ORCA Documentation

Release Alpago

Antoine Hoarau
Ryan Lober

Jan 22, 2020

Getting Started

ORCA is a c++ whole-body reactive controller meant to compute the desired actuation torque of a robot given some tasks to perform and some constraints.

CHAPTER 1

Motivation

1.1 Table of Contents

1.1.1 Installation and Configuration

This guide will take you through the steps to install ORCA on your machine. ORCA is cross platform so you should be able to install it on Linux, OSX, and Windows.

Dependencies

- A modern **c++11** compiler (gcc > 4.8 or clang > 3.8)
- **cmake** > 3.1
- **iDynTree** (optional, shipped)
- **qpOASES** 3 (optional, shipped)
- **Eigen** 3 (optional, shipped)
- **Gazebo** 8 (optional)

ORCA is self contained! That means that is ships with both **iDynTree** and **qpOASES** inside the project, allowing for fast installations and easy integration on other platforms. Therefore you can start by simply building ORCA from source and it will include the necessary dependencies so you can get up and running.

Always keep in mind that it's better to install the dependencies separately if you plan to use **iDynTree** or **qpOASES** in other projects. For now only **iDynTree** headers appear in public headers, but will be removed eventually to ease the distribution of this library.

If you want to install the dependencies separately please read the following section: *Installing the dependencies*. Otherwise, if you just want to get coding, then jump ahead to *Installing ORCA*.

Note: You can almost always avoid calling sudo, by calling `cmake .. -DCMAKE_INSTALL_PREFIX=/some/dir` and exporting the `CMAKE_PREFIX_PATH` variable: `export CMAKE_PREFIX_PATH=$CMAKE_PREFIX_PATH:/some/dir`.

Installing the dependencies

This installation requires you to build the dependencies separately, but will give you better control over versioning and getting the latest features and bug fixes.

Eigen

```
wget http://bitbucket.org/eigen/eigen/get/3.3.4.tar.bz2
tar xjvf 3.3.4.tar.bz2
cd eigen-eigen-dc6cfdf9bcec
mkdir build ; cd build
cmake --build .
sudo cmake --build . --target install
```

qpOASES

```
wget https://www.coin-or.org/download/source/qpOASES/qpOASES-3.2.1.zip
unzip qpOASES-3.2.1.zip
cd qpOASES-3.2.1
mkdir build ; cd build
cmake .. -DCMAKE_CXX_FLAGS="-fPIC" -DCMAKE_BUILD_TYPE=Release
cmake --build .
sudo cmake --build . --target install
```

iDynTree

```
git clone https://github.com/robotology/idyntree
cd idyntree
mkdir build ; cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build .
sudo cmake --build . --target install
```

Gazebo

Examples are built with Gazebo 8. They can be adapted of course to be backwards compatible.

```
curl -ssL http://get.gazebosim.org | sh
```

Installing ORCA

Whether or not you have installed the dependencies separately, you are now ready to clone, build and install ORCA. Hooray.

```
git clone https://github.com/syroco/orca
cd orca
mkdir build ; cd build
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake --build .
sudo cmake --build . --target install
```

Testing your installation

Assuming you followed the directions to the letter and encountered no compiler errors along the way, then you are ready to get started with ORCA. Before moving on to the *Examples*, check out the *Quick Start Guide* to test your install and awe in the epicness of ORCA!

1.1.2 Quick Start Guide

First off, make sure you have followed the *Installation and Configuration* guide step by step.

If you have successfully installed ORCA then we can go ahead and try out one of the examples to get things up and running. To do so we will launch the example: `06-trajectory_following` (more info here: *Minimum jerk Cartesian trajectory following*)

This example assumes you have Gazebo >=8.0 installed on your machine. If not please follow the Gazebo tutorial for your system (http://gazebosim.org/tutorials?cat=install) and rebuild the ORCA library.

Once you have Gazebo, to launch the example open a terminal and run:

```
06-trajectory_following [path_to_orca]/examples/resources/lwr.urdf
```

Important: Make sure to replace `[path_to_orca]` with the real path to the ORCA repo on your system.

Now, open a second terminal and run:

```
gzclient
```

If everything goes well then you should see the robot moving back and forth like this:

What's next?

Check out *Where to go from here?* for more info.

1.1.3 Where to go from here?

Check out the examples

A number of examples have been included in the source code to help you better understand how ORCA works and how you can use it. The examples are grouped based on the concepts they demonstrate. We also provide some examples for using 3rd party libraries together with ORCA.

Want to use ORCA in you project?

Check out the *Using ORCA in your projects* page for information on how to include the ORCA library into your next control project.

Check out the API Documentation

You can find the Doxygen generated API documentation at the following link: *API Documentation*. This will help you navigate the ORCA API for your projects.

ROS or OROCOS user?

We have written ROS and OROCOS wrappers for the ORCA library and done most of the heavy lifting so you can get started using the contoller right away. To learn more about these projects please check out their respective pages:

`ORCA_ROS`: https://github.com/syroco/orca_ros

`RTT_ORCA`: https://github.com/syroco/rtt_orca (Compatible with ORCA < version 2.0.0)

1.1.4 Building the documentation

The ORCA documentation is composed of two parts. The **user's manual** (what you are currently reading) and the **API Reference**. Since ORCA is written entirely in `c++` the API documentation is generated with Doxygen. The manual, on the otherhand, is generated with python Sphinx… because frankly it is prettier.

Obviously, you can always visit the url: insert_url_here

to read the documentation online, but you can also generate it locally easily thanks to the magical powers of python.

How to build

First we need to install some dependencies for python and of course doxygen.

Python dependencies

```
pip3 install -U --user pip sphinx sphinx-autobuild recommonmark sphinx_rtd_theme
```

or if using Python 2.x

```
pip2 install -U --user pip sphinx sphinx-autobuild recommonmark sphinx_rtd_theme
```

Doxygen

You can always install Doxygen from source by following:

```
git clone https://github.com/doxygen/doxygen.git
cd doxygen
mkdir build
cd build
cmake -G "Unix Makefiles" ..
make
sudo make install
```

but we would recommend installing the binaries.

Linux:

```
sudo apt install doxygen
```

OSX:

```
brew install doxygen
```

Windows:

Download the executable file here: http://www.stack.nl/~dimitri/doxygen/download.html and follow the install wizard.

Building the docs with Sphinx

```
cd [orca_root]
cd docs/
make html
```

`[orca_root]` is the path to wherever you cloned the repo i.e. `/home/$USER/orca/`.

How to browse

Since Sphinx builds static websites you can simply find the file `docs/build/html/index.html` and open it in a browser.

If you prefer to be a fancy-pants then you can launch a local web server by navigating to `docs/` and running:

```
make livehtml
```

This method has the advantage of automatically refreshing when you make changes to the `.rst` files. You can browse the site at: http://127.0.0.1:8000.

1.1.5 Using ORCA in your projects

If you want to you ORCA in your project you can either use pure `CMake` or `catkin`.

CMake

```
# You need at least version 3.1 to use the modern CMake targets.
cmake_minimum_required(VERSION 3.1.0)

# Your project's name
project(my_super_orca_project)

# Tell CMake to find ORCA
find_package(orca REQUIRED)

# Add your executable(s) and/or library(ies) and their corresponding source files.
add_executable(${PROJECT_NAME} my_super_orca_project.cc)

# Point CMake to the ORCA targets.
target_link_libraries(${PROJECT_NAME} orca::orca)
```

catkin

Note: As of now, `catkin` does not support modern cmake targets and so you have some superfluous cmake steps to do when working with `catkin` workspaces.

```
# You need at least version 2.8.3 to use the modern CMake targets.
cmake_minimum_required(VERSION 2.8.3)

# Your project's name
project(my_super_orca_catkin_project)

# Tell CMake to find ORCA
find_package(orca REQUIRED)

# Tell catkin to find ORCA
find_package(catkin REQUIRED COMPONENTS orca)

# Include the catkin headers
include_directories(${catkin_INCLUDE_DIRS})

# Add your executable(s) and/or library(ies) and their corresponding source files.
add_executable(${PROJECT_NAME} my_super_orca_catkin_project.cc)

# Point CMake to the catkin and ORCA targets.
target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES} orca::orca)
```

1.1.6 API Reference

All of the API documentation is autogenerated using Doxygen. Click the link below to be redirected.

API Documentation

1.1.7 Basic

Simple controller

Note: The source code for this example can be found in `[orca_root]/examples/basic/01-simple_controller.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/basic/01-simple_controller.cc

---

Objective

In this example we want to show the basics of using ORCA. Here, we create a minimal controller with one task and some common constraints.

Introduction

First we need to include the appropriate headers and use the right namespaces. When you are getting started the easiest solution is to use the helper header `orca.h` and helper namespace `orca::all` which include all the necessary headers and opens up all their namespaces. This helps with reducing the verbosity of the examples here but is not recommended for production builds because it will cause code bloat.

```
#include <orca/orca.h>
using namespace orca::all;
```

We then create our `main()` function…

```
int main(int argc, char const *argv[])
```

and parse the command line arguments:

```
if(argc < 2)
{
    std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l␣
↪debug/info/warning/error)" << "\n";
    return -1;
}
std::string urdf_url(argv[1]);

orca::utils::Logger::parseArgv(argc, argv);
```

ORCA provides a utility class called `Logger` which, as its name implies, helps log output. See the API documentation for more information on logging levels.

Setup

Now we get to the good stuff. We start by creating a robot model which gives us access to the robot's kinematics and dynamics.

```
auto robot_model = std::make_shared<RobotModel>();
robot->loadModelFromFile(urdf_url);
robot->setBaseFrame("base_link");
robot->setGravity(Eigen::Vector3d(0,0,-9.81));
```

We first instantiate a `shared_ptr` to the class `RobotModel`. We can pass a robot name, but if we don't, it is extracted from the urdf, which is loaded from a file in `robot->loadModelFromFile(urdf_url);`. If the URDF is parsed then we need to set the base frame in which all transformations (e.g. end effector pose) are expressed in `robot->setBaseFrame("base_link");`. Finally we manually set the gravity vector `robot->setGravity(Eigen::Vector3d(0,0,-9.81));` (this is optional).

The next step is to set the initial state of the robot. For your convenience, ORCA provides a helper class called `EigenRobotState` which stores the whole state of the robot as eigen vectors/matrices. This class is totally optional, it is just meant to keep consistency for the sizes of all the vectors/matrices. You can use it to fill data from either a real robot or simulated robot.

```
EigenRobotState eigState;
eigState.resize(robot->getNrOfDegreesOfFreedom());
eigState.jointPos.setZero();
eigState.jointVel.setZero();
robot->setRobotState(eigState.jointPos,eigState.jointVel);
```

First we resize all the vectors/matrices to match the robot configuration and set the joint positions and velocities to zero. Initial joint positions are often non-zero but we are lazy and `setZero()` is so easy to type. Finally, we set the robot state, `robot->setRobotState(eigState.jointPos,eigState.jointVel);`. Now the robot is considered 'initialized'.

Note: Here we only set $\boldsymbol{q}, \dot{\boldsymbol{q}}$ because in this example we are dealing with a fixed base robot.

Creating the Controller

With the robot created and initialized, we can construct a `Controller`:

```
// Instanciate an ORCA Controller
orca::optim::Controller controller(
    "controller"
    ,robot
    ,orca::optim::ResolutionStrategy::OneLevelWeighted
    ,QPSolver::qpOASES
);
```

To do so we pass a name, `"controller"`, the robot model, `robot`, a `ResolutionStrategy`, `orca::optim::ResolutionStrategy::OneLevelWeighted`, and a solver, `QPSolver::qpOASES`.

Note: As of now, the only supported solver is `qpOASES`, however `OSQP` will be integrated in a future release.

Note: Other `ResolutionStrategy` options include: `MultiLevelWeighted`, and `Generalized`. Please be aware that these strategies are not yet officially supported.

If your robot's low level controller takes into account the gravity and coriolis torques already (Like with KUKA LWR) then you can tell the controller to remove these components from the torques computed by the solver. Setting them to false keeps the components in the solution (this is the default behavior).

```
controller.removeGravityTorquesFromSolution(true);
controller.removeCoriolisTorquesFromSolution(true);
```

Adding Tasks

With the controller created we can now start adding tasks. In this introductory example, we add only a Cartesian acceleration task for the end-effector.

```
auto cart_task = std::make_shared<CartesianTask>("CartTask_EE");
controller.addTask(cart_task);
```

A `shared_ptr` to a `CartesianTask` is created with a unique name, `CartTask_EE`. The task is then added to the controller to initialize it.

For this task, we want to control `link_7`,

```
cart_task->setControlFrame("link_7");
```

And set its desired pose:

```
Eigen::Affine3d cart_pos_ref;
cart_pos_ref.translation() = Eigen::Vector3d(1.,0.75,0.5); // x,y,z in meters
cart_pos_ref.linear() = Eigen::Quaterniond::Identity().toRotationMatrix();
```

We also set the desired cartesian velocity and acceleration to zero.

```
Vector6d cart_vel_ref = Vector6d::Zero();
Vector6d cart_acc_ref = Vector6d::Zero();
```

Note: Rotation is done with a Matrix3x3 and it can be initialized in a few ways. Note that each of these methods produce equivalent Rotation matrices in this case.

Example 1: create a quaternion from Euler anglers ZYZ convention

```
Eigen::Quaterniond quat;
quat = Eigen::AngleAxisd(0, Eigen::Vector3d::UnitZ())
     * Eigen::AngleAxisd(0, Eigen::Vector3d::UnitY())
     * Eigen::AngleAxisd(0, Eigen::Vector3d::UnitZ());
cart_pos_ref.linear() = quat.toRotationMatrix();
```

Example 2: create a quaternion from RPY convention

```
cart_pos_ref.linear() = quatFromRPY(0,0,0).toRotationMatrix();
```

Example 3: create a quaternion from Kuka Convention

```
cart_pos_ref.linear() = quatFromKukaConvention(0,0,0).toRotationMatrix();
```

Example 4: use an Identity quaternion

```
cart_pos_ref.linear() = Eigen::Quaterniond::Identity().toRotationMatrix();
```

The desired values are set on the servo controller because `CartesianTask` expects a cartesian acceleration, which is computed automatically by the servo controller.

```
cart_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_
↪ref);
```

Now set the servoing PID

```
Vector6d P;
P << 1000, 1000, 1000, 10, 10, 10;
cart_task->servoController()->pid()->setProportionalGain(P);
Vector6d D;
D << 100, 100, 100, 1, 1, 1;
cart_task->servoController()->pid()->setDerivativeGain(D);
```

Adding Constraints

Now we add some constraints. We start with a joint torque constraint for all the actuated DoF. To create it we first get the number of actuated joints from the model.

```
const int ndof = robot->getNrOfDegreesOfFreedom();
```

The joint torque limit is usually given by the robot manufacturer and included in most robot descriptions, but for now it is not parsed directely from the URDF - so we need to add it manually.

```
auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit");
controller.addConstraint(jnt_trq_cstr);
Eigen::VectorXd jntTrqMax(ndof);
jntTrqMax.setConstant(200.0);
jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);
```

We first create a `shared_ptr` with a unique name, `auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit");` and add it to the controller `controller.addConstraint(jnt_trq_cstr);`. We then set the torque limits to $\pm 200Nm$.

Contrary to torque limits, joint position limits are automatically extracted from the URDF model. Note that you can set them if you want by simply doing `jnt_pos_cstr->setLimits(jntPosMin,jntPosMax)`.

```
auto jnt_pos_cstr = std::make_shared<JointPositionLimitConstraint>("JointPositionLimit
↪");
controller.addConstraint(jnt_pos_cstr);
```

Joint velocity limits are usually given by the robot manufacturer but like the torque limits, must be added manually for now.

```
auto jnt_vel_cstr = std::make_shared<JointVelocityLimitConstraint>("JointVelocityLimit
↪");
controller.addConstraint(jnt_vel_cstr);
Eigen::VectorXd jntVelMax(ndof);
jntVelMax.setConstant(2.0);
jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);
```

With the tasks anc constraints created and added to the controller, we can begin the control loop.

Control Loop

The control loop is where the robot model is updated using the current state information from the real or simulated robot, the control problem is formulated and solved, and the resultant joint torques are sent to the robot actuators. For this example, we simply calculate the joint torques τ at each control time step and do nothing with them. This is because we are not interacting with a real robot or a simulated robot.

```
double dt = 0.001;
double current_time = 0;

controller.activateTasksAndConstraints();

for (; current_time < 2.0; current_time +=dt)
{
    // Here you can get the data from your robot (API is robot-specific)
    // Something like :
        // eigState.jointPos = myRealRobot.getJointPositions();
        // eigState.jointVel = myRealRobot.getJointVelocities();

    robot->setRobotState(eigState.jointPos,eigState.jointVel);
    controller.update(current_time, dt);
    if(controller.solutionFound())
    {
        const Eigen::VectorXd& trq_cmd = controller.getJointTorqueCommand();

        // Send torques to the REAL robot (API is robot-specific)
        // myRealRobot.set_joint_torques(trq_cmd);
    }
    else
    {
        // WARNING : Optimal solution is NOT found
        // Perform some fallback strategy (see below)
    }
}
```

First, since we are manually stepping the time, we initialize the `current_time` to zero and the `dt=0.001`.

The next important step is to activate the tasks and constraints: `controller.activateTasksAndConstraints();`. This **must** be done before the controller update is called, or else no solution will be found.

Now that the tasks and constraints are activated, we step into the control loop, which increments `current_time` from `0.0` to `2.0` seconds by `dt`:

```
for (; current_time < 2.0; current_time +=dt)
```

At the begining of each loop, we must first retrieve the robot's state information so that we can update our robot model being used in the controller. This step depends on the robot-specific API being used and is up to the user to implement.

Note: In future examples we demonstrate how to do this with the Gazebo simulator.

After we get the appropriate state information from our robot (in this case, the joint positions and velocities) we update the robot model: `robot->setRobotState(eigState.jointPos,eigState.jointVel);` . With the model updated we now update the controller, `controller.update(current_time, dt);`. The controller update first updates all of the tasks and constraints, then formulates the optimal control problem, then solves said problem. If the controller found a solution to the optimal control problem then `controller.`

`solutionFound()` will return true and this tells you that you can get that result and use it to control your robot. Here we extract the optimal control torques, `const Eigen::VectorXd& trq_cmd = controller.getJointTorqueCommand();` and then send them to our robot, using robot specific functions.

Note: In this example, we extract only the optimal torques, but you of course have access to the full solution:

```
// The whole optimal solution [AccFb, Acc, Tfb, T, eWrenches]
const Eigen::VectorXd& full_solution = controller.getSolution();
// The optimal joint torque command
const Eigen::VectorXd& trq_cmd = controller.getJointTorqueCommand();
// The optimal joint acceleration command
const Eigen::VectorXd& trq_acc = controller.getJointAccelerationCommand();
```

If the controller fails to find a solution to the problem then `controller.solutionFound()` returns `false`, and you must implement some **fallback** strategy. By fallback, we mean some strategy to be used when we have no idea what torques to send to the robot. A simple but effective strategy, is to simply brake the robot and stop its motion.

Important: If the optimal control problem has no solution it is generally because the tasks and constraints are ill-defined and not because no solution exists. For this reason, one can implement fallback strategies which are slightly more intelligent than simply stopping the robot. For example: - Compute KKT Solution and send to the robot (solutions without inequality constraints) - PID around the current position (to slow to a halt) - Switch controllers - etc.

Shutting Things Down

Once we are finished using the controller and want to bring everything to a stop, we need to gradually deactivate the tasks and constraints to avoid any erratic behaviors at the end of the motion. To do so, we start by deactivating the tasks and constraints:

```
controller.deactivateTasksAndConstraints();
```

We then need to update the controller so the tasks and constraints can slowly ramp down to total deactivation.

```
while(!controller.tasksAndConstraintsDeactivated())
{
    current_time += dt;
    controller.update(current_time,dt);
}
```

Our controller is now deactivated and can be `deleted` or destroyed without any issues.

Typically at the end of the execution you would either stop the robot or put it into some robot-specific control mode (position control, gravity compensation, etc.).

Conclusion

In this example you have seen all of the necessary steps to getting an ORCA controller up and running. In the next examples we will look at more realistic examples where the controller interacts with a robot/simulation.

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/


#include <orca/orca.h>
using namespace orca::all;

int main(int argc, char const *argv[])
{
    // Get the urdf file from the command line
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l debug/info/warning/error)" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

```

(continues on next page)

(continued from previous page)

```cpp
    //  Parse logger level as --log_level (or -l) debug/warning etc
    orca::utils::Logger::parseArgv(argc, argv);

    // Create the kinematic model that is shared by everybody. Here you can pass a robot name
    auto robot_model = std::make_shared<RobotModel>();

     //  If you don't pass a robot name, it is extracted from the urdf
    robot_model->loadModelFromFile(urdf_url);

    // All the transformations (end effector pose for example) will be expressed wrt this base frame
    robot_model->setBaseFrame("base_link");

    // Sets the world gravity (Optional)
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81));

    // This is an helper function to store the whole state of the robot as eigen vectors/matrices. This class is totally optional, it is just meant to keep consistency for the sizes of all the vectors/matrices. You can use it to fill data from either real robot and simulated robot.
    RobotState eigState;

    // resize all the vectors/matrices to match the robot configuration
    eigState.resize(robot_model->getNrOfDegreesOfFreedom());

    // Set the initial state to zero (arbitrary). @note: here we only set q,qot because this example asserts we have a fixed base robot
    eigState.jointPos.setZero();
    eigState.jointVel.setZero();

    // Set the first state to the robot
    robot_model->setRobotState(eigState.jointPos,eigState.jointVel);
    // Now is the robot is considered 'initialized'


    // Instanciate an ORCA Controller
    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolverImplType::qpOASES
    );
    // Other ResolutionStrategy options: MultiLevelWeighted, Generalized


    // Create the servo controller that the cartesian task needs
    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("servo_controller");

    // Set the pose desired for the link_7
    Eigen::Affine3d cart_pos_ref;

    // Setting the translational components.
    cart_pos_ref.translation() = Eigen::Vector3d(1.,0.75,0.5); // x,y,z in meters

    // Rotation is done with a Matrix3x3 and it can be initialized in a few ways. Note that each of these methods produce equivalent Rotation matrices in this case.
```

(continues on next page)

(continued from previous page)

```

    // Example 1 : create a quaternion from Euler anglers ZYZ convention
    Eigen::Quaterniond quat;
    quat = Eigen::AngleAxisd(0, Eigen::Vector3d::UnitZ())
         * Eigen::AngleAxisd(0, Eigen::Vector3d::UnitY())
         * Eigen::AngleAxisd(0, Eigen::Vector3d::UnitZ());
    cart_pos_ref.linear() = quat.toRotationMatrix();

    // Example 2 : create a quaternion from RPY convention
    cart_pos_ref.linear() = quatFromRPY(0,0,0).toRotationMatrix();

    // Example 3 : create a quaternion from Kuka Convention
    cart_pos_ref.linear() = quatFromKukaConvention(0,0,0).toRotationMatrix();

    // Example 4 : use an Identity quaternion
    cart_pos_ref.linear() = Eigen::Quaterniond::Identity().toRotationMatrix();

    // Set the desired cartesian velocity and acceleration to zero
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();

    // Now set the servoing PID
    Vector6d P;
    P << 1000, 1000, 1000, 10, 10, 10;
    cart_acc_pid->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_acc_pid->pid()->setDerivativeGain(D);

    cart_acc_pid->setControlFrame("link_7");
    // The desired values are set on the servo controller. Because cart_task->setDesired expects a cartesian acceleration. Which is computed automatically by the servo controller
    cart_acc_pid->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

    // Cartesian Task
    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    // Set the servo controller to the cartesian task
    cart_task->setServoController(cart_acc_pid);

    // Get the number of actuated joints
    const int ndof = robot_model->getNrOfDegreesOfFreedom();

    // Joint torque limit is usually given by the robot manufacturer
    auto jnt_trq_cstr = controller.addConstraint<JointTorqueLimitConstraint>("JointTorqueLimit");
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);

    // Joint position limits are automatically extracted from the URDF model.
    // Note that you can set them if you want. by simply doing jnt_pos_cstr->setLimits(jntPosMin,jntPosMax).
    auto jnt_pos_cstr = controller.addConstraint<JointPositionLimitConstraint>("JointPositionLimit");

    // Joint velocity limits are usually given by the robot manufacturer
```

(continues on next page)

(continued from previous page)

```
    auto jnt_vel_cstr = controller.addConstraint<JointVelocityLimitConstraint>("JointVelocityLimit");
    Eigen::VectorXd jntVelMax(ndof);
    jntVelMax.setConstant(2.0);
    jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);


    double dt = 0.5;
    double current_time = 0;

    controller.activateTasksAndConstraints();


    // If your robot's low level controller takes into account the gravity and coriolis torques already (Like with KUKA LWR) then you can tell the controller to remove these components from the torques computed by the solver. Setting them to false keeps the components in the solution (this is the default behavior).
    controller.removeGravityTorquesFromSolution(true);
    controller.removeCoriolisTorquesFromSolution(true);

    // Now you can run the control loop
    for (; current_time < 2.0; current_time +=dt)
    {
        // Here you can get the data from you REAL robot (API is robot-specific)
        // Something like :
            // eigState.jointPos = myRealRobot.getJointPositions();
            // eigState.jointVel = myRealRobot.getJointVelocities();

        // Now update the internal kinematic model with data from the REAL robot
        std::cout << "Setting robot state to : \n"
            << "Joint Pos : " << eigState.jointPos.transpose() << '\n'
            << "Joint Vel : " << eigState.jointVel.transpose() << '\n';

        robot_model->setRobotState(eigState.jointPos,eigState.jointVel);

        // Step the controller + solve the internal optimal problem
        std::cout << "Updating controller..." ;
        controller.update(current_time, dt);
        std::cout << "OK" << '\n';

        // Do what you want with the solution
        if(controller.solutionFound())
        {
            // The whole optimal solution [AccFb, Acc, Tfb, T, eWrenches]
            const Eigen::VectorXd& full_solution = controller.getSolution();
            // The optimal joint torque command
            const Eigen::VectorXd& trq_cmd = controller.getJointTorqueCommand();
            // The optimal joint acceleration command
            const Eigen::VectorXd& trq_acc = controller.getJointAccelerationCommand();

            // Send torques to the REAL robot (API is robot-specific)
            //real_tobot->set_joint_torques(trq_cmd);
        }
        else
        {
            // WARNING : Optimal solution is NOT found
            // Switching to a fallback strategy
```

(continues on next page)

(continued from previous page)

```
            // Typical are :
            // - Stop the robot (robot-specific method)
            // - Compute KKT Solution and send to the robot (dangerous)
            // - PID around the current position (dangerous)

            // trq = controller.computeKKTTorques();
            // Send torques to the REAL robot (API is robot-specific)
            // real_tobot->set_joint_torques(trq_cmd);
        }
    }

    // Print the last computed solution (just for fun)
    const Eigen::VectorXd& full_solution = controller.getSolution();
    const Eigen::VectorXd& trq_cmd = controller.getJointTorqueCommand();
    const Eigen::VectorXd& trq_acc = controller.getJointAccelerationCommand();
    std::cout << "Full solution : " << full_solution.transpose() << '\n';
    std::cout << "Joint Acceleration command : " << trq_acc.transpose() << '\n';
    std::cout << "Joint Torque command       : " << trq_cmd.transpose() << '\n';

    // At some point you want to close the controller nicely
    controller.deactivateTasksAndConstraints();


    // Let all the tasks ramp down to zero
    while(!controller.tasksAndConstraintsDeactivated())
    {
        current_time += dt;
        controller.update(current_time,dt);
    }

    // All objets will be destroyed here
    return 0;
}
```

Simulating the controller performance

Note: The source code for this example can be found in `[orca_root]/examples/basic/02-simulating_results.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/basic/02-simulating_results.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
```

(continues on next page)

(continued from previous page)

```
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
using namespace orca::all;



int main(int argc, char const *argv[])
{
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l debug/info/warning/error)" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    orca::utils::Logger::parseArgv(argc, argv);

    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->setBaseFrame("base_link");
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81));
    RobotState eigState;
    eigState.resize(robot_model->getNrOfDegreesOfFreedom());
    eigState.jointPos.setZero();
    eigState.jointVel.setZero();
    robot_model->setRobotState(eigState.jointPos,eigState.jointVel);
```

(continues on next page)

(continued from previous page)

```

    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolverImplType::qpOASES
    );

    // Create the servo controller that the cartesian task needs
    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("servo_controller");
    // Now set the servoing PID
    Vector6d P;
    P << 1000, 1000, 1000, 10, 10, 10;
    cart_acc_pid->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_acc_pid->pid()->setDerivativeGain(D);

    cart_acc_pid->setControlFrame("link_7");

    Eigen::Affine3d cart_pos_ref;
    cart_pos_ref.translation() = Eigen::Vector3d(1.,0.75,0.5); // x,y,z in meters
    cart_pos_ref.linear() = Eigen::Quaterniond::Identity().toRotationMatrix();

    // Set the desired cartesian velocity and acceleration to zero
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();

    // The desired values are set on the servo controller. Because cart_task->setDesired expects a cartesian acceleration. Which is computed automatically by the servo controller
    cart_acc_pid->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);
    // Set the servo controller to the cartesian task
    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    cart_task->setServoController(cart_acc_pid);

    // ndof
    const int ndof = robot_model->getNrOfDegreesOfFreedom();

    auto jnt_trq_cstr = controller.addConstraint<JointTorqueLimitConstraint>("JointTorqueLimit");
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);

    auto jnt_pos_cstr = controller.addConstraint<JointPositionLimitConstraint>("JointPositionLimit");

    auto jnt_vel_cstr = controller.addConstraint<JointVelocityLimitConstraint>("JointVelocityLimit");
    Eigen::VectorXd jntVelMax(ndof);
    jntVelMax.setConstant(2.0);
    jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);


    controller.activateTasksAndConstraints();
```

(continues on next page)

(continued from previous page)

```
    // for each task, it calls task->activate(), that can call onActivationCallback() if it is set.
    // To set it :
    // task->setOnActivationCallback([&]()
    // {
    //      // Do some initialisation here
    // });
    // Note : you need to set it BEFORE calling
    // controller.activateTasksAndConstraints();





    double dt = 0.001;
    double current_time = 0.0;
    Eigen::VectorXd trq_cmd(ndof);
    Eigen::VectorXd acc_new(ndof);

    controller.update(current_time, dt);
    current_time += dt;


    controller.print();

    std::cout << "\n\n\n" << '\n';
    std::cout << "=====================================" << '\n';
    //std::cout << "Initial State:\n" << cart_task->servoController()->getCurrentCartesianPose() << '\n';
    std::cout << "Desired State:\n" << cart_pos_ref.matrix() << '\n';
    std::cout << "=====================================" << '\n';
    std::cout << "\n\n\n" << '\n';
    std::cout << "Begining Simulation..." << '\n';

    int print_counter = 0;
    for (; current_time < 10.0; current_time +=dt)
    {


        if(print_counter == 100)
        {
            std::cout << "Task position at t = " << current_time << "\t---\t" << cart_acc_pid->getCurrentCartesianPose().block(0,3,3,1).transpose() << '\n';
            print_counter = 0;
        }
        ++print_counter;

        controller.update(current_time, dt);

        if(controller.solutionFound())
        {
            trq_cmd = controller.getJointTorqueCommand();
        }
        else
        {
            std::cout << "[warning] Didn't find a solution. Stopping simulation." << '\n';
```

(continues on next page)

(continued from previous page)

```
            break;
        }

        acc_new = robot_model->getMassMatrix().ldlt().solve(trq_cmd - robot_model->
↪getJointGravityAndCoriolisTorques());

        eigState.jointPos += eigState.jointVel * dt + ((acc_new*dt*dt)/2);
        eigState.jointVel += acc_new * dt;

        robot_model->setRobotState(eigState.jointPos,eigState.jointVel);

    }
    std::cout << "Simulation finished." << '\n';
    std::cout << "\n\n\n" << '\n';
    std::cout << "=====================================" << '\n';
    //std::cout << "Final State:\n" << cart_task->servoController()->
↪getCurrentCartesianPose() << '\n';
    //std::cout << "Position error:\n" << cart_task->servoController()->
↪getCurrentCartesianPose().block(0,3,3,1) - cart_pos_ref.translation() << '\n';




    // All objets will be destroyed here
    return 0;
}
```

1.1.8 Intermediate

An introduction to the ORCA callback system

Note: The source code for this example can be found in `[orca_root]/examples/intermediate/02-using_callbacks.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/intermediate/01-using_callbacks.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
```

(continues on next page)

(continued from previous page)

```
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
#include <chrono>
using namespace orca::all;

class TaskMonitor {
private:
    bool is_activated_ = false;
    bool is_deactivated_ = false;


public:
    TaskMonitor ()
    {
        std::cout << "TaskMonitor class constructed." << '\n';
    }
    bool isActivated(){return is_activated_;}
    bool isDeactivated(){return is_deactivated_;}

    void onActivation()
    {
        std::cout << "[TaskMonitor] Called 'onActivation' callback." << '\n';
    }

    void onActivated()
    {
        std::cout << "[TaskMonitor] Called 'onActivated' callback." << '\n';
        is_activated_ = true;
    }

    void onUpdateEnd(double current_time, double dt)
    {
```

(continues on next page)

(continued from previous page)

```
        std::cout << "[TaskMonitor] Called 'onUpdateBegin' callback." << '\n';
        std::cout << "  >> current time: " << current_time << '\n';
        std::cout << "  >> dt: " << dt << '\n';
    }

    void onUpdateBegin(double current_time, double dt)
    {
        std::cout << "[TaskMonitor] Called 'onUpdateEnd' callback." << '\n';
        std::cout << "  >> current time: " << current_time << '\n';
        std::cout << "  >> dt: " << dt << '\n';
    }
    void onDeactivation()
    {
        std::cout << "[TaskMonitor] Called 'onDeactivation' callback." << '\n';
    }

    void onDeactivated()
    {
        std::cout << "[TaskMonitor] Called 'onDeactivated' callback." << '\n';
        is_deactivated_ = true;
    }
};




int main(int argc, char const *argv[])
{
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l debug/info/warning/error)" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    orca::utils::Logger::parseArgv(argc, argv);

    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->setBaseFrame("base_link");
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81));
    RobotState eigState;
    eigState.resize(robot_model->getNrOfDegreesOfFreedom());
    eigState.jointPos.setZero();
    eigState.jointVel.setZero();
    robot_model->setRobotState(eigState.jointPos,eigState.jointVel);

    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolverImplType::qpOASES
    );

    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("servo_controller");
```

(continues on next page)

(continued from previous page)

```
    Vector6d P;
    P << 1000, 1000, 1000, 10, 10, 10;
    cart_acc_pid->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_acc_pid->pid()->setDerivativeGain(D);
    cart_acc_pid->setControlFrame("link_7");
    Eigen::Affine3d cart_pos_ref;
    cart_pos_ref.translation() = Eigen::Vector3d(0.3,-0.5,0.41); // x,y,z in meters
    cart_pos_ref.linear() = orca::math::quatFromRPY(M_PI,0,0).toRotationMatrix();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();
    cart_acc_pid->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    cart_task->setServoController(cart_acc_pid);

    const int ndof = robot_model->getNrOfDegreesOfFreedom();

    auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit");
    controller.addConstraint(jnt_trq_cstr);
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);

    auto jnt_pos_cstr = std::make_shared<JointPositionLimitConstraint>("JointPositionLimit");
    controller.addConstraint(jnt_pos_cstr);

    auto jnt_vel_cstr = std::make_shared<JointVelocityLimitConstraint>("JointVelocityLimit");
    controller.addConstraint(jnt_vel_cstr);
    Eigen::VectorXd jntVelMax(ndof);
    jntVelMax.setConstant(2.0);
    jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);

    double dt = 0.1;
    double current_time = 0.0;
    int delay_ms = 500;

    // The good stuff...

    auto task_monitor = std::make_shared<TaskMonitor>();

    cart_task->onActivationCallback(std::bind(&TaskMonitor::onActivation, task_monitor));
    cart_task->onActivatedCallback(std::bind(&TaskMonitor::onActivated, task_monitor));
    cart_task->onComputeBeginCallback(std::bind(&TaskMonitor::onUpdateBegin, task_monitor, std::placeholders::_1, std::placeholders::_2));
    cart_task->onComputeEndCallback(std::bind(&TaskMonitor::onUpdateEnd, task_monitor, std::placeholders::_1, std::placeholders::_2));
    cart_task->onDeactivationCallback(std::bind(&TaskMonitor::onDeactivation, task_monitor));
    cart_task->onDeactivatedCallback(std::bind(&TaskMonitor::onDeactivated, task_monitor));
```

(continues on next page)

(continued from previous page)

```
    std::cout << "[main] Activating tasks and constraints." << '\n';
    controller.activateTasksAndConstraints();
    std::this_thread::sleep_for(std::chrono::milliseconds(delay_ms));

    std::cout << "[main] Starting 'RUN' while loop." << '\n';
    while(!task_monitor->isActivated()) // Run 10 times.
    {
        std::cout << "[main] 'RUN' while loop. Current time: " << current_time << '\n
↪';
        controller.update(current_time, dt);
        current_time +=dt;
        std::this_thread::sleep_for(std::chrono::milliseconds(delay_ms));
    }
    std::cout << "[main] Exiting 'RUN' while loop." << '\n';

    std::cout << "-----------------\n";

    std::cout << "[main] Deactivating tasks and constraints." << '\n';
    controller.deactivateTasksAndConstraints();
    std::this_thread::sleep_for(std::chrono::milliseconds(delay_ms));

    std::cout << "[main] Starting 'DEACTIVATION' while loop." << '\n';

    while(!task_monitor->isDeactivated())
    {
        std::cout << "[main] 'DEACTIVATION' while loop. Current time: " << current_
↪time << '\n';
        controller.update(current_time, dt);
        current_time += dt;
        std::this_thread::sleep_for(std::chrono::milliseconds(delay_ms));
    }
    std::cout << "[main] Exiting 'DEACTIVATION' while loop." << '\n';


    std::cout << "[main] Exiting main()." << '\n';
    return 0;
}
```

Using lambda functions in the callbacks

Note: The source code for this example can be found in `[orca_root]/examples/intermediate/02-using_lambda_callbacks.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/intermediate/02-using_lambda_callbacks.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
```

(continues on next page)

(continued from previous page)

```
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
using namespace orca::all;

class MinJerkPositionTrajectory {
private:
    Eigen::Vector3d alpha_, sp_, ep_;
    double duration_ = 0.0;
    double start_time_ = 0.0;
    bool first_call_ = true;
    bool traj_finished_ = false;



public:
    MinJerkPositionTrajectory (double duration)
    : duration_(duration)
    {
    }

    bool isTrajectoryFinished(){return traj_finished_;}

```

(continues on next page)

(continued from previous page)

```
    void resetTrajectory(const Eigen::Vector3d& start_position, const Eigen::Vector3d& end_position)
    {
        sp_ = start_position;
        ep_ = end_position;
        alpha_ = ep_ - sp_;
        first_call_ = true;
        traj_finished_ = false;
    }

    void getDesired(double current_time, Eigen::Vector3d& p, Eigen::Vector3d& v, Eigen::Vector3d& a)
    {
        if(first_call_)
        {
            start_time_ = current_time;
            first_call_ = false;
        }
        double tau = (current_time - start_time_) / duration_;
        if(tau >= 1.0)
        {
            p = ep_;
            v = Eigen::Vector3d::Zero();
            a = Eigen::Vector3d::Zero();

            traj_finished_ = true;
            return;
        }
        p =                         sp_ + alpha_ * ( 10*pow(tau,3.0) - 15*pow(tau,4.0)  + 6*pow(tau,5.0)   );
        v = Eigen::Vector3d::Zero() + alpha_ * ( 30*pow(tau,2.0) - 60*pow(tau,3.0)  + 30*pow(tau,4.0)  );
        a = Eigen::Vector3d::Zero() + alpha_ * ( 60*pow(tau,1.0) - 180*pow(tau,2.0) + 120*pow(tau,3.0) );
    }
};




int main(int argc, char const *argv[])
{
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l debug/info/warning/error)" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    orca::utils::Logger::parseArgv(argc, argv);

    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->setBaseFrame("base_link");
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81));
    RobotState eigState;
```

(continues on next page)

(continued from previous page)

```
    eigState.resize(robot_model->getNrOfDegreesOfFreedom());
    eigState.jointPos.setZero();
    eigState.jointVel.setZero();
    robot_model->setRobotState(eigState.jointPos,eigState.jointVel);

    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolverImplType::qpOASES
    );

    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("servo_controller
↪");
    Vector6d P;
    P << 1000, 1000, 1000, 10, 10, 10;
    cart_acc_pid->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_acc_pid->pid()->setDerivativeGain(D);
    cart_acc_pid->setControlFrame("link_7");
    Eigen::Affine3d cart_pos_ref;
    cart_pos_ref.translation() = Eigen::Vector3d(0.3,-0.5,0.41); // x,y,z in meters
    cart_pos_ref.linear() = orca::math::quatFromRPY(M_PI,0,0).toRotationMatrix();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();
    cart_acc_pid->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    cart_task->setServoController(cart_acc_pid);

    const int ndof = robot_model->getNrOfDegreesOfFreedom();

    auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit
↪");
    controller.addConstraint(jnt_trq_cstr);
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);

    auto jnt_pos_cstr = std::make_shared<JointPositionLimitConstraint>(
↪"JointPositionLimit");
    controller.addConstraint(jnt_pos_cstr);

    auto jnt_vel_cstr = std::make_shared<JointVelocityLimitConstraint>(
↪"JointVelocityLimit");
    controller.addConstraint(jnt_vel_cstr);
    Eigen::VectorXd jntVelMax(ndof);
    jntVelMax.setConstant(2.0);
    jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);

    double dt = 0.001;
    double current_time = 0.0;

    // The good stuff...

    MinJerkPositionTrajectory traj(5.0);
```

(continues on next page)

(continued from previous page)

```
    int traj_loops = 0;
    bool exit_control_loop = true;
    Eigen::Vector3d start_position, end_position;


    cart_task->onActivationCallback([](){
        std::cout << "Activating CartesianTask..." << '\n';
    });

    cart_task->onActivatedCallback([&](){
        //start_position = cart_task->servoController()->getCurrentCartesianPose().block(0,3,3,1);
        end_position = cart_pos_ref.translation();
        traj.resetTrajectory(start_position, end_position);
        std::cout << "CartesianTask activated. Begining trajectory." << '\n';
    });

    cart_task->onComputeBeginCallback([&](double current_time, double dt){
        Eigen::Vector3d p, v, a;
        traj.getDesired(current_time, p, v, a);
        cart_pos_ref.translation() = p;
        cart_vel_ref.head(3) = v;
        cart_acc_ref.head(3) = a;
        //cart_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);
    });

    cart_task->onComputeEndCallback([&](double current_time, double dt){
        if (traj.isTrajectoryFinished()  )
        {
            if (traj_loops < 4)
            {
                traj.resetTrajectory(end_position, start_position);
                std::cout << "Changing trajectory direction." << '\n';
                ++traj_loops;
            }
            else
            {
                std::cout << "Trajectory looping finished." << '\n';
                exit_control_loop = true;
            }
        }
    });

    cart_task->onDeactivationCallback([](){
        std::cout << "Deactivating task." << '\n';
    });

    cart_task->onDeactivatedCallback([](){
        std::cout << "CartesianTask deactivated. Stopping controller" << '\n';
    });

    controller.activateTasksAndConstraints();

    // Control loop
    while(traj_loops < 4)
    {
```

(continues on next page)

(continued from previous page)

```
        controller.update(current_time, dt);
        current_time +=dt;
    }
    std::cout << "Out of control loop." << '\n';

    controller.deactivateTasksAndConstraints();


    while(!controller.tasksAndConstraintsDeactivated())
    {
        controller.update(current_time, dt);
        current_time += dt;
    }
    return 0;
}
```

1.1.9 Gazebo

Simulating a single robot

Note: The source code for this example can be found in `[orca_root]/examples/gazebo/01-single_robot.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/gazebo/01-single_robot.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
```

(continues on next page)

(continued from previous page)

```cpp
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/gazebo/GazeboServer.h>
#include <orca/gazebo/GazeboModel.h>

using namespace orca::gazebo;

int main(int argc, char** argv)
{
    // Get the urdf file from the command line
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    // Instanciate the gazebo server with de dedfault empty world
    // This is equivalent to GazeboServer gz("worlds/empty.world")
    GazeboServer s;
    // Insert a model onto the server and create the GazeboModel from the return value
    // You can also set the initial pose, and override the name in the URDF
    auto m = GazeboModel(s.insertModelFromURDFFile(urdf_url));

    // This is how you can get the full state of the robot
    std::cout << "Model \'" << m.getName() << "\' State :\n" << '\n';
    std::cout << "- Gravity "                  << m.getGravity().transpose()            << '\n';
    std::cout << "- Base velocity\n"           << m.getBaseVelocity().transpose()       << '\n';
    std::cout << "- Tworld->base\n"            << m.getWorldToBaseTransform().matrix()       << '\n';
    std::cout << "- Joint positions "          << m.getJointPositions().transpose()     << '\n';
    std::cout << "- Joint velocities "         << m.getJointVelocities().transpose()    << '\n';
    std::cout << "- Joint external torques "   << m.getJointExternalTorques().transpose()    << '\n';
    std::cout << "- Joint measured torques "   << m.getJointMeasuredTorques().transpose()    << '\n';

    // You can optionally register a callback that will be called
    // after every WorldUpdateEnd, so the internal gazebo model is updated
    // and you can get the full state (q,qdot,Tworld->base, etc)
```

(continues on next page)

(continued from previous page)

```
    m.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,double dt)
    {
        std::cout << "[" << m.getName() << "]" << '\n'
            << "- iteration    " << n_iter << '\n'
            << "- current time " << current_time << '\n'
            << "- dt           " << dt << '\n';
        // Example : get the minimal state
        const Eigen::VectorXd& q = m.getJointPositions();
        const Eigen::VectorXd& qdot = m.getJointVelocities();

        std::cout << "ExtTrq " << m.getJointExternalTorques().transpose() << '\n';
        std::cout << "MeaTrq " << m.getJointMeasuredTorques().transpose() << '\n';
    });

    // Run the main simulation loop.
    // This is a blocking call that runs the simulation steps
    // It can be stopped by CTRL+C
    // You can optionally add a callback that happends after WorldUpdateEnd
    s.run();
    return 0;
}
```

Simulating multiple robots

Note: The source code for this example can be found in `[orca_root]/examples/gazebo/02-multi_robot.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/gazebo/02-multi_robot.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
```

(continues on next page)

(continued from previous page)

```
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/gazebo/GazeboServer.h>
#include <orca/gazebo/GazeboModel.h>

using namespace orca::gazebo;
using namespace Eigen;

int main(int argc, char** argv)
{
    // Get the urdf file from the command line
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    // Instanciate the gazebo server with de dedfault empty world
    // This is equivalent to GazeboServer gz("worlds/empty.world")
    GazeboServer gz_server;

    // Insert a model onto the server and create the GazeboModel from the return value
    // You can also set the initial pose, and override the name in the URDF
    auto gz_model_one = GazeboModel(gz_server.insertModelFromURDFFile(urdf_url
        ,Vector3d(-2,0,0)
        ,quatFromRPY(0,0,0)
        ,"one"));

    // Insert a second model with a different pose and a different name
    auto gz_model_two = GazeboModel(gz_server.insertModelFromURDFFile(urdf_url
        ,Vector3d(2,0,0)
        ,quatFromRPY(0,0,0)
        ,"two"));

    // You can optionally register a callback for each GazeboModel so you can do individual updates on it
    // The function is called after every WorldUpdateEnd, so the internal gazebo model is updated
    // and you can get the full state (q,qdot,Tworld->base, etc)
```

(continues on next page)

(continued from previous page)

```
    gz_model_two.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,
↪double dt)
    {
        std::cout << "gz_model_two \'" << gz_model_two.getName() << "\' callback " <<
↪'\n'
            << "- iteration    " << n_iter << '\n'
            << "- current time " << current_time << '\n'
            << "- dt           " << dt << '\n';
        // Example : get the joint positions
        // gz_model_two.getJointPositions()
    });

    // Run the main simulation loop.
    // This is a blocking call that runs the simulation steps
    // It can be stopped by CTRL+C
    // You can optionally add a callback that happends after WorldUpdateEnd
    gz_server.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,double␣
↪dt)
    {
        std::cout << "GazeboServer callback " << '\n'
            << "- iteration    " << n_iter << '\n'
            << "- current time " << current_time << '\n'
            << "- dt           " << dt << '\n';
    });
    gz_server.run();
    return 0;
}
```

Set robot state

Note: The source code for this example can be found in `[orca_root]/examples/gazebo/03-set_robot_state.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/gazebo/03-set_robot_state.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
```

(continues on next page)

(continued from previous page)

```
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
#include <orca/gazebo/GazeboServer.h>
#include <orca/gazebo/GazeboModel.h>

using namespace orca::all;
using namespace orca::gazebo;

int main(int argc, char** argv)
{
    // Get the urdf file from the command line
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    // Instanciate the gazebo server with de dedfault empty world
    GazeboServer gz_server(argc,argv);
    // This is equivalent to GazeboServer gz("worlds/empty.world")
    // Insert a model onto the server and create the GazeboModel from the return value
    // You can also set the initial pose, and override the name in the URDF
    auto gz_model = GazeboModel(gz_server.insertModelFromURDFFile(urdf_url));

    // Create an ORCA robot
    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->print();

    // Update the robot on at every iteration
    gz_model.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,double␣
↪dt)
```

(continues on next page)

(continued from previous page)

```
    {
        std::cout << "Gazebo iteration " << n_iter << " current time: " << current_time << " dt: " << dt << '\n';

        robot_model->setRobotState(gz_model.getWorldToBaseTransform().matrix()
                                    ,gz_model.getJointPositions()
                                    ,gz_model.getBaseVelocity()
                                    ,gz_model.getJointVelocities()
                                    ,gz_model.getGravity()
                                );
    });

    // Run the main simulation loop.
    // This is a blocking call that runs the simulation steps
    // It can be stopped by CTRL+C
    // You can optionally add a callback that happends after WorldUpdateEnd
    gz_server.run();
    return 0;
}
```

Set robot state with gravity compensation

Note: The source code for this example can be found in `[orca_root]/examples/gazebo/04-set_robot_state_gravity_compensation.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/gazebo/04-set_robot_state_gravity_compensation.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
```

(continues on next page)

(continued from previous page)

```
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
#include <orca/gazebo/GazeboServer.h>
#include <orca/gazebo/GazeboModel.h>

using namespace orca::all;
using namespace orca::gazebo;

int main(int argc, char** argv)
{
    // Get the urdf file from the command line
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    // Instanciate the gazebo server with de dedfault empty world
    GazeboServer gz_server(argc,argv);
    // This is equivalent to GazeboServer gz("worlds/empty.world")
    // Insert a model onto the server and create the GazeboModel from the return value
    // You can also set the initial pose, and override the name in the URDF
    auto gz_model = GazeboModel(gz_server.insertModelFromURDFFile(urdf_url));

    // Create an ORCA robot
    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->print();

    // Set the gazebo model init pose
    // auto joint_names = robot_model->getJointNames();
    // std::vector<double> init_joint_positions(robot_model->
↪getNrOfDegreesOfFreedom(),0);

    // gz_model.setModelConfiguration(joint_names,init_joint_positions);
    // or like this
    // gz_model.setModelConfiguration({"joint_2","joint_5"},{1.5,0.0});

    // Update the robot on at every iteration
    gz_model.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,double␣
↪dt)
```

(continues on next page)

(continued from previous page)

```
    {
        robot_model->setRobotState(gz_model.getWorldToBaseTransform().matrix()
                                    ,gz_model.getJointPositions()
                                    ,gz_model.getBaseVelocity()
                                    ,gz_model.getJointVelocities()
                                    ,gz_model.getGravity()
                                );
        gz_model.setJointGravityTorques(robot_model->getJointGravityTorques());
    });

    // Run the main simulation loop.
    // This is a blocking call that runs the simulation steps
    // It can be stopped by CTRL+C
    // You can optionally add a callback that happends after WorldUpdateEnd
    std::cout << "Simulation running... (GUI with \'gzclient\')" << "\n";
    gz_server.run();
    return 0;
}
```

Using Gazebo to simulate an ORCA controller

Note: The source code for this example can be found in `[orca_root]/examples/gazebo/05-orca_gazebo.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/gazebo/05-orca_gazebo.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
```

(continues on next page)

(continued from previous page)

```
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
#include <orca/gazebo/GazeboServer.h>
#include <orca/gazebo/GazeboModel.h>

using namespace orca::all;
using namespace orca::gazebo;



int main(int argc, char const *argv[])
{
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l debug/info/warning/error)" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    GazeboServer gz_server(argc,argv);
    auto gz_model = GazeboModel(gz_server.insertModelFromURDFFile(urdf_url));
    gz_model.setModelConfiguration( { "joint_0", "joint_3","joint_5"} , {1.0,-M_PI/2.,M_PI/2.});

    orca::utils::Logger::parseArgv(argc, argv);

    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->setBaseFrame("base_link");
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81));


    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolverImplType::qpOASES
    );


    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("servo_controller");
```

(continues on next page)

(continued from previous page)

```
    cart_acc_pid->pid()->setProportionalGain({1000, 1000, 1000, 10, 10, 10});
    cart_acc_pid->pid()->setDerivativeGain({100, 100, 100, 1, 1, 1});
    cart_acc_pid->setControlFrame("link_7");

    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    cart_task->setServoController(cart_acc_pid);

    const int ndof = robot_model->getNrOfDegreesOfFreedom();

    auto jnt_trq_cstr = controller.addConstraint<JointTorqueLimitConstraint>("JointTorqueLimit");
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);

    auto jnt_pos_cstr = controller.addConstraint<JointPositionLimitConstraint>("JointPositionLimit");

    auto jnt_vel_cstr = controller.addConstraint<JointVelocityLimitConstraint>("JointVelocityLimit");
    jnt_vel_cstr->setLimits(Eigen::VectorXd::Constant(ndof,-2.0),Eigen::VectorXd::Constant(ndof,2.0));


    // Lets decide that the robot is gravity compensated
    // So we need to remove G(q) from the solution
    controller.removeGravityTorquesFromSolution(true);
    gz_model.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,double dt)
    {
        robot_model->setRobotState(gz_model.getWorldToBaseTransform().matrix()
                            ,gz_model.getJointPositions()
                            ,gz_model.getBaseVelocity()
                            ,gz_model.getJointVelocities()
                            ,gz_model.getGravity()
                        );
        // Compensate the gravity at least
        gz_model.setJointGravityTorques(robot_model->getJointGravityTorques());
        // All tasks need the robot to be initialized during the activation phase
        if(n_iter == 1)
            controller.activateTasksAndConstraints();

        controller.update(current_time, dt);

        if(controller.solutionFound())
        {
            gz_model.setJointTorqueCommand( controller.getJointTorqueCommand() );
        }
        else
        {
            gz_model.setBrakes(true);
        }
    });

    std::cout << "Simulation running... (GUI with \'gzclient\')" << "\n";

    // If you want to pause the simulation before starting it uncomment these lines
```

(continues on next page)

(continued from previous page)

```
    // Note that to unlock it either open 'gzclient' and click on the play button
    // Or open a terminal and type 'gz world -p false'
    //
    std::cout << "Gazebo is paused, open gzclient to unpause it or type 'gz world -p␣
↪false' in a new terminal" << '\n';
    gazebo::event::Events::pause.Signal(true);

    gz_server.run();
    return 0;
}
```

Minimum jerk Cartesian trajectory following

Note: The source code for this example can be found in `[orca_root]/examples/gazebo/06-trajectory_following.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/gazebo/06-trajectory_following.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
```

(continues on next page)

(continued from previous page)

```cpp
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
#include <orca/gazebo/GazeboServer.h>
#include <orca/gazebo/GazeboModel.h>

using namespace orca::all;
using namespace orca::gazebo;

class MinJerkPositionTrajectory {
private:
    Eigen::Vector3d alpha_, sp_, ep_;
    double duration_ = 0.0;
    double start_time_ = 0.0;
    bool first_call_ = true;
    bool traj_finished_ = false;

public:
    MinJerkPositionTrajectory (double duration)
    : duration_(duration)
    {
    }

    bool isTrajectoryFinished(){return traj_finished_;}

    void resetTrajectory(const Eigen::Vector3d& start_position, const Eigen::Vector3d& end_position)
    {
        sp_ = start_position;
        ep_ = end_position;
        alpha_ = ep_ - sp_;
        first_call_ = true;
        traj_finished_ = false;
    }

    void getDesired(double current_time, Eigen::Vector3d& p, Eigen::Vector3d& v, Eigen::Vector3d& a)
    {
        if(first_call_)
        {
            start_time_ = current_time;
            first_call_ = false;
        }
        double tau = (current_time - start_time_) / duration_;
        if(tau >= 1.0)
        {
            p = ep_;
            v = Eigen::Vector3d::Zero();
            a = Eigen::Vector3d::Zero();

            traj_finished_ = true;
```

(continues on next page)

(continued from previous page)

```
            return;
        }
        p = sp_ + alpha_ * ( 10*pow(tau,3.0) - 15*pow(tau,4.0)  + 6*pow(tau,5.0)   );
        v = Eigen::Vector3d::Zero() + alpha_ * ( 30*pow(tau,2.0) - 60*pow(tau,3.0)  + 30*pow(tau,4.0)  );
        a = Eigen::Vector3d::Zero() + alpha_ * ( 60*pow(tau,1.0) - 180*pow(tau,2.0) + 120*pow(tau,3.0) );
    }
};


int main(int argc, char const *argv[])
{
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -l debug/info/warning/error)" << "\n";
        return -1;
    }
    std::string urdf_url(argv[1]);

    orca::utils::Logger::parseArgv(argc, argv);

    auto robot_model = std::make_shared<RobotModel>();
    robot_model->loadModelFromFile(urdf_url);
    robot_model->setBaseFrame("base_link");
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81));

    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolverImplType::qpOASES
    );

    const int ndof = robot_model->getNrOfDegreesOfFreedom();


    auto joint_pos_task = controller.addTask<JointAccelerationTask>("JointPosTask");

    // Eigen::VectorXd P(ndof);
    // P.setConstant(100);
    joint_pos_task->pid()->setProportionalGain(Eigen::VectorXd::Constant(ndof, 100));

    // Eigen::VectorXd I(ndof);
    // I.setConstant(1);
    joint_pos_task->pid()->setDerivativeGain(Eigen::VectorXd::Constant(ndof, 1));

    // Eigen::VectorXd windupLimit(ndof);
    // windupLimit.setConstant(10);
    joint_pos_task->pid()->setWindupLimit(Eigen::VectorXd::Constant(ndof, 10));

    // Eigen::VectorXd D(ndof);
    // D.setConstant(10);
    joint_pos_task->pid()->setDerivativeGain(Eigen::VectorXd::Constant(ndof, 10));

    joint_pos_task->setWeight(1.e-6);
```

(continues on next page)

(continued from previous page)

```
    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("CartTask_EE-servo_controller");
    Vector6d P;
    P << 1000, 1000, 1000, 10, 10, 10;
    cart_acc_pid->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_acc_pid->pid()->setDerivativeGain(D);
    cart_acc_pid->setControlFrame("link_7");

    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    cart_task->setServoController(cart_acc_pid);



    auto jnt_trq_cstr = controller.addConstraint<JointTorqueLimitConstraint>("JointTorqueLimit");
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax);

    auto jnt_pos_cstr = controller.addConstraint<JointPositionLimitConstraint>("JointPositionLimit");

    auto jnt_vel_cstr = controller.addConstraint<JointVelocityLimitConstraint>("JointVelocityLimit");
    Eigen::VectorXd jntVelMax(ndof);
    jntVelMax.setConstant(2.0);
    jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);

    GazeboServer gzserver(argc,argv);
    auto gz_model = GazeboModel(gzserver.insertModelFromURDFFile(urdf_url));
    gz_model.setModelConfiguration( { "joint_0", "joint_3","joint_5"} , {1.0,-M_PI/2.,M_PI/2.});

    ///////////////////////////////////////
    ///////////////////////////////////////
    ///////////////////////////////////////
    ///////////////////////////////////////

    MinJerkPositionTrajectory traj(5.0);
    int traj_loops = 0;
    Eigen::Vector3d start_position, end_position;
    Eigen::VectorXd controller_torques(ndof);
    Eigen::Affine3d desired_cartesian_pose;
    Vector6d desired_cartesian_vel = Vector6d::Zero();
    Vector6d desired_cartesian_acc = Vector6d::Zero();

    cart_task->onActivationCallback([](){
        std::cout << "Activating CartesianTask..." << '\n';
    });

    cart_task->onActivatedCallback([&](){
        desired_cartesian_pose = cart_acc_pid->getCurrentCartesianPose();
        Eigen::Quaterniond quat = orca::math::quatFromRPY(M_PI,0,0); // make it point to the table
```

(continues on next page)

(continued from previous page)

```
        desired_cartesian_pose.linear() = quat.toRotationMatrix();

        start_position = desired_cartesian_pose.translation();
        end_position = start_position + Eigen::Vector3d(0,-0.35,-.3);
        traj.resetTrajectory(start_position, end_position);
    });

    cart_task->onComputeBeginCallback([&](double current_time, double dt){
        if (cart_task->getState() == TaskBase::State::Activated)
        {
            Eigen::Vector3d p, v, a;
            traj.getDesired(current_time, p, v, a);

            desired_cartesian_pose.translation() = p;
            desired_cartesian_vel.head(3) = v;
            desired_cartesian_acc.head(3) = a;

            cart_acc_pid->setDesired(desired_cartesian_pose.matrix(),desired_
↪cartesian_vel,desired_cartesian_acc);
        }
    });

    cart_task->onComputeEndCallback([&](double current_time, double dt){
        if (cart_task->getState() == TaskBase::State::Activated)
        {
            if (traj.isTrajectoryFinished()  )
            {
                if (traj_loops < 10)
                {
                    // flip start and end positions.
                    auto ep = end_position;
                    end_position = start_position;
                    start_position = ep;
                    traj.resetTrajectory(start_position, end_position);
                    std::cout << "Changing trajectory direction. [" << traj_loops <<
↪" of 10]" << '\n';
                    ++traj_loops;
                }
                else
                {
                    std::cout << "Trajectory looping finished. Deactivating task and␣
↪starting gravity compensation." << '\n';
                    cart_task->deactivate();
                }
            }
        }
    });

    cart_task->onDeactivationCallback([&](){
        std::cout << "Deactivating task." << '\n';
        std::cout << "\n\n\n" << '\n';
        std::cout << "Last controller_torques:\n" << controller_torques << '\n';
    });

    cart_task->onDeactivatedCallback([&](){
        std::cout << "CartesianTask deactivated." << '\n';
    });
```

(continues on next page)

(continued from previous page)

```


    // Lets decide that the robot is gravity compensated
    // So we need to remove G(q) from the solution
    controller.removeGravityTorquesFromSolution(true);
    gz_model.executeAfterWorldUpdate([&](uint32_t n_iter,double current_time,double dt)
    {
        robot_model->setRobotState(gz_model.getWorldToBaseTransform().matrix()
                            ,gz_model.getJointPositions()
                            ,gz_model.getBaseVelocity()
                            ,gz_model.getJointVelocities()
                            ,gz_model.getGravity()
                        );
        gz_model.setJointGravityTorques(robot_model->getJointGravityTorques());
        // All tasks need the robot to be initialized during the activation phase
        if(n_iter == 1)
            controller.activateTasksAndConstraints();

        controller.update(current_time, dt);

        if(controller.solutionFound())
        {
            controller_torques = controller.getJointTorqueCommand();
            gz_model.setJointTorqueCommand( controller_torques );
        }
        else
        {
            gz_model.setBrakes(true);
        }
    });

    std::cout << "Simulation running... (GUI with \'gzclient\')" << '\n';
    // If you want to pause the simulation before starting it uncomment these lines
    // Note that to unlock it either open 'gzclient' and click on the play button
    // Or open a terminal and type 'gz world -p false'
    //
    std::cout << "Gazebo is paused, open gzclient to unpause it or type 'gz world -p false' in a new terminal" << '\n';
    gazebo::event::Events::pause.Signal(true);

    gzserver.run();
    return 0;
}
```

1.1.10 Plotting

Using the internal plotting tools

Note: The source code for this example can be found in `[orca_root]/examples/plotting/01-plotting_torques.cc`, or alternatively on github at: https://github.com/syroco/orca/blob/dev/examples/plotting/01-plotting_torques.cc

Full Code Listing

```
// This file is a part of the ORCA framework.
// Copyright 2017, ISIR / Universite Pierre et Marie Curie (UPMC)
// Copyright 2018, Fuzzy Logic Robotics
// Main contributor(s): Antoine Hoarau, Ryan Lober, and
// Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
//
// ORCA is a whole-body reactive controller framework for robotics.
//
// This software is governed by the CeCILL-C license under French law and
// abiding by the rules of distribution of free software.  You can  use,
// modify and/ or redistribute the software under the terms of the CeCILL-C
// license as circulated by CEA, CNRS and INRIA at the following URL
// "http://www.cecill.info".
//
// As a counterpart to the access to the source code and  rights to copy,
// modify and redistribute granted by the license, users are provided only
// with a limited warranty  and the software's author,  the holder of the
// economic rights,  and the successive licensors  have only  limited
// liability.
//
// In this respect, the user's attention is drawn to the risks associated
// with loading,  using,  modifying and/or developing or reproducing the
// software by the user in light of its specific status of free software,
// that may mean  that it is complicated to manipulate,  and  that  also
// therefore means  that it is reserved for developers  and  experienced
// professionals having in-depth computer knowledge. Users are therefore
// encouraged to load and test the software's suitability as regards their
// requirements in conditions enabling the security of their systems and/or
// data to be ensured and,  more generally, to use and operate it in the
// same conditions as regards security.
//
// The fact that you are presently reading this means that you have had
// knowledge of the CeCILL-C license and that you accept its terms.

/** @file
 @copyright 2018 Fuzzy Logic Robotics <info@fuzzylogicrobotics.com>
 @author Antoine Hoarau
 @author Ryan Lober
*/

#include <orca/orca.h>
#include <matplotlibcpp/matplotlibcpp.h>
using namespace orca::all;

namespace plt = matplotlibcpp;

int main(int argc, char const *argv[])
{
    // Get the urdf file from the command line
    if(argc < 2)
    {
        std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf (optionally -
↪l debug/info/warning/error)" << "\n";
        return -1;
    }
```

(continues on next page)

(continued from previous page)

```
    std::string urdf_url(argv[1]);

    // Parse logger level as --log_level (or -l) debug/warning etc
    orca::utils::Logger::parseArgv(argc, argv);

    // Create the kinematic model that is shared by everybody
    auto robot_model = std::make_shared<RobotModel>(); // Here you can pass a robot name
    robot_model->loadModelFromFile(urdf_url); // If you don't pass a robot name, it is extracted from the urdf
    robot_model->setBaseFrame("base_link"); // All the transformations (end effector pose for example) will be expressed wrt this base frame
    robot_model->setGravity(Eigen::Vector3d(0,0,-9.81)); // Sets the world gravity (Optional)

    // This is an helper function to store the whole state of the robot as eigen vectors/matrices
    // This class is totally optional, it is just meant to keep consistency for the sizes of all the vectors/matrices
    // You can use it to fill data from either real robot and simulated robot
    RobotState eigState;
    eigState.resize(robot_model->getNrOfDegreesOfFreedom()); // resize all the vectors/matrices to match the robot configuration
    // Set the initial state to zero (arbitrary)
    // NOTE : here we only set q,qot because this example asserts we have a fixed base robot
    eigState.jointPos.setZero();
    eigState.jointVel.setZero();
    // Set the first state to the robot
    robot_model->setRobotState(eigState.jointPos,eigState.jointVel); // Now is the robot is considered 'initialized'

    // Instanciate an ORCA Controller
    orca::optim::Controller controller(
        "controller"
        ,robot_model
        ,orca::optim::ResolutionStrategy::OneLevelWeighted // MultiLevelWeighted, Generalized
        ,QPSolverImplType::qpOASES
    );

    auto cart_acc_pid = std::make_shared<CartesianAccelerationPID>("servo_controller");
    Vector6d P;
    P << 1000, 1000, 1000, 10, 10, 10;
    cart_acc_pid->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_acc_pid->pid()->setDerivativeGain(D);
    cart_acc_pid->setControlFrame("link_7");
    Eigen::Affine3d cart_pos_ref;
    cart_pos_ref.translation() = Eigen::Vector3d(0.3,-0.5,0.41); // x,y,z in meters
    cart_pos_ref.linear() = orca::math::quatFromRPY(M_PI,0,0).toRotationMatrix();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();
    cart_acc_pid->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

```

(continues on next page)

(continued from previous page)

```
    auto cart_task = controller.addTask<CartesianTask>("CartTask_EE");
    cart_task->setServoController(cart_acc_pid);

    // Get the number of actuated joints
    const int ndof = robot_model->getNrOfDegreesOfFreedom();

    // Joint torque limit is usually given by the robot manufacturer
    auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit
↪");
    controller.addConstraint(jnt_trq_cstr); // Add the constraint to the controller␣
↪to initialize it
    Eigen::VectorXd jntTrqMax(ndof);
    jntTrqMax.setConstant(200.0);
    jnt_trq_cstr->setLimits(-jntTrqMax,jntTrqMax); // because not read in the URDF␣
↪for now

    // Joint position limits are automatically extracted from the URDF model
    // Note that you can set them if you want
    // by simply doing jnt_pos_cstr->setLimits(jntPosMin,jntPosMax);
    auto jnt_pos_cstr = std::make_shared<JointPositionLimitConstraint>(
↪"JointPositionLimit");
    controller.addConstraint(jnt_pos_cstr); // Add the constraint to the controller␣
↪to initialize it

    // Joint velocity limits are usually given by the robot manufacturer
    auto jnt_vel_cstr = std::make_shared<JointVelocityLimitConstraint>(
↪"JointVelocityLimit");
    controller.addConstraint(jnt_vel_cstr); // Add the constraint to the controller␣
↪to initialize it
    Eigen::VectorXd jntVelMax(ndof);
    jntVelMax.setConstant(2.0);
    jnt_vel_cstr->setLimits(-jntVelMax,jntVelMax);  // because not read in the URDF␣
↪for now

    double dt = 0.001;
    double total_time = 1.0;
    double current_time = 0;

    // Shortcut : activate all tasks
    controller.activateTasksAndConstraints();

    // Now you can run the control loop
    std::vector<double> time_log;
    int ncols = std::ceil(total_time/dt);
    Eigen::MatrixXd torqueMat(ndof,ncols);
    torqueMat.setZero();

    for (int count = 0; current_time < total_time; current_time +=dt)
    {
        time_log.push_back(current_time);

        // Here you can get the data from you REAL robot (API might vary)
        // Some thing like :
        //      eigState.jointPos = myRealRobot.getJointPositions();
        //      eigState.jointVel = myRealRobot.getJointVelocities();

        // Now update the internal kinematic model with data from REAL robot
```

(continues on next page)

(continued from previous page)

```
        robot_model->setRobotState(eigState.jointPos,eigState.jointVel);

        // Step the controller
        if(controller.update(current_time,dt))
        {

            // Get the controller output
            const Eigen::VectorXd& full_solution = controller.getSolution();

            torqueMat.col(count) = controller.getJointTorqueCommand();

            const Eigen::VectorXd& trq_acc = controller.getJointAccelerationCommand();

            // Here you can send the commands to you REAL robot
            // Something like :
            // myRealRobot.setTorqueCommand(trq_cmd);
        }
        else
        {
            // Controller could not get the optimal torque
            // Now you have to save your robot
            // You can get the return code with controller.getReturnCode();
        }

        count++;

        std::cout << "current_time  " << current_time << '\n';
        std::cout << "total_time  " << total_time << '\n';
        std::cout << "time log size  " << time_log.size() << '\n';
        std::cout << "torqueMat.cols " << torqueMat.cols() << '\n';
    }

    // Print the last computed solution (just for fun)
    const Eigen::VectorXd& full_solution = controller.getSolution();
    const Eigen::VectorXd& trq_cmd = controller.getJointTorqueCommand();
    const Eigen::VectorXd& trq_acc = controller.getJointAccelerationCommand();
    LOG_INFO << "Full solution : " << full_solution.transpose();
    LOG_INFO << "Joint Acceleration command : " << trq_acc.transpose();
    LOG_INFO << "Joint Torque command       : " << trq_cmd.transpose();

    // At some point you want to close the controller nicely
    controller.deactivateTasksAndConstraints();
    // Let all the tasks ramp down to zero
    while(!controller.tasksAndConstraintsDeactivated())
    {
        current_time += dt;
        controller.print();
        controller.update(current_time,dt);
    }

    // Plot data
    for (size_t i = 0; i < torqueMat.rows(); i++)
    {
        std::vector<double> trq(time_log.size());
        Eigen::VectorXd::Map(trq.data(),time_log.size()) = torqueMat.row(i);
        plt::plot(time_log,trq);
    }
```

(continues on next page)

(continued from previous page)

```
    plt::show();
    return 0;
}
```

1.1.11 Overview

The most generic representation of the whole-body controller used in ORCA can be summarized by the following optimization problem,

$$\begin{aligned} \underset{\chi}{\arg\min} \quad & f^{\text{task}}(\chi) \\ \text{s.t.} \quad & G\chi \leq \boldsymbol{h} \\ & A\chi = \boldsymbol{b}. \end{aligned} \tag{1.1}$$

- s.t.: subject to

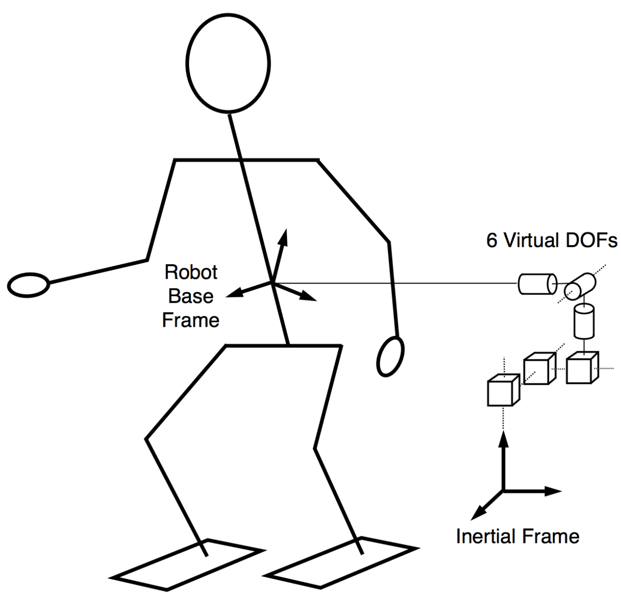
The objective, $f^{\text{task}}(\chi)$, is a function of the optimization variable, χ, and is determined by control objectives, or tasks. The resolution of the objective is subject to (s.t.) the affine inequality and equality constraints, which ensure that the control constraints are respected.

To understand how whole-body controllers are formulated in ORCA, we begin with a brief description of the free-floating rigid body dynamics. The parameterization of the dynamics forms the optimization variable. The control objectives, or tasks, and constraints are then detailed and written in terms of the optimization variable. Finally, task prioritization schemes are discussed.

1.1.12 Dynamics

Free-Floating Rigid Body Dynamics

For robots whose root link can float freely in Cartesian space, e.g. humanoids, it is necessary to consider the pose of the root link with respect to (wrt) the inertial reference frame. The primary method for doing so is to account for the root link pose directly in the generalized coordinates, $\boldsymbol{q}$, of the robot as shown by:



Todo: add citations

The generalized configuration parameterization for floating base robots,

$$\boldsymbol{q} = \begin{Bmatrix} \boldsymbol{\xi}_{fb} \\ \boldsymbol{q}_j \end{Bmatrix}, \tag{1.2}$$

therefore contains the pose of the base link wrtthe inertial reference frame, $\boldsymbol{\xi}_{fb}$, and the joint space coordinates, $\boldsymbol{q}_j$. Set brackets are used in (1.2) because $\boldsymbol{\xi}_{fb}$ is a homogeneous transformation matrix in $\mathbb{R}^{4\times4}$ and $\boldsymbol{q}_j$ is a vector in $\mathbb{R}^n$, with n the number of dofof the robot, thus $\boldsymbol{\xi}_{fb}$ and $\boldsymbol{q}_j$ cannot be concatenated into a vector. However, the twist of the base, $\boldsymbol{v}_{fb}$, with the joint velocities, $\dot{\boldsymbol{q}}_j$, can be concatenated in vector notation, along with the base and joint accelerations to obtain,

$$\boldsymbol{\nu} = \begin{bmatrix} \boldsymbol{v}_{fb} \\ \dot{\boldsymbol{q}}_j \end{bmatrix}, \quad \text{and} \quad \dot{\boldsymbol{\nu}} = \begin{bmatrix} \dot{\boldsymbol{v}}_{fb} \\ \ddot{\boldsymbol{q}}_j \end{bmatrix}. \tag{1.3}$$

These representations provide a complete description of the robot's state and its rate of change, and allow the equations of motion to be written as,

$$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \underbrace{C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{g}(\boldsymbol{q})}_{\boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu})} = S^\top \boldsymbol{\tau} + {}^eJ^\top(\boldsymbol{q})^e\boldsymbol{\omega}. \tag{1.4}$$

In (1.4), $M(\boldsymbol{q})$ is the generalized mass matrix, $C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}$ and $\boldsymbol{g}(\boldsymbol{q})$ are the Coriolis-centrifugal and gravitational terms, S is a selection matrix indicating the actuated degrees of freedom, ${}^e\boldsymbol{\omega}$ is the concatenation of the external contact wrenches, and ${}^eJ$ their concatenated Jacobians.

Grouping $C(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}$ and $\boldsymbol{g}(\boldsymbol{q})$ together into $\boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu})$, the equations can by simplified to

$$M(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu}) = S^\top \boldsymbol{\tau} + {}^eJ^\top(\boldsymbol{q})^e\boldsymbol{\omega}. \tag{1.5}$$

The joint torques induced by friction force could also be included in (1.5), but are left out for the sake of simplicity. Additionally, the variables $\dot{\boldsymbol{\nu}}$, τ, and ${}^e\boldsymbol{\omega}$, can be grouped into the same vector,

$$\boldsymbol{\chi} = \begin{bmatrix} \dot{\boldsymbol{\nu}} \\ \boldsymbol{\tau} \\ {}^e\boldsymbol{\omega} \end{bmatrix}, \tag{1.6}$$

forming the optimization variable from (1.1), and allowing (1.5) to be rewritten as,

$$\begin{bmatrix} -M(\boldsymbol{q}) & S^\top & {}^eJ^\top(\boldsymbol{q}) \end{bmatrix} \boldsymbol{\chi} = \boldsymbol{n}(\boldsymbol{q},\boldsymbol{\nu}). \tag{1.7}$$

Equation (1.7) provides an equality constraint which can be used to ensure that the minimization of the control objectives respects the system dynamics.

1.1.13 Optimization

Optimization Vector

In *Free-Floating Rigid Body Dynamics* we expressed the equations of motion as an affine function of our optimization variable, $\boldsymbol{\chi}$. Here, we look at each component in $\boldsymbol{\chi}$ and detail its meaning, position in the overall vector, and dimensions.

$$\boldsymbol{\chi} = \begin{bmatrix} \dot{\boldsymbol{\nu}}_{fb} \\ \dot{\boldsymbol{\nu}}_j \\ \boldsymbol{\tau}_{fb} \\ \boldsymbol{\tau}_j \\ {}^e\boldsymbol{\omega}_0 \\ \vdots \\ {}^e\boldsymbol{\omega}_n \end{bmatrix}$$

- $\dot{\boldsymbol{\nu}}_{fb}$: Floating base joint acceleration (6×1)
- $\dot{\boldsymbol{\nu}}_{j}$: Joint space acceleration $(n_{DoF} \times 1)$
- $\boldsymbol{\tau}_{fb}$: Floating base joint torque (6×1)
- $\boldsymbol{\tau}_{j}$: Joint space joint torque $(n_{DoF} \times 1)$
- ${}^{e}\boldsymbol{\omega}_{n}$: External wrench (6×1)

Each of these variables are termed **Control Variables** in ORCA and are used to define every task and constraint.

These variables can of course be combined for convenience:

- $\dot{\boldsymbol{\nu}}$: Generalised joint acceleration, concatenation of $\dot{\boldsymbol{\nu}}_{fb}$ and $\dot{\boldsymbol{\nu}}_{j}$ $(6 + n_{DoF} \times 1)$
- $\boldsymbol{\tau}$: Generalised joint torque, concatenation of $\boldsymbol{\tau}_{fb}$ and $\boldsymbol{\tau}_{j}$ $(6 + n_{DoF} \times 1)$
- ${}^{e}\boldsymbol{\omega}$: External wrenches $(n_{\text{wrenches}}6 \times 1)$
- $\boldsymbol{\chi}$: The whole optimization vector $(6 + n_{DoF} + 6 + n_{DoF} + n_{wrenches}6 \times 1)$

With our optimization varible well defined, we can now formulate the optimization problem.

The Optimization Problem

Returning to our generic representation of a whole-body controller presented in *Overview*,

$$\begin{aligned} \underset{\boldsymbol{\chi}}{\arg\min} \quad & f^{\text{task}}(\boldsymbol{\chi}) \\ \text{s.t.} \quad & G\boldsymbol{\chi} \leq \boldsymbol{h} \\ & A\boldsymbol{\chi} = \boldsymbol{b}, \end{aligned} \tag{1.8}$$

we make some important assumptions about the structure of the problem. Firstly, we make the assumtion that our control problem is continous and has size = n, i.e. $\boldsymbol{\chi} \in \mathbb{R}^n$. Next we impose that $f^{\text{task}}(\boldsymbol{\chi})$ be quadratic in $\boldsymbol{\chi}$, leaving us with an unconstrained **Quadratic Program**, or QP:

$$\begin{aligned} \underset{\boldsymbol{\chi}}{\arg\min} \quad f(\boldsymbol{\chi}) &= \frac{1}{2}\boldsymbol{\chi}^\top H\boldsymbol{\chi} + \boldsymbol{g}^\top\boldsymbol{\chi} + r \\ &= \boldsymbol{\chi}^\top(E^\top E)\boldsymbol{\chi} - 2(E^\top \mathbf{f})^\top\boldsymbol{\chi} + \mathbf{f}^\top\mathbf{f} \\ &= \|E\boldsymbol{\chi} - \mathbf{f}\|_2^2, \end{aligned} \tag{1.9}$$

In (1.9), the first line is the classical formulation of a QP:

- $\boldsymbol{\chi}$ the optimization vector
- H the hessian matrix $(n \times n)$
- $\boldsymbol{g}$ the gradient vector $(n \times 1)$
- E the linear matrix of the affine function $(n \times n)$
- f the origin vector $(n \times 1)$

The last line of (1.9), $\|E\boldsymbol{\chi} - \mathbf{f}\|_2^2$, is the least-squares formulation. We will continue using the least squares version, which admits an analytical minimum-norm solution, $\boldsymbol{\chi}^*$, in the unconstrained case.

$$\boldsymbol{\chi}^* = \underset{\boldsymbol{\chi}}{\arg\min} \ \|E\boldsymbol{\chi} - \mathbf{f}\|_2^2 = E^\dagger\mathbf{f}, \tag{1.10}$$

where $E^\dagger$ is the Moore-Penrose pseudoinverse of the E matrix. This solution will be found assuming the rank of the linear system is consistent.

Adding an affine equality constraint produces a constrained least squares problem,

$$\begin{aligned} \arg\min_{\chi} \quad & \|E\chi - \mathbf{f}\|_2^2 \\ \text{s.t.} \quad & A\chi = \boldsymbol{b}, \end{aligned} \tag{1.11}$$

which can be solved analytically, assuming a solution exists, using the **Karush Kuhn Tucker (KKT) equations**,

$$\begin{aligned} & \underbrace{\begin{bmatrix} E^\top E & A^\top \\ A & \mathbf{0} \end{bmatrix}}_{\text{KKT Matrix}} \begin{bmatrix} \chi \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} E^\top \mathbf{f} \\ \boldsymbol{b} \end{bmatrix} \\ \Leftrightarrow & \begin{bmatrix} \chi \\ \boldsymbol{z} \end{bmatrix} = \begin{bmatrix} E^\top E & A^\top \\ A & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} E^\top \mathbf{f} \\ \boldsymbol{b} \end{bmatrix}, \end{aligned} \tag{1.12}$$

where $\boldsymbol{z}$ is the solution to the dual problem and contains the **Lagrange multipliers**.

Adding an affine inequality constraint to the problem produces the following QP,

$$\begin{aligned} \arg\min_{\chi} \quad & \|E\chi - \mathbf{f}\|_2^2 \\ \text{s.t.} \quad & A\chi = \boldsymbol{b} \\ & G\chi \leq \boldsymbol{h}. \end{aligned} \tag{1.13}$$

Equation (1.13) can no longer be solved analytically and one must use numerical methods such as interior point, or active set methods.

Note: For more details on convex optimization, check out Boyd and Vandenberghe's book: http://web.stanford.edu/~boyd/cvxbook/

Resolution of (1.13) with a numerical solver, such as `qpOASES`, will provide a globally optimal solution for χ^* provided that the constraint equations are consistent, i.e. the set of possible solutions is not empty.

Objective Function Implementation

Within ORCA the QP objective function is formulated as a weighted Euclidean norm of an affine function,

$$\|E\chi - \mathbf{f}\|_W^2 \Leftrightarrow \left\| \sqrt{W} \left(E\chi - \mathbf{f} \right) \right\|^2 \tag{1.14}$$

In (1.14), W is the weight of the euclidean norm $(n \times n)$ and must be a positive symmetric definite matrix.

In ORCA, W is actually composed of two components, the norm weighting W' and the selection matrix, S,

$$W = SW' \tag{1.15}$$

S is a matrix with either 1's or 0's on the diagonal which allows us to ignore all or parts of the affine function we are computing. Concretely this means we can ignore components of the task error. More information on tasks is provided in the *Control Objectives (Tasks)* section.

For example…

For a Cartesian position task, setting the low 3 entries on the diagonal of S to 0 allows us to ignore orientation errors.

For practicality's sake we set S from a vector with the function `setSelectionVector(const Eigen::VectorXd& s)`, which creates a diagonal matrix from `s`.

Given W from (1.15), the hessian and gradient are calculated as,

$$\frac{1}{2}\boldsymbol{\chi}^\top H\boldsymbol{\chi} + \boldsymbol{g}^\top\boldsymbol{\chi}$$
$$\Leftrightarrow \boldsymbol{\chi}^\top(E^\top W E)\boldsymbol{\chi} - 2(WE^\top\mathbf{f})^\top\boldsymbol{\chi}$$

Note: $r = \mathbf{f}^\top\mathbf{f}$ is dropped from the objective function because it does not change the optimal solution of the QP.

In the code, these calculations can be found in `WeightedEuclidianNormFunction`:

```
void WeightedEuclidianNormFunction::QuadraticCost::computeHessian(const␣
↪Eigen::VectorXd& SelectionVector
                                                            , const Eigen::MatrixXd& Weight
                                                            , const Eigen::MatrixXd& A)
{
    Hessian_.noalias() = SelectionVector.asDiagonal() * Weight * A.transpose() * A ;
}

void WeightedEuclidianNormFunction::QuadraticCost::computeGradient(const␣
↪Eigen::VectorXd& SelectionVector
                                                            , const Eigen::MatrixXd& Weight
                                                            , const Eigen::MatrixXd& A
                                                            , const Eigen::VectorXd& b)
{
    Gradient_.noalias() =  2.0 * SelectionVector.asDiagonal() * Weight * A.
↪transpose() * b ;
}
```

Constraint Implementation

Constraints are written as double bounded linear functions,

$$\boldsymbol{lb} \leq C\boldsymbol{\chi} \leq \boldsymbol{ub}.$$

- C the constraint matrix ($n \times n$)
- $\boldsymbol{lb}$ and $\boldsymbol{ub}$ the lower and upper bounds of $C\boldsymbol{\chi}$ ($n \times 1$)

Thus to convert our standard affine constraint forms we have the following relationships:

$$A\boldsymbol{\chi} = \boldsymbol{b} \Leftrightarrow \boldsymbol{b} \leq A\boldsymbol{\chi} \leq \boldsymbol{b}$$

$$G\boldsymbol{\chi} \leq \boldsymbol{h} \Leftrightarrow \begin{bmatrix} G\boldsymbol{\chi} \\ -G\boldsymbol{\chi} \end{bmatrix} \leq \begin{bmatrix} \boldsymbol{ub_h} \\ -\boldsymbol{lb_h} \end{bmatrix} \Leftrightarrow \boldsymbol{lb_h} \leq G\boldsymbol{\chi} \leq \boldsymbol{ub_h}$$

ORCA QP

In ORCA the full QP is expressed as,

$$\begin{aligned} \underset{\boldsymbol{\chi}}{\arg\min} \quad & \frac{1}{2}\boldsymbol{\chi}^\top H\boldsymbol{\chi} + \boldsymbol{g}^\top\boldsymbol{\chi} \\ \text{s.t.} \quad & \boldsymbol{lb} \leq \boldsymbol{\chi} \leq \boldsymbol{ub} \\ & \boldsymbol{lb} \leq C\boldsymbol{\chi} \leq \boldsymbol{ub}, \end{aligned}$$

Note: For convenience an explicit constraint on the optimization variable $\boldsymbol{\chi}$ is included in the problem because it is so common. This constraint is identical to the second line: $\boldsymbol{lb} \leq C\boldsymbol{\chi} \leq \boldsymbol{ub}$ when C is the identity matrix.

In the next sections we show how to formulate the different task and constraint types one might need to control a robot. In section *Multi-Objective Optimization*, we show how to combine multiple objective functions (tasks) in one controller allowing us to exploit the redundancy of the system.

Note: Multiple constraints can be combined through vertical concatenation of their matrices and vectors. I.e.

$$\begin{bmatrix} \boldsymbol{lb}_1 \\ \boldsymbol{lb}_2 \\ \vdots \\ \boldsymbol{lb}_{n_C} \end{bmatrix} \leq \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_{n_C} \end{bmatrix} \boldsymbol{\chi} \leq \begin{bmatrix} \boldsymbol{ub}_1 \\ \boldsymbol{ub}_2 \\ \vdots \\ \boldsymbol{ub}_{n_C} \end{bmatrix}$$

1.1.14 Tasks

Control Objectives (Tasks)

The basic problem of control is to drive a system from some initial state to some desired state. The control of robots is no different, but the term state takes on greater ambiguity. For simple systems, such as the double integrator, linearized inverted pendulum, etc., state-space control is sufficient for virtually any high-level objective one could envision for the system. However, for a robot, describing the control problem solely in terms of its state, i.e. $\boldsymbol{q}$ and $\boldsymbol{\nu}$, is limiting and one may also want to describe it in terms of the pose and twist of an end-effector, or possibly even a wrench on some link (although not technically a state in the classical control sense). Far from being a detriment, this variability is what makes robots so useful but requires a bit of abstraction from classical state-space control vocabulary. For this reason, the term **task** is commonly used to indicate a control objective for a robot. Tasks, in second-order controllers, can be driven by desired accelerations, wrenches, or torques, and in operational-space or joint-space. They are expressed in the whole-body controller as functions of the errors between the desired and current values of the task. In this work, the square of the l^2-norm is used to create a quadratic objective function. Consequently, the task errors are expressed in the least-squares formulation.

Cartesian Acceleration Task

Probably the most important, if not most prevalent, task is to move a link on the robot from one pose to another. Typically it is the end-effector(s) which are of interest. These tasks, which are generally expressed as desired positions or orientations, are converted to **acceleration tasks**, through means of task servoing. More details on task servoing are provided in *Task Servoing*. Once given a desired operational-space acceleration for a link, $\ddot{\boldsymbol{\xi}}_i^{\text{des}}$, an acceleration task consists in finding the joint-space values which produce $\ddot{\boldsymbol{\xi}}_i^{\text{des}}$,

$$\ddot{\boldsymbol{\xi}}_i^{\text{des}} = J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{J}_i(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu}, \tag{1.16}$$

where $J_i(\boldsymbol{q})$ and $\dot{J}_i(\boldsymbol{q}, \boldsymbol{\nu})$ are the link Jacobian and its derivative. For the control objective, one simply rewrites the task as an error which must be minimized,

$$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \dot{J}_i(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu} - \ddot{\boldsymbol{\xi}}_i^{\text{des}} \right\|_2^2. \tag{1.17}$$

Using the squared l^2-norm produces a quadratic error term, which defines the objective function $f_i^{\ddot{\boldsymbol{\xi}}}$ to be minimized. The objective function $f_i^{\ddot{\boldsymbol{\xi}}}$ is then rewritten in terms of the optimization variable, $\boldsymbol{\chi}$,

$$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| \begin{bmatrix} J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} - \left(\ddot{\boldsymbol{\xi}}_i^{\text{des}} - \dot{J}_i(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu} \right) \right\|_2^2 . \tag{1.18}$$

In (1.18) the term $\mathbf{0}$ represents a matrix of zeros. Regrouping terms as,

$$E^{\ddot{\boldsymbol{\xi}}} = \begin{bmatrix} J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix} \tag{1.19}$$

$$\mathbf{f}^{\ddot{\boldsymbol{\xi}}} = \ddot{\boldsymbol{\xi}}_i^{\text{des}} - \dot{J}_i(\boldsymbol{q}, \boldsymbol{\nu})\boldsymbol{\nu}, \tag{1.20}$$

allows (1.18) to be written in the classical least-squares form as,

$$f_i^{\ddot{\boldsymbol{\xi}}} = \left\| E^{\ddot{\boldsymbol{\xi}}} \boldsymbol{\chi} - \mathbf{f}^{\ddot{\boldsymbol{\xi}}} \right\|_2^2 . \tag{1.21}$$

The dependencies of $E^{\ddot{\boldsymbol{\xi}}}$ and $\mathbf{f}^{\ddot{\boldsymbol{\xi}}}$ have been removed for brevity.

$$w_{task} . \| \mathbf{E}x + \mathbf{f} \|_{W_{norm}}$$

$$\underset{n\times 1}{\mathrm{Y}} = \underset{n\times p}{X} \times \underset{p\times 1}{\theta} + \underset{n\times 1}{\varepsilon}$$

Joint Acceleration Task

Acceleration tasks can be expressed in either joint-space or in operational-space. Here, the operational-space form is presented but the joint-space form can easily be produced as,

$$f_i^{\dot{\boldsymbol{\nu}}} = \left\| \dot{\boldsymbol{\nu}} - \dot{\boldsymbol{\nu}}_i^{\text{des}} \right\|_2^2 , \tag{1.22}$$

with

$$E^{\dot{\boldsymbol{\nu}}} = \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \tag{1.23}$$

$$\mathbf{f}^{\dot{\boldsymbol{\nu}}} = \dot{\boldsymbol{\nu}}_i^{\text{des}}, \tag{1.24}$$

where I is the identity matrix. Substituting (1.23) and (1.24) into (1.22) gives,

$$f_i^{\dot{\boldsymbol{\nu}}} = \left\| E^{\dot{\boldsymbol{\nu}}} \boldsymbol{\chi} - \mathbf{f}^{\dot{\boldsymbol{\nu}}} \right\|_2^2 . \tag{1.25}$$

Wrench Task

In order for robots to work properly in their environment, they must be able to interact with it. Not only does this allow the robot to manipulate and modify its environment, but it also allows the robot to exploit the environment to compensate for its underactuation and more generally to dynamically perform complex behaviors. Walking and balance are two pertinent examples of such behaviors because to achieve them, contact forces with the ground must be properly exploited. For details on this see…

Todo: add citations

In order to interact with the environment, **wrench tasks** can be formulated to manage the interaction forces and torques,

$${}^{e}\boldsymbol{\omega}_i = {}^{e}\boldsymbol{\omega}_i^{\text{des}} . \tag{1.26}$$

where ${}^{e}\boldsymbol{\omega}_i^{\text{des}}$ is the desired external wrench to affect, and ${}^{e}\boldsymbol{\omega}_i$ is the wrench applied on the environment. Again, to formulate a control objective function, $f_i^{\boldsymbol{\omega}}$, the task is rewritten as the squared norm of a task error,

$$f_i^{\boldsymbol{\omega}} = \left\| {}^{e}\boldsymbol{\omega}_i - {}^{e}\boldsymbol{\omega}_i^{\text{des}} \right\|_2^2 . \tag{1.27}$$

Rewriting (1.27) in terms of $\boldsymbol{\chi}$ gives,

$$f_i^{\boldsymbol{\omega}} = \left\| \begin{bmatrix} \mathbf{0} & S_i^{\boldsymbol{\omega}} \end{bmatrix} \boldsymbol{\chi} - {}^{e}\boldsymbol{\omega}_i^{\text{des}} \right\|_2^2 , \tag{1.28}$$

where $S_i^{\boldsymbol{\omega}}$ is a wrench selection matrix which allows the i^{th} wrench to be controlled. Using,

$$E^{\boldsymbol{\omega}} = \begin{bmatrix} \mathbf{0} & S_i^{\boldsymbol{\omega}} \end{bmatrix} \tag{1.29}$$

$$\mathbf{f}^{\boldsymbol{\omega}} = {}^{e}\boldsymbol{\omega}_i^{\text{des}}, \tag{1.30}$$

(1.28) can be written as,

$$f_i^{\boldsymbol{\omega}} = \left\| E^{\boldsymbol{\omega}} \boldsymbol{\chi} - \mathbf{f}^{\boldsymbol{\omega}} \right\|_2^2 . \tag{1.31}$$

Torque Task

Finally, it may also be desirable to specify **torque tasks** for purposes of regularization, among other possibilities. As with wrench tasks, torque tasks, can be written simply as,

$$\boldsymbol{\tau} = \boldsymbol{\tau}^{\text{des}}. \tag{1.32}$$

To formulate the control objective function, $f^{\boldsymbol{\tau}}$, the square norm of the task error is written,

$$f^{\boldsymbol{\tau}} = \left\| \boldsymbol{\tau} - \boldsymbol{\tau}^{\text{des}} \right\|_2^2 , \tag{1.33}$$

which can be reformulated in terms of $\boldsymbol{\chi}$ as,

$$f^{\boldsymbol{\tau}} = \left\| \begin{bmatrix} \mathbf{0} & S^{\top} & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} - \boldsymbol{\tau}^{\text{des}} \right\|_2^2 . \tag{1.34}$$

Once again regrouping terms,

$$E^{\boldsymbol{\tau}} = \begin{bmatrix} \mathbf{0} & S^{\top} & \mathbf{0} \end{bmatrix} \tag{1.35}$$

$$\mathbf{f}^{\boldsymbol{\tau}} = \boldsymbol{\tau}^{\text{des}}, \tag{1.36}$$

the least-squares form of the torque task is written,

$$f^{\boldsymbol{\tau}} = \left\| E^{\boldsymbol{\tau}} \boldsymbol{\chi} - \mathbf{f}^{\boldsymbol{\tau}} \right\|_2^2 . \tag{1.37}$$

Task Servoing

The desired terms, $\ddot{\boldsymbol{\xi}}_i^{\text{des}}$, $\dot{\boldsymbol{\nu}}_i^{\text{des}}$, ${}^{e}\boldsymbol{\omega}_i^{\text{des}}$, and $\boldsymbol{\tau}^{\text{des}}$, from (1.16), (1.22), (1.26), and (1.32), respectively are provided by higher-level task servoing. Commonly, the high-level reference of a task is simply to attain some pose, and in the case of a wrench task, some force and/or torque. For acceleration tasks, if the desired task value is expressed as a pose, position, or orientation, then it must be converted to an acceleration. This is done here using a feedforward (PD) controller,

$$\ddot{\boldsymbol{\xi}}_i^{\text{des}}(t + \Delta t) = \ddot{\boldsymbol{\xi}}_i^{\text{ref}}(t + \Delta t) + K_p \boldsymbol{\epsilon}_i(t) + K_d \dot{\boldsymbol{\epsilon}}_i(t), \tag{1.38}$$

noindent where $\ddot{\boldsymbol{\xi}}_i^{\text{ref}}(t+\Delta t)$ is the feedforward frame acceleration term, $\boldsymbol{\epsilon}_i(t)$ and $\dot{\boldsymbol{\epsilon}}_i(t)$ are the current pose error and its derivative, with K_p and $K_d = 2\sqrt{K_p}$, their proportional and derivative gains respectively. This term also serves to remove drift at the controller level and stabilize the output of the task. The terms, $\boldsymbol{\epsilon}_i(t)$ and $\dot{\boldsymbol{\epsilon}}_i(t)$, are not explicitly defined here because they are representation dependent (see citep{Siciliano2008}). For wrench and torque tasks a similar servoing controller can be developed using a Proportional-Integral (PI) controller.

$$\boldsymbol{\omega}^{des}(t+\Delta t) = \boldsymbol{\omega}^{ref}(t+\Delta t) + K_p \boldsymbol{\epsilon}_{\boldsymbol{\omega}}(t) + K_i \int \boldsymbol{\epsilon}_{\boldsymbol{\omega}}(t) dt \tag{1.39}$$

This servoing helps stabilize the whole-body controller by driving the desired task values to some fixed state in asymptotically stable manner. Without the servoing the the task error objective term, $f_i^{\text{task}}(\boldsymbol{\chi})$, could change discontinuously between time steps resulting in discontinuous jumps in the optimal joint torques determined between two time steps.

1.1.15 Constraints

Control Constraints

As with all real world control problems, there are limits to what the system being controlled can do. In this particular case, the main constraint is that of the system dynamics, i.e. the equations of motion. This means that any solution found must be dynamically feasible. Apart from this, the control input is typically bounded. For robots with revolute joints, this means that the torque which can be generated by the actuators is limited to plus or minus some value. Likewise, the joints themselves generally have limited operating ranges for various mechanical reasons. In addition to these common limiting factors, other phenomena such as unilateral and bilateral contacts can come into play.

Dynamics Constraints

The rigid body dynamics of the robot are governed by the equations of motion from equations_of_motion_in_optvar. This constraint ultimately dictates the achievable dynamics of the system, and is formulated as the following equality constraint,

$$\underbrace{\begin{bmatrix} -M(\boldsymbol{q}) & S^\top & {}^eJ^\top(\boldsymbol{q}) \end{bmatrix}}_{A^d} \boldsymbol{\chi} = \underbrace{\boldsymbol{n}(\boldsymbol{q}, \boldsymbol{\nu})}_{\boldsymbol{b}^d}. \tag{1.40}$$

The terms A^d and $\boldsymbol{b}^d$ are used to distinguish the equality constraint matrix and vector, respectively, for the dynamic constraints.

Important: To put this into ORCA standard form we have,

$$\boldsymbol{b}^d \leq A^d \boldsymbol{\chi} \leq \boldsymbol{b}^d$$

Actuator Limit Constraints

Here, we assume that all articulations are revolute and therefore all actuation limits are torque limits, however, expression of force limits for prismatic joints would be another possibility. Writing these limits as an inequality provides an upper and lower bound on the amount of torque which can be exerted to accomplish the tasks.

$$\boldsymbol{\tau}_{\text{min}} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{\text{max}}. \tag{1.41}$$

Expressing torque_limits in terms of $\boldsymbol{\chi}$ creates the following linear inequality,

$$\underbrace{\begin{bmatrix} \mathbf{0} & S^\top & \mathbf{0} \\ \mathbf{0} & -S^\top & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{\tau}}} \boldsymbol{\chi} \leq \underbrace{\begin{bmatrix} \boldsymbol{\tau}_{\text{max}} \\ -\boldsymbol{\tau}_{\text{min}} \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{\tau}}}. \tag{1.42}$$

Important: To put this into ORCA standard form we have,

$$\boldsymbol{\tau}_{\text{min}} \leq \begin{bmatrix} \mathbf{0} & S^{\top} & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} \leq \boldsymbol{\tau}_{\text{max}}$$

Joint Limit Constraints

Probably the most common limitation of any robot is the range of motion which each joint can achieve. Whether linear or angular, most joints have a finite range through which they can move thus limiting $\boldsymbol{q}$. These joint limits can easily be expressed as a inequality on $\boldsymbol{q}$,

$$\boldsymbol{q}_{\text{min}} \leq \boldsymbol{q} \leq \boldsymbol{q}_{\text{max}}. \tag{1.43}$$

Similarly to these position limits, we can also define limits on the joint velocities and accelerations,

$$\boldsymbol{\nu}_{\text{min}} \leq \boldsymbol{\nu} \leq \boldsymbol{\nu}_{\text{max}} \tag{1.44}$$

$$\dot{\boldsymbol{\nu}}_{\text{min}} \leq \dot{\boldsymbol{\nu}} \leq \dot{\boldsymbol{\nu}}_{\text{max}}. \tag{1.45}$$

The joint position limits, unlike the torque limits, must be manipulated somewhat in order to be properly expressed in $\boldsymbol{\chi}$. To formulate this constraint, $\boldsymbol{q}$ needs to be calculated while taking into account a second order prediction of the joint-space movement,

$$\boldsymbol{q}(t+h) = \boldsymbol{q}(t) + h\boldsymbol{\nu}(t) + \frac{h^2}{2}\dot{\boldsymbol{\nu}}(t), \tag{1.46}$$

where h is the prediction period, which is generally some multiple of the control period. Note that the floating base components of the configuration variable are not subject to articular limits, and their corresponding components in $\boldsymbol{q}$, $\boldsymbol{\nu}$, and $\dot{\boldsymbol{\nu}}$, are disregarded in (1.46). Dropping the time dependencies, the limits are written,

$$\begin{aligned} \boldsymbol{q}_{\text{min}} \leq \boldsymbol{q} + h\boldsymbol{\nu} + \frac{h^2}{2}\dot{\boldsymbol{\nu}} \leq \boldsymbol{q}_{\text{max}} \\ \Leftrightarrow \frac{2}{h^2}\left[\boldsymbol{q}_{\text{min}} - (\boldsymbol{q} + h\boldsymbol{\nu})\right] \leq \dot{\boldsymbol{\nu}} \leq \frac{2}{h^2}\left[\boldsymbol{q}_{\text{max}} - (\boldsymbol{q} + h\boldsymbol{\nu})\right]. \end{aligned}$$

Using $\boldsymbol{\chi}$, (1.47) can be rewritten as,

$$\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{q}}} \boldsymbol{\chi} \leq \underbrace{\frac{2}{h^2}\begin{bmatrix} \boldsymbol{q}_{\text{max}} - (\boldsymbol{q} + h\boldsymbol{\nu}) \\ -\left[\boldsymbol{q}_{\text{min}} - (\boldsymbol{q} + h\boldsymbol{\nu})\right] \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{q}}}. \tag{1.47}$$

From (1.47), one can of course naturally derive joint velocity and acceleration limits,

$$\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\boldsymbol{\nu}}} \boldsymbol{\chi} \leq \underbrace{\frac{1}{h}\begin{bmatrix} \boldsymbol{\nu}_{\text{max}} - \boldsymbol{\nu} \\ -(\boldsymbol{\nu}_{\text{min}} - \boldsymbol{\nu}) \end{bmatrix}}_{\boldsymbol{h}^{\boldsymbol{\nu}}} \tag{1.48}$$

$$\underbrace{\begin{bmatrix} I & \mathbf{0} \\ -I & \mathbf{0} \end{bmatrix}}_{G^{\dot{\boldsymbol{\nu}}}} \boldsymbol{\chi} \leq \underbrace{\begin{bmatrix} \dot{\boldsymbol{\nu}}_{\text{max}} \\ -\dot{\boldsymbol{\nu}}_{\text{min}} \end{bmatrix}}_{\boldsymbol{h}^{\dot{\boldsymbol{\nu}}}}. \tag{1.49}$$

The choice of the prediction period, h, in the joint-space limits is crucial to the proper functioning of these constraints. Smaller values of h lead to more aggressive approaches to the joint limits, while larger values produce a more conservative treatment. This variability is due to the fact that the prediction does not take into account the deceleration capabilities of the joints.

Important: To put these constraints into ORCA standard form we have,

$$\frac{2}{h^2}\left[\boldsymbol{q}_{\text{min}} - (\boldsymbol{q} + h\boldsymbol{\nu})\right] \leq \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} \leq \frac{2}{h^2}\left[\boldsymbol{q}_{\text{max}} - (\boldsymbol{q} + h\boldsymbol{\nu})\right]$$

$$\frac{1}{h}\left[\boldsymbol{\nu}_{\text{max}} - \boldsymbol{\nu}\right] \leq \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} \leq \frac{1}{h}\left[\boldsymbol{\nu}_{\text{max}} - \boldsymbol{\nu}\right]$$

$$\dot{\boldsymbol{\nu}}_{\text{max}} \leq \begin{bmatrix} I & \mathbf{0} \end{bmatrix} \boldsymbol{\chi} \leq \dot{\boldsymbol{\nu}}_{\text{max}}$$

Contact Constraints

When a robot interacts with its environment, it does so through contacts. These contacts can be **unilateral contacts**, or **bilateral contacts**. Simply put, unilateral contacts are those the robot can only push, e.g. foot contact with the floor, and bilateral contacts are those which allow the robot to push or pull, e.g. gripping the rung of a ladder.

Todo: add citations: Following the formulations in citep{Salini2011} and citep{Saab2013}

For unilateral contact constraints, a linearized approximation of the Coulomb friction cone is employed. A friction contact constraint in the controller must ensure that the linear velocity at the contact point is zero,

$$^{F}J_i(\boldsymbol{q})\dot{\boldsymbol{\nu}} + {}^{F}\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} = \mathbf{0}, \tag{1.50}$$

and that the wrench remains within a linearized approximation of a friction cone,

$$^{F}C_i{}^{F}\boldsymbol{\omega}_i \leq \mathbf{0}. \tag{1.51}$$

In (1.50), ^{F}J and $^{F}\dot{J}$ contain the linear components of the i^{th} contact Jacobian. In (1.51), $^{F}C_i$ is a matrix which linearly approximates the second-order norm cone,

$$\left\| {}^{F}\boldsymbol{\omega}_i - ({}^{F}\boldsymbol{\omega}_i \cdot \hat{\boldsymbol{n}}_i)\hat{\boldsymbol{n}}_i \right\|_2 \leq \mu_i({}^{F}\boldsymbol{\omega}_i \cdot \hat{\boldsymbol{n}}_i), \tag{1.52}$$

where $^{F}\boldsymbol{\omega}_i$ is are the force components of the i^{th} contact wrench, $\hat{\boldsymbol{n}}_i$ is the normal vector of the contact, and μ_i is the friction coefficient. Finally, expressing these two constraints in terms of $\boldsymbol{\chi}$, and defining $^{F}\boldsymbol{\omega}_i = S_i^F\boldsymbol{\chi}$, gives the following coupled equality and inequality constraints,

$$\underbrace{\begin{bmatrix} {}^{F}J_i(\boldsymbol{q}) & \mathbf{0} \end{bmatrix}}_{A^{\boldsymbol{\omega}}} \boldsymbol{\chi} = \underbrace{-{}^{F}\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu}}_{\boldsymbol{b}^{\boldsymbol{\omega}}} \tag{1.53}$$

$$\underbrace{\begin{bmatrix} \mathbf{0} & {}^{F}C_i S_i^F \end{bmatrix}}_{G^{\boldsymbol{\omega}}} \boldsymbol{\chi} \leq \underbrace{\mathbf{0}}_{\boldsymbol{h}^{\boldsymbol{\omega}}}, \tag{1.54}$$

where S_i^F selects the i^{th} contact force vector. Equations (1.53) and (1.54) are valid for a single contact point. For surface contacts, e.g. a foot sole, multiple points on the surface can be used for friction contact constraints — usually the four corners of the foot. Equation (1.53) introduces 3 equality constraints for the linear velocity of the contact point. The number of inequality constraints introduced by (1.54) depends on the number of polygon edges used to approximate the friction cone. Here, 6 edges are used, and because of symmetry, this introduces 3 inequality constraints per contact to the controller.

Important: To put these constraints into ORCA standard form we have,

$$\boldsymbol{b}^{\boldsymbol{\omega}} \leq A^{\boldsymbol{\omega}} \leq \boldsymbol{b}^{\boldsymbol{\omega}}$$

$$-\inf \leq G^{\boldsymbol{\omega}} \boldsymbol{\chi} \leq \boldsymbol{h}^{\boldsymbol{\omega}}$$

For bilateral contacts, it is sufficient to ensure no relative motion between the two links, i and j in contact. It should be noted that here a link can be some part of the environment for which a kinematic model exists. To ensure no motion between the links, the following relationship must be true,

$$\left(J_i(\boldsymbol{q}) - J_j(\boldsymbol{q})\right)\dot{\boldsymbol{\nu}} + \left(\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu}) - \dot{J}_j(\boldsymbol{q},\boldsymbol{\nu})\right)\boldsymbol{\nu} = \mathbf{0}, \tag{1.55}$$

where $J_i(\boldsymbol{q})$, $\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu})$, $J_j(\boldsymbol{q})$, and $\dot{J}_j(\boldsymbol{q},\boldsymbol{\nu})$, are the Jacobians and their derivatives for the $i$textsuperscript{th} and $j$textsuperscript{th} links respectively. Putting (1.55) in terms of $\boldsymbol{\chi}$ produces,

$$\underbrace{\begin{bmatrix}(J_i(\boldsymbol{q}) - J_j(\boldsymbol{q})) & \mathbf{0}\end{bmatrix}}_{A^{bc}}\boldsymbol{\chi} = \underbrace{-\left(\dot{J}_i(\boldsymbol{q},\boldsymbol{\nu}) - \dot{J}_j(\boldsymbol{q},\boldsymbol{\nu})\right)\boldsymbol{\nu}}_{\boldsymbol{b}^{bc}}. \tag{1.56}$$

Important: To put this constraint into ORCA standard form we have,

$$\boldsymbol{b}^{bc} \leq A^{bc} \leq \boldsymbol{b}^{bc}$$

1.1.16 Resolution Strategies

Multi-Objective Optimization

Objective functions represent the intentions of the problem designer: what meaningful quantity or measure is to be minimized to best solve some issue. As is often the case, there may be more than one quantity or measure which must be minimized and therefore multiple objective functions are combined together. When multiple objective functions, $f_i(\boldsymbol{\chi})$, are considered simultaneously, a **multi-objective optimization** problem (a.k.a. multicriteria, multicriterion, or Pareto optimization) is created. One common method of solving multi-objective optimization problems is through textit{scalarization}. Scalarization is the process of combining of multiple objective costs into one scalar cost. There are a multitude of scalarization techniques but weighted summation is of the most common,

$$\underset{\boldsymbol{\chi}}{\arg\min} \sum_{i=1}^{n_o} w_i f_i(\boldsymbol{\chi}) = \sum_{i=1}^{n} w_i \left\|E_i\boldsymbol{\chi} - \mathbf{f}_i\right\|_2^2. \tag{1.57}$$

In (1.57), n_o is the total number of objective functions. This scalarization can be written compactly by concatenating the individual objectives as,

$$\underset{\boldsymbol{\chi}}{\arg\min} \quad \left\|E_w\boldsymbol{\chi} - \mathbf{f}_w\right\|_2^2 \tag{1.58}$$

where

$$E_w = \begin{bmatrix}\sqrt{w_1}E_1\\ \sqrt{w_2}E_2\\ \vdots\\ \sqrt{w_n}E_{n_o}\end{bmatrix} \quad \text{and} \quad \mathbf{f}_w = \begin{bmatrix}\sqrt{w_1}\mathbf{f}_1\\ \sqrt{w_2}\mathbf{f}_2\\ \vdots\\ \sqrt{w_n}\mathbf{f}_{n_o}\end{bmatrix}. \tag{1.59}$$

Each weight, $w_i \geq 0$, dictates the relative importance of its objective $f_i(\boldsymbol{\chi})$ and therefore its impact on the solution. In (1.58) the weights are assumed to be scalars, but it is also possible to use matrices of different weights as long as they remain positive semi-definite.

As an alternative to scalarization, the objective functions can be minimized hierarchically in order of importance to ensure that the most important objective(s) are minimized as much as possible without influence of the lower priority objectives. This is known as **lexicographic optimization** in multi-objective optimization. To achieve this, the objectives are treated individually as a cascade of QPs where the solutions are reused as equality constraints in the subsequent QP minimizations.

Resolution (Prioritization) Strategies for Whole-Body Control

If multiple task objective functions are combined (using operations that preserve convexity) in the resolution of the control problem, then they can be performed simultaneously. In these cases, it is important to select a strategy for the resolution of the optimization problem. In turn, the strategy determines how tasks interact/interfere with one another. The two prevailing methods for dealing with multiple tasks are hierarchical and weighted prioritization.

Hierarchical Prioritization

In **hierarchical prioritization**, the tasks are organized by order of importance in discrete levels. Each task error is minimized in descending order of its importance and the solution to the optimization problem is then used in the equality constraints for the proceeding optimizations.

Hierarchical Prioritization Algorithm

$$\text{for} \quad (i = 1 \ldots n_{\text{task}})$$

$$\begin{aligned}
\boldsymbol{\chi}_i^* = \arg\min_{\boldsymbol{\chi}} \quad & f_i^{\text{task}}(\boldsymbol{\chi}) + w_0 f_0^{\text{task}}(\boldsymbol{\chi}) \\
\text{s.t.} \quad & G\boldsymbol{\chi} \leq \boldsymbol{h} \\
& A_i\boldsymbol{\chi} = \boldsymbol{b}_i \\
A_{i+1} \leftarrow & \begin{bmatrix} A_i \\ E_i \end{bmatrix} \\
\boldsymbol{b}_{i+1} \leftarrow & \begin{bmatrix} \boldsymbol{b}_i \\ \boldsymbol{\chi}_i^* \end{bmatrix} \\
\boldsymbol{\chi}^* \leftarrow & \boldsymbol{\chi}_i^*
\end{aligned}$$

$$\text{return} \quad \boldsymbol{\chi}^*$$

This algorithm is tantamount to null-space projection in the dynamic domain; however, inequality constraints can be accounted for. As a note, the regularization term, $w_0 f_0^{\text{task}}(\boldsymbol{x})$, in each optimization cascade serves to remove solution redundancy when the objective function has a null space, but this redundancy is necessary for executing the subsequent tasks. The operation, $A_{i+1} \leftarrow \begin{bmatrix} A_i \\ E_i \end{bmatrix}$, propagates the null space of the objective function, which has just been solved, to the proceeding objective functions through the equality constraint.

Resolving the whole-body control problem hierarchically has the benefit of strictly ensuring the optimization of one task error over another; however, it makes task transitioning and blending more difficult. Using continuous, or soft, priorities can alleviate some of these issues.

Weighted Prioritization

In multi-objective optimization, task weights dictate where, on the Pareto front of solutions, the QP calculates an optimum. Consequently, the optimum found favors the minimization of tasks with higher weights. This affords a method of prioritization, which ensures that critical tasks, such as those for balance, are preferentially accomplished, in situations where other less-critical tasks, such as a reach, have conflicting optima.

Weighted Prioritization Algorithm

$$
\begin{aligned}
\boldsymbol{\chi}^* = \arg\min_{\boldsymbol{\chi}} \quad & \sum_{i=1}^{n_{\text{task}}} w_i f_i^{\text{task}}(\boldsymbol{\chi}) + w_0 f_0^{\text{task}}(\boldsymbol{\chi}) \\
\text{s.t.} \quad & G\boldsymbol{\chi} \leq \boldsymbol{h} \\
& A\boldsymbol{\chi} = \boldsymbol{b}. \\
\text{return} \quad & \boldsymbol{\chi}^*
\end{aligned}
$$

However, using continuous priorities between tasks cannot guarantee that the tasks will not interfere with one another.

Important: In fact, each task will assuredly impact the ensemble but that impact can be rendered numerically negligible.

Hybrid Schemes

It can be seen that the weighted strategy is a subset of the hierarchical strategy, by observing that each level in a hierarchical scheme can be solved as a weighted problem. This **hybrid prioritization strategy** can provide the best of both hierarchical and weighted methods, but at the cost of increase implementation and computational complexity.

Generalized Hierarchical Prioritization

In addition to the simple mixing of weights and hierarchies, continuous generalized projection schemes are developed by citep{Liu2016}. These methods allow priorities to continuously vary from weighted to purely hierarchical through scalar values. Such approaches can provide smooth transitions between tasks, as is common in complex activities such as walking, but require substantially more computation time than purely weighted or hierarchical methods.

Resolution Strategies in ORCA

ORCA provides three strategies for resolving a multi-objective QP which containts multiple tasks and/or constraints.

1. `OneLevelWeighted` (weighted prioritization)
2. `MultiLevelWeighted` (hybrid prioritization)
3. `Generalized` (generalized hierarchical prioritization)

Note: these strategies are in the namespace `orca::optim::ResolutionStrategy`

The strategies are implemented in `Controller.cc` on the controller update:

```
bool Controller::update(double current_time, double dt)
{
    MutexLock lock(mutex);
    solution_found_ = false;

    switch (resolution_strategy_)
    {
        case ResolutionStrategy::OneLevelWeighted:
        {
            ...
        }
        case ResolutionStrategy::MultiLevelWeighted:
        {
            ...
        }
        case ResolutionStrategy::Generalized:
        {
            not implemented yet
        }
        default:
            orca_throw(Formatter() << "unsupported resolution strategy");
    }
}
```

Each of these strategies is detailed in the following sections.

One Level Weighted

```
case ResolutionStrategy::OneLevelWeighted:
{
    updateTasks(current_time,dt);
    updateConstraints(current_time,dt);
    auto problem = getProblemAtLevel(0);
    problem->build();
    solution_found_ = problem->solve();

    if(this->update_cb_)
        this->update_cb_(current_time,dt);

    static bool print_warning = true;
    if(solution_found_ && isProblemDry(problem) && print_warning)
    {
        print_warning = false;
        LOG_WARNING << "\n\n"
            <<" Solution found but the problem is dry !\n"
            << "It means that an optimal solution is found but the problem \n"
            << "only has one task computing anything, ans it's the"
            << "GlobalRegularisation task (This will only be printed once)\n\n"
            << "/!\\ Resulting torques will cause the robot to fall /!\\";
    }

    return solution_found_;
}
```

Multi-Level Weighted

Todo: Not yet implemented...

```
case ResolutionStrategy::MultiLevelWeighted:
{
    updateTasks(current_time,dt);
    updateConstraints(current_time,dt);
    auto problem = getProblemAtLevel(0);
    problem->build();
    solution_found_ = problem->solve();

    if(this->update_cb_)
        this->update_cb_(current_time,dt);

    static bool print_warning = true;
    if(solution_found_ && isProblemDry(problem) && print_warning)
    {
        print_warning = false;
        LOG_WARNING << "\n\n"
            <<" Solution found but the problem is dry !\n"
            << "It means that an optimal solution is found but the problem \n"
            << "only has one task computing anything, ans it's the"
            << "GlobalRegularisation task (This will only be printed once)\n\n"
            << "/!\\ Resulting torques will cause the robot to fall /!\\";
    }

    return solution_found_;
}
```

Generalized

Todo: Not yet implemented as of ORCA `v.2.0.0`

1.1.17 License

CeCILL-C FREE SOFTWARE LICENSE AGREEMENT

Notice

This Agreement is a Free Software license agreement that is the result of discussions between its authors in order to ensure compliance with the two main principles guiding its drafting:

- firstly, compliance with the principles governing the distribution of Free Software: access to source code, broad rights granted to users,
- secondly, the election of a governing law, French law, with which it is conformant, both as regards the law of torts and intellectual property law, and the protection that it offers to both authors and holders of the economic rights over software.

The authors of the CeCILL-C (for Ce[a] C[nrs] I[nria] L[ogiciel] L[ibre]) license are:

Commissariat à l'Energie Atomique - CEA, a public scientific, technical and industrial research establishment, having its principal place of business at 25 rue Leblanc, immeuble Le Ponant D, 75015 Paris, France.

Centre National de la Recherche Scientifique - CNRS, a public scientific and technological establishment, having its principal place of business at 3 rue Michel-Ange, 75794 Paris cedex 16, France.

Institut National de Recherche en Informatique et en Automatique - INRIA, a public scientific and technological establishment, having its principal place of business at Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay cedex, France.

Preamble

The purpose of this Free Software license agreement is to grant users the right to modify and re-use the software governed by this license.

The exercising of this right is conditional upon the obligation to make available to the community the modifications made to the source code of the software so as to contribute to its evolution.

In consideration of access to the source code and the rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors only have limited liability.

In this respect, the risks associated with loading, using, modifying and/or developing or reproducing the software by the user are brought to the user's attention, given its Free Software status, which may make it complicated to use, with the result that its use is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the suitability of the software as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions of security. This Agreement may be freely reproduced and published, provided it is not altered, and that no provisions are either added or removed herefrom.

This Agreement may apply to any or all software for which the holder of the economic rights decides to submit the use thereof to its provisions.

Article 1 - DEFINITIONS

For the purpose of this Agreement, when the following expressions commence with a capital letter, they shall have the following meaning:

Agreement: means this license agreement, and its possible subsequent versions and annexes.

Software: means the software in its Object Code and/or Source Code form and, where applicable, its documentation, "as is" when the Licensee accepts the Agreement.

Initial Software: means the Software in its Source Code and possibly its Object Code form and, where applicable, its documentation, "as is" when it is first distributed under the terms and conditions of the Agreement.

Modified Software: means the Software modified by at least one Integrated Contribution.

Source Code: means all the Software's instructions and program lines to which access is required so as to modify the Software.

Object Code: means the binary files originating from the compilation of the Source Code.

Holder: means the holder(s) of the economic rights over the Initial Software.

Licensee: means the Software user(s) having accepted the Agreement.

Contributor: means a Licensee having made at least one Integrated Contribution.

Licensor: means the Holder, or any other individual or legal entity, who distributes the Software under the Agreement.

Integrated Contribution: means any or all modifications, corrections, translations, adaptations and/or new functions integrated into the Source Code by any or all Contributors.

Related Module: means a set of sources files including their documentation that, without modification to the Source Code, enables supplementary functions or services in addition to those offered by the Software.

Derivative Software: means any combination of the Software, modified or not, and of a Related Module.

Parties: mean both the Licensee and the Licensor.

These expressions may be used both in singular and plural form.

Article 2 - PURPOSE

The purpose of the Agreement is the grant by the Licensor to the Licensee of a non-exclusive, transferable and worldwide license for the Software as set forth in Article 5 hereinafter for the whole term of the protection granted by the rights over said Software.

Article 3 - ACCEPTANCE

3.1 The Licensee shall be deemed as having accepted the terms and conditions of this Agreement upon the occurrence of the first of the following events:

- (i) loading the Software by any or all means, notably, by downloading from a remote server, or by loading from a physical medium;
- (ii) the first time the Licensee exercises any of the rights granted hereunder.

3.2 One copy of the Agreement, containing a notice relating to the characteristics of the Software, to the limited warranty, and to the fact that its use is restricted to experienced users has been provided to the Licensee prior to its acceptance as set forth in Article 3.1 hereinabove, and the Licensee hereby acknowledges that it has read and understood it.

Article 4 - EFFECTIVE DATE AND TERM

4.1 EFFECTIVE DATE

The Agreement shall become effective on the date when it is accepted by the Licensee as set forth in Article 3.1.

4.2 TERM

The Agreement shall remain in force for the entire legal term of protection of the economic rights over the Software.

Article 5 - SCOPE OF RIGHTS GRANTED

The Licensor hereby grants to the Licensee, who accepts, the following rights over the Software for any or all use, and for the term of the Agreement, on the basis of the terms and conditions set forth hereinafter.

Besides, if the Licensor owns or comes to own one or more patents protecting all or part of the functions of the Software or of its components, the Licensor undertakes not to enforce the rights granted by these patents against successive Licensees using, exploiting or modifying the Software. If these patents are transferred, the Licensor undertakes to have the transferees subscribe to the obligations set forth in this paragraph.

5.1 RIGHT OF USE

The Licensee is authorized to use the Software, without any limitation as to its fields of application, with it being hereinafter specified that this comprises:

1. permanent or temporary reproduction of all or part of the Software by any or all means and in any or all form.
2. loading, displaying, running, or storing the Software on any or all medium.
3. entitlement to observe, study or test its operation so as to determine the ideas and principles behind any or all constituent elements of said Software. This shall apply when the Licensee carries out any or all loading, displaying, running, transmission or storage operation as regards the Software, that it is entitled to carry out hereunder.

5.2 RIGHT OF MODIFICATION

The right of modification includes the right to translate, adapt, arrange, or make any or all modifications to the Software, and the right to reproduce the resulting software. It includes, in particular, the right to create a Derivative Software.

The Licensee is authorized to make any or all modification to the Software provided that it includes an explicit notice that it is the author of said modification and indicates the date of the creation thereof.

5.3 RIGHT OF DISTRIBUTION

In particular, the right of distribution includes the right to publish, transmit and communicate the Software to the general public on any or all medium, and by any or all means, and the right to market, either in consideration of a fee, or free of charge, one or more copies of the Software by any means.

The Licensee is further authorized to distribute copies of the modified or unmodified Software to third parties according to the terms and conditions set forth hereinafter.

5.3.1 DISTRIBUTION OF SOFTWARE WITHOUT MODIFICATION

The Licensee is authorized to distribute true copies of the Software in Source Code or Object Code form, provided that said distribution complies with all the provisions of the Agreement and is accompanied by:

1. a copy of the Agreement,
2. a notice relating to the limitation of both the Licensor's warranty and liability as set forth in Articles 8 and 9,

and that, in the event that only the Object Code of the Software is redistributed, the Licensee allows effective access to the full Source Code of the Software at a minimum during the entire period of its distribution of the Software, it being understood that the additional cost of acquiring the Source Code shall not exceed the cost of transferring the data.

5.3.2 DISTRIBUTION OF MODIFIED SOFTWARE

When the Licensee makes an Integrated Contribution to the Software, the terms and conditions for the distribution of the resulting Modified Software become subject to all the provisions of this Agreement.

The Licensee is authorized to distribute the Modified Software, in source code or object code form, provided that said distribution complies with all the provisions of the Agreement and is accompanied by:

1. a copy of the Agreement,
2. a notice relating to the limitation of both the Licensor's warranty and liability as set forth in Articles 8 and 9,

and that, in the event that only the object code of the Modified Software is redistributed, the Licensee allows effective access to the full source code of the Modified Software at a minimum during the entire period of its distribution of the Modified Software, it being understood that the additional cost of acquiring the source code shall not exceed the cost of transferring the data.

5.3.3 DISTRIBUTION OF DERIVATIVE SOFTWARE

When the Licensee creates Derivative Software, this Derivative Software may be distributed under a license agreement other than this Agreement, subject to compliance with the requirement to include a notice concerning the rights over the Software as defined in Article 6.4. In the event the creation of the Derivative Software required modification of the Source Code, the Licensee undertakes that:

1. the resulting Modified Software will be governed by this Agreement,
2. the Integrated Contributions in the resulting Modified Software will be clearly identified and documented,
3. the Licensee will allow effective access to the source code of the Modified Software, at a minimum during the entire period of distribution of the Derivative Software, such that such modifications may be carried over in a subsequent version of the Software; it being understood that the additional cost of purchasing the source code of the Modified Software shall not exceed the cost of transferring the data.

5.3.4 COMPATIBILITY WITH THE CeCILL LICENSE

When a Modified Software contains an Integrated Contribution subject to the CeCILL license agreement, or when a Derivative Software contains a Related Module subject to the CeCILL license agreement, the provisions set forth in the third item of Article 6.4 are optional.

Article 6 - INTELLECTUAL PROPERTY

6.1 OVER THE INITIAL SOFTWARE

The Holder owns the economic rights over the Initial Software. Any or all use of the Initial Software is subject to compliance with the terms and conditions under which the Holder has elected to distribute its work and no one shall be entitled to modify the terms and conditions for the distribution of said Initial Software.

The Holder undertakes that the Initial Software will remain ruled at least by this Agreement, for the duration set forth in Article 4.2.

6.2 OVER THE INTEGRATED CONTRIBUTIONS

The Licensee who develops an Integrated Contribution is the owner of the intellectual property rights over this Contribution as defined by applicable law.

6.3 OVER THE RELATED MODULES

The Licensee who develops a Related Module is the owner of the intellectual property rights over this Related Module as defined by applicable law and is free to choose the type of agreement that shall govern its distribution under the conditions defined in Article 5.3.3.

6.4 NOTICE OF RIGHTS

The Licensee expressly undertakes:

1. not to remove, or modify, in any manner, the intellectual property notices attached to the Software;
2. to reproduce said notices, in an identical manner, in the copies of the Software modified or not;
3. to ensure that use of the Software, its intellectual property notices and the fact that it is governed by the Agreement is indicated in a text that is easily accessible, specifically from the interface of any Derivative Software.

The Licensee undertakes not to directly or indirectly infringe the intellectual property rights of the Holder and/or Contributors on the Software and to take, where applicable, vis-à-vis its staff, any and all measures required to ensure respect of said intellectual property rights of the Holder and/or Contributors.

Article 7 - RELATED SERVICES

7.1 Under no circumstances shall the Agreement oblige the Licensor to provide technical assistance or maintenance services for the Software.

However, the Licensor is entitled to offer this type of services. The terms and conditions of such technical assistance, and/or such maintenance, shall be set forth in a separate instrument. Only the Licensor offering said maintenance and/or technical assistance services shall incur liability therefor.

7.2 Similarly, any Licensor is entitled to offer to its licensees, under its sole responsibility, a warranty, that shall only be binding upon itself, for the redistribution of the Software and/or the Modified Software, under terms and conditions that it is free to decide. Said warranty, and the financial terms and conditions of its application, shall be subject of a separate instrument executed between the Licensor and the Licensee.

Article 8 - LIABILITY

8.1 Subject to the provisions of Article 8.2, the Licensee shall be entitled to claim compensation for any direct loss it may have suffered from the Software as a result of a fault on the part of the relevant Licensor, subject to providing evidence thereof.

8.2 The Licensor's liability is limited to the commitments made under this Agreement and shall not be incurred as a result of in particular: (i) loss due the Licensee's total or partial failure to fulfill its obligations, (ii) direct or consequential loss that is suffered by the Licensee due to the use or performance of the Software, and (iii) more generally, any consequential loss. In particular the Parties expressly agree that any or all pecuniary or business loss (i.e. loss of data, loss of profits, operating loss, loss of customers or orders, opportunity cost, any disturbance to business activities) or any or all legal proceedings instituted against the Licensee by a third party, shall constitute consequential loss and shall not provide entitlement to any or all compensation from the Licensor.

Article 9 - WARRANTY

9.1 The Licensee acknowledges that the scientific and technical state-of-the-art when the Software was distributed did not enable all possible uses to be tested and verified, nor for the presence of possible defects to be detected. In this respect, the Licensee's attention has been drawn to the risks associated with loading, using, modifying and/or developing and reproducing the Software which are reserved for experienced users.

The Licensee shall be responsible for verifying, by any or all means, the suitability of the product for its requirements, its good working order, and for ensuring that it shall not cause damage to either persons or properties.

9.2 The Licensor hereby represents, in good faith, that it is entitled to grant all the rights over the Software (including in particular the rights set forth in Article 5).

9.3 The Licensee acknowledges that the Software is supplied "as is" by the Licensor without any other express or tacit warranty, other than that provided for in Article 9.2 and, in particular, without any warranty as to its commercial value, its secured, safe, innovative or relevant nature.

Specifically, the Licensor does not warrant that the Software is free from any error, that it will operate without interruption, that it will be compatible with the Licensee's own equipment and software configuration, nor that it will meet the Licensee's requirements.

9.4 The Licensor does not either expressly or tacitly warrant that the Software does not infringe any third party intellectual property right relating to a patent, software or any other property right. Therefore, the Licensor disclaims any and all liability towards the Licensee arising out of any or all proceedings for infringement that may be instituted in respect of the use, modification and redistribution of the Software. Nevertheless, should such proceedings be instituted against the Licensee, the Licensor shall provide it with technical and legal assistance for its defense. Such technical and legal assistance shall be decided on a case-by-case basis between the relevant Licensor and the Licensee pursuant to a memorandum of understanding. The Licensor disclaims any and all liability as regards the Licensee's use of the name of the Software. No warranty is given as regards the existence of prior rights over the name of the Software or as regards the existence of a trademark.

Article 10 - TERMINATION

10.1 In the event of a breach by the Licensee of its obligations hereunder, the Licensor may automatically terminate this Agreement thirty (30) days after notice has been sent to the Licensee and has remained ineffective.

10.2 A Licensee whose Agreement is terminated shall no longer be authorized to use, modify or distribute the Software. However, any licenses that it may have granted prior to termination of the Agreement shall remain valid subject to their having been granted in compliance with the terms and conditions hereof.

Article 11 - MISCELLANEOUS

11.1 EXCUSABLE EVENTS

Neither Party shall be liable for any or all delay, or failure to perform the Agreement, that may be attributable to an event of force majeure, an act of God or an outside cause, such as defective functioning or interruptions of the electricity or telecommunications networks, network paralysis following a virus attack, intervention by government authorities, natural disasters, water damage, earthquakes, fire, explosions, strikes and labor unrest, war, etc.

11.2 Any failure by either Party, on one or more occasions, to invoke one or more of the provisions hereof, shall under no circumstances be interpreted as being a waiver by the interested Party of its right to invoke said provision(s) subsequently.

11.3 The Agreement cancels and replaces any or all previous agreements, whether written or oral, between the Parties and having the same purpose, and constitutes the entirety of the agreement between said Parties concerning said purpose. No supplement or modification to the terms and conditions hereof shall be effective as between the Parties unless it is made in writing and signed by their duly authorized representatives.

11.4 In the event that one or more of the provisions hereof were to conflict with a current or future applicable act or legislative text, said act or legislative text shall prevail, and the Parties shall make the necessary amendments so as to comply with said act or legislative text. All other provisions shall remain effective. Similarly, invalidity of a provision of the Agreement, for any reason whatsoever, shall not cause the Agreement as a whole to be invalid.

11.5 LANGUAGE

The Agreement is drafted in both French and English and both versions are deemed authentic.

Article 12 - NEW VERSIONS OF THE AGREEMENT

12.1 Any person is authorized to duplicate and distribute copies of this Agreement.

12.2 So as to ensure coherence, the wording of this Agreement is protected and may only be modified by the authors of the License, who reserve the right to periodically publish updates or new versions of the Agreement, each with a separate number. These subsequent versions may address new issues encountered by Free Software.

12.3 Any Software distributed under a given version of the Agreement may only be subsequently distributed under the same version of the Agreement or a subsequent version.

Article 13 - GOVERNING LAW AND JURISDICTION

13.1 The Agreement is governed by French law. The Parties agree to endeavor to seek an amicable solution to any disagreements or disputes that may arise during the performance of the Agreement.

13.2 Failing an amicable solution within two (2) months as from their occurrence, and unless emergency proceedings are necessary, the disagreements or disputes shall be referred to the Paris Courts having jurisdiction, by the more diligent Party.

Version 1.0 dated 2006-09-05.

CHAPTER 2

Authorship

Work on ORCA initially began in 2017 at the Institut des Systèmes Intelligents et de Robotique (ISIR). Since January 2018, active maintenance and development has been taken over by Fuzzy Logic Robotics S.A.S.

2.1 Maintainers

- Antoine Hoarau
- Ryan Lober
- Fuzzy Logic Robotics (info@fuzzylogicrobotics.com)

2.2 Contributors

- Vincent Padois

2.3 Related Publications

2.4 Partner Institutions