A Kinematic Whole-body Control System for Highly Redundant Robots

Oscar E. Ramos
Department of Electrical Engineering
Universidad de Ingenieria y Tecnologia - UTEC, Lima, Peru

***Abstract*—Whole-body control is crucial to fully exploit all the possible motions that redundant robots with a free-floating base can achieve. This paper presents a new open source and platform independent system that integrates whole-body operational-space control schemes based upon inverse kinematics for highly redundant robots. Contrary to other works, which heavily rely on complex underlying platforms and are difficult to integrate in different frameworks, the proposed system does not rely on specific middlewares. It is completely platform independent, and its integration in different frameworks is easy and straightforward. The control system has been tested with a redundant humanoid robot.**

I. Introduction

Highly redundant robots such as humanoids, quadrupeds or other multi-limbed robots possess several degrees of freedom which allow them to achieve different objectives at the same time with multiple configurations. This complexity makes their control difficult. If these robots are considered as a fixed torso with limbs, controlling each limb as a classical robotic manipulator, the whole motion is reduced and over-constrained. Therefore, some otherwise possible movements are not achievable, and the whole system becomes underexploited. To overcome this limitation and exploit the redundant robotic system, a whole-body motion control is necessary.

In recent years, several frameworks have been proposed to deal with multi-objective whole-body motion aiming at finding joint angular positions, velocities, accelerations or torques for a given objective. The usual way to specify this objective is through tasks which belong to the so-called *operational space* [1] or *task space* [2]. For instance, the Cartesian space, the visual feature space [3], and even the joint space itself are examples of an operational space. Several teams have developed their own platforms and middlewares to implement kinematic or dynamic control. Some examples of freely available architectures are the Stack of Tasks (SoT) software, which relies on a data-flow system called dynamic graph for all its modules, and the library called OpenSoT, which relies on YARP, a platform for data-communication. Other control systems for whole-body motion exist, but most of them are private and constrained to specific research institutions.

One of the main problems with the currently existing robotic control systems is that they are oriented towards specific robotic platforms and, therefore, they are completely dependent on a middleware, which complicates their integration with a different framework. Therefore, in order to use those systems, or even a small component of them, it is necessary to adopt the whole platform, or to adapt the needed component to the desired platform. The former alternative typically represents the installation of unnecessary (and usually very heavy and complex) software. The latter alternative is time consuming and needs the understanding of the middleware for which they were developed, which is usually not straightforward. Thus, currently a big problem for robotic whole-body motion is system integration.

978-1-5090-2532-9/16/$31.00

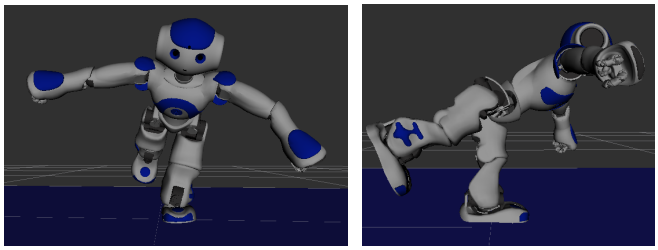

Fig. 1: Motion generated with the proposed system. A task keeps the center of mass above the supporting foot so that the robot does not fall down while moving the arms and legs in single support.

This paper presents a whole-body control system for multi-articulated robots with a free-floating base which is self-contained and does not depend on any specific platform. It constitutes a new control library that implements an Operational Space Inverse Kinematics (OSIK) control which generates whole-body motion using tasks. Due to this reason, the library is called *osik-control*. The system provides both a weighted task scheme based on QPs (Quadratic Programs), and a prioritized task scheme based on nullspace projections and HQPs (Hierarchical Quadratic Programs). An example of a motion generated with the proposed system is shown in Fig. 1.

The main contributions of this work are the software architecture that supports a generic whole-body inverse kinematics control, the independence of third-party middlewares, the open source system implementation, the reduction in software complexity for the usage of the library, and the possibility to use weighted or prioritized schemes within the same framework, for research or educational purposes, which is currently not provided by any other system. Each implementation deals with joint angular and velocity limits, which is fundamental for the feasibility of the motion in a real hardware.

II. Operational Space Inverse Kinematics

This section presents the inverse kinematics control schemes that are integrated in the proposed system. These schemes are called solvers and generate motion based upon the formalism of kinematic tasks which determine the motion objectives.

A. Kinematic Tasks

The configuration of an n degree of freedom robot with a free-floating base is represented as $q = [x_b \ q_a]^T$, where $x_b \in SE(3)$ is the pose of the floating base[1], and $q_a \in \mathbb{R}^n$ is the vector of joint angular values. The motion of a robot, with joint configuration q and n degrees of freedom can be described by specifying a set of tasks, defined by task functions [2]. Let a task i be denoted by e_i, and its analytic Jacobian be $J_i = \frac{\partial e_i}{\partial q}$ such that

$$\dot{e}_i = J_i \dot{q} \tag{1}$$

where $\dot{e}_i$ is the velocity of task i, and $\dot{q}$ is the generalized joint velocity vector. A task e is defined as the difference between the current configuration x and the desired configuration x^*, where both x and x^* belong to the same operational space. That is,

$$e = x \ominus x^* \tag{2}$$

where $\ominus$ is a task difference operator. If $x \in \mathbb{R}^3$ is a position, then $\ominus$ is simply a term by term difference. If $x \in SO(3)$ is an orientation, a usual choice is to set the error as the axis/angle representation of $R(x)R^T(x^*)$, where $R(x)$ is an operator that converts x into a rotation matrix. Several other tasks can be defined to generate different types of motion [4]. The reference for a kinematic task is given by $\dot{e}^* = -\lambda e$, where $\lambda > 0$ represents the kinematic task gain and guarantees an exponential decrease of the error to zero (asymptotic stability). This gain can be a constant or it can be chosen as a function of the error as

$$\lambda = (\lambda_{max} - \lambda_{min}) e^{-k\|x \ominus x^*\|} + \lambda_{min} \tag{3}$$

where $k > 0$ is the rate of descent. Note that this gain increases when the error decreases, achieving a faster convergence.

B. Weighted Inverse Kinematics Solver

Let the control system be composed of t tasks defining the desired motion, and let task i have a weight $w_i > 0$. The approach to find the generalized joint velocity $\dot{q}$ that will satisfy this weighted scheme consists in the following optimization problem

$$\min_{\dot{q}} \left\{ \sum_{i=1}^{t} w_i \|\dot{e}_i - J_i \dot{q}\|^2 \right\}. \tag{4}$$

In this case, tasks with a greater weight will be more closely satisfied, but if there are conflicting tasks, it might happen that none is completely satisfied but there exists a compromise amongst them [5]. For the motion to be feasible in a real robotic platform, joint limits need to be considered. Let $\underline{q}$, $\overline{q}$ be the lower and upper joint angular limits, and $\underline{\dot{q}}$, $\overline{\dot{q}}$ be the lower and upper joint velocity limits. Since $\|y\|^2 = y^T y$, it can be shown that the optimization (4), considering joint limits, can be written as the following quadratic program (QP):

$$\begin{aligned} &\min_{\dot{q}} \left\{ \dot{q}^T W \dot{q} + \dot{q}^T p \right\} \\ &\text{s.t.} \\ &\max\left\{ \frac{\underline{q} - q}{\Delta t}, \underline{\dot{q}} \right\} \leq \dot{q} \leq \min\left\{ \frac{\overline{q} - q}{\Delta t}, \overline{\dot{q}} \right\} \end{aligned} \tag{5}$$

where

$$W = \sum_{i=1}^{t} w_i J_i^T J_i, \quad p = -2 \sum_{i=1}^{t} w_i J_i^T \dot{e}_i,$$

and Δt is the control time between each control signal. The bounds relating the joint angular limits to the joint velocities are obtained solving for $\dot{q}_k$ in the Taylor expansion $q_{k+1} = q_k + \dot{q}_k \Delta t$. Note that the terms containing $\dot{e}_i^T \dot{e}_i$ in the expansion of (4) have been ignored in the minimization (5) since they are independent of $\dot{q}$ and only act as a constant offset without altering the optimal solution. The minimum and maximum in the constraint in (5) are taken element-wise and ensure that both angular and velocity limits will be satisfied.

C. Prioritized Solver with Nullspace Projection

In prioritized schemes each task is associated with a priority $i \in \mathbb{N}^+$, where a lower i indicates a higher priority. For a single task, the solution to the problem $\min_{\dot{q}} \|\dot{e}_1^* - J_1 \dot{q}\|^2$ is

$$\dot{q}^* = J_1^{\#} \dot{e}_1^* + P_1 z_2 \tag{6}$$

where $J^{\#}$ is any reflexive generalized inverse of J_1 (usually the Moore-Penrose pseudoinverse [6]), $P_1 = I - J_1^{\#} J_1$ is a projector onto the nullspace of J_1, and z_2 is an arbitrary vector. It is possible to use z_2 to generate additional motion without interfering with the first task. By doing successive projections onto the nullspace, it can be shown that the solution for n prioritized tasks is

$$\dot{q}^* = \dot{q}^*_{1,2,\cdots,n} + P_n z_{n+1} \tag{7}$$

where

$$\begin{aligned} \dot{q}^*_{1,2,\cdots,n} &= \sum_{k=1}^{n} (J_k P_{k-1})^{\#} (\dot{e}_k^* - J_k \dot{q}^*_{1,2,\cdots,k-1}) \\ P_k &= P_{k-1}(I - (J_k P_{k-1})^{\#}(J_k P_{k-1})) \end{aligned}$$

with initial values $P_0 = I$, $\dot{q}_0^* = 0$. These initial values are only used for the algorithm to be general and have no meaning (although, since $P_0 = I$, the nullspace of the zero task is the whole task space). To deal with inequality tasks, this approach uses potential functions $f_p(q)$ that set $\dot{q} = \alpha \frac{\partial f_p(q)}{\partial q}$, but the disadvantage is that they enforce soft constrains rather than hard constraints. Joint limits can be taken into account by defining the potential function

$$f_p(q) = \frac{1}{2n} \sum_{i=1}^{n} \left(\frac{\overline{q}_i - \underline{q}_i}{(\overline{q}_i - q_i)(q_i - \underline{q}_i)} \right) \tag{8}$$

[1]Typically, $x_b = [x \ y \ z \ \varphi_x \ \varphi_y \ \varphi_z]^T$, where x, y, z are the Cartesian position and $\varphi_x, \varphi_y, \varphi_z$ are Euler angles or an axis/angle representation. A frequent alternative is to represent the orientation with a quaternion.

which gives a minimum value at $\frac{1}{2}(\overline{q}_i+\underline{q}_i)$ and a high-potential when approaching the joint limit. The resulting task is $q = [q_1, \cdots, q_n]^T$ where

$$\dot{q}_i = \frac{\alpha(\overline{q}_i - \underline{q}_i)}{2n}\left(\frac{1}{(\overline{q}_i - q_i)^2(q_i - \underline{q}_i)} - \frac{1}{(\overline{q_i} - q_i)(q_i - \underline{q}_i)^2}\right)$$

The nullspace prioritization scheme in (7) is sometimes called a *forward* priority approach since the solution starts with the highest priority task and moves to the lowest priority. Recently, a *reverse* priority approach was proposed [7] but it is not yet clear the advantages of that methodology and it has not been considered in this work.

D. Prioritized Solver with QPs

A hierarchical order of tasks can also be solved using a cascade of QPs [8]. In this approach, the optimization function at the k^{th} priority level is written as

$$\begin{aligned} &\min_{\dot{q}, w_k} \|w_k\|^2 \\ &\text{s.t.} \\ &\overline{\dot{e}}^s_{k-1} \leq J^s_{k-1}\dot{q} + w^{s*}_{k-1} \leq \underline{\dot{e}}^s_{k-1} \\ &\overline{\dot{e}}_k \leq J_k\dot{q} + w_k \leq \underline{\dot{e}}_k \end{aligned} \tag{9}$$

where $\underline{\dot{e}}_k$, $\overline{\dot{e}}_k$ are the lower and upper task bounds for the k^{th} task, and w_k is a slack variable at the k^{th} level. The stacked bounds of the previous tasks $\overline{\dot{e}}^s_{k-1} = [\overline{\dot{e}}^T_{k-1}, \cdots, \overline{\dot{e}}^T_1]^T$, $\underline{\dot{e}}^s_{k-1} = [\underline{\dot{e}}^T_{k-1}, \cdots, \underline{\dot{e}}^T_1]^T$, the stacked Jacobian $J^s_{k-1} = [J^T_{k-1}, \cdots, J^T_1]^T$, and the stacked optimized slack variables $w^{s*}_{k-1} = [w^{*T}_{k-1}, \cdots, w^{*T}_1]^T$ are considered in (9) in order to account for priority. More specifically, w^*_i represents the solution found in the previous optimization steps. The optimization (9) is solved first for task 1 obtaining w^*_1. Then, it is solved for task 2 using w^*_1 as a hard constraint for task 1. By including w^*_1, the solution for task 2 will also be optimal with respect to task 1 achieving a priority among tasks.

One advantage of the prioritized scheme using HQP is that it allows to naturally include inequality tasks. Examples of these tasks are joint limits, visual fields, obstacle avoidance, collision (and self-collision) avoidance among others. When both bounds are the same ($\overline{\dot{e}} = \underline{\dot{e}}$), the task becomes an equality task and the problem is equivalent to a nullspace projection. Therefore, this approach can be considered as a generalization of nullspace projections. Note that an equivalent using inverse dynamics exists but it is computationally more expensive.

III. System Integration

The operational space inverse kinematics schemes described in Section II are integrated in this work in a unified library that is open source and freely available[2]. This section describes the integration with more detail.

[2]The library called *osik-control* is freely available as a git repository at https://github.com/utec-robotics/osik-control.git

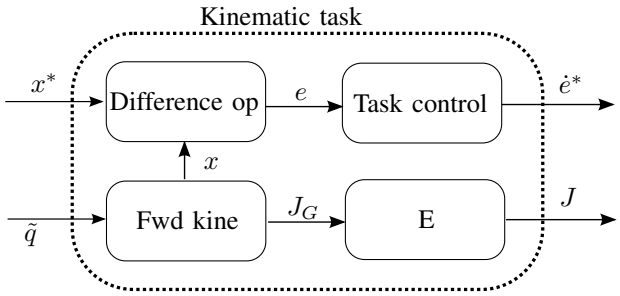

Fig. 2: Scheme showing the architecture of a generic kinematic task.

A. Architecture

The proposed system is composed of a robot model, tasks and different kinematic solvers. The full robot model is parsed and it provides basic kinematic information such as positions and orientations of the rigid bodies composing the robot. It also provides the joint angular and velocity limits, which are fundamental for a kinematically feasible motion.

Tasks are the basic building blocks for motion since they specify the objectives in the Cartesian space, Joint space or any other task space. Fig 2 shows the architecture of a generic kinematic task. The inputs are the desired task configuration x^* and the current joint configuration $\tilde{q}$ as measured by the robot. Using forward kinematics the current task configuration x is obtained from $\tilde{q}$ and is fed to the block that makes the difference between the current and the desired task values (section II-A) obtaining the task error e. Then, the task control signal $\dot{e}^*$ is obtained as a function of this error. The forward kinematics block also provides the geometric Jacobian J_G such that $[v^T \ \omega^T]^T = J_G\dot{q}$, where v and ω are the linear and angular velocity of an operational point (e.g. the hand, the ankle, etc.). This Jacobian needs to be multiplied with a matrix E, which is dependent upon the orientation representation, to become the task Jacobian J, also called the analytic Jacobian.

To obtain the joint position control q, which will drive a position controlled robot, the operational space inverse kinematics solver (osik) is used as shown in Fig. 3. The solver obtains the task Jacobian J and task control error $\dot{e}^*$ from each task and computes the joint velocity command $\dot{q}^*$ using any of the methods described in Section II. Then, this velocity can be integrated (e.g. using Euler integration) to get the joint position command q^*, for a position controlled robot, or it can be directly used if the robot is velocity controlled. The sensed joint configuration $\tilde{q}$, obtained from a real or simulated robot, is then used to update the tasks and therefore the inverse kinematics command.

B. Platform Independence

The proposed system does not depend on any specific platform and can be integrated into any complex robotic architecture. It depends only on independent state-of-the art libraries used for the basic kinematics and for the solution of optimization problems. The robot model is currently specified in URDF (unified robotic description framework)

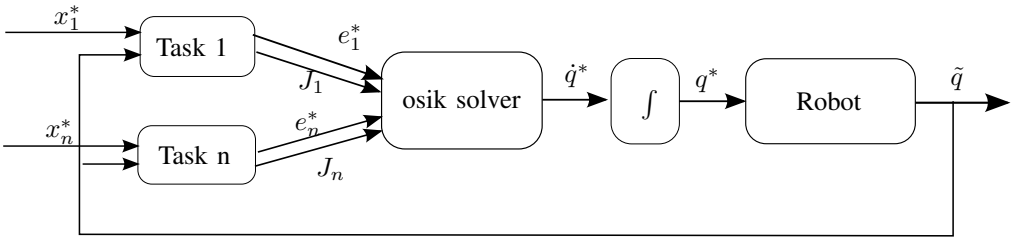

Fig. 3: Scheme for the operational space inverse kinematics (osik) solver.

format, which is an xml file containing the model description, together with its kinematic and dynamic parameters as well as rendering information. It is possible to extend the support to other formats, but URDFs have become the *de-facto* standard in robotics research. For computing the basic kinematics and dynamics of rigid bodies, it depends on RBDL (Rigid Body Dynamics Library), but it extends this library to be able to consider a free-floating base, which is crucial for multi-articulated robot that are not fixed in their environment. The current architecture also allows to develop plugins for other robot dynamic libraries, in case they are needed for a specific purpose. The QPs needed for the weighted scheme and for the HQP scheme are solved using qpOASES [9], which uses active sets to efficiently find the solution to QPs.

IV. Results

The proposed system was tested in a simulation that uses the full kinematic and dynamic model of NAO, a small humanoid robot with 26 degrees of freedom. The framework used to test the proposed system was ROS (Robot Operating System). It is important to point out that this choice is completely arbitrary: any other framework could have been used since the library itself is completely independent of the platform and URDFs can be parsed without using ROS. Fig 4 shows an example of whole-body motion for NAO. The highest priority tasks are assigned to both feet so as to keep them still on the ground. This is accomplished with tasks that control the full pose of an operational point, in this case, the robot ankles. The next priority level is for the center of mass to be always above the support polygon, which is achieved with an inequality task. Then, a task is assigned to the waist of the robot, for making it move to the left (from the robot's point of view). Joint limits for angular positions and velocities are also included. Note that there is no specific joint for the waist and the robot needs to coordinate the motion of the legs to achieve this task.

The proposed system was also tested with the real NAO robot. Some of these results are shown in [10], where the robot is intended to imitate in real time part of the motion of a human performer. In [10], joint limits are taken into account as constraints for the optimization problem. The gains for the tasks were set to high values to closely follow the trajectory.

V. Conclusion

This work introduced a novel whole-body control library (osik-control) that uses operational space inverse kinematics and is both platform and middleware independent, allowing an easy integration with more complex architectures. The proposed system effectively and easily generates complex whole-body behaviors based on tasks that can be expressed as equalities or inequalities, providing simple and generic interfaces. The purpose of this work has been to develop an open source library for research on whole-body motion. Future work will deal with whole-body inverse dynamics control as well as trajectory optimization, considering middleware and platform independence.

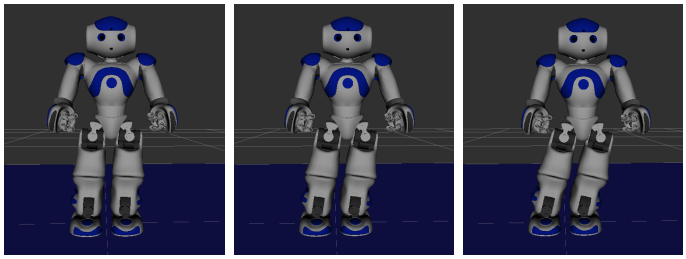

Fig. 4: Snapshots of a simulated humanoid robot NAO moving its waist laterally through the control of its whole-body.

References

[1] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal of Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[2] C. Samson, B. Espiau, and M. L. Borgne, *Robot control: the task function approach*. Oxford University Press, 1991.

[3] B. Espiau, F. Chaumette, and P. Rives, "A new approach to visual servoing in robotics," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 313–326, 1992.

[4] O. Ramos, "Generation of whole-body motion for humanoid robots with the complete dynamics," Ph.D. dissertation, Université de Toulouse, LAAS-CNRS, Toulouse, France, 2014.

[5] 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.

[6] K. L. Doty, C. Melchiorri, and C. Bonivento, "A Theory of Generalized Inverses Applied to Robotics," *The International Journal of Robotics Research*, vol. 12, pp. 1–19, 1993.

[7] F. Flacco and A. De Luca, "A reverse priority approach to multi-task control of redundant robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2014, pp. 2421–2427.

[8] O. Kanoun, F. Lamiraux, P. Wieber, F. Kanehiro, E. Yoshida, and J. Laumond, "Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[9] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[10] J. Avalos, S. Cortez, K. Vasquez, V. Murray, and O. Ramos, "Telepresence using the kinect sensor and the nao robot," in *2016 IEEE 7th Latin American Symp. on Circuits & Systems*, 2016, pp. 303–306.