# Robot Dynamics with URDF & CasADi

Lill Maria Gjerde Johannessen[1], Mathias Hauan Arbo[1], and Jan Tommy Gravdahl[1]

***Abstract*—Fast, accurate evaluation of the dynamics parameters is a key ingredient for accurate control, estimation, and simulation of robots. As these are time-consuming to compute by hand, a software library for generating the rigid body dynamics symbolically can be of great use for robotics researchers. In this paper, we propose a library to efficiently compute and evaluate robot dynamics and its derivatives. Based on a URDF description of the robot's kinematics, three major rigid body dynamics algorithms are used to retrieve the dynamics symbolically in the CasADi framework. To validate the numerical accuracy, the numerical evaluation of the solutions are compared against three other well-established rigid body dynamics libraries, namely RBDL, KDL, and PyBullet. We conduct a timing comparison between the libraries, and we show that the evaluation times of the symbolic expressions are at most one order of magnitude higher than the evaluation times of the numerical libraries. Last, it is shown that the evaluation times of the dynamics derivatives remain of the same order as the evaluation times of the dynamics expressions.**

***Keywords*—Manipulator Dynamics, Robot Kinematics, Robot Programming.**

## I. Introduction

Defining advanced feedback control techniques for robots requires the use of kinematics and dynamics. Oftentimes one requires both the forward or inverse mapping and its derivatives. These can be tedious to compute by hand, and many symbolic solver systems result in functions that have long evaluation time, making them impractical for use in feedback control.

Robotic Operating System (ROS) [1] is a software solution with a growing community among robotics programmers. ROS presents a Universal Robot Description Format (URDF), an XML file describing the robot's kinematics as a kinematic tree of frames with inertial, collision, and visual properties.

A library for using symbolic equations that is growing in popularity among robotics researchers is CasADi [2]. It is an open-source tool for algorithmic differentiation (AD) and numerical optimization. This framework provides the ability to rapidly prototype optimization algorithms and symbolic equations that are close to production ready.

In this paper, we present *urdf2casadi* (u2c), a software library for obtaining functions of the robot's dynamics that can be used with symbolic expressions in the CasADi framework, based on a URDF description of the robot. The library provides forward and inverse dynamics, as well as the Coriolis and gravitational terms, and the inertia matrix. The library is implemented in Python for a cross-platform functionality, and the functions returned by u2c use CasADi's autogenerated C-code to minimize evaluation time.

The library also represents an opportunity to efficiently obtain the derivatives of the dynamics. In standard approaches for controlling complex robotic systems, such as trajectory optimization and optimal control [3, 4, 5], the system dynamics and their derivatives are a crucial part of the optimization problems. Yet, a large amount of the computational time of the optimization algorithms [6] is spent on computing these derivatives.

There are several approaches for evaluating the derivatives. Finite differences is considered the simplest method, but it is sensitive to rounding errors [6], and is dependent on fine parallelization [7, 8]. Further, the manual process of deriving the derivatives is both complex and error-prone, and the analytic derivatives are often difficult to optimize and implement efficiently. By obtaining the dynamics expressions with u2c, the derivatives are easily obtainable using CasADi's built-in Jacobian functionality. Similar to the use of AutoDiff in [6], CasADi uses AD to obtain the derivatives, and the method for calculating the Jacobian involves advanced algorithms exploiting sparsity and symmetry patterns [2]. The result yields efficient calculation of derivatives, making them appropriate for use in robotics research.

### A. Rigid Body Dynamics Libraries

Due to its importance in robotics research, there are several libraries for generating robot dynamics based on a URDF. The *Kinematics and Dynamics Library* [9] (KDL) contains special object types and functions so that one can take a kinematic chain and evaluate the dynamic parameters, i.e the inertia matrix, the Coriolis matrix, and the gravitational force. The routines are real-time safe and implemented in C++.

The *Rigid Body Dynamics Library* (RBDL) [10] is a C++ library inspired by the algorithms presented in Featherstone's *Rigid Body Dynamics Algorithms* [11]. RBDL provides forward dynamics, inverse dynamics, and the dynamic parameters.

PyBullet [12] is an open-source collision detection and rigid body dynamics library, mainly used for physics simulation, robotics, and deep reinforcement learning. It provides the inverse dynamics, and the dynamic parameters.

The aforementioned libraries are numerical. This restricts the user to the built-in function rather than being able to take as many partial derivatives as necessary for the controller formulation. However, it is chosen to compare u2c against these libraries for several reasons. KDL is a well-established library in the ROS community, PyBullet is widely used within machine learning, and RBDL is, similar to u2c, implemented

[1] Lill Maria Gjerde Johannessen (lmjohann@stud.ntnu.no), Mathias Hauan Arbo, and Jan Tommy Gravdahl are with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

[2] The work reported in this paper is based on activities within centre for research based innovation SFI Manufacturing in Norway, and is partially funded by the Research Council of Norway under contract number 237900.

based on [11]. Further, these libraries all provide Python bindings and URDF loadings, similar to u2c.

### B. Rigid Body Dynamics

The dynamics of a multi-body rigid system can be described by the equation of motion [13]:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})q + G(q) - \sum_i J_i(q)^T f_i^{ext} \tag{1}$$

where $M$ is the inertia matrix giving the relationship between the generalized joint forces, $\tau$, and the generalized joint accelerations, $\ddot{q}$. $C$ is the Coriolis matrix, encompassing the Coriolis effects, and $G$ encompasses the gravitational forces. For brevity, the dependent variables of $M$, $C$, and $G$ are omitted henceforth and the generalized joints are referred to as joints.

The inverse dynamics (ID) is defined as the joint torques required for the joints to produce a desired joint acceleration for a given joint position, velocity, and external forces:

$$\tau = \mathrm{ID}(model, q, \dot{q}, \ddot{q}, f^{ext}). \tag{2}$$

Similarly, the forward dynamics (FD) is defined as the joint acceleration according to a joint position, torque input and external forces:

$$\ddot{q} = \mathrm{FD}(model, q, \dot{q}, \tau, f^{ext}). \tag{3}$$

u2c provides the forward and inverse dynamics, as well as the Coriolis matrix, the gravitational effects and the inertia matrix using three rigid body dynamics algorithms. The *recursive Newton-Euler algorithm* (RNEA) is used to obtain the inverse dynamics, the Coriolis matrix, and the gravitational force. The *articulated body algorithm* (ABA) is used to obtain the forward dynamics, and the *composite rigid body algorithm* (CRBA) is used to obtain the inertia matrix. The algorithms are implemented using spatial algebra, as presented by Featherstone [11].

## II. Spatial Notation

To give insight into the algorithms, this section gives a brief introduction to spatial vector algebra. Spatial vectors are 6D vectors that contain the linear and angular characteristics of rigid body motion and forces. Spatial vector algebra provides a compact notation to study the dynamics of a multi-body rigid system. A spatial vector contains the information of two 3D vectors and thus replace two or more 3D equations. Hence, dynamics algorithms can be derived quickly and expressed in a compact form leading to efficient computer code.

*1) Spatial Transforms:* The placement of an isolated body $B_i$ with a fixed frame $i$ relative to a world frame 0 is described by a spatial transform denoted ${}^iX_0$. When dealing with multi-body rigid systems, one has to consider several bodies connected through joints. The connecting joint thus represents a constraint for the relative placement between the connection bodies. This constraint is expressed through a quantity referred to as the joint motion matrix, denoted $S_i$. When finding the transform from a child body, denoted $\lambda_i$, to its parent body, $i$, one must consider the placement from the parent body to the connecting joint, $X_J$, and the placement from the joint to the child body, $X_T$. The total transform between two bodies are hence given by ${}^iX_{\lambda_i} = X_J X_T$. The quantity $X_T$ is a fixed placement given by the robot's kinematics, while $X_J$ varies with the joint motion constraint represented by $S_i$.

*2) Spatial Inertia:* When a rigid body has mass, the spatial inertia tensor at an origin $O$ is given by:

$$I_O = \begin{pmatrix} \bar{I}_C + mc \times c\times^T & mc\times \\ mc\times^T & m1 \end{pmatrix}, \tag{4}$$

where $m$ is the mass of the body, $c$ is the body's center of mass, 1 is the identity matrix, and $\times$ represent a spatial cross product operator, which is further explained in the latter. The upper left element of the inertia matrix ($\bar{I}_C + mc \times c\times^T$) is the rotational inertia of the body around $O$. An important advantage of spatial inertia is that if one has to find the total inertia of several bodies, it becomes the sum of all rigid body inertias. To illustrate, if two bodies, having inertia $I_1$ and $I_2$, are rigidly connected and form a composite body, then the total inertia becomes $I_{tot} = I_1 + I_2$.

*3) Motion and force vectors:* Featherstone [11] distinguishes between two groups of spatial vectors, namely *motion* vectors and *force* vectors. Spatial velocity and acceleration belong to the motion group, and spatial force and momentum belong to the force group. Quantities of motion vectors are generally denoted by $m$ and quantities of force vectors are denoted $f$. We distinguished between spatial force transforms and spatial motion transforms such that:

$$m_A = {}^AX_B m_B, \tag{5}$$

$$f = {}^AX^* f_B, \tag{6}$$

where $A$ and $B$ represent two Cartesian frames, ${}^AX_B$ represent the motion transform, and ${}^AX_B^*$ represent the force transform. The relationship between the motion and force transform is that one is the inverse transpose of the other:

$${}^BX_A^* = {}^BX_A^{-T}. \tag{7}$$

The spatial inertia can be seen as a mapping between the two groups as the spatial momentum is given by $h_i = I_i v_i$ and spatial force is given by $f_i = I_i a_i + v_i \times^* h_i$. The spatial force thus correspond to the to the time derivative of the spatial momentum.

Motion vectors can also operate on both motion and force vectors through spatial cross product operators. They are similar to classic time derivatives, and it is distinguished between spatial motion and force cross products:

$$\dot{m} = v_A \times m_B, \tag{8}$$

$$\dot{f}_A = v_A \times_B^* f_B. \tag{9}$$

$\times^*$ can be viewed as the dual of $\times$, and their relationship is similar to the relationship between $X$ and $X^*$.

Spatial velocity is defined as the time derivative of the spatial transform. Thus, the spatial joint velocity, denoted $v_J$, is

found by taking the derivative of the aforementioned quantity $X_J$:

$$v_J = \frac{\partial X_J}{\partial q_i}\dot{q} = S_i\dot{q}_i, \tag{10}$$

where $S_i$ is the aforementioned joint motion matrix. $S_i$ is considered time independent for 1-DOF joints.

Last, the joint acceleration is defined as the derivative of the joint velocity:

$$\begin{aligned} a_{Ji} &= S_i\ddot{q}_i + \dot{S}_i\dot{q}_i \\ &= S_i\ddot{q}_i + v_i \times S_i\dot{q}_i \\ &= S_i\ddot{q}_i + v_i \times v_{Ji}. \end{aligned} \tag{11}$$

## III. Rigid Body Dynamics Algorithms

### A. The Recursive Newton-Euler Algorithm

Although various algorithms have been proposed to retrieve ID, RNEA remains the most efficient, whose complexity is $O(n)$ with n being the number of bodies of the robotic system. RNEA was first proposed by [14], and was later renewed by [11] to exploit the advantages of spatial algebra. Compared to [11], u2c explicitly calculates model quantities, i.e ${}^iX_{\lambda_i}$, $S_i$, $I_i$, beforehand in a model calculation routine, and ${}^iX^*_{\lambda_i}$ is found using (7). This accounts for the other algorithm implementations as well. Algorithm 1 shows the compact result of using spatial algebra. As can be observed, RNEA is a two-pass algorithm which propagates the kinematic quantities in a forward pass, followed by retrieving the joint forces in a backward pass.

Although RNEA was originally developed to obtain ID, u2c exploits modifications of RNEA to obtain the Coriolis and gravitational terms as these are subsets of the inverse dynamics problem. ID can be viewed as:

$$ID = \text{RNEA}(model, q, \dot{q}, \ddot{q}, f^{ext}), \tag{12}$$

while $C$ and $G$ can be obtained by:

$$C = \text{RNEA}(model, q, \dot{q}, 0, 0), \tag{13}$$

$$G = \text{RNEA}(model, q, 0, 0, 0). \tag{14}$$

**Algorithm 1** Recursive Newton-Euler Algorithm

**Input:** $X$, $S$, $I$
**Output:** $\tau$

$v_0 = 0$
$a_0 = -a_g$
**for** $i = 1$ **to** $n_B$ **do**
  $v_i = {}^iX_{\lambda_i}v_{\lambda_i} + S_i\dot{q}_i$
  $a_i = {}^iX_{\lambda_i}a_{\lambda_i} + S_i\ddot{q}_i + v_i \times S_i\dot{q}_i$
  $f_i = Ia_i + v_i \times^* Iv_i$ - ${}^iX^*_{\lambda_i}f^{ext}_i$
**end for**
**for** $i = n_B - 1$ **to** 0 **do**
  $\tau_i = S_i^T f_i$
  **if** $\lambda_i \neq 0$ **then**
    $f_{\lambda_i} = f_{\lambda_i} + {}^{\lambda_i}X^*_i f_i$
  **end if**
**end for**

### B. The Articulated Body Algorithm

One of the most efficient algorithms for computing the forward dynamics is ABA, whose complexity is, similar to RNEA, $O(n)$. The algorithm was first presented by [15] although various versions have been proposed since then. ABA does not rely on computing the inverse of the inertia matrix, but is generally more complex than RNEA. It is composed of three main passes, whereas the first pass is a forward recursion collecting the spatial forces acting on the bodies. The quantities obtained in the first pass are used to retrieve the articulated body inertia and the articulated body forces in a backward pass, followed by obtaining the joint and body accelerations in a forward recursion.

### C. The Composite Rigid Body Algorithm

CRBA is used to compute the inertia matrix. The physical interpretation of $M$ is that it relates the force acting on each joint to the acceleration of each joint. By using the definition of the kinetic energy of each body, while treating the bodies as composite rigid bodies, the algorithm recursively calculates each element of the matrix. For further details about CRBA and ABA, we refer to Featherstone's Rigid Body Dynamics Algorithms [11].

CRBA can also be used to solve the forward dynamics problem. [11] presents the equation of motion for a multi-body rigid system as:

$$\tau = M(q)\ddot{q} + C_2(q, \dot{q}, f^{ext}) \tag{15}$$

where $C_2$ encompasses the Coriolis effects, the gravitational force, and the effects of external forces, if any. This quantity can, similar to the Coriolis matrix, be found by a call to RNEA where $\ddot{q} = 0$ and the external and gravitational forces are considered. Thus, FD can be found by solving the equation for $\ddot{q}$:

$$\ddot{q} = M^{-1}(\tau - C_2), \tag{16}$$

where the dependencies of $M$ and $C_2$ are omitted. It should also be mentioned that this method for finding FD has a worst case of $O(nd^2)$ with $d$ being the depth of the kinematic tree. In cases where the kinematic tree contains few bodies, these algorithms can exceed the speed of $O(n)$-algorithms. Thus, this approach has also been implemented in u2c, and both approaches for FD are evaluated in the next section.

## IV. Results

In this section, the performance of u2c with regard to numerical accuracy, efficiency of evaluating the dynamics expressions, and its derivatives, are reported. The tests are performed on a Ubuntu 16.04, 3.5 GHz x 12 Intel Xeon CPU processor in a Python 2.7 environment, and the -OFast compiler flag is used for optimization of the generated C-code. Various robots are used to evaluate the performance: a 2-DOF pendulum, a 6-DOF UR5, and a 16-DOF snake robot.

TABLE I: Numerical differences between libraries for a 6-DOF UR5 robot for 1000 random samples.

| | KDL/u2c | RBDL/u2c | PyBullet/u2c |
|---|---|---|---|
| $G\ (N)$ | $4.42 \cdot 10^{-12}$ | $1.96 \cdot 10^{-07}$ | $1.83 \cdot 10^{-03}$ |
| $C\ (N)$ | $1.03 \cdot 10^{-11}$ | $4.30 \cdot 10^{-07}$ | $3.12 \cdot 10^{-03}$ |
| ID $(Nm)$ | | $4.41 \cdot 10^{-07}$ | $4.12 \cdot 10^{-03}$ |
| $M\ (kgm^2)$ | $1.40 \cdot 10^{-12}$ | $1.46 \cdot 10^{-08}$ | $2.32 \cdot 10^{-03}$ |
| FD $(m/s^2)$ | | $7.88 \cdot 10^{-07}$ | |

### A. Numerical Results

The numerical results for the UR5, displayed in Table I, are obtained by generating 1000 samples of configurations, velocities, and accelerations or torques, uniformly distributed within the joint limits. The result shows that u2c and KDL have very similar results, implying a numerical difference of factor $10^{-15}$ for a single sample of $M$ and $G$. For $C$, the numerical difference is one order of magnitude higher. This increase, from $G$ to $C$, is due to the extra variable that has to be considered in the RNEA calculations, namely $\dot{q}$, when evaluating $C$. One can also observe that the numerical differences between RBDL and u2c, thus also between KDL and RBDL, is of factor $10^{-07}$ and $10^{-08}$. This indicates a numerical difference of at most $10^{-10}$ for a single sample, which can be considered satisfactory. The similarity between KDL and u2c may be due to more similar data types. Further, the numerical results imply a numerical difference of $10^{-06}$ per sample, between PyBullet and the remaining libraries, which may indicate a small inaccuracy for PyBullet. It is speculated that the error is a matter of single versus double precision floats, or a floating point cancellation issue. The developers of PyBullet have been informed about this, but the numerical difference is negligible in most use cases.

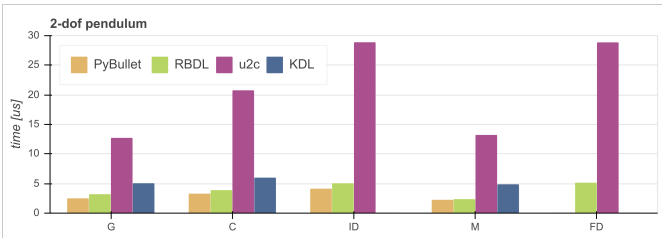
The numerical tests for the 2-DOF pendulum and the 16-DOF snake [16] yielded the same result as for the 6-DOF UR5. Hence, one can assume that the numerical accuracy is independent of the number of DOF.

### B. Timing Results

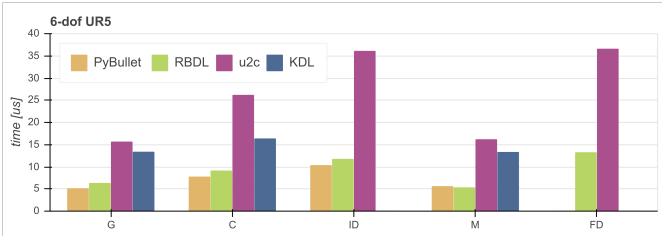
TABLE II: Summary table of number of operations for the dynamics expressions.

| | pendulum | ur5 | snake |
|---|---|---|---|
| G | 18 | 199 | 224 |
| C | 79 | 648 | 572 |
| ID | 110 | 696 | 710 |
| M | 34 | 872 | 1217 |
| FD (CRBA) | 132 | 2354 | 14786 |
| FD (ABA) | 142 | 1948 | 2272 |

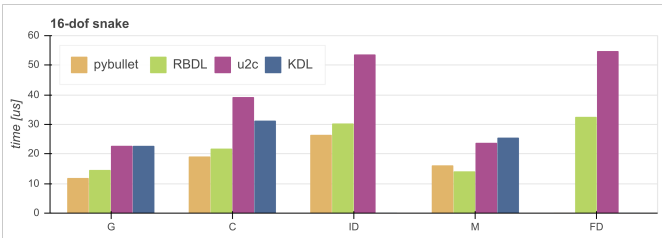
For obtaining the timing results, random configuration, velocity, and acceleration or force vectors are uniformly sampled within the joint limits. Measurements of 1000 samples are stored and the median times spent on evaluating the dynamics are listed in Figure 1. The results show that u2c overall uses a longer evaluation time compared to PyBullet, RBDL, and




(a) Median evaluation times for pendulum.




(b) Median evaluation times for UR5.

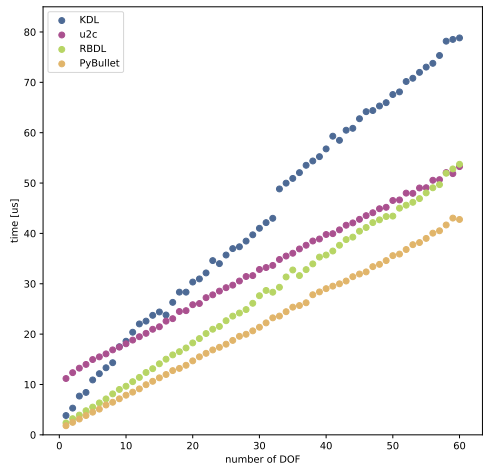



(c) Median evaluation times for snake.

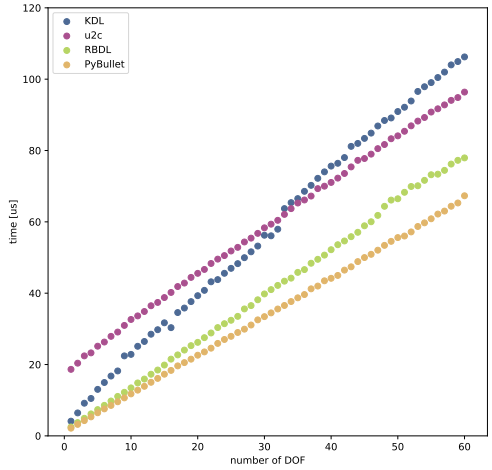
Fig. 1: Median evaluation times for $G$, $C$, ID, $M$, and FD for PyBullet, RBDL, u2c, and KDL.

KDL. This is a natural consequence of u2c generated functions supporting symbolic data types, and thus some overhead is expected. Hence, the functions that only require one symbolic variable, i.e $G$ and $M$, have less overhead than the functions that require two or three symbolic variables.

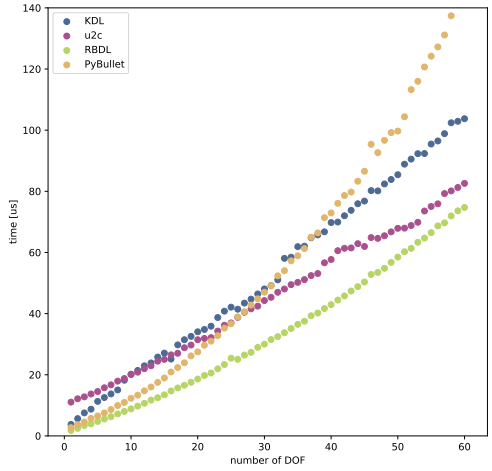
u2c is at most one order of magnitude slower than the numerical libraries, which is the case for the 2-DOF pendulum. While for the 16-DOF snake, u2c and the numerical libraries yield evaluation times of the same order of magnitude. By evaluating Table IV and Figure 1, it is clear that the evaluation times of u2c are mostly affected by the number of symbolic variables, and are less sensitive to the number of operations in the expression. To illustrate, ID and FD(aba) encompass three symbolic variables and are both obtained with $O(n)$-algorithms. FD for the snake requires 1562 more operations than ID. Yet, the difference in evaluation time is only a few microseconds. Hence, it is reasonable to assume that the increase in evaluation time from $G$ and $M$, to $C$, and up to ID and FD is mostly due to the additional symbolic variables required. From Figure 1 it is also seen that the dynamics expressions with a higher number of symbolic variables are more affected by an increase in the number of DOF. This is reasonable as this leads to symbolic vectors with a higher number of elements.




(a) Median evaluation times for $G$.




(b) Median evaluation times for $C$.




(c) Median evaluation times for $M$.

Fig. 2: Median evaluation times for robots of 1 to 60 DOF.

As a result of the CasADi functions being insensitive to an increase in number of operations, and thus also to an increase in DOF, one can observe from Figure 1 that the overall difference in evaluation time between u2c and the numerical libraries decreases as the number of DOF increases. Due to this finding, it was wanted to investigate how the evaluation times evolved for a higher increase in DOF. Thus, we have constructed 60 experimental URDFs and the evaluation times for $G$, $C$, and $M$ from 1 DOF to 60 DOF are shown in Figure 2. For $G$, it is observed that even though u2c starts out as the library with the slowest evaluation times, it ends up being faster than KDL and RBDL. Further, one can observe that the overhead related to an additional variable makes $C$ slower and more sensitive to an increase in DOF, compared to $G$. Yet, u2c still exceeds KDL for robots over 30 DOF and remain the same order of magnitude as RBDL and PyBullet. For $M$ it is shown that u2c exceeds KDL for robots over 10 DOF, and PyBullet for robots over 20 DOF. Hence, we have shown that for the dynamics functions that only encompass a few symbolic variables, the total increase in overhead related to u2c is smaller than the increase in overhead for several of the numerical libraries. This is partly due to the minimal overhead related to the increase in operations, provided by CasADi [2]. For ID and FD, the overhead associated with three symbolic variables prevents u2c from exceeding the numerical libraries, but the evaluation times remain the same order of magnitude as the numerical libraries.

### C. Derivatives Timing

TABLE III: Summary table of median evaluation times for dynamics the derivatives with respect to $q$, $\dot{q}$, $\ddot{q}$ and $\tau$.

| | pendulum | UR5 | snake |
|---|---|---|---|
| derivatives of $G$ | 12.63 µs | 15.38 µs | 23.47 µs |
| derivatives of $C$ | 20.57 µs | 29.62 µs | 46.78 µs |
| derivatives of ID | 28.91 µs | 40.08 µs | 67.68 µs |
| derivatives of $M$ | 12.54 µs | 17.14 µs | 36.72 µs |
| derivatives of FD (crba) | 29.07 µs | 48.35 µs | 286.73 µs |
| derivatives of FD (aba) | 29.10 µs | 50.52 µs | 82.48 µs |

TABLE IV: Summary table of number of operations for the dynamics derivatives.

| | pendulum | UR5 | snake |
|---|---|---|---|
| derivatives of $G$ | 29 | 881 | 2347 |
| derivatives of $C$ | 239 | 5157 | 11792 |
| derivatives of ID | 234 | 4731 | 16159 |
| derivatives of $M$ | 44 | 4095 | 11635 |
| derivatives of FD (crba) | 378 | 17764 | 431178 |
| derivatives of FD (aba) | 390 | 13517 | 40901 |

The derivatives are easily obtained using CasADi's built-in Jacobian functionality, allowing the user to explicitly define which variable one wishes to find the derivative with respect to.

The results, summarized in Table III and Figure 3, show that the evaluation times of the derivatives are not much longer than

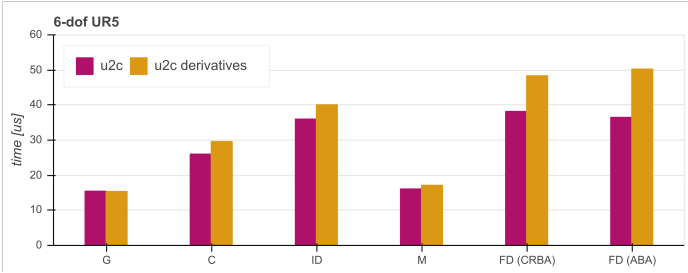



Fig. 3: Median evaluation times of the dynamics and the dynamics derivatives for a UR5.

the dynamics expressions themselves. Figure 3 displays this graphically for a UR5. The evaluation times of the derivatives are the same order of magnitude as the evaluation times of the derivatives, the only exception being the derivative of FD using CRBA for the 16-DOF snake as it must invert a very large inertia matrix.

Table IV shows the number of operations for the dynamics derivatives expressions. As the dynamics functions and their related derivative functions contain the same number of symbolic variables, the change in evaluation time can be assumed to be mainly due to the increase in operations. The evaluation time of the derivative of $G$ for the UR5 increases with 617 operations, which does not remarkably affect the evaluation time. FD using CRBA is the method that is most exposed to an increase in number of operations, as it has a complexity of $O(nd^2)$ and requires three symbolic variables. The derivative of FD using CRBA for the snake increases with 445647 operations, leading to a 217.81 µs increase in evaluation time. This indicates 0.49 µs longer evaluation time for an increase of 1000 operations, and substantiates the fact that the evaluation times of CasADi functions are not heavily affected by an increase in operations. It is the number of symbolic variables involved that is most essential in this matter.

## V. Conclusion

The paper has proposed a software library to efficiently compute the dynamics expressions of a robot based on a URDF description of it's kinematics. To achieve this, we have combined the CasADi framework with the implementation of rigid body dynamics algorithms using spatial algebra. By combining RNEA, ABA, and CRBA, urdf2casadi provides the inverse and forward dynamics expressions, as well as the Coriolis and gravitational terms, and the inertia matrix. Our approach leads to efficient algorithms, resulting in evaluation times comparable to numerical approaches represented by KDL, RBDL, and PyBullet.

By extracting the expressions in the CasADi framework, the dynamics derivatives are easily obtainable using CasADi's Jacobian functionality. We have demonstrated that the evaluation times of the dynamics derivatives are of the same order of magnitude as the evaluation times of the dynamics expressions, thus making them suitable for use within trajectory optimization, optimal control, and other approaches where dynamics derivatives are needed.

We provide the complete open-source Python implementation of this library, thus providing a multi-platform, easily installable library ideal for use in robotics research. More details can be found in Johannessen [17].

## References

[1] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, (Kobe, Japan), May 2009.

[2] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[3] M. H. Arbo, E. I. Grtli, and J. T. Gravdahl, "On model predictive path following and trajectory tracking for industrial robots," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pp. 100–105, Aug 2017.

[4] D. Verscheure, M. Diehl, J. De Schutter, and J. Swevers, "On-line time-optimal path tracking for robots," in *2009 IEEE International Conference on Robotics and Automation*, pp. 599–605, May 2009.

[5] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[6] J. Carpentier and N. Mansard, "Analytical Derivatives of Rigid Body Dynamics Algorithms," in *Robotics: Science and Systems (RSS 2018)*, (Pittsburgh, United States), June 2018.

[7] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913, Oct 2012.

[8] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the HRP-2 humanoid," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3346–3351, Sep. 2015.

[9] R. Smits, "KDL: Kinematics and Dynamics Library." http://www.orocos.org/kdl, 2014. Accessed: 2018-12-10.

[10] M. L. Felis, "RBDL: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, pp. 1–17, 2016.

[11] R. Featherstone, *Rigid Body Dynamics Algorithms*. Springer, 2008.

[12] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning." http://pybullet.org, 2016–2018. Accessed: 2018-12-9.

[13] B. Siciliano and K. Oussama, *Handbook of Robotics*. Springer, 2008.

[14] J. Luh, M. Walker, and R. Paul, "Resolved-acceleration control of mechanical manipulators," *IEEE Transactions on Automatic Control*, vol. 25, pp. 468–474, June 1980.

[15] R. Featherstone, "The Calculation of Robot Dynamics Using Articulated-Body Inertias," *The International Journal of Robotics Research*, vol. 2, no. 1, pp. 13–30, 1983.

[16] I.-L. Borlaug, K. Pettersen, and J. Gravdahl, "Trajectory tracking for an articulated intervention AUV using a super-twisting algorithm in 6 DOF.," *IFAC-PapersOnLine*, vol. 51, no. 29, pp. 311 – 316, 2018. 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018.

[17] L. M. G. Johannessen, "Robot Dynamics with URDF & CasADi," Master's thesis, Norwegian University of Technology and Science, Trondheim, Norway, 2019.