L_k^T P_k$. In our case, we use `LSSOL` [17] for solving the QP (4), which directly accepts the factorized form (P_k, L'_k). This avoids an internal Cholesky factorization so that our regularization does not add too much time to the overall process of building and solving the QP.

The code for $\psi \circ \varphi$, its gradient and the vector transport needs only to be implemented once for each elementary manifold (it is then trivial to get those functions for Cartesian products of manifolds). The composition with f' and c' is done automatically. The expression of those functions is adapted from [18].

III. Posture Generation, variables and architecture

Writing a posture generation problem can easily become cumbersome without the appropriate tools. Common pitfalls are for example writing the derivative of a function, managing how the Jacobian matrices of the already implemented functions are modified when a variable is added to the problem, adding a new type of constraint, or correctly writing a function on a sub-manifold of the problem manifold. A fair amount of bookkeeping is always necessary, which should not be the charge of the user writing the constraints. In our PG, we propose an architecture automating most of the problematic tasks, so that the user can focus on the mathematical formulation of the problem.

A. Geometric expressions

Most constraints are geometric. In order to simplify the writing of functions, we use a dedicated system of expression

graph encapsulated in a set of geometric objects. The main idea is to separate the purely mathematical logic from the geometric one. As an example if P_r and V_r are a point and a vector attached to the camera of the robot, and P_e is a fixed point in the environment, the constraint $(P_e - P_r).V_r = 0$ can be used to have the robot look at P_e. With our system, the user creates only those objects and write the code `(Pe-Pr).dot(Vr)` to get the value needed. The geometric layer takes care that all the quantities are expressed in the correct frame, the mathematical layer performs the corresponding operations. If q is a variable object, `(Pe-Pr).dot(Vr).diff(q)` returns automatically the differential of the expression w.r.t. q. This makes the writing of the constraints very easy.

At the mathematical level, we consider 5 types of expressions which can be either variables or constants:

- Scalar, a 1-dimensional element of $\mathbb{R}$
- Coordinates, a 3-dimensional element of $\mathbb{R}^3$
- Rotation, a 3×3 matrix representing a 3D rotation
- Transformation, a 4×4 matrix representation of a 3D isometry
- Array, a dynamic size array

The meaningful unary (inverse, opposite, norm...) and binary (multiplication, addition, subtraction, dot product...) operations (with their derivatives by chain rule) are implemented. We also have a Function class for more complicated expressions, for example expressing $q \mapsto T_i(q)$ where T_i is the transformation between the reference frame of the robot and the frame of its i-th body [1]. The combinations of those elementary operations defines a computation graph.

The geometric layer consists of physical or geometric objects, named features, which exist independently of their mathematical expression in a given reference frame. We have so far 4 objects:

- A Frame, defined by a Transformation expression and a reference frame.
- A Point and a Vector, defined by a Coordinates expression and a reference frame.
- A Wrench, defined by a pair of Coordinates expressions and a reference frame.

We have a special World Frame object to serve as starting reference frame.

For each feature, one can get its expression in a given frame. Basic operations are defined between those features (when applicable). For example, the subtraction between two Points gives a Vector. The geometric logic resides in the change of frame and those operations.

B. Automatic mapping

The manifold $\mathcal{M}$, on which the optimization takes place, is a Cartesian product of several sub-manifolds. Same goes for their representation spaces:

$$\begin{aligned} \mathcal{M} &= \mathcal{M}_1 \times \mathcal{M}_2 \times \mathcal{M}_3 \times \dots \\ \mathbb{E} &= \mathbb{E}_1 \times \mathbb{E}_2 \times \mathbb{E}_3 \times \dots \end{aligned} \tag{6}$$

[1]The kinematics of rigid body systems is handled by the RBDyn library (`https://github.com/jorisv/RBDyn`)

From the solver's viewpoint, the entry space of each function is the complete manifold. But for the developer, writing a function on the complete $\mathbb{E}$ is cumbersome because (i) of the need to manage indexes, and (ii) when the function is implemented, the complete $\mathbb{E}$ may not be known. A user-written function f is usually defined on a subset of $\mathbb{E}$, say $\mathbb{E}_I = \mathbb{E}_i \times \mathbb{E}_j \times \mathbb{E}_k \dots$, that is minimalist for that function, and should not account for unrelated manifolds. One does not want to think about the values of the forces when writing a geometric constraint for example. Our automatic mapping tool generates the correct projection functions π_I such that the developer can write a function f on $\mathbb{E}_I$ while the solver receives it as a function $f \circ \pi_I$ on $\mathbb{E}$. This idea is illustrated by the example in Fig. 3

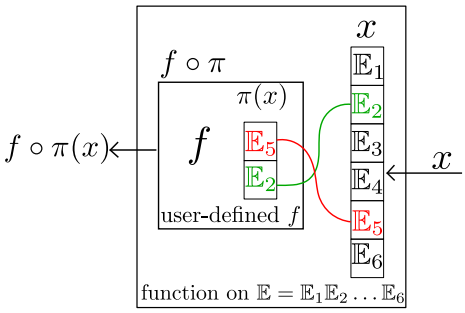

Fig. 3: automatic variable mapping

C. Problem Generator

The problem generator is the tool constructing the optimization problem. It registers all the variables and the functions related to a given problem. Each function is likely to bring additional variables with it. For each contact contributing to the balance, a variable on $\mathbb{R}^3$ representing the contact force is added to the problem. The associated wrench is added to the stability constraints. Once the registration is complete, the complete manifold of the problem is generated and uses the information of the Automatic mapping to "plug" each function with the correct sub-manifold. Subsequently, the optimization problem can be generated and passed to the solver. The communication between the solver and the generated problem is made through the RobOptim framework[2].

IV. Problem formulations

Let $q = [q_F^T; q_r^T] \in \mathbb{R}^3 \times SO(3) \times \mathcal{M}_r$ be the combination of the free-flyer of the robot $q_F \in \mathbb{R}^3 \times SO(3)$ and the articular parameters $q_r \in \mathcal{M}_r$. Let $\mathcal{W}_i(p) = \{f_i, m_i(p)\}$ be the wrench (force+moment) applied by the environment onto the robot at contact i and expressed on point p. A frame F is composed of a reference point and an orthonormal basis of 3 vectors $F = \{O, (x, y, z)\}$.

Here is a list of constraints that we consider in our problem (implementation of other ones is on-going):

- Joint limits $q^- \leq q_r \leq q^+$:
  These cannot be directly translated on manifolds other than $\mathbb{R}^n$. For example, spherical joints can be

[2]`http://www.roboptim.net/`

parametrized on $S2 \times \mathbb{R}$, then the $S2$ part can be limited by a cone, and the $\mathbb{R}$ part can have real bounds.

- The contact constraint consists in identifying the features of two frames F_1 and F_2. For example, for a planar contact, we get the set of equation 7.

$$\begin{aligned}(O_2 - O_1).z_1 &= 0 \\ z_2.z_1 &\leq 0 \\ z_2.x_1 &= 0 \\ z_2.y_1 &= 0\end{aligned} \tag{7}$$

  Note that on F_2 only the point O_2 and the vector z_2 are necessary. Other types of contacts can be created that way, by equalizing other features, as explained in [1].
- The stability constraint ensures that the Euler-Newton equation (8) is balanced for the set of external wrenches applied to the robot (gravity $\mathcal{W}_G$ and contact forces $\mathcal{W}_i$).

$$\sum_i \mathcal{W}_i(p) + \mathcal{W}_G(p) = 0 \tag{8}$$

  For each contact that bears forces ("stability" contact), a wrench applied on the robot at the contact point is added to the problem. That wrench is parametrized on a subset of $\mathbb{R}^6$ depending on the type of contact. For punctual contacts, the moment part is null on the application point. Only a parametrization of the force part on $\mathbb{R}^3$ is needed. We model planar contacts as a combination of punctual forces applied at each vertex of the contact polygon. In the case of interaction forces between 2 robots, only one wrench is created and it is used as is in the stability equation of one robot and its opposite is used for the stability of the second robot.
- The friction cone constraint limits the tangential part of every forces to avoid slippage. We write it as 9 (with μ the friction coefficient)

$$\begin{aligned}\mu^2 f_z^2 - f_x^2 - f_y^2 &\geq 0 \\ f_z &\geq 0\end{aligned} \tag{9}$$

The frame in which the constraints are written matters critically. Most often, the frame's configuration depends on a part of the optimization variables, that must be accounted for in computing the constraints' Jacobian. Our framework computes such dependencies automatically.

Our current PG (i.e. coding state) does not include yet collisions and auto-collisions, nor torque limits. Their implementation is on-going and is simply the matter of coding time. Another important part is its cost function. We only mention the cost function that have specificities when dealing with manifolds, the distance to a reference posture q_0. On a robot that has all its articulations parametrized on $\mathbb{R}$ the distance can be expressed simply with the Euclidean norm $d = \|q_j - q_0\|^2$. Since we work on non-Euclidean manifolds, the logarithm function on the manifold must be used. It gives the distance vector between two points in the tangent space, the norm of this vector can be used as a distance. So we get $d = \left\|\log_{q_0}(q_r)\right\|^2$.

V. Simulation Results

Here, we present several posture generation problems resolution that leverage the specific capabilities of our software.

A. Application to plan-sphere contact

When we consider a planar contact, having a frame F_S fixed in reference to F_B is sufficient because the equations describing that contact are invariant w.r.t. the point's location. But for different contact topologies, the location of the contact point in the body's frame F_B matters. We propose to parametrize the location and normal of the contact point with an additional variable.

We consider the contact between a body's flat surface S_B of normal n_B, with the surface of a sphere S_s of center c_s, radius r_s, and let p_s and n_s be a point and its normal to S_S. The most general way to express such a constraint is to ensure that p_s is on S_B and that n_s and n_B are opposite. This means creating a variable v_{S2} on the manifold $S2$ and map p_S and n_S on it. In our framework, this constraint is expressed exactly as the contact between 2 planar surfaces, once the mappings of $p_s(v_{S2})$ and $n_s(v_{S2})$ are done. In a framework that does not handle manifolds (as we do), it would require to setup a specific constraint, ensuring that the distance between c_s and S_B is equal to r_S.

In Fig. 4 we show the results obtained by solving a problem where the HRP2-Kai robot has to keep its feet in contact with the ground at fixed positions, touch a sphere with a side of its right wrist and point as far as possible in a given direction d with its left hand, under balance constraints. The top row of Fig. 4 shows the results for this problem with several different d. In every situation, the projection of the CoM is outside the polygon of support, meaning that such postures would not be reached without leaning on the sphere.

B. Contact with parametrized wrist

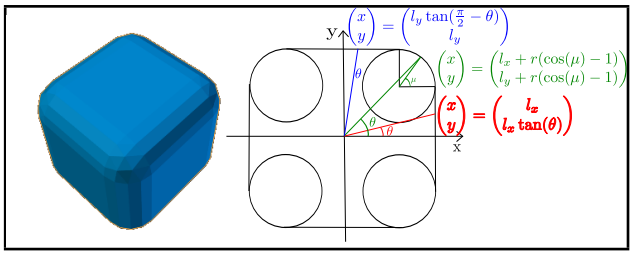

Fig. 5: Parametrization of the wrist of HRP2-Kai

Being able to choose the location of the contact point on the sphere is interesting, but a limitation of this formulation is that the contact point on the wrist of the robot is restricted to one single user-defined face. Instead, we describe the shape of the wrist body as a parametric function and let the contact point on the wrist as well as its counterpart on the sphere, result from the optimization process. The section of HRP2-Kai's wrist is a square with rounded edges. We parametrize this shape as shown in Fig. 5: we consider the angular coordinate θ of the point on the section. It is added as a variable to the problem. The shape of a quarter of section $[0; \pi/2]$ is a succession of a vertical line, a quarter

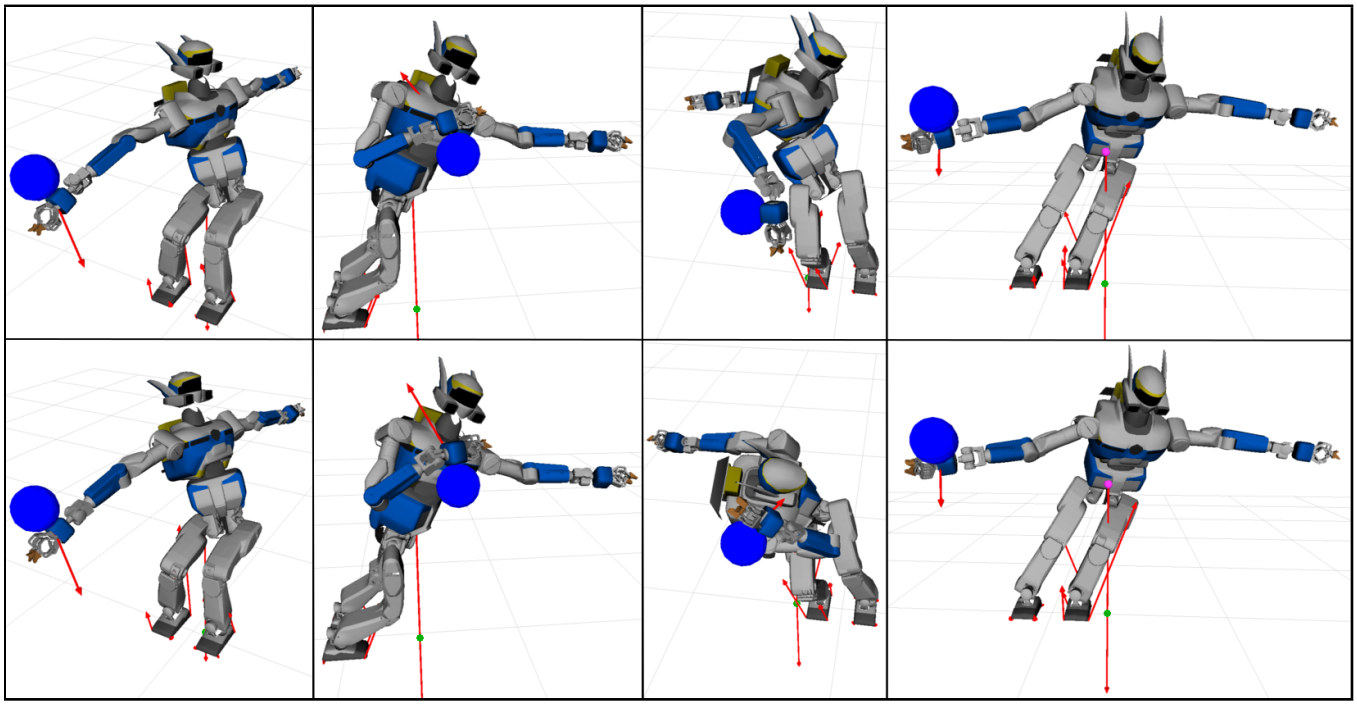

Fig. 4: HRP2-Kai leaning on sphere with right wrist to point the left gripper as far as possible in 4 cardinal directions. Top row: semi-predefined contact; Bottom row: free contact with parametrized wrist. Projection of the CoM on the ground (green dots)

of circle and a horizontal line. This pattern is repeated for the 3 other quarters. The equations are given in Fig. 5. In our framework, we define the function describing the shape of the wrist, create a frame parametrized by that function and then define the contact between that frame and the point and normal on the sphere. This formulation not only is very easy to implement, but most importantly, allows for richer posture generations. The optimization algorithm chooses the contact point on the sphere as well as the contact point on the wrist, which leads to a wider accessibility range, and a better satisfaction of the cost function. The bottom row of Fig. 4 displays the results of this simulation for the robot pointing in 4 directions. Notice that on the 2nd and the 4th (pointing forward and to the left) images, the results for the 2 types of models are nearly identical. Whereas in the 1st and 3rd images, different faces of the wrist have been chosen(On the 1st, the wrist is rotated by $180°$, and $90°$) on the 3rd). In these 4 cases, the contact with parametrized wrist gives a better cost of the objective function. This observation scales: we solved this problem for 5000 random pointing directions, and in average, the contact with parametrized wrist allows to reach 5mm further. The success rate of the solver is 98.5% in the parametrized wrist case against 99.9% when the face is fixed. The numbers of iterations are similar.

This method is certainly scalable, and can be used for any kind of humanoid robot and environment. Yet, it requires to have a parametric equation of the surface. We plan to implement a method to generate a parametrized surface point and its normal directly from the 3D mesh of an object. The accompanying video shows the optimization process for the problem with parametrized wrist. Notice on the video that the contact point on the wrist changes sides all along the iterations.

C. Contact with an object parametrized on $S2$

In this simulation case, we want the HRP-4 robot (anther model) to carry a cube with its two hands. The most general way to do it is to select a face of the cube for each contact, and enforce the contact between that face and the hand's surface. We propose to approximate the cube with a superellipsoid and to parametrize the resulting shape on $S2$. The implicit equation of a superellipsoid is $S(x, y, z) = 0$, with

$$S(x,y,z) = \left(\left|\frac{x}{A}\right|^r + \left|\frac{y}{B}\right|^r\right)^{\frac{t}{r}} + \left|\frac{z}{C}\right|^t - 1 \qquad (10)$$

A point in $S2$ is represented by a vector $v = (x, y, z)$ in $\mathbb{E} = \mathbb{R}^3$. To a given unit vector v we associate a point αv on the surface of the superellipsoid by solving $S(\alpha v) = 0$ for α. At this point, the normal is given by $\frac{\nabla S(\alpha v)}{\|\nabla S(\alpha v)\|}$ which simplifies into $\frac{\nabla S(v)}{\|\nabla S(v)\|}$. Given this parametrization, we write a contact constraint between the frame of the hand of the robot and the point and normal on the surface of the superellipsoid.

In Fig. 1 we present some results for a posture generation problem with manipulation: On the left side, the feet are free to move on a sphere, and, on the right side, the left foot position is fixed and the right foot is free to move on the ground. The hands must be in contact with the cube. The cube is free to move (parametrized by $\mathbb{R}^3 \times SO(3)$) and has it own set of Euler-Newton equations, which must be fulfilled. On the accompanying video, one can observe how the contact points on the cube evolve along with the optimization.

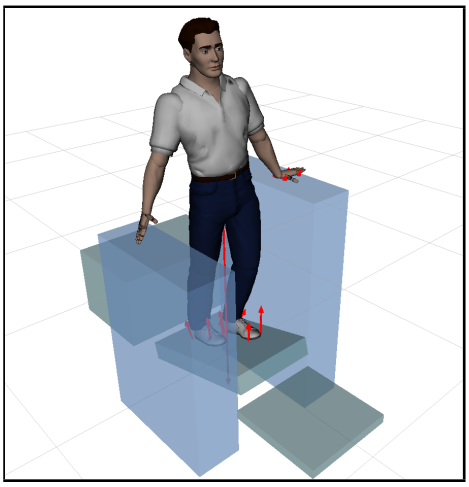

Fig. 6: Posture generation for a human avatar

D. Posture Generation with a human model

The geometric model of a human is much more complex than the one of a humanoid robot in terms of topology. Even with the simplest models, the shoulders, wrists, ankles, or hips need to be described as spherical joints, and therefore be parameterized on $SO(3)$. We showcase that our solver is able to handle such complex models as a human avatar in Fig. 6. Our human model has spherical joints on its wrists, shoulders, torso, hips and ankles. With the addition to the free-flyer, the manifold that contains the articular space of that human model is: $\mathcal{M}_H = SO(3) \times \mathbb{R}^3 \times (SO(3))^9 \times \mathbb{R}^4$. We fixed the neck's joints as well as the fingers to avoid unnecessary variables. In this simulation, we require the human to stand on an inclined slope while leaning on the left side's wall. Since we have not yet implemented the boundary limits on spherical joints, the model could reach non desired configurations. That issue limits for the time being the number of scenarios that we can solve. But the use of a cost function on the posture of type $d = \left\| \log_{q_0}(q_j) \right\|^2$, attracting the avatar to a basic standing posture, allowed us to have acceptable results.

VI. Discussion and Conclusion

Writing the posture generation problem as a non-linear optimization problem on non-Euclidean manifolds proves to be an elegant approach in terms of code structuring and user interface, in addition to mathematical readability. This work is still on-going and only preliminary results of the current state of the implementation are shown. We illustrate some posture generation problems with the HRP2-Kai and the HRP-4 humanoid robots; the results are very promising indeed. Our future (on-going) work is focused on the following streams:

- complement other functionalities as ready-to-use templates (e.g. constraints on task forces, collision avoidance, etc.);
- specialize the solver to humanoid PG problems and benchmark various numerical approaches (e.g. for the choice of the Hessian approximation, the trust region, tuning some parameters, etc.) and exploiting, if any, robotic model properties;
- improve convergence, numerical robustness and computation time of the PG. Currently, the resolution of any of the presented problems takes a few seconds on a laptop with Intel Core i7-3840QM CPU @ 2.80GHz. We aim at reducing it to a tenth of a second.

Once the code is more stable and finalized, we plan to make it open-source.

References

[1] A. Escande, A. Kheddar, and S. Miossec, "Planning contact points for humanoid robots," *Robotics and Autonomous Systems*, vol. 61, no. 5, pp. 428–442, 2013.

[2] C. Lawrence, J. L. Zhou, and A. L. Tits, "User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints," 1997.

[3] K. Bouyarmane and A. Kheddar, "Humanoid robot locomotion and manipulation step planning," *Advanced Robotics*, vol. 26, no. 10, pp. 1099–1126, 2012.

[4] A. Wächter and L. Biegleri, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.

[5] J. Vaillant, A. Kheddar, H. Audren, F. Keith, S. Brossette, K. Kaneko, M. Morisawa, E. Yoshida, and F. Kanehiro, "Vertical Ladder Climbing by HRP-2 Humanoid Robot," in *IEEE-RAS Int. Conf. Humanoid Robot.*, Madrid, Spain, 2014, pp. 671–676.

[6] K. Hauser, T. Bretl, and J.-C. Latombe, "Non-gaited humanoid locomotion planning," in *IEEE-RAS Int. Conf. Humanoid Robots*, December 5-7 2005, pp. 7–12.

[7] A. Aristidou and J. Lasenby, "Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver," University of Cambridge, Tech. Rep., 2009.

[8] P. Baerlocher and R. Boulic, "Parametrization and range of motion of the ball-and-socket joint," in *Deform. Avatars*, 2001, pp. 180–190.

[9] K. Bouyarmane and A. Kheddar, "On the dynamics modeling of free-floating-base articulated mechanisms and applications to humanoid whole-body dynamics and control," in *IEEE-RAS Int. Conf. Humanoid Robots*, 2012.

[10] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

[11] J. Schulman, Y. Duan, J. Ho, a. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *Int. J. Rob. Res.*, 2014.

[12] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. Springer, 2006.

[13] K. Bouyarmane and A. Kheddar, "On the dynamics modeling of free-floating-base articulated mechanisms and applications to humanoid whole-body dynamics and control," in *IEEE-RAS Int. Conf. Humanoid Robot.*, Osaka, Japan, Nov. 29 - Dec. 1 2012, pp. 36–42.

[14] R. Fletcher and S. Leyffer, "Nonlinear programming without a penalty function," *Mathematical Programming*, vol. 91, pp. 239–269, 2000.

[15] J. Nocedal and Y.-x. Yuan, "Analysis of a self-scaling quasi-newton method," *Mathematical Programming*, vol. 61, no. 1-3, pp. 19–37, 1993.

[16] G. Golub and C. Van Loan, *Matrix computations*, 3rd ed. John Hopkins University Press, 1996.

[17] P. E. E. Gill, S. J. Hammarling, W. Murray, M. A. Saunders, and M. H. Wright, "User's guide for lssol (version 1.0): a fortran package for constrained linear least-squares and convex quadratic programming," Standford University, Standord, California 94305, Tech. Rep. 86-1, January 1986.

[18] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, "Manopt, a matlab toolbox for optimization on manifolds," *Journal of Machine Learning Research*, vol. 15, pp. 1455–1459, 2014.