dqrobotics

Release 19.10.0

Bruno V. Adorno and Murilo M. Marinho

Apr 27, 2023

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 About this project

DQ Robotics is a standalone open-source (LGPLv3) library for robot modelling and control.

1.1.1 Core features

It provides dual quaternion algebra and kinematic calculation algorithms in Python3, Matlab, and C++11.

- Most users will benefit from using the Python3 version at first. It is easy and computationally efficient (C++ code runs under the hood for fast performance).
- Use the MATLAB version if you want to test your ideas fast while having convenient visualization tools, provided you have access to the MathWorks software.
- Use the C++11 version for real-time high-performance applications and if you're not afraid of pointers.

1.1.2 Support

Warning: Any requests regarding unsupported systems will most likely be ignored.

1. Python3 (Ubuntu LTS excluding EOL versions)
2. MATLAB (> 2016a)
3. C++11 (Ubuntu LTS excluding EOL versions)

1.2 Installation

1.2.1 Python3 Installation

Note: DQ Robotics Python3 is recommended for most users.

Note: DQ Robotics Python3 is distributed as a LGPLV3 licensed package. Interface submodules might have additional licenses.

Note: Had any issues? Report them at the Python-Issue-Tracker.

Installation (Release)

The DQ Robotics Python3 is delivered through PyPI. On a Ubuntu LTS (Excluding EOL versions) system, open a terminal and run:

```
python3 -m pip install --user dqrobotics
```

Warning: The installation will fail for any unsupported system, even if that system has pip.

Installation (Development)

Warning: This package is unstable, which means that the API can suddenly change.

The development version of the library can be installed using the additional `--pre` flag.

```
python3 -m pip install --user dqrobotics --pre
```

Updates

You can get updates with

```
python3 -m pip install --user dqrobotics --upgrade
```

Import

All the code is under the `dqrobotics` module.

```python
from dqrobotics import *
a = DQ(1,2,3,4,5,6,7,8)
print(a)
```

Using with the Robot Operating System (ROS)

The recommended `pip` installation installs the library for a given user and should be visible to any ROS code executed by that user.

Interface modules

Warning: We offer support (on a voluntary basis) for the interface modules but no support whatsoever for the third-party software they interface with. For that, refer to their vendors.

Interface modules for DQRobotics Python3 are installed together with the core module.

Interface with CoppeliaSim (Formely V-REP)

Note: The V-REP interface (also compatible with CoppeliaSim) module uses BSD-licensed code distributed by CoppeliaRobotics. Refer to their license for details.

You do not need to do anything specific to make DQRobotics work with CoppeliaSim. It should work with the default configurations of CoppeliaSim. If it doe s not work out-of-the-box, be sure that the connection port is correctly configured (refer to CoppeliaSimRemoteAPI). The standard way is to have port `19997` configured in your `remoteApiConnections.txt`.

Warning: Known issue: For MacOS users, CoppeliaSim's default remote API port 19997 does not seem to open correctly on application startup. This is a CoppeliaSim issue, not an issue with DQRobotics. Another way way to open the port is to:

1. add `simRemoteApi.start(19997)` to the main script of the scene
2. start the simulation.

After that, the :`DQ_VrepInterface` can be used normally. Note that this way, will you be unable to start and stop the simulation remotely.

The minimal example below will obtain the pose of the *Floor* on the default scene in CoppeliaSim.

```
from dqrobotics import *
from dqrobotics.interfaces.vrep import DQ_VrepInterface

remote_api_port = 19997 # This port needs to be configured correctly in your CoppeliaSim!

vi = DQ_VrepInterface()
try:
    if not vi.connect(19997, 100, 10):
        raise RuntimeError("Unable to connect to CoppeliaSim, be sure CoppeliaSim is␣
↪opened in the default scene "
                           "and that port {} is correctly opened.".format(remote_api_
↪port))

    x_floor = vi.get_object_pose("Floor")
    print("The pose of the floor is {}".format(x_floor))
    print("The translation of the floor is {}".format(translation(x_floor)))
    print("The rotation of the floor is {}".format(rotation(x_floor)))

except KeyboardInterrupt:
```

(continues on next page)

(continued from previous page)

```
    print("Interrupted by user, finishing cleanly.")
except Exception as e:
    print("Exception caught: {}, finishing cleanly.".format(e))
finally:
    vi.disconnect()
```

When it works correctly, the result will be

```
The pose of the floor is 1 + E*( - 0.05k)
The translation of the floor is  - 0.1k
The rotation of the floor is 1
```

Interface with quadprog

Note: The quadprog package is licensed under GPLv2+. Refer to their license for details. The wrapper class `DQ_QuadprogSolver` is licensed under the terms of DQRobotics.

To use the `DQ_QuadprogSolver` (a wrapper of quadprog), you have to install quadprog. To do so, open a terminal and run:

```
python3 -m pip install quadprog --user
```

You can then import `DQ_QuadprogSolver` as follows

```
from dqrobotics.solvers import DQ_QuadprogSolver
```

1.2.2 Matlab Installation

Note: DQ Robotics Matlab is adequate if you want to test your ideas fast while having convenient visualization tools, provided you have access to the Mathworks software.

Note: DQ Robotics Matlab is distributed as a LGPLV3 licensed package. Matlab, however, is not free software and other third-party toolboxes may also not be free.

Note: Had any issues? Report them at the Matlab-Issue-Tracker.

Installation

Assuming that you already have Matlab installed on you computer, download the most recent Matlab toolbox of DQ Robotics here.

After downloading the file dqrobotics-YY-MM.mltbx, where YY-MM stands for the year and month of release, just open it and Matlab should copy the files to the folder `Toolboxes/dqrobotics-YY-MM` in your `$HOME` folder and appropriately set the Matlab path.

To test if the toolbox was installed correctly, just go to the prompt and type

```
>> DQ
ans =
  0
```

If you receive an error instead, it means that the toolbox was not properly installed and you should open an issue here.

Development branch

Those wanting the results of our latest developments can checkout the master branch of the Matlab repository. In order to use DQ Robotics on your MATLAB installation, and supposing you did the checkout at **[PATH_TO_DQ_ROBOTICS_FOLDER]**, just add

```
[PATH_TO_DQ_ROBOTICS_FOLDER]/matlab/
```

and subfolders to your MATLAB path.

Note however, that the development branch is unstable and should not be used in production environments.

Type `help DQ` or `doc DQ` to see the embedded documentation and all available functions.

1.2.3 C++11 Installation

> **Warning:** If terms such as C++11, CMAKE, PPA, and linking are unfamiliar to you, consider using the Python3 or MATLAB version of DQ Robotics instead.

Note: DQ Robotics C++11 is distributed as a LGPLV3 licensed package. Interface packages might have additional licenses.

Note: Had any issues? Report them at the CPP-Issue-Tracker.

Release PPA

The official support is for Ubuntu LTS (Excluding EOL versions) using our Stable-PPA.

```
sudo add-apt-repository ppa:dqrobotics-dev/release
sudo apt-get update
sudo apt-get install libdqrobotics
```

All library updates will be delivered together with your regular Ubuntu updates.

Development PPA

Warning: This repository is unstable, which means that the API can suddenly change.

Warning: **DO NOT** add both PPAs at once. If you already used the Release PPA, you first have to remove it.

```
sudo apt remove libdqrobotics*
sudo add-apt-repository --remove ppa:dqrobotics-dev/release
```

The development PPA targets Ubuntu LTS (Excluding EOL versions) using our Devel-PPA.

```
sudo add-apt-repository ppa:dqrobotics-dev/development
sudo apt-get update
sudo apt-get install libdqrobotics
```

Including

Note: After installation, the DQ Robotics headers can be included without any extra configuration.

You can list up the installed headers with

```
find /usr/include/dqrobotics
```

The headers can be added to your code in the following way

```
#include <dqrobotics/DQ.h>
#include <dqrobotics/robot_modeling/DQ_Kinematics.h>
#include <dqrobotics/robot_modeling/DQ_SerialManipulator.h>
#include <dqrobotics/utils/DQ_Geometry.h>
//Any other headers
```

Linking

The shared objects are installed at /usr/lib and will be found naturally by the linker on Ubuntu. For example, using CMAKE,

```
target_link_libraries(my_binary dqrobotics)
```

Interface packages

Warning: We offer support (on a voluntary basis) for the interface packages but no support whatsoever for the third-party software they interface with. For that, refer to their vendors.

Interfaces of DQ Robotics with other libraries are available as separate packages in the PPA, and they can be listed with

```
sudo apt-get update
apt-cache search libdqrobotics*
```

V-REP Interface Package

Note: The V-REP interface package uses BSD-licensed code distributed by CoppeliaRobotics. Refer to their license for details.

The interface between DQ Robotics and V-REP can be installed with

```
sudo apt-get install libdqrobotics-interface-vrep
```

The following headers will be installed in your system:

```
#include<dqrobotics/interfaces/vrep/DQ_VrepInterface.h>
#include<dqrobotics/interfaces/vrep/DQ_VrepRobot.h>
#include<dqrobotics/interfaces/vrep/robots/LBR4pVrepRobot.h>
#include<dqrobotics/interfaces/vrep/robots/YouBotVrepRobot.h>
```

This interface package also requires linking. Using CMAKE, for example:

```
target_link_libraries(my_binary dqrobotics dqrobotics-interface-vrep)
```

CPLEX Interface Package

The interface between DQ Robotics and CPLEX is header-only and can be installed as follows:

```
sudo apt-get install libdqrobotics-interface-cplex
```

The following header will be installed in your system

```
#include<dqrobotics/solvers/DQ_CPLEXSolver.h>
```

If you are using CPLEX, you have to install, configure, and link to it according to its documentation.

Json11 Interface Package

Note: The Json11 interface package uses MIT-licensed code by Dropbox. Refer to their license for details.

Warning: The Json11 interface package for now has limited functionality and can only import `DQ_SerialManipulator` instances.

The interface between DQ Robotics and Json11 can be installed with

```
sudo apt-get install libdqrobotics-interface-json11
```

The following header will be installed in your system:

```
#include<dqrobotics/interfaces/json11/DQ_JsonReader.h>
```

This interface package also requires linking. Using CMAKE, for example:

```
target_link_libraries(my_binary dqrobotics dqrobotics-interface-json11)
```

Using with the Robot Operating System (ROS)

DQ Robotics C++11 and all interface packages install as system-wide packages, so they can be added to your ROS code using the CMAKE directives shown above.

Building from source in another OS

Warning: There is no support whatosever for other operating systems besides Ubuntu LTS.

You might be able to build from source as long as you have Eigen3, CMake, and a C++11 compatible compiler.

1.3 Class overview

1.4 Basics

Note: We use the mathematical notation of [VA17] unless otherwise stated.

1.4.1 Preliminaries

Consider these important definitions that apply to all following explanations.

The quaternion set is given by

$\mathbb{H} \triangleq \{h_1 + h_2 + h_3 + h_4 \,:\, h_1, h_2, h_3, h_4 \in \mathbb{R}\}$

in which the imaginary units , , and have the following properties:

$\hat{\imath}^2 = \hat{\jmath}^2 = \hat{k}^2 = \hat{\imath}\hat{\jmath}\hat{k} = -1$

The dual quaternion set is given by

$\mathcal{H} \triangleq \left\{h + h' \,:\, h, h' \in \mathbb{H},\ ^2 = 0,\ \neq 0\right\}$

where $^2 = 0$ but $\neq 0$.

1.4.2 Programing

Python3 Basics

Preliminaries

All code in this section expects you to have the followoing import in the beginning of your file

```
from dqrobotics import *
```

Binary Operations

Preliminaries

Suppose you have the dual quaternions

$h_1 = 1 + + + +(1 + ++)$

and

$h_2 = 2 + 2 + 2 + 2 + (2 + 2 + 2 + 2).$

They can be declared in DQ Robotics as

```
h1 = DQ([1,1,1,1,1,1,1,1])
h2 = DQ([2,2,2,2,2,2,2,2])
```

Operations

1. Sum $h_3 = h_1 + h_2$

```
h3 = h1+h2 #Python
```

```
h3 = h1+h2; %Matlab
```

```
DQ h3 = h1+h2; //cpp
```

2. Subtraction $h_3 = h_1 - h_2$

```
h3 = h1-h2
```

3. Multiplication $h_3 = h_1 h_2$

```
h3 = h1*h2
```

C++11 Basics

Preliminaries

All code in this section expects you to have the `include` and `using` in the beginning of your file

```
#include<dqrobotics/DQ.h>

using namespace DQ_robotics;
```

Binary Operations

Preliminaries

Suppose you have the dual quaternions

$h_1 = 1 + + + + (1 + + +)$

and

$h_2 = 2 + 2 + 2 + 2 + (2 + 2 + 2 + 2)$.

They can be declared in DQ Robotics as

```
DQ h1(1,1,1,1,1,1,1,1);
DQ h2(2,2,2,2,2,2,2,2);
```

Operations

1. Sum $h_3 = h_1 + h_2$

```
DQ h3 = h1+h2;
```

2. Subtraction $h_3 = h_1 - h_2$

```
DQ h3 = h1-h2;
```

3. Multiplication $h_3 = h_1 h_2$

```
DQ h3 = h1*h2;
```

1.5 Advanced topics

1.5.1 Vector-field inequalities

Note: This section is based on the results presented in [MAHM19].

Warning: For brevity, the code is shown using Python3. The other versions of the library also have the same methods.

Distance Jacobians

Note: All the distance Jacobians were implemented as `static` methods of the class `DQ_Kinematics`.

To have access to these methods, use

```
from dqrobotics.robot_modeling import DQ_Kinematics
```

As an usage example, suppose that we are using the `KukaYoubot` robot

```
import numpy as np
from math import pi
from dqrobotics import *
from dqrobotics.robot_modeling import DQ_Kinematics
from dqrobotics.robots import KukaYoubotRobot

# Get robot kinematics
robot = KukaYoubotRobot.kinematics()

# Define an arbirary posture
q = np.array([0, 0, 0, 0, pi/2.0, pi/2.0, 0, 0]) # Arbitrary value

# The pose of the robot can be used to retrieve robot primitives
pose = robot.fkm(q)
# The end effector translation (Could be another point)
robot_point = translation(pose)
# Line is the z-axis of the end-effector (Could be another line)
robot_line = Ad(rotation(pose), k_) + E_ * cross(translation(pose), Ad(rotation(pose), k_
↪))
# Plane is normal to the z-axis of the end-effector (Could be another plane)
robot_plane = Ad(rotation(pose), k_) + E_ * dot(translation(pose), Ad(rotation(pose), k_
↪))

# These Jacobians are used to calculate the distance Jacobians
pose_jacobian = robot.pose_jacobian(q)
translation_jacobian = DQ_Kinematics.translation_jacobian(pose_jacobian, pose)
line_jacobian = DQ_Kinematics.line_jacobian(pose_jacobian, pose, k_)
plane_jacobian = DQ_Kinematics.plane_jacobian(pose_jacobian, pose, k_)
```

(continues on next page)

(continued from previous page)

```
# Workspace primitives are calculated with dual-quaternion algebra
workspace_point = 0.5*i_ + 0.2*j_ # A point in (0.5, 0.2, 0.0) in world-coordinates
workspace_line = i_ # The x-axis in world-coordinates
workspace_plane = k_ # Normal to the z-axis in world-coodinates (the x-y plane)
```

Robot-point to point distance Jacobian, $J_{t,p}$

Note: Mathematically defined in Eq. (22) of [MAHM19].

The Jacobian relating the joint velocities with the derivative of the squared-distance between a point in the manipulator and a point in the workspace.

```
result = DQ_Kinematics.point_to_point_distance_jacobian(translation_jacobian, robot_
↪point, workspace_point)
```

Robot-point to line distance Jacobian, $J_{t,l}$

Note: Mathematically defined in Eq. (32) of [MAHM19].

The Jacobian relating the joint velocities with the derivative of the squared-distance between a point in the manipulator and a line in the workspace.

```
result = DQ_Kinematics.point_to_line_distance_jacobian(translation_jacobian, robot_point,
↪ workspace_line)
```

Robot-line to point distance Jacobian, $J_{l,p}$

Note: This method provides a generalized version of Eq. (34) of [MAHM19] to any line in the manipulator.

The Jacobian relating the joint velocities with the derivative of the squared-distance between a line in the manipulator and a point in the workspace.

```
result = DQ_Kinematics.line_to_point_distance_jacobian(line_jacobian, robot_line,␣
↪workspace_point)
```

Robot-line to line distance Jacobian, $J_{l,l}$

Note: This method provides a generalized version of Eq. (48) of [MAHM19] to any line in the manipulator.

The Jacobian relating the joint velocities with the derivative of the squared-distance between a line in the manipulator and a line in the workspace.

```
result = DQ_Kinematics.line_to_line_distance_jacobian(line_jacobian, robot_line,␣
↪workspace_line)
```

Robot-plane to point distance Jacobian, $J_{\pi,l}$

Note: This method provides a generalized version of Eq. (56) of [MAHM19] to any plane in the manipulator.

The Jacobian relating the joint velocities with the derivative of the distance between a plane in the manipulator and a point in the workspace.

```
result = DQ_Kinematics.plane_to_point_distance_jacobian(plane_jacobian, workspace_point)
```

Robot-point to plane distance Jacobian, $J_{p,\pi}$

Note: Mathematically defined in Eq. (59) of [MAHM19].

The Jacobian relating the joint velocities with the derivative of the distance between a point in the manipulator and a plane in the workspace.

```
result = DQ_Kinematics.point_to_plane_distance_jacobian(translation_jacobian, robot_
↪point, workspace_plane)
```

1.6 References

BIBLIOGRAPHY

[MAHM19] Murilo Marques Marinho, Bruno Vilhena Adorno, Kanako Harada, and Mamoru Mitsuishi. Dynamic active constraints for surgical robots using vector-field inequalities. *IEEE Transactions on Robotics*, 35(5):1166–1185, oct 2019. doi:10.1109/tro.2019.2920078.

[VA17] Bruno Vilhena Adorno. *Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra — Part I: Fundamentals*. unpublished, February 2017. working paper or preprint. URL: https://hal.archives-ouvertes.fr/hal-01478225.