An Open Source Extensible Software Package to Create Whole-Body Compliant Skills in Personal Mobile Manipulators

Roland Philippsen*, Luis Sentis†, and Oussama Khatib*
*Stanford University, †University of Texas at Austin

***Abstract*— Whole-body operational space control is a powerful compliant control approach for robots that physically interact with their environment. The underlying mathematical and algorithmic principles have been laid in a large body of published work, and novel research keeps advancing its formulation and variations. However the lack of a reusable and robust shared implementation has hindered its widespread adoption.**

To fill this gap, we present an open-source implementation of whole-body operational space control that provides runtime configurability, ease of reuse and extension, and independence from specific middlewares or operating systems. Our libraries are highly portable, and the application code contains a thin adaptation layer for specific development and runtime environments such as ROS.

In this paper, we briefly survey the foundations of whole-body control for mobile manipulation and describe the structure of our software. We performed experiments on two quite different robots to demonstrate that the software is mature enough for building a community of users and developers who can work on extensions and applications.

I. Introduction

The foundations for controlling robots that physically interact with objects and persons in environments made for humans have been investigated for approximately three decades. In addition to recognizing and developing the required mechatronic capabilities, such as backdriveable actuation and effective control of output torques, this large body of work provides mathematical and algorithmic foundations for creating robotic skills similar to those of humans in terms of movement coordination, stiffness regulation, and contact with the environment. However, despite the availability of mechatronically advanced manipulators such as the KUKA LWR, Barrett WAM, and Meka arm, this knowledge is only slowly making its way into applications. These machines indicate a definite trend toward robots better suited for physical interaction with everyday environments, and we believe that the remaining hurdle in producing agile interactive robot applications now lies in the complexity of the required models and the difficulty of producing reusable extensible software frameworks that provide advanced control methodologies.

To address this hurdle, we launched the `stanford-wbc` open-source project in 2009, with the aim of making the whole-body operational-space formulation developed at the Stanford Robotics and AI Lab [17], [12] readily available for widespread reuse. In this paper, we present the first fruits of our effort to design, implement and test a software package for compliant whole-body control, complete with resources to create complex mobile manipulation behaviors by composition and extension. Having worked with open-source robotics software and frameworks such as ROS, Player/Stage, XCF, GenoM, and Orocos, we understand the need for robust, flexible, and documented foundations that can be re-composed in application-specific ways, and our objectives explicitly include these non-technical challenges.

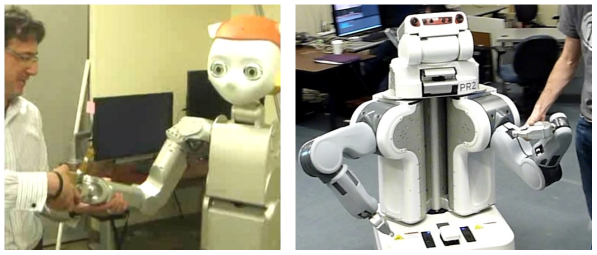


Fig. 1. To date, the `stanford-wbc` software has been integrated on UT Austin's Dreamer and Willow Garage's PR2, which are quite distinct in their mechatronics as well as their development and runtime system environments.

The project homepage is `http://stanford-wbc.sourceforge.net` which contains announcements, documentation, mailing list archives, and links to download source code releases and clone the `git` repository. The majority of the source code is released under the LGPLv3 license [3], with some third-party code under other licenses.

To date, we have integrated the project on two robots (Fig. 1): within the ROS-based control architecture on PR2 at Willow Garage and Stanford; and on Dreamer at University of Texas at Austin within the RTAI shared-memory interface provided by Meka Robotics. The approach underlying our software has been applied to other robots, such as PUMA and ASIMO, and back-porting `stanford-wbc` to these systems would be easy.

II. Related Work

There is a large body of work that focuses on models and control structures. Torque control strategies include the work of Khatib [11] and the German aerospace center, DLR [6], [14], the latter focusing on impedance control of semi-flexible robot arms. At the whole-body level, torque control has been applied by Mizuuchi et al. for compliant tendon contraction [13], by Whitman and Atkeson [20] to coordinate multiple optimal controllers for achieving complex behaviors, by Hyon [8] for compliant multi-contact, and ourselves [16] for the execution of whole-body skills. Position control strategies still dominate the field; at the whole-body level, they include work at AIST (e.g. momentum-based models by Kajita et al. [9]), on the Honda Asimo (e.g. inverse kinematic position control coupled with an inverted pendulum model by Hirai et al. [5]), and on Reem-B (inverse kinematics approach by Tellez et al. [19]).

The research underlying `stanford-wbc` focuses on models of whole-body compliant skills and on hierarchical control structures that resolve task conflicts during execution [16], [18]. These rely on contact interactions for manipulators, especially results on dynamics and force control by Khatib [10] and Raibert and Craig [15], as well as our work on mobile and multi-grasp manipulation [7], [21], [1].

III. Whole-Body Control Foundations

This section summarizes algorithmic foundations and introduces the main concepts of: *(i)* operational space tasks, *(ii)* their adaptive hierarchical arrangement into sensorimotor whole-body skills, and *(iii)* the prioritized control structure which dynamically decouples tasks. Algorithm 1 in Sec. IV-B provides a dense summary in pseudo-code with simplified notation.

A **task** is defined via a mapping between the robot's N-dimensional joint configuration and some M-dimensional space which describes an objective that the controller should achieve. For any joint configuration, the robot is considered a point in the M-dimensional **operational space**, and the task controls the motion of this point. At this stage, a task is a kinematic entity with an associated tangent space that can be represented by a Jacobian, which maps velocities and forces between joint and operational spaces.

For the (usual) case of $N > M$, a task-point defines a manifold in joint-space. This manifold represents the redundancy of the robot with respect to the task. We can explore tangents to the current configuration to determine joint-space motions which keep the task-point constant. This is referred to as the **null space** of the task, and it provides the basis for task decoupling via projection.

In order to facilitate the management of tasks and hierarchies, we introduce **sensorimotor skills** to translate between higher-level goals (such as provided by planning algorithms) and the operational-space tasks. A skill is responsible for configuring the tasks such that an overall motion objective is achieved. Skills can be considered containers and managers of tasks, and the active skill provides the current task hierarchy to the control structure.

The intuition behind the hierarchical **control structure** is is to instantiate several tasks, each of which tries to drive the robot toward some state. The task contributions are accumulated using null space projections to ensure that lower-priority tasks do not interfere with higher levels. The motion is thus determined by each task *in combination with* their priorities. This structuring provides two orthogonal ways of changing robot behavior, either by influencing the tasks (e.g. changing their gains or goals) or by rearranging the hierarchy (e.g. inserting tasks or locally inverting their ordering).

The model of the robot under prioritized control was developed in [16] which provides the generalized Jacobian

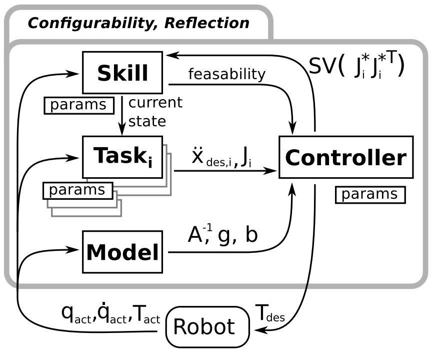


Fig. 2. Architectural overview of the Whole-Body Control software. SV stands for *singular values*, T_{des} is the desired joint-torque vector, and $\{q_{\text{act}}, \dot{q}_{\text{act}}, T_{\text{act}}\}$ are the actual joint positions, velocities, and torques. The rest of the notation is the same as in Algorithm 1.

of an operational task for a given priority, written

$$J^*_{k|\mathcal{P}_k} \triangleq J_{\text{task}_k} \; N_{\mathcal{P}_k}, \tag{1}$$

where J_{task_k} is the Jacobian of the k-th level task manifold with respect to an inertial frame,

$$N_{\mathcal{P}_k} \triangleq I - \sum_{i=1}^{k-1} \overline{J}^*_{i|\mathcal{P}_i} \; J^*_{i|\mathcal{P}_i} \tag{2}$$

is the recursive dynamically-consistent null space describing the task's k-th priority state, $k|\mathcal{P}_k$ is used to express that the k-th task operates in the null space of all higher priority tasks, and $\overline{J}^*_{i|\mathcal{P}_k}$ is the dynamically consistent generalized inverse of the i-th prioritized Jacobian. The Principle of Virtual Work allows us to determine the torques that accomplish an operational force:

$$\Gamma_k = J^{*T}_{k|\mathcal{P}_k} \; F_k \tag{3}$$

where F_k is the force or impedance command to control and interact with the task point, $J^*_{k|\mathcal{P}_k}$ is the whole-body task Jacobian including the priority, and Γ_k is the task's contribution to the whole-body command of torques sent to the actuators. For command summation, we use the control structure

$$\Gamma = \sum_k \Gamma_k = \sum_k \left(J^{*T}_{k|\mathcal{P}_k} F_k \right). \tag{4}$$

This structure is a variant of our previous work on whole-body compliant control [18].

IV. Implementation

We adhere to three main objectives during the design and implementation of `stanford-wbc`: *(i)* separate concerns into mathematical foundations, runtime configurability, and independence from specific implementation environments; *(ii)* support two types of end-users, namely control integrators (library support for creating interesting applications) and researchers (investigate and modify models and control structures); *(iii)* ease experimentation and system integration by

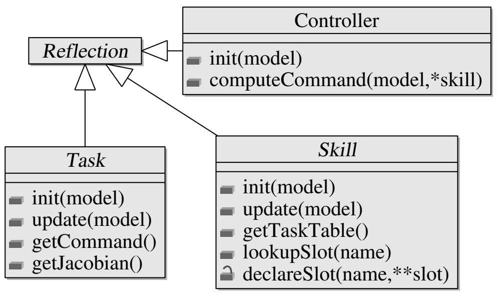


Fig. 3. Controllers, skills, and tasks follow a similar layout by providing `init()` and `update()` methods (the latter is called `computeCommand()` in the controller to make its role more explicit).

providing reusable mechanisms for data logging, parameter reflection, and interactive online configuration.

We follow the tried and trusted separation into layered libraries that are then pulled into OS- and application-specific executables. We rely on object-oriented programming (C++) to allow extension by inheritance and composition. While relying on some widespread open-source libraries, we carefully avoid dependencies on any particular middleware or operating system. We strive to avoid feature bloat and keep all interfaces clear and minimal.

This section begins with a high-level architectural overview, then presents the code structure from the bottom up, and finally describes the patterns and utilities for object creation during startup and on-line reconfiguration of the running tasks and skills.

A. System Architecture

Fig. 2 depicts the main components of our framework and the information that flows during operation. The **Controller** implements Eq. (4) to compute the torque commands Γ_{des} that are sent to the robot, based on the Jacobians J_i and the desired accelerations $\ddot{x}_{\text{des},i}$ or forces $F_{des,i}$ that are provided by the **Task** objects in the currently active task hierarchy. The **Model** estimates the joint-space kinematics and dynamics, used by tasks for internal updates and by the controller for gravity compensation and dynamic decoupling. The **Skill** is responsible for determining the currently active task hierarchy. The latter is denoted *current state* in the figure because hierarchy switching is typically implemented using a finite-state machine that processes external sensorimotor feedback as well as internal task feasibility information.

The system architecture diagram hints at the runtime configurability and reflection capabilities by indicating *parameter* lists associated with the skill, tasks, and the controller. These parameters are declared with type and name in the source code, and at runtime they can be enumerated, inspected, and changed by outside processes.

B. Implementation Structure

Fig. 3 shows the main aspects of the controller, skill, and task base classes. The common `Reflection` base class provides uniform parameter introspection capabilities which are described in Sec. IV-C.

Algorithm 1 Hierarchical Task Decomposition *(simplified)*.

```
**Input:** joint-space inverse mass-inertia $\mathbf{A}^{-1}$
  joint-space gravity torque vector $\mathbf{g}$
  desired task accelerations $\ddot{\mathbf{x}}_{\text{des},i}$
  task Jacobians $\mathbf{J}_i$
**Output:** joint-space torque vector $\mathbf{\Gamma}$
  $\mathbf{\Gamma} \leftarrow 0$
  $N_0^* \leftarrow I_{n\times n}$
  for all $0 \leq i < N_{\text{tasks}}$ do
    $J_i^* \leftarrow \mathbf{J}_i N_i^*$
    $\Lambda_i^* \leftarrow \left(J_i^* \mathbf{A}^{-1} J^{*T}\right)^+$
    $p_i^* \leftarrow \Lambda_i^* J_i^* \mathbf{A}^{-1}\mathbf{g}$
    $F_{\text{comp},i} \leftarrow \Lambda_i^* J_i^* \mathbf{A}^{-1}\mathbf{\Gamma}$
    $\mathbf{\Gamma} \leftarrow \mathbf{\Gamma} + J_i^{*T}\left(\Lambda_i^* \ddot{\mathbf{x}}_{\text{des},i} + p_i^* - F_{\text{comp},i}\right)$
    if $i < N_{\text{tasks}} - 1$ then
      $N_{i+1}^* \leftarrow \left(I_{n\times n} - \mathbf{A}^{-1} J_i^{*T} \Lambda_i^* J_i^*\right) N_i^*$
    end if
  end for
  return $\Gamma$
```

Algorithm 1 gives simplified pseudo-code for the main method (called `computeCommand`) of the default **Controller** subclass. Here, the "$(\cdot)^+$" denotes a pseudo-inverse computed by singular-value decomposition and thresholding at a task-dependent and configurable value. Several have been omitted from the pseudo-code: task feasibility feedback to the skill, switching to a safe fallback controller in case of emergency, and skipping of inactive tasks.

The **Model** is a facade [4] that hides the specifics of the kinematic and dynamic model behind a clear interface. Some of its methods involve pointers that can be treated as opaque handles but provide direct access to the dynamics engine for experts.

The **Task** class is the extension point for programming operational-space controllers independently of the specific robot. Each task takes care of controlling a point in a virtual task space that is mapped to and from the robot's joint space using the model.

The **Skill** base class is the starting point for composing specific tasks into sensorimotor skills that result in compliant whole-body behavior of the robot. A skill instance is responsible for updating its tasks and providing a valid fully defined task hierarchy at each control cycle. One intuitive way of implementing skills is to define a finite set of states, each with an associated task hierarchy, and have the skill perform state transitions based on sensor readings or other feedback methods, but the software framework does not enforce any particular task management approach.

Although a skill defines what needs to be coordinated by the tasks and the hierarchy, it (usually) does not define specific goals. Instead, a skill definition file is parsed to create and configure the required tasks, and their goals and parameters get influenced at runtime using the reflection approach described in Sec. IV-C.

Besides providing tasks to the controller, skill classes can signal emergency situations by inspecting the singular values of $J_i^* J_i^{*T}$ computed by the controller. When an essential task becomes infeasible, the controller can be asked to switch

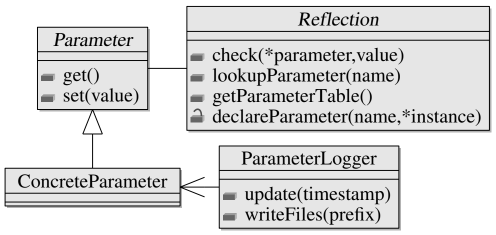


Fig. 4. Parameter reflection infrastructure. There are five types of `ConcreteParameter`: string, integer, floating point, vector, and matrix.

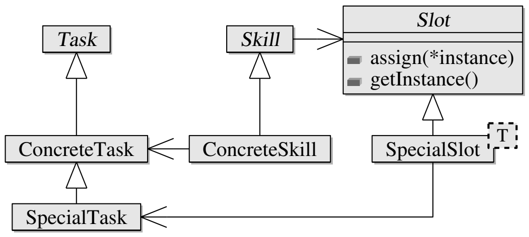


Fig. 5. Task slots allow skill subclasses to declare a placeholder for a task, and then rely on the parser and factory (not shown) to assign a task instance at runtime when the controller starts up.

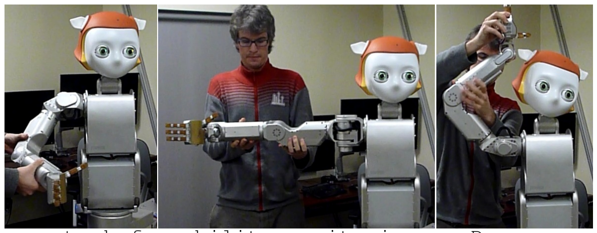
to a configurable safe fallback control law.

C. Configurability and Reflection

One of the most important features to make a software framework reusable by others is runtime configurability. For real-time capabilities, however, computational overhead is also important. We have thus traded-off generality, amount of code, and computational resources. Two features make `stanford-wbc` configurable: parameter reflection and task slots.

Parameter reflection (see Fig. 4) connects the implementation to the parser and runtime configuration engine. The provided infrastructure maintains named parameters and also provides generic logging capabilities. A subclass of `Reflection` exposes its fields as parameters simply by calling the `declareParameter()` method of its base class. Optionally, parameters can be flagged read-only or excluded from automatic logging, and reflection subclasses can intercept write requests to enforce arbitrary constraints.

Task slots (see Fig. 5) allow skills to be written independently of specific task subclasses or instances, a separation of concerns important for extensibility and reusability. For instance, if a skill requires a certain task to control the end-effector position, it should not be concerned how exactly a task achieves this. So, instead of hard-coding task types, skill subclasses call the `declareSlot()` method in their constructor, passing a name and a pointer to one of their task fields. The rest is handled by the parser and factory: the configuration file first defines tasks and their parameters, and then instantiates skills fills their slots with the appropriate task instances. In case of type mismatch, human readable error messages are generated.



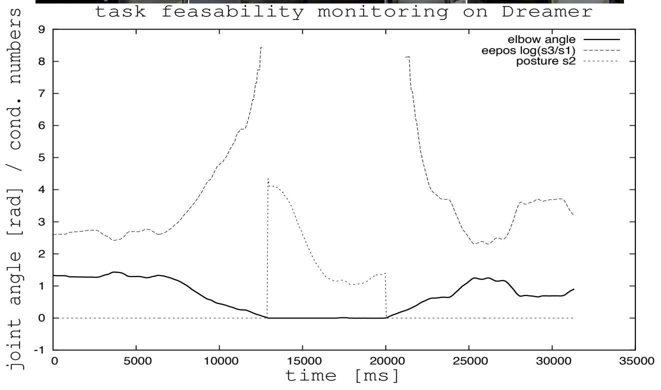


Fig. 6. Task-singularity monitoring on Dreamer. When the elbow is stretched out, the end-effector position task becomes infeasible (its condition number becomes infinite) and the joint posture task gains an extra dimension (its second eigenvalue becomes non-zero).

V. Experiments

In order to demonstrate the portability of `stanford-wbc` and its basic operation, we have performed experiments on two robots which differ significantly from each other, in terms of their mechatronics as well as concerning their development and runtime environments.

Dreamer is a humanoid upper body made by Meka Robotics for the Human-Centered Robotics Lab at the University of Texas at Austin. Its joints contain series-elastic actuators that provide high-fidelity torque control. Here, `stanford-wbc` is integrated with an RTAI application that has one real-time whole-body control thread which communicates via shared memory with the hardware driver, and one non-real-time thread for runtime configuration and debug output.

The second robot we used is **PR2** at the Stanford Robotics and AI Lab, a dual-arm mobile manipulator made by Willow Garage. PR2 ships with ROS [2] and we integrated `stanford-wbc` along with related packages into a ROS stack called `whole_body_control`. We implemented a ROS controller plugin that uses POSIX message queues as communication mechanism between real-time driver and non-real-time whole-body control application.

A. Studying Task Feasibility on Dreamer

As explained earlier, if parts of a task become infeasible, its dynamically consistent Jacobian drops rank. This is not always a problem: in fact, many if not most of the tasks will be partially infeasible, and this is naturally handled by the control structure. It is up to the skill to ensure that the ones which are critical to the behavior remain at full rank.

Fig. 7. Demonstration of the interactive `HelloByebyeSkill` on Dreamer, the Meka robot of the University of Texas at Austin, during a show and tell at Willow Garage on March 4, 2011.

In order to illustrate how readily available the required information is, Fig. 6 shows a skill with three tasks: an end-effector orientation task at the top, an end-effector position task in the middle, and a joint posture control at the bottom of the hierarchy. The orientation task is always feasible (this is a property of Dreamer's kinematic structure). The 3-dimensional position task thus has 4 dimensions to work in. It might be expected that it is always full rank, but due to kinematic singularities, some directions of the end-effector position can become uncontrollable. This is shown by the plots of the condition number, which shoots up as we manually push the arm toward specific configurations. The joint posture controller, which gets projected into the nullspaces of both the orientation and position tasks, thus has between 1 and 2 dimensions to work with. This can be seen by looking at its 2nd singular value, which is non-zero only when the end-effector position task drops rank.

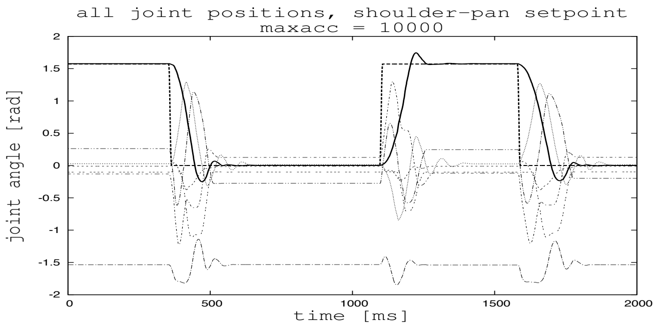
B. *Interactive Skill Demo on Dreamer*

Fig. 7 shows some video stills from demonstrating an interactive skill. The `HelloByebyeSkill` contains four tasks subdivided among two states: the `SHAKE_HANDS` and `WAVE` states each have a Cartesian end-effector position tasks and a joint-posture task. The skill starts in the `SHAKE_HANDS` state which holds the robot's hand out in front of the torso, waiting for a human to shake it to "say hello." Once that ends (detected by looking at the end-effector position error), the skill switches to the `WAVE` state which "waves goodbye" by alternating between two end-effector position goals above the robot's head.

This behavior was written in an afternoon and is fully integrated into the parameter reflection and task slot mechanisms. It thus illustrates how easy it is to extend the existing framework with novel behaviors while leveraging the existing tasks and infrastructure for wider system integration: we also implemented a thin bridge between the parameter reflection of `stanford-wbc` and ROS messages and services, thus allowing direct interaction with the running whole-body controller via the ROS command line tools or other ROS nodes.

C. *Compensating for Motor Saturation on PR2*

One of the main objectives for the mechatronic design of PR2 was to make the arms extremely safe. This was achieved by relying on a mechanical counter-balance for gravity compensation, which in turn allows the arm motors



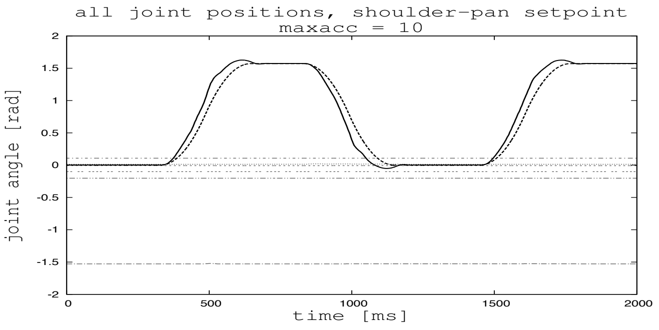


Fig. 8. In this experiment we send a step in the setpoint to the shoulder-pan joint of the left arm of PR2. All other joint setpoints remain constant. The setpoint and trajectory is shown with a thick line, and all the other joint trajectories overlaid with thin lines in order to show the coupling (before) and its absence (after). Top: without acceleration limit. Bottom: with $\ddot{x}_{\text{max}} = 10 \text{rad/s}$

to be very low-powered. While this is great for safety, it severely limits the control torques, especially in the first 4 degrees of freedom. This motor torque saturation, along with running-belt stretch in some of the transmissions, are the main reasons for the difficulties in using whole-body operational space control on PR2. A naive application of `stanford-wbc` leads to arm motions that are reminiscent of excessive dynamic coupling and severely under-damped PD controllers.

The experiments shown in Fig. 8 demonstrate that these mechatronic limitations can be partially overcome: the running-belt stretch can be compensated with experimental PR2 controller packages provided by Willow Garage, and motor torque saturation can be avoided by limiting the acceleration of task points. Moving just the first joint with a simple PD controller creates large coupling in all the other joints, but if we generate acceleration- and velocity-bounded trajectories, the coupling disappears almost entirely.

More work remains to be done to make the acceleration limitation adaptive during task execution. Also, the implications of the nullspace projections on acceleration-bounded task-points are not yet fully understood. But we have shown that it is possible to employ whole-body control on PR2 as long as the required fidelity is not very high.

D. *Demonstrating Base-Velocity Integration on PR2*

An important aspect of mobile manipulators is of course that they are not bolted to a fixed base. Fig. 9 shows video

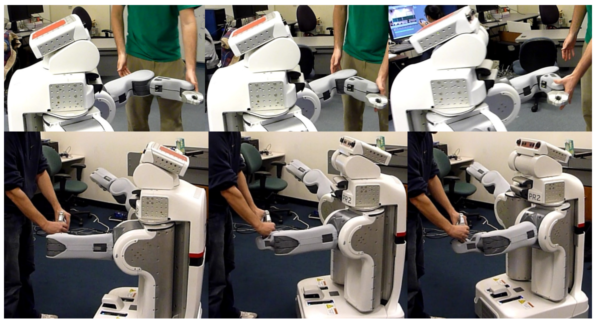


Fig. 9. Translating errors in the end-effector position and orientation into virtual torques at the base, and then translating these torques into base velocity commands using a viscosity approximation, allows to easily control the robot by pushing and twisting its hand.

stills taken during a demonstration of our integration of PR2's base velocity controller into the whole-body control framework. We use 3 virtual joints to represent the base motion to `stanford-wbc` via a simple yet effective dynamic model that treats the base as a large mass in a purely viscous medium. This maps desired torques into velocities which we send to PR2's base velocity controller.

VI. Conclusion

The much-touted advent of personal robots that will physically and safely interact with people and objects in everyday environments is not likely to come about unless we start employing mechatronics and control approaches that are designed from the ground up to properly deal with forces and dynamics. Many (if not most) of the required principles are now sufficiently understood, and recent commercial hardware developments for mobile manipulation indicate that the required technology is within our grasp. The whole-body control software presented in this paper leverages an advanced control approach that inherently supports physical interaction and can handle multiple competing control objectives in a very flexible manner. With the release as an open-source project, we are fostering its widespread adoption for upcoming developments, for research as well as for applications. In particular, the ROS stack which integrates our core libraries with an executable that uses the controller plugin architecture developed for PR2, significantly lowers the entrance barrier. Our experimental results show once again that whole-body control works, but this time the implementation is available to a large community of roboticists for testing, use, and modification.

Acknowledgement

Josh Petersen implemented PR2 code for handling motor torque saturations and base velocity integration. Samir Menon, Francois Conti, Torsten Kröger, and Ellen Klingbeil contributed to design discussions and early refactoring attempts. Kurt Konolige, Günter Niemeyer, Eric Berger, Stu Glaser, Sachin Chitta, John Hsu, Aaron Edsinger, and Robby Kelbs provided valuable support, technically as well as for the project in general.

References

[1] K.C. Chang and O. Khatib. Operational space dynamics: Efficient algorithms for modeling and control of branching mechanisms. In *Proceedings of the IEEE International Conference on Robotics and Automation*, April 2000.

[2] S. Cousins, B. Gerkey, K. Conley, and W. Garage. Sharing software with ros [ros topics]. *Robotics Automation Magazine, IEEE*, 17(2):12 –14, june 2010.

[3] Free Software Foundation. LGPLv3 (GNU Lesser General Public License version 3). `http://www.gnu.org/licenses/lgpl.html`.

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1321–1326, Leuven, Belgium, 1998.

[6] G. Hirzinger, J. Bals, M. Otter, and J. Stelter. The dlr-kuka success story: robotics research improves industrial robots. *Robotics Automation Magazine, IEEE*, 12(3):16 – 23, sept. 2005.

[7] R. Holmberg and O. Khatib. Development and control of a holonomic mobile robot for mobile manipulation tasks. *International Journal of Robotics Research*, 19(11):1066–1074, 2000.

[8] Sang-Ho Hyon. A motor control strategy with virtual musculoskeletal systems for compliant anthropomorphic robots. *Mechatronics, IEEE/ASME Transactions on*, 14(6):677 –688, dec. 2009.

[9] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Resolved momentum control: Humanoid motion planning based on the linear and angular momentum. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1644–1650, Las Vegas, USA, October 2003.

[10] O. Khatib. *Commande Dynamique dans l'Espace Opérationnel des Robots Manipulateurs en Présence d'Obstacles*. PhD thesis, l'École Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse, France, 1980.

[11] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *International Journal of Robotics Research*, 3(1):43–53, 1987.

[12] O. Khatib, L. Sentis, and J. Park. A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In *Springer Tracts in Advanced Robotics - STAR Series*, Prague, Czech Republic, March 2008.

[13] I. Mizuuchi, Y. Nakanishi, Y. Sodeyama, Y. Namiki, T. Nishino, N. Muramatsu, J. Urata, K. Hongo, T. Yoshikai, and M. Inaba. An advanced musculoskeletal humanoid Kojiro. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 294 –299, dec. 2007.

[14] Christian Ott. *Cartesian impedance control of redundant flexible-joint robots*. Springer, 2008.

[15] M.H. Raibert and J.J. Craig. Hybrid position force control of manipulators. *ASME J. Dyn. Sys. Measurement Contr.*, 103(2):126–133, 1981.

[16] L. Sentis and O. Khatib. Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2(4):505–518, December 2005.

[17] L. Sentis and O. Khatib. A whole-body control framework for humanoids operating in human environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Orlando, USA, May 2006.

[18] L. Sentis, J. Park, and O. Khatib. Compliant control of multi-contact and center of mass behaviors in humanoid robots. *IEEE Transactions on Robotics*, 26(3):483–501, June 2010.

[19] R. Tellez, F. Ferro, S. Garcia, E. Gomez, E. Jorge, D. Mora, D. Pinyol, J. Oliver, O. Torres, J. Velazquez, and D. Faconti. Reem-B: An autonomous lightweight human-size humanoid robot. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 462 –468, dec. 2008.

[20] E.C. Whitman and C.G. Atkeson. Control of instantaneously coupled systems applied to humanoid walking. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 210 –217, dec. 2010.

[21] D. Williams and O. Khatib. The virtual linkage: A model for internal forces in multi-grasp manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1025–1030, Atlanta, USA, October 1993.