ORCA Documentation

Release Alpago

Antoine Hoarau

Jun 03, 2018

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*, let's first test the installation.

To do so simply run the following command:

```
orca_install_test
```

What's next?

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

1.1.2 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)

1.1.3 API Reference

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

API Documentation

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)
```

(continues on next page)

(continued from previous page)

```
# 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 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

```
#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)
    {
```

(continues on next page)

(continued from previous page)

```
        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]);

    //  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 = std::make_shared<RobotDynTree>();

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

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

    // Sets the world gravity (Optional)
    robot->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->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->setRobotState(eigState.jointPos,eigState.jointVel);
    // Now is the robot is considered 'initialized'


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

    // Cartesian Task
    auto cart_task = std::make_shared<CartesianTask>("CartTask-EE");
    // Add the task to the controller to initialize it.
    controller.addTask(cart_task);
    // Set the frame you want to control. Here we want to control the link_7.
    cart_task->setControlFrame("link_7"); //
```

(continues on next page)

(continued from previous page)

```
    // 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.

    // 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_task->servoController()->pid()->setProportionalGain(P);
    Vector6d D;
    D << 100, 100, 100, 1, 1, 1;
    cart_task->servoController()->pid()->setDerivativeGain(D);


    // 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_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

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

    // Joint torque limit is usually given by the robot manufacturer
    auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit");

    // Add the constraint to the controller to initialize - it is not read from the URDF for now.
```

(continues on next page)

(continued from previous page)

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

    // Add the constraint to the controller to initialize
    controller.addConstraint(jnt_pos_cstr);

    // Joint velocity limits are usually given by the robot manufacturer
    auto jnt_vel_cstr = std::make_shared<JointVelocityLimitConstraint>("JointVelocityLimit");

    // Add the constraint to the controller to initialize - it is not read from the URDF for now.
    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;

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

        // Step the controller + solve the internal optimal problem
        controller.update(current_time, dt);

        // Do what you want with the solution
        if(controller.solutionFound())
        {
            // The whole optimal solution [AccFb, Acc, Tfb, T, eWrenches]
```

(continues on next page)

(continued from previous page)

```
            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
            // 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();
    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.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

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

    orca::optim::Controller controller(
        "controller"
        ,robot
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolver::qpOASES
    );

    auto cart_task = std::make_shared<CartesianTask>("CartTask-EE");
    controller.addTask(cart_task);
    cart_task->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();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();

    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);

    cart_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

    const int ndof = robot->getNrOfDegreesOfFreedom();

    auto jnt_trq_cstr = std::make_shared<JointTorqueLimitConstraint>("JointTorqueLimit");
```

(continues on next page)

(continued from previous page)

```
    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);


    controller.activateTasksAndConstraints();
    // 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);

    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';

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

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

        // if(current_time % 0.1 == 0.0)
        // {
        //
        // }
```

(continues on next page)

(continued from previous page)

```
        std::cout << "Task position at t = " << current_time << "\t---\t" << cart_
↪task->servoController()->getCurrentCartesianPose().block(0,3,3,1).transpose() << '\n
↪';

        controller.update(current_time, dt);

        if(controller.solutionFound())
        {
            trq_cmd = controller.getJointTorqueCommand();
        }
        else
        {
            std::cout << "[warning] Didn't find a solution, using last valid solution.
↪" << '\n';
        }

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

        eigState.jointPos += eigState.jointVel * dt + ((acc_new*dt*dt)/2);
        eigState.jointVel += acc_new * dt;
    }
    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(). - cart_pos_ref.translation() << '\n';




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

1.1.7 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/02-using_callbacks.cc

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

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

(continues on next page)

(continued from previous page)

```
    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)
    {
        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;
```

(continues on next page)

(continued from previous page)

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

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

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

    orca::optim::Controller controller(
        "controller"
        ,robot
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolver::qpOASES
    );

    auto cart_task = std::make_shared<CartesianTask>("CartTask-EE");
    controller.addTask(cart_task);
    cart_task->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();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();

    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);

    cart_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_
↪acc_ref);

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

(continues on next page)

(continued from previous page)

```
    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));

    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';
```

(continues on next page)

(continued from previous page)

```
166
167
168     std::cout << "[main] Exiting main()." << '\n';
169     return 0;
170 }
```

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

1.1.8 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

```
1  #include <orca/gazebo/GazeboServer.h>
2  #include <orca/gazebo/GazeboModel.h>
3
4  using namespace orca::gazebo;
5
6  int main(int argc, char** argv)
7  {
8      // Get the urdf file from the command line
9      if(argc < 2)
10     {
11         std::cerr << "Usage : " << argv[0] << " /path/to/robot-urdf.urdf" << "\n";
12         return -1;
13     }
14     std::string urdf_url(argv[1]);
15
16     // Instanciate the gazebo server with de dedfault empty world
17     // This is equivalent to GazeboServer gz("worlds/empty.world")
18     GazeboServer s;
19     // Insert a model onto the server and create the GazeboModel from the return value
20     // You can also set the initial pose, and override the name in the URDF
21     auto m = GazeboModel(s.insertModelFromURDFFile(urdf_url));
22
23     // This is how you can get the full state of the robot
24     std::cout << "Model \'" << m.getName() << "\' State :\n" << '\n';
25     std::cout << "- Gravity "                    << m.getGravity().transpose()
↪        << '\n';
26     std::cout << "- Base velocity\n"             << m.getBaseVelocity().transpose()
↪        << '\n';
27     std::cout << "- Tworld->base\n"              << m.getWorldToBaseTransform().
↪matrix()       << '\n';
```

(continues on next page)

(continued from previous page)

```
    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)
    m.setCallback([&](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

```
#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;
```

(continues on next page)

(continued from previous page)

```
    }
    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)
    gz_model_two.setCallback([&](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.run([&](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';
    });
    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

```cpp
#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 gzserver(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 gzrobot = GazeboModel(gzserver.insertModelFromURDFFile(urdf_url));

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

    // Update the robot on at every iteration
    gzrobot.setCallback([&](uint32_t n_iter,double current_time,double dt)
    {
        robot->setRobotState(gzrobot.getWorldToBaseTransform().matrix()
                             ,gzrobot.getJointPositions()
                             ,gzrobot.getBaseVelocity()
                             ,gzrobot.getJointVelocities()
                             ,gzrobot.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
    gzserver.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

```cpp
#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 gzserver(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 gzrobot = GazeboModel(gzserver.insertModelFromURDFFile(urdf_url));

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

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

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

    // Update the robot on at every iteration
    gzrobot.setCallback([&](uint32_t n_iter,double current_time,double dt)
    {
        robot_kinematics->setRobotState(gzrobot.getWorldToBaseTransform().matrix()
                            ,gzrobot.getJointPositions()
                            ,gzrobot.getBaseVelocity()
                            ,gzrobot.getJointVelocities()
                            ,gzrobot.getGravity()
                        );
        gzrobot.setJointGravityTorques(robot_kinematics->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";
```

(continues on next page)

(continued from previous page)

```
    gzserver.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

```
#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]);

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

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

    orca::optim::Controller controller(
        "controller"
        ,robot
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolver::qpOASES
    );

    auto cart_task = std::make_shared<CartesianTask>("CartTask-EE");
    controller.addTask(cart_task);
    cart_task->setControlFrame("link_7"); //
    Eigen::Affine3d cart_pos_ref;
    cart_pos_ref.translation() = Eigen::Vector3d(0.5,-0.5,0.8); // x,y,z in meters
```

(continues on next page)

(continued from previous page)

```
    cart_pos_ref.linear() = Eigen::Quaterniond::Identity().toRotationMatrix();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();

    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);
    cart_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_
↪acc_ref);


    const int ndof = robot->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);

    GazeboServer gzserver(argc,argv);
    auto gzrobot = GazeboModel(gzserver.insertModelFromURDFFile(urdf_url));

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

    bool cart_task_activated = false;

    cart_task->onActivatedCallback([&cart_task_activated](){
        std::cout << "CartesianTask activated. Removing gravity compensation and␣
↪begining motion." << '\n';
        cart_task_activated = true;
    });

    gzrobot.setCallback([&](uint32_t n_iter,double current_time,double dt)
    {
        robot->setRobotState(gzrobot.getWorldToBaseTransform().matrix()
                             ,gzrobot.getJointPositions()
                             ,gzrobot.getBaseVelocity()
                             ,gzrobot.getJointVelocities()
                             ,gzrobot.getGravity()
```

(continues on next page)

(continued from previous page)

```
                        );
        // All tasks need the robot to be initialized during the activation phase
        if(n_iter == 1)
            controller.activateTasksAndConstraints();

        controller.update(current_time, dt);
        if (cart_task_activated)
        {
            if(controller.solutionFound())
            {
                gzrobot.setJointTorqueCommand( controller.getJointTorqueCommand() );
            }
            else
            {
                gzrobot.setBrakes(true);
            }
        }
        else
        {
            gzrobot.setJointGravityTorques(robot->getJointGravityTorques());
        }
    });

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

```
#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)
    {
```

(continues on next page)

(continued from previous page)

```
    }

    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;
            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 = std::make_shared<RobotDynTree>();
    robot->loadModelFromFile(urdf_url);
    robot->setBaseFrame("base_link");
```

(continues on next page)

(continued from previous page)

```
    robot->setGravity(Eigen::Vector3d(0,0,-9.81));
    RobotState eigState;
    eigState.resize(robot->getNrOfDegreesOfFreedom());
    eigState.jointPos.setZero();
    eigState.jointVel.setZero();
    robot->setRobotState(eigState.jointPos,eigState.jointVel);

    orca::optim::Controller controller(
        "controller"
        ,robot
        ,orca::optim::ResolutionStrategy::OneLevelWeighted
        ,QPSolver::qpOASES
    );

    auto cart_task = std::make_shared<CartesianTask>("CartTask-EE");
    controller.addTask(cart_task);
    cart_task->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();
    Vector6d cart_vel_ref = Vector6d::Zero();
    Vector6d cart_acc_ref = Vector6d::Zero();

    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);


    const int ndof = robot->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;

    GazeboServer gzserver(argc,argv);
    auto gzrobot = GazeboModel(gzserver.insertModelFromURDFFile(urdf_url));

```

(continues on next page)

(continued from previous page)

```
    ////////////////////////////////////////
    ////////////////////////////////////////
    ////////////////////////////////////////
    ////////////////////////////////////////

    MinJerkPositionTrajectory traj(5.0);
    int traj_loops = 0;
    bool exit_control_loop = true;
    Eigen::Vector3d start_position, end_position;


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

    bool cart_task_activated = false;

    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. Removing gravity compensation and␣
↪begining motion." << '\n';
        cart_task_activated = true;
    });

    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);
            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 (cart_task->getState() == TaskBase::State::Activated)
        {
            if (traj.isTrajectoryFinished()  )
            {
                if (traj_loops < 5)
                {
                    // 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." << '\n';
                    ++traj_loops;
                }
                else
                {
```

(continues on next page)

(continued from previous page)

```
                    std::cout << "Trajectory looping finished. Deactivating task and starting gravity compensation." << '\n';
                    cart_task->deactivate();
                }
            }
        }
    });

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

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



    gzrobot.setCallback([&](uint32_t n_iter,double current_time,double dt)
    {
        robot->setRobotState(gzrobot.getWorldToBaseTransform().matrix()
                            ,gzrobot.getJointPositions()
                            ,gzrobot.getBaseVelocity()
                            ,gzrobot.getJointVelocities()
                            ,gzrobot.getGravity()
                        );
        // All tasks need the robot to be initialized during the activation phase
        if(n_iter == 1)
            controller.activateTasksAndConstraints();

        controller.update(current_time, dt);

        if (cart_task_activated)
        {
            if(controller.solutionFound())
            {
                gzrobot.setJointTorqueCommand( controller.getJointTorqueCommand() );
            }
            else
            {
                gzrobot.setBrakes(true);
            }
        }
        else
        {
            gzrobot.setJointGravityTorques(robot->getJointGravityTorques());
        }
    });

    std::cout << "Simulation running... (GUI with \'gzclient\')" << "\n";
    gzserver.run();
    return 0;
}
```

1.1.9 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

```cpp
#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;
    }
    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 = std::make_shared<RobotDynTree>(); // Here you can pass a robot name
    robot->loadModelFromFile(urdf_url); // If you don't pass a robot name, it is extracted from the urdf
    robot->setBaseFrame("base_link"); // All the transformations (end effector pose for example) will be expressed wrt this base frame
    robot->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->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->setRobotState(eigState.jointPos,eigState.jointVel); // Now is the robot is considered 'initialized'

    // Instanciate an ORCA Controller
```

(continues on next page)

(continued from previous page)

```
    orca::optim::Controller controller(
        "controller"
        ,robot
        ,orca::optim::ResolutionStrategy::OneLevelWeighted // MultiLevelWeighted,Generalized
        ,QPSolver::qpOASES
    );

    // Cartesian Task
    auto cart_task = std::make_shared<CartesianTask>("CartTask-EE");
    controller.addTask(cart_task); // Add the task to the controller to initialize it
    // Set the frame you want to control
    cart_task->setControlFrame("link_7"); // We want to control the link_7

    // Set the pose desired for the link_7
    Eigen::Affine3d cart_pos_ref;
    // Translation
    cart_pos_ref.translation() = Eigen::Vector3d(1.,0.75,0.5); // x,y,z in meters
    // Rotation is done with a Matrix3x3
    Eigen::Quaterniond quat;
    // Example 1 : create a quaternion from Euler anglers ZYZ convention
    quat = Eigen::AngleAxisd(0, Eigen::Vector3d::UnitZ())
         * Eigen::AngleAxisd(0, Eigen::Vector3d::UnitY())
         * Eigen::AngleAxisd(0, Eigen::Vector3d::UnitZ());
    // 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();

    // Set the desired cartesian velocity to zero
    Vector6d cart_vel_ref;
    cart_vel_ref.setZero();

    // Set the desired cartesian velocity to zero
    Vector6d cart_acc_ref;
    cart_acc_ref.setZero();

    // 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);
    // 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_task->servoController()->setDesired(cart_pos_ref.matrix(),cart_vel_ref,cart_acc_ref);

    // Get the number of actuated joints
    const int ndof = robot->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
```

(continues on next page)

(continued from previous page)

```
    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
        robot->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();

```

(continues on next page)

(continued from previous page)

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

1.1.10 Overview

The problem is written as a **quadratic problem** :

$$\min_x \frac{1}{2}x^t Hx + x^t g$$
$$\text{subject to}$$
$$lb \leq x \leq ub$$
$$lb_A \leq Ax \leq ub_A$$

- `x` the optimization vector
- `H` the hessian matrix ($size(x) \times size(x)$)
- `g` the gradient vector ($size(x) \times 1$)
- `A` the constraint matrix ($size(x) \times size(x)$)
- `lb` and `ub` the lower and upper bounds of `x` ($size(x) \times 1$)
- `lbA` and `ubA` the lower and upper bounds of `A` ($size(x) \times 1$)

Tasks are written as **weighted euclidian distance function** :

$$w_{task}\|\mathbf{E}x + \mathbf{f}\|^2_{W_{norm}}$$

- `x` the optimization vector, or **part** of the optimization vector
- `E` the linear matrix of the affine function ($size(x) \times size(x)$)
- `f` the origin vector ($size(x) \times 1$)
- `w task` the weight of the tasks in the overall quadratic cost (scalar $[0:1]$)
- `W norm` the weight of the euclidean norm ($size(x) \times size(x)$)

Given n_t tasks, the **overall cost function** is such that:

$$\frac{1}{2}x^t Hx + x^t g = \frac{1}{2}\sum_{i=1}^{n_t} w_{task,i}\|\mathbf{E}_i x + \mathbf{f}_i\|^2_{W_{norm,i}}$$

Constraints are written as **double bounded linear function** :

$$lb_C \leq Cx \leq ub_C$$

- `C` the constraint matrix ($size(x) \times size(x)$)
- `lbC` and `ubC` the lower and upper bounds of `A` ($size(x) \times 1$)

1.1.11 Optimization Vector

The optimization vector in the quadratic problem is written as follows :

$$X = \begin{pmatrix} \dot{\nu}^{fb} \\ \dot{\nu}^{j} \\ \tau^{fb} \\ \tau^{j} \\ {}^{e}w_0 \\ \vdots \\ {}^{e}w_n \end{pmatrix}$$

- $\dot{\nu}^{fb}$: Floating base joint acceleration (6×1)
- $\dot{\nu}^{j}$: Joint space acceleration $(n_{dof} \times 1)$
- τ^{fb} : Floating base joint torque (6×1)
- τ^{j} : Joint space joint torque $(n_{dof} \times 1)$
- ${}^{e}w_{n}$: External wrench (6×1)
- τ^{fb} : Floating base joint torque (6×1)
- τ^{j} : Joint space joint torque $(n_{dof} \times 1)$
- ${}^{e}w_{n}$: External wrench (6×1)

In ORCA those are called *Control variables* and should be used to define every task and constraint. In addition to those necessary variables, you can specify also a combination :

- $\dot{\nu}$: Generalised joint acceleration, concatenation of $\dot{\nu}^{fb}$ and $\dot{\nu}^{j}$ $(6 + n_{dof} \times 1)$
- τ : Generalised joint torque, concatenation of τ^{fb} and τ^{j} $(6 + n_{dof} \times 1)$
- X : The whole optimization vector $(6 + n_{dof} + 6 + n_{dof} + n_{wrenches}6 \times 1)$
- ${}^{e}w$: External wrenches $(n_{wrenche}6 \times 1)$
- X : The whole optimization vector $(6 + n_{dof} + 6 + n_{dof} + n_{wrenches}6 \times 1)$
- ${}^{e}w$: External wrenches $(n_{wrenche}6 \times 1)$

1.1.12 Cartesian Acceleration

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

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

1.1.13 Dynamics Equation

- **Control variable** : X (whole optimization vector)
- **Type** : Equality constraint
- **Size** : $ndof \times size(X)$

$$\begin{bmatrix} -M & S_{\tau} & J_{^{e}w} \end{bmatrix} X = C + G$$

```
orca::constraint::DynamicsEquationConstraint dyn_eq;
dyn_eq.loadRobotModel( urdf );
dyn_eq.setGravity( Eigen::Vector3d(0,0,-9.81) );
dyn_eq.update(); // <-- Now initialized

dyn_eq.activate(); // <-- Now activated
dyn_eq.insertInProblem(); // <-- Now part of the optimization problem
```

1.1.14 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 world-wide 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


UPMC
SORBONNE UNIVERSITÉS
Fuzzy Logic
ROBOTICS