




The ARC-OPT Library for Whole-Body Control of Robotic Systems

Dennis Mronga ¹ and Frank Kirchner ^{1,2}

¹ German Research Center for Artificial Intelligence (DFKI), Bremen, Germany ² University of Bremen, Bremen, Germany

DOI: [10.21105/joss.06696](https://doi.org/10.21105/joss.06696)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Summary

ARC-OPT (Adaptive Robot Control using OPTimization) is a C++ library for Whole-Body Control (WBC) ([Sentis & Khatib, 2006](#)) of complex robotic systems, such as humanoids, quadrupedal robots, or mobile manipulators.

In general, WBC describes a robot control problem in terms of costs and constraints of a quadratic program (QP). The cost function thereby minimizes the residuals of multiple feedback controllers, each dedicated to a specific robot task, along with further objectives. In each control cycle, the QP is solved and the solution, which should fulfill all objectives if possible, is sent to the robot's actuators. WBC is a reactive control approach, which targets redundant robots and is able to control multiple tasks simultaneously, like, e.g., grasping and balancing on a humanoid robot.

Editor: [Adi Singh](#)  

Reviewers:

- [@mhubii](#)
- [@sea-bass](#)
- [@JHartzler](#)

Submitted: 07 March 2024

Published: 30 December 2024

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Statement of need

ARC-OPT supports the software developer in designing such Whole-Body Controllers by providing configuration options for different pre-defined WBC problems. Today, the methodology of WBC is well understood and several mature frameworks exist. Task Space Inverse Dynamics (TSID) ([Prete et al., 2016](#)) implements a control algorithm for legged robots on acceleration level, while the approach presented in Posa et al. (2016) operates on torque level. In Smits et al. (2009) a generalized velocity-IK framework is implemented, which is, however, tightly coupled to the Orocos project. Similarly, Pink ([Caron et al., 2024](#)) is a weighted task-based framework for differential inverse kinematics implemented in Python. The IHMC Whole-Body Controller has been developed for the ATLAS robot ([Feng et al., 2015](#)), providing control algorithms for walking and manipulation based on QPs. Drake ([Tedrake & Drake Development Team, 2019](#)) is a collection of libraries for model-based design and control of complex robots. It provides interfaces to several open-source and commercial solvers, including linear least-squares, quadratic programming, and non-linear programming. Finally, ControlIt! ([University of Texas at Austin, 2021](#)) is a middleware built around the whole-body operational space control algorithm first introduced by Sentis & Khatib (2006).

In contrast to the existing libraries, ARC-OPT implements unified interfaces for different WBC problems on velocity, acceleration and torque level, as well as options to benchmark different QP solvers and rigid body dynamics libraries on these problems. Furthermore, it provides a novel WBC approach for robots with parallel kinematic loops, which is described in Mronga et al. (2022).

Description

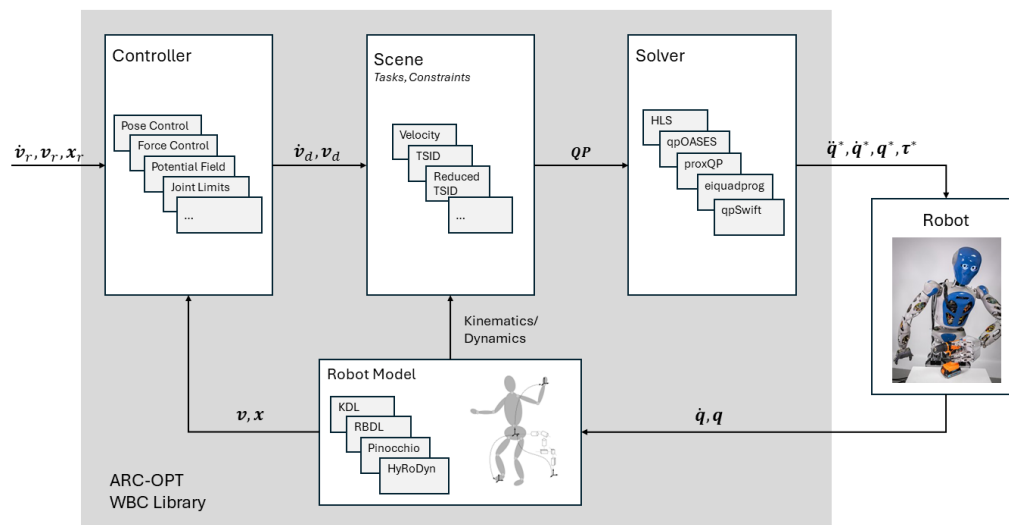


Figure 1: ARC-OPT library overview

Figure 1 shows an overview of the ARC-OPT library. ARC-OPT separates the implementation of controllers, robot model, solver, and scene, which allows a modular composition of the WBC problem:

- A **controller** implements a function in the robot's task space, e.g., for maintaining balance, avoiding an obstacle, or following a trajectory. ARC-OPT provides various controllers in joint or Cartesian space, like PD-Controllers, or repulsive potential fields.
- The **scene** sets up the QP, where the costs can be configured at runtime, and the constraints are specific for the implemented scene. Different scenes are currently implemented on velocity and acceleration level.
- The **robot model** computes the kinematic and dynamic information that the scene requires to set up the QP, like Jacobians, mass-inertia matrices, and gravity terms. ARC-OPT implements multiple robot models based on Pinocchio (Carpentier et al., 2019), RBDL (Felis, 2016), and HyroDyn (Kumar et al., 2020).
- The **solver** solves the QP and generates the required control input for the robot joints. ARC-OPT provides various QP solvers based on open-source implementations, e.g., qpOASES (Ferreau et al., 2014), eiquadprog (Buondonno, 2021), proxQP (Bambade et al., 2022), and qpSwift (Pandala et al., 2019).

Apart from this, ARC-OPT implements various concepts typically used in WBC. These include floating base dynamics and friction cone constraints, which are required for walking robots. Furthermore, task weighting and hierarchies can be used to prioritize one task over another. The software provides tutorials explaining most of these concepts.

Example

This example shows how to set up an acceleration-level WBC. Here, the tasks are formulated in the cost function. Equations of motion, rigid contacts and joint torque limits are implemented as constraints. The decision variables are the joint accelerations \ddot{q} , joint torques τ and contact wrenches f . Mathematically, this can be expressed by the following QP:

$$\begin{aligned} \min_{\mathbf{q}, \dot{\mathbf{q}}, \mathbf{f}} \quad & \left\| \sum_i \mathbf{w}_i^T (\mathbf{J}_i \ddot{\mathbf{q}} + \dot{\mathbf{J}}_i \dot{\mathbf{q}} - \dot{\mathbf{v}}_{d,i}) \right\|_2 \\ \text{s.t.} \quad & \mathbf{H} \ddot{\mathbf{q}} + \mathbf{h} = \mathbf{B} \boldsymbol{\tau} + \mathbf{J}_c \mathbf{f} \\ & \mathbf{J}_c \ddot{\mathbf{q}} = -\dot{\mathbf{J}}_c \dot{\mathbf{q}} \\ & \boldsymbol{\tau}_m \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_M \end{aligned}$$

where \mathbf{w}_i are the task weights for the i -th task, \mathbf{J}_i is the respective robot Jacobian, $\dot{\mathbf{v}}_{d,i}$ the desired task space acceleration, $\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}$ the joint positions, velocities, and accelerations, \mathbf{H} the mass-inertia matrix, \mathbf{h} the vector of gravity and Coriolis forces, $\boldsymbol{\tau}$ the robot joint torques, \mathbf{B} the control input matrix, \mathbf{f} the contact wrenches, \mathbf{J}_c the contact Jacobian, and $\boldsymbol{\tau}_m, \boldsymbol{\tau}_M$ the joint torque limits. To implement a simple Cartesian position controller for, e.g., controlling the end effector of a robot manipulator, the following PD-controller can be used:

$$\dot{\mathbf{v}}_d = \dot{\mathbf{v}}_r + \mathbf{K}_d(\mathbf{v}_r - \mathbf{v}) + \mathbf{K}_p(\mathbf{x}_r - \mathbf{x})$$

where $\mathbf{K}_p, \mathbf{K}_d$ are gain matrices, \mathbf{x}, \mathbf{v} the end effector position and velocity, $\dot{\mathbf{v}}_r, \mathbf{v}_r, \mathbf{x}_r$ the reference acceleration, velocity, and position. Figure 2 shows a C++ code snippet from ARC-OPT, which implements the above QP on a KUKA iiwa robot arm (no contacts, fixed base robot). In the code example, at first the robot model is set up using the URDF file, then the QP-solver (qpOASES), the WBC scene and a Cartesian controller are configured. In the subsequent control loop, a circular trajectory is tracked in Cartesian space. The full example can be found in the ARC-OPT tutorials¹. Figure 3 shows a visualization of the resulting robot motion. While this example only implements simple Cartesian control, more complex problems with multiple objectives/controllers can easily be composed, such as tracking the leg positions and center of mass on a walking robot.

```

RobotModelPtr robot_model = make_shared<RobotModelPinocchio>();
if(!robot_model->configure(RobotModelConfig("../models/kuka/urdf/kuka_iiwa.urdf")))
    return -1;

QPSolverPtr solver = make_shared<QP0ASESSolver>();

double dt = 0.01;
AccelerationSceneTSID scene(robot_model, solver, dt);
TaskConfig cart_task("cart_pos_ctrl", 0, "kuka_lbr_l_link_0", "kuka_lbr_l_tcp", "kuka_lbr_l_link_0", 1.0);
if(!scene.configure({cart_task}))
    return -1;

CartesianPosPDController ctrl;
ctrl.setPGain(base::Vector6d::Constant(100.0));
ctrl.setDGain(base::Vector6d::Constant(30.0));
ctrl.setFFGain(base::Vector6d::Constant(1));

RigidBodyStateSE3 setpoint, ctrl_output;
setpoint.pose.position = Eigen::Vector3d(0.0, 0.0, 1.0);
setpoint.pose.orientation.setIdentity();
setpoint.twist.setZero();

uint nj = robot_model->noOfJoints();
Joints joint_state = initialJointState(vector<double>(nj, 0.1), vector<double>(nj, 0.0), robot_model->jointNames());

JointIntegrator integrator;
for(double t = 0; t < 10; t+=dt){
    setpoint.pose.position = base::Vector3d(0.1*sin(2*M_PI*0.3*t), 0.1*sin(2*M_PI*0.3*t), 1.138);
    setpoint.twist.linear = base::Vector3d(0.1*cos(2*M_PI*0.3*t), 0.1*cos(2*M_PI*0.3*t), 0.0);

    robot_model->update(joint_state);
    RigidBodyStateSE3 feedback = robot_model->rigidBodyState(cart_task.root, cart_task.tip);
    scene.setReference(cart_task.name, ctrl.update(setpoint, feedback));
    Joints solver_output = scene.solve(scene.update());
    integrator.integrate(joint_state, solver_output, dt);

    joint_state = solver_output;
    usleep(dt * 1e6);
}

```

Figure 2: Minimal code example for Cartesian position control on a KUKA iiwa robot

¹https://github.com/ARC-OPT/wbc/blob/master/tutorials/kuka_iiwa/cart_pos_ctrl_dynamic.cpp

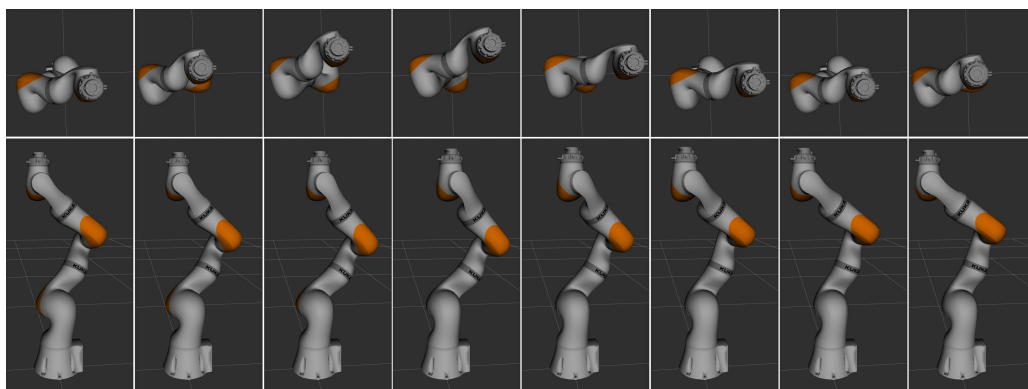


Figure 3: Screenshots of the resulting robot motion in the example

ROS 2 integration and Python Bindings

The WBC library is integrated into ROS 2 using the *ros2_control* framework. Each WBC controller is represented by a separate ROS 2 controller. Another ROS 2 controller (whole-body controller) integrates the robot model, the WBC scene and the solver. In each control step, the whole-body controller updates the robot model, sets up the costs and constraints of the QP, solves it and sends the solution to the hardware interfaces. Typical control rates for WBC are 500Hz - 1Khz. Most configuration options are exposed as ROS 2 parameters and the WBC problem can be configured according to the user's needs using .yaml configuration files.

Additionally, Python bindings for most of the library functions are available².

The ARC-OPT library for Whole-Body Control has been used in various scientific works (Mronga et al., 2022, 2020; Mronga & Kirchner, 2021; Popescu et al., 2022), and evaluated on different robots, like, e.g., the RH5 humanoid (Eßer et al., 2021) shown in Figure 4.

Acknowledgements

ARC-OPT is supported through grants from the German Federal Ministry of Education and Research (BMBF), grant numbers 01IW21002 (M-Rock project), 01IW20004 (VeryHuman project) and 01IW24008 (CoEx project).

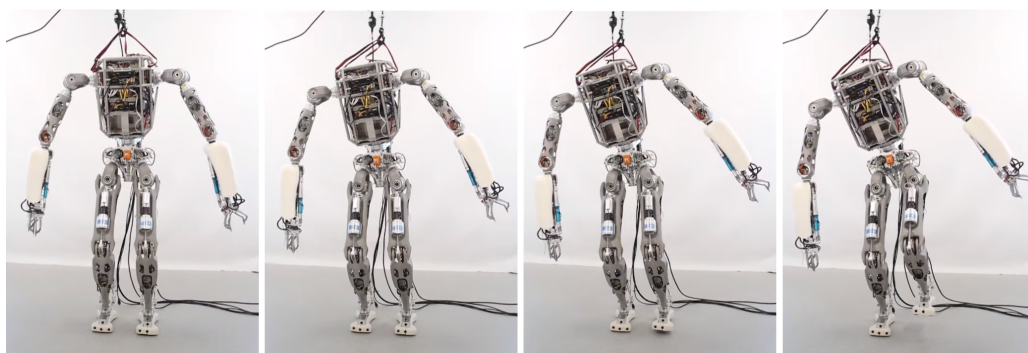


Figure 4: RH5 Humanoid robot standing on one leg using the ARC-OPT library

²https://github.com/ARC-OPT/wbc_py

References

- Bambade, A., El-Kazdadi, S., Taylor, A., & Carpentier, J. (2022, June). PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond. *RSS 2022 - Robotics: Science and Systems*. <https://doi.org/10.15607/rss.2022.xviii.040>
- Buondonno, G. (2021). *Eiquadprog*. <https://github.com/stack-of-tasks/eiquadprog>
- Caron, S., De Mont-Marin, Y., Budhiraja, R., Bang, S. H., Domrachev, I., & Nedelchev, S. (2024). *Pink: Python inverse kinematics based on Pinocchio* (Version 3.1.0). <https://github.com/stephane-caron/pink>
- Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiroux, F., Stasse, O., & Mansard, N. (2019). The Pinocchio C++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. *IEEE International Symposium on System Integrations (SII)*. <https://doi.org/10.1109/sii.2019.8700380>
- Eßer, J., Kumar, S., Peters, H., Bargsten, V., Gea, J. de, Mastalli, C., Stasse, O., & Kirchner, F. (2021). Design, analysis and control of the series-parallel hybrid RH5 humanoid robot. *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, 400–407. <https://doi.org/10.1109/HUMANOIDS47582.2021.9555770>
- Felis, M. L. (2016). RBDL: An efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 1–17. <https://doi.org/10.1007/s10514-016-9574-0>
- Feng, S., Whitman, E., Xinjilefu, X., & Atkeson, C. G. (2015). Optimization based full body control for the atlas robot. *IEEE-RAS International Conference on Humanoid Robots*, 120–127. <https://doi.org/10.1109/HUMANOIDS.2014.7041347>
- Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., & Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4), 327–363. <https://doi.org/10.1007/s12532-014-0071-1>
- Kumar, S., Szadkowski, K. A. von, Mueller, A., & Kirchner, F. (2020). An analytical and modular software workbench for solving kinematics and dynamics of series-parallel hybrid robots. *Journal of Mechanisms and Robotics*, 12(2). <https://doi.org/10.1115/1.4045941>
- Mronga, D., & Kirchner, F. (2021). Learning context-adaptive task constraints for robotic manipulation. *Robotics and Autonomous Systems*, 141, 103779. <https://doi.org/10.1016/j.robot.2021.103779>
- Mronga, D., Knobloch, T., Gea Fernández, J. de, & Kirchner, F. (2020). A constraint-based approach for human-robot collision avoidance. *Advanced Robotics*, 1–17. <https://doi.org/10.1080/01691864.2020.1721322>
- Mronga, D., Kumar, S., & Kirchner, F. (2022). Whole-body control of series-parallel hybrid robots. *2022 International Conference on Robotics and Automation (ICRA)*, 228–234. <https://doi.org/10.1109/ICRA46639.2022.9811616>
- Pandala, A. G., Ding, Y., & Park, H.-W. (2019). qpSWIFT: A real-time sparse quadratic program solver for robotic applications. *IEEE Robotics and Automation Letters*, 4(4), 3355–3362. <https://doi.org/10.1109/LRA.2019.2926664>
- Popescu, M., Mronga, D., Bergonzani, I., Kumar, S., & Kirchner, F. (2022). Experimental investigations into using motion capture state feedback for real-time control of a humanoid robot. *Sensors*, 22(24). <https://doi.org/10.3390/s22249853>
- Posa, M., Kuindersma, S., & Tedrake, R. (2016). Optimization and stabilization of trajectories for constrained dynamical systems. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 1366–1373. <https://doi.org/10.1109/ICRA.2016.7487270>
- Prete, A. del, Mansard, N., Ramos Ponce, O. E., Stasse, O., & Nori, F. (2016). Imple-

- menting Torque Control with High-Ratio Gear Boxes and without Joint-Torque Sensors. *International Journal of Humanoid Robotics*, 13(1), 1550044. <https://doi.org/10.1142/s0219843615500449>
- Sentis, L., & Khatib, O. (2006). A whole-body control framework for humanoids operating in human environments. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2641–2648. <https://doi.org/10.1109/ROBOT.2006.1642100>
- Smits, R., De Laet, T., Claes, K., Bruyninckx, H., & De Schutter, J. (2009). iTASC: A tool for multi-sensor integration in robot manipulation. In H. Hahn, H. Ko, & S. Lee (Eds.), *Multisensor fusion and integration for intelligent systems: An edition of the selected papers from the IEEE international conference on multisensor fusion and integration for intelligent systems 2008* (pp. 235–254). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-89859-7_17
- Tedrake, R., & Drake Development Team, the. (2019). *Drake: Model-based design and verification for robotics*. <https://drake.mit.edu>
- University of Texas at Austin, H.-C. R. L. of the. (2021). *ControlIt! - a whole body operational space control middleware*. <https://github.com/liangfok/controlit>