Extending a constrained hybrid dynamics solver for energy-optimal robot motions in the presence of static friction

Djordje Vukcevic

Publisher: Dean Prof. Dr. Wolfgang Heiden

Hochschule Bonn-Rhein-Sieg – University of Applied Sciences,
Department of Computer Science

Sankt Augustin, Germany

May 2018

Technical Report 03-2018

Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

ISSN 1869-5272 ISBN 978-3-96043-063-6

Digital Object Identifier doi:10.18418/978-3-96043-063-6
DOI-Resolver http://dx.doi.org/

Abstract

Friction effects impose a requirement for the supplementary amount of torque to be produced in actuators for a robot to move, which in turn increases energy consumption. We cannot eliminate friction, but we can optimize motions to make them more energy efficient, by considering friction effects in motion computations. Optimizing motions means computing efficient joint torques/accelerations based on different friction torques imposed in each joint. Existing friction forces can be used for supporting certain types of arm motions, e.g standing still. Reducing energy consumption of robot's arms will provide many benefits, such as longer battery life of mobile robots, reducing heat in motor systems, etc.

The aim of this project is extending an already available constrained hybrid dynamic solver, by including static friction effects in the computations of energy optimal motions. When the algorithm is extended to account for static friction factors, a convex optimization (maximization) problem must be solved.

The author of this hybrid dynamic solver has briefly outlined the approach for including static friction forces in computations of motions, but without providing a detailed derivation of the approach and elaboration that will show its correctness. Additionally, the author has outlined the idea for improving the computational efficiency of the approach, but without providing its derivation.

In this project, the proposed approach for extending the originally formulated algorithm has been completely derived and evaluated in order to show its feasibility. The evaluation is conducted in simulation environment with one DOF robot arm, and it shows correct results from the computation of motions. Furthermore, this project presents the derivation of the outlined method for improving the computational efficiency of the extended solver.

Contents

List of Figures

List of Tables

Acronyms

Abbreviation		**Meaning**
ABA	=	Articulated-Body Algorithm
COM	=	Center Of Mass
CORBA	=	Common Object Request Broker Architecture
CRBA	=	Composite-Rigid-Body Algorithm
DART	=	Dynamic Animation and Robotics Toolkit
DOF	=	Degrees Of Freedom
FD	=	Forward Dynamics
GIK	=	Generalized Inverted Kinematics
HD	=	Hybrid Dynamics
HQP	=	Hierarchical Quadratic Problem
ID	=	Inverse Dynamics
iTaSC	=	instantaneous Task Specification and Control
KDL	=	Kinematics and Dynamics Library
LCP	=	Linear Complementary Problem
ODE	=	Open Dynamics Engine
OSF	=	Operational Space Formulation
RBDL	=	Rigid Body Dynamics Library
RNE	=	Recursive Newton-Euler
ROS	=	Robot Operating System
SoT	=	Stack of Tasks
URDF	=	Unified Robot Description Format
WBC	=	Whole Body Control
WBOSC	=	Whole Body Operational Space Control

List of Symbols

$\ddot{\phi}$	Separation acceleration between robot and object in environment
$\ddot{X}$	Cartesian acceleration vector - Spatial vector in Plücker coordinates
δ	Convex function (to be optimized) defined by a task - iTaSC control framework
Λ	Convex set of dissipative forces
μ	Vector of dissipative joint forces - reaction joint forces from friction effects and joint position limits
ν	Lagrange multiplier - Vector of magnitudes of constraint forces
ω	Vector of three angular velocity components
ϕ	Separation distance between robot and object in environment
τ	Vector of force variables in joint space
τ_c	Contact force converted in joint space
$\times, \times^*$	Spatial cross product operators, for motion and force respectively - Plücker coordinates
υ	Vector of three linear velocity components
ζ	Slack variable in optimization problem definitions
A_N	$6 \times m$ matrix of 6×1 unit column vectors of Cartesian constraint forces imposed on segment N
b_N	6×1 vector of acceleration energy setpoint for segment N
C	Control block in iTaSC control diagram
$C(q, \dot{q})$	Bias force vector consisting of Coriolis, centrifugal and gravity forces in joint space
D	Combined inertia matrix of a segment inertia and its associated joint rotor inertia

d	Joint rotor inertia
E	3×3 rotation matrix
e	Error function in SoT task formulation
f	Linear force acting on a body
F_c	Vector of constraint forces - Spatial vector in Plücker coordinates
$H(q)$	Joint space inertia matrix
$J(q)$	Jacobian matrix
K	Feedback gain in iTaSC control scheme
$l(q, x_f, x_u)$	Loop closer function in iTaSC control scheme
$M + E$	Model Update and Estimation block in iTaSC control diagram
N	Null space projection matrix in whole body operational space control framework
n	Angular force (moment) acting on a body
P	Control plant in iTaSC control diagram
P^A	Projection operator for articulated body inertias and forces
Q	Vector representation for residual parts of constraint equation
q, $\dot{q}$, $\ddot{q}$	Position, velocity and acceleration vectors in joint space respectively
r	3×1 position vector
S	Matrix for defining subspace (freedom) of a motion vector
s	Feature vector in SoT task formulation
$s(x, u)$	State function in iTaSC control scheme
T	Contact normal vector, defined in the joint space
u	Control input vector in iTaSC control framework
W	Selection matrix for choosing elements of dissipative force vector
x_f	Feature coordinates in iTaSC control framework
Y	Matrix representation for residual parts of constraint equation
y	Control output vector in iTaSC control framework
Z	Acceleration energy induced in the system - Gauss function

z	Measurements from the sensors in iTaSC control diagram
$(\cdot)^T$	Transpose of a matrix
$(\cdot)^{-1}$	Inverse of a matrix
${}^{i+1}X_i, {}^{i+1}X_i^*$	Matrices for transformation of coordinates for motion and force vectors respectively, from coordinate $\{i\}$ to coordinate $\{i+1\}$
F^{ext}	External force acting on a rigid body - Spatial vector in Plücker coordinates
F_{bias}	Bias force acting on a rigid body (Coriolis, centrifugal and external forces) - Spatial vector in Plücker coordinates
x_u	Various geometric uncertainties in iTaSC control diagram

Chapter 1

Introduction

High energy consumption is one of the problems with industrial manipulators and service robots such as KUKA youBot and Fraunhofer IPA Care-O-bot. Unfortunately, actuation systems of robots consume a lot of energy [1], and one of the reasons are non-efficient motions. Optimization of such motions, in terms of energy consumption, is important for:

- Extending the battery life of autonomous mobile robots.
- Reducing collision impacts and thus ensuring a better safety in robot environment.
- Reducing heat in motor systems, induces by non-optimum force commands.

Static friction in robot joints imposes an additional load on motors. Actuators need to produce a supplementary amount of torque in order to overcome friction effects and move [2]. This, of course, requires more energy from the power supply [3]. One of the solutions for overcoming the problem of energy usage minimization is extending existing motion control algorithms for taking friction forces into account.

A natural way of specifying constrained motions (robot tasks) is the operational space formulation, where desired motion of a robot is defined in Cartesian space. Fortunately, Popov and Vereshchagin in [4], [5] developed domain specific-solver for computing control commands based on imposed constraints. This constrained hybrid dynamic solver represent a great contribution to robotics community. This algorithm can not only compute control commands based on many different task specifications, but it also resolves robot redundancy by computing unique and energy optimal instantaneous motions.

Aforementioned properties of the solver, define a perfect candidate for an algorithm to be extended for computation of energy-optimal robot motions in the presence of static friction. Moreover, the ingenuity of the Popov-Vereshchagin algorithm represents a great motivation for the research conducted in this project.

In [4], Vereshchagin proposed and briefly outlined the approach for including dissipative forces from static friction effects in computations of motions, but without providing a detailed derivation of the approach and elaboration that will show its correctness. Additionally and also in [4], Vereshchagin outlined the idea for improving the computational efficiency of the approach, but without providing its derivation.

This research and development project report presents the complete and detailed derivation of the procedure for integrating static friction factors in the originally formulated Popov-Vereshchagin algorithm. Furthermore, it presents the evaluation of the proposed approach, in order to show its feasibility. An additional section in this report is dedicated for deriving the outlined method for improving the efficiency of the extended algorithm.

The content of this project report is organized as follows. Chapter 2 provides a background knowledge required for understanding the field of robot dynamics. Furthermore, it provides an overview of the state of the art work. That is, the work in fields of friction compensation, computations of energy-efficient motions and whole body - task based robot control. Chapter 3 describes the problem addressed, and the task of this project. In chapter 4, a detailed description of the Popov-Vereshchagin solver is presented. The complete derivation of the extension for this algorithm is presented in chapter 5, along with the derivation of the proposed method for improving the efficiency of the extended algorithm. Evaluation of the proposed approach for integrating static friction factors in computations of motion is described in chapter 6. Finally, the conclusion of this work and aims of the future research are presented in last chapter 7.

Chapter 2

State of the Art

2.1 Robot Dynamics

A robot can be of many types, such as: wheeled robot, manipulator, aerial robot, etc. For us, the most important representative is robot manipulator, which is a robot system that consists of rigid bodies called *links* and they are connected by joints. By connecting two *links*, the *joint* introduces a constraint on relative motion between them, in such a way that constraint force reduces number of allowed direction in which motion can be executed.

Motion of rigid bodies, or to be specific, their forces and accelerations are studied by the field of dynamics. Dynamic equations that describe these motions are evaluated by dynamic algorithms, which perform numerical calculations for variables of interest. In robotics, two main types of problems for which calculations are performed are *forward* and *inverse* dynamics [6].

Here, *forward* dynamics (FD) deals with computation of accelerations that represent reactions on given/applied forces. The equation of motion which is evaluated by FD is expressed as:

$$H(q)^{-1}(\tau - C(q, \dot{q})) = \ddot{q} \tag{2.1}$$

Here, term H represents joint space inertia matrix and q, $\dot{q}$, $\ddot{q}$ are position, velocity and acceleration vectors in joint space, respectively. C represents the bias force vector which consists of Coriolis, centrifugal and gravity forces in joint space, but also additional(external) forces which may be imposed on the system [6]. Finally τ represent vector of force variables in joint space.

On another side, *inverse* dynamics (ID) deals with computation of forces which need to be generated in the robot system, in order to result in desired/given accelerations. The equation of motion which is evaluated by ID is expressed as:

$$H(q)\ddot{q} + C(q, \dot{q}) = \tau \tag{2.2}$$

Additional problems for which dynamic algorithms are used include: *hybrid* dynamics (HD) and identification of the inertial parameters for particular robot system [7]. Here, *hybrid* dynamics deals with calculation of unknown accelerations and forces, based on already available/given force and acceleration values for particular joints.

Different types of dynamic algorithms are used for finding solutions for aforementioned problems:

- *Recursive Newton-Euler* algorithm (RNE) used for solving *inverse* dynamics problem.
- *Articulated-Body Algorithm* (ABA) used for solving *forward* dynamics problem.
- *Popov-Vereshchagin* algorithm used for solving *hybrid* dynamics problem [8].
- *Articulated-Body Hybrid Dynamics Algorithm* used for solving *hybrid* dynamics problem and it represents adaptation/extension of aforementioned ABA [6].
- *Composite-Rigid-Body Algorithm* (CRBA), a *forward* dynamics solver used for computation of joint-space inertia matrix of a robot system [9].

Constraints imposed on a robot system define the number of allowed motion directions. They can be artificial (e.g. desired motion specified by the task definition) and physical (e.g. contact between two bodies). Artificial constraints in terms of task specification will be covered in section 2.4. However, contact constraint defines that two bodies must not penetrate when a contact between them occurs [6], and it is expressed as inequality constraint of the form:

$$\phi(q) \geq 0. \tag{2.3}$$

This equation defines separation distance ϕ between two objects, such that the distance must be equal or grater than 0 in order to avoid penetration.
For describing a motion of the rigid body, which is in contact with another object, equation 2.2 is expanded in form of:

$$H(q)\ddot{q} + C(q,\dot{q}) = \tau + T\nu \tag{2.4}$$

Here, term ν defines magnitude (initially unknown) of the contact force and T represent contact normal vector, defined in the joint space. Finally, complete expression $T\nu = \tau_c$ represent contact force (converted in joint space) that impose this type of constraint on the system. For resolving these constraints and finding solution for the control input, the problem can be formulated in two ways, as *linear complementary problem* (LCP) and as *quadratic program.*
In LCP, which is expressed in the form:

$$\ddot{\phi} = Y\nu + Q, \ \ddot{\phi} \geq 0, \ \nu \geq 0, \ \ddot{\phi}^T\nu = 0, \tag{2.5}$$

the aim is to find the vector ν. Here, $\ddot{\phi}$ represents separation acceleration, and matrix Y and vector Q represent residual parts of the constraint equation. Properties of this problem formulation are such that a unique solution exist if matrix Y is positive definite. However, if matrix Y is not positive definite, than unique solution is not guaranteed, and furthermore infinitely many solutions may exist or there may not be solution at all [6].
On another side, with formulation as *quadratic program*, defined as [10]:

$$minimize \quad \frac{1}{2}(\ddot{q} - H^{-1}(\tau - C))^T H(\ddot{q} - H^{-1}(\tau - C)) \tag{2.6}$$

$$\textit{subject to} \qquad T^T\ddot{q} + \dot{T}^T\dot{q} \geq 0,$$

a unique solution is guaranteed, if one exist. Furthermore, this procedure for computing the solution is less computationally expensive. This formulation represent application of the *Gauss' least constraint principle* for resolving inequality constraints in domain of rigid body dynamics.

Today many researchers use simulation and modelling approaches to contribute research and development in robotics. Due to many capabilities, simulators can be used for predicting energy consumption and validation of control solutions for robots, before using it in real environment [11]. Among many existing software solutions, the most used physics engines are [11]: MuJoCo physics engine [12], ODE (Open Dynamics Engine) [13], DART (Dynamic Animation and Robotics Toolkit) [14], Bullet physics engine [15], SimBody [16], etc. Aforementioned simulation libraries are using various dynamics solvers, and most used ones are: *Recursive Newton-Euler* (RNE) and Featherstone *articulated-body* algorithms. Additionally, some of them include solutions for handling contacts and collisions with environment, and also friction in the joints.

Apart from simulators, other important software solutions are the ones which are used for the control of real robots. The best representative are following open source softwares:

- Kinematics and Dynamics Library (KDL): used for constructing kinematic chains of the robots and additionally for computation of robot motions [17]. Beside kinematics algorithms, library also includes implementation of two inverse dynamic algorithms: *Recursive Newton-Euler* solver and *Popov-Vereshchagin* solver. Moreover, this library can be used as standalone software tool or even in connection with Robot Operating System (ROS) [18] framework.
- Rigid Body Dynamics Library (RBDL) [9]: contains implementation of three dynamic algorithms: *Recursive Newton Euler Algorithm*, *Composite Rigid Body Algorithm* and *Articulated Body Algorithm.* Additionally, library includes algorithms for solving forward and inverse kinematics problems, as well as contact handling problems.
- Other examples include: Pinocchio [19] and JRL-RBDyn [20] libraries.

2.2 Friction compensation

Several strategies have been developed for compensating friction, in diverse control applications for robotics [2], [21]. Presented approaches consider various ways to counteract the friction effects in control systems, such as:

- By estimating friction parameters off-line.
- Model-based adaptive algorithms applied for on-line estimation of friction parameters.
- The ones which are not based on a particular friction model, but using fuzzy, neural, and genetic algorithms to estimate parameters.

Nevertheless, in order to account for positioning errors and stick-slip effects, and improve motion of robots, in all of aforementioned methods reaction friction forces have been identified and included in calculations of control commands.
Many different controller designs have been developed, and the most important ones are inverse dynamics control schemes [2], [22]. In this case, compensation is performed by applying force/torque commands which are greater, but opposite to the friction forces. The goal is the prediction of reaction forces from friction effects and computation of opposed control inputs, in order to ensure correct execution of a desired motion. Nevertheless, none of the controllers involve any optimization process in computations of motion.

2.3 Minimizing energy consumption

In order to improve manufacturing systems' efficiency, researches have developed many different strategies to decrease energy consumption of manipulators [1]. For us, the most important methods are the ones which consider developing algorithms for computation of energy-efficient motions [23]. We can classify them as the algorithms which deal with trajectories optimizations and algorithms which deal with optimization of instantaneous motions.

- Many approaches for solving energy-optimal motion problems, are considering various methods for optimizing trajectory cost functions. They used Newton and quasi-Newton optimization algorithms [24], [25], or even sequential quadratic programming methods [26], where trajectories were parametrized in terms of B-splines functions.
  Here, for computing exact analytic gradients and Hessians, formulations of dynamic equations of the motion are based on Lie group theory.
  Moreover, in [3] authors were also including static friction forces in cost function among other dynamic effects, and also using gradient based methods for optimizing it.
- A different, but important approach is presented in [27], where time/energy optimal path (trajectory) tracking problem has been reformulated as convex

optimal control problem. Furthermore, authors have also accounted for static friction forces in the equation of motion.

- Other strategies are considering genetic algorithms [28], [29] for optimizing the trajectories.
- Minimization of energy can be also considered as criterion for resolving redundancy in control of highly redundant robots. In order to find unique instantaneous motion, authors in [30] developed an approach that is based on Gauss' least constraint principle. For minimizing the Gauss' function, authors' method relies on generalize inverse technique, where inertia matrix is used for weighting. The output of this approach are accelerations which produce energy optimal motions. Nevertheless, the authors did not consider friction effects in computation of motions.
- For computations of instantaneous energy-optimal motions, the most important representative is Popov-Vereshchagin hybrid dynamics solver [4]. The algorithm is based on Gauss' least constraint principle (as the cost function to be optimized) [8]. The linear-time dynamics solver computes optimal control commands, based on the imposed motion constraints, but does not consider friction effects.

2.4 Whole body - task based control approaches

Highly redundant robot mechanisms with multiple end-effectors, such as humanoid robots, require specification of control tasks based not only on the behaviour of a single end-effector, but also on motions of other parts of the robot, such as legs, head, torso, etc. [31]. Several different approaches were considered [32]–[34] for resolving whole body - task based control problems with highly redundant robots.

2.4.1 Whole Body Operational Space Control

In order to model and enable task-oriented whole body control of the robots, the operational space formulation (OSF) approach was introduced in [32] and later on extended to whole body operational space control (WBOSC) framework [31], [35]–[38]. This framework establishes prioritized control hierarchy among three different control categories: 1) constraint-handling tasks (e.g. contacts, 2) avoiding near-body objects, 3) joint-limits and self-collisions), operational tasks (i.e manipulation and locomotion) and posture tasks (e.g. maintaining balance) [37].

To ensure the safety of the robot and environment, the approach defines constraints at the highest level of hierarchy, where the secondary control categories (operational tasks) are projected in the constraint null-space and the posture is controlled within residual null space of the robot, defined by the operational task (remaining degrees of freedom). This methodology formulates tasks dynamics in such way that prevents violation (conflicting) of the higher priority tasks by the lower priority tasks, and additionally enables runtime monitoring of the robot's task (behavior) feasibilities [31], [36], [37]. For example, if unexpected obstacle occurs in the preplanned trajectory of the robot, the controller will detect the task in-feasibility before the collision occurs, and based on user input or strategy, the task can be modified to avoid obstacle (without violating the constraints) or stop the motion of the robot. The unique characteristic of a prioritized hierarchy enables the controller to detect the in-feasibility of a certain task, by checking if the *Jacobian* of the respective task (J_t) has become singular or not, in the computations of the instantaneous motions [36].

The operational space formulation computes the desired joint torques for the given task (constraint-handling, operational or posture task) based on externally provided force information [39].
The command torque can be expressed in terms of robot's joint space dynamics:

$$H(q)\ddot{q} + C(q, \dot{q}) = \tau \tag{2.7}$$

This torque can be also defined by the force transformation [40]:

$$\tau_t = J_t^T F_t \tag{2.8}$$

The desired task is defined as an acceleration vector $\ddot{X}_t$ and the force required for the task execution is formulated as:

$$F_t = I_t \ddot{X}_t + F_{bias,t} \tag{2.9}$$

Where I_t is operational space inertia matrix, $F_{bias,t}$ is bias force vector. Finally, the general formulation for the controller of robot's dynamic behaviour in the related task space is:

$$\tau_t = J_t^T (I_t \ddot{X}_t + F_{bias,t}) \tag{2.10}$$

τ_t can represent the control command for any type of task, from the set of constraint, operational and posture tasks.

The hierarchical control of whole body is defined such that the complete command torque τ is decomposed in different torque vectors:

$$\tau = \tau_{constraints} + N_{constraints}^T (\tau_{tasks} + N_{tasks}^T \tau_{postures}) \tag{2.11}$$

Where $\tau_{constraints}$, τ_{tasks}, $\tau_{postures}$ represent torque commands for handling/controlling constraints, operational tasks and postures respectively. $N_{constraints}$ and N_{tasks} are the null space projection matrices.

$\tau_{constraints}$ is defined as:

$$\tau_{constraints} = J_{constraints}^T F_{constraints} \tag{2.12}$$

Where $J_{constraints}$ is the Jacobian which maps constraint forces from Cartesian space into joint space, and $F_{constraints}$ is a vector of constraint forces specified in Cartesian space.

τ_{task} is defined as:

$$\tau_{tasks} = J_{tasks}^T F_{tasks} \tag{2.13}$$

Each robot's end-effector has its associated Jacobian matrix and J_{tasks} is a matrix constructed from the Jacobian matrices of all end-effectors. An operational task specifies a force vector for each end-effector and F_{tasks} is the matrix composed of all defined force vectors [35].

$\tau_{postures}$ is defined as:

$$\tau_{postures} = J^T_{postures} F_{postures} \tag{2.14}$$

Where $F_{postures}$ is the force vector used for commanding the desired posture, and $J_{postures}$ is the posture Jacobian. One example for robot posture command can be the specification of the torso position and orientation in Cartesian space.

The null space projection matrices $N_{constraints}$ and N_{tasks}, enable *dynamic consistency* of the lower level tasks, with the respect to higher priority tasks. The concept of dynamic consistency is a property which guarantees that lower level task behavior will be executed without dynamically effecting the higher level task [31].

2.4.2 *ControlIt!* framework

ControlIt! is an open source software framework which provides a software solution for implementation of whole body controllers in simulation and on real robot [41]. It is integrated with ROS middleware (Hydro and Indigo) [18] and enables its usage on different robot platforms.

The current implementation supports only the Whole Body Operational Space Control (WBOSC) analytical solver, but due to its plugin-based modular architecture [41], it enables future addition/implementation of other types of whole body control algorithms [41].

The software architecture is divided in three main components: configuration, whole body controller (WBC) and hardware abstraction layer (robot interface and clock), where the configuration component includes: robot model, set of prioritized tasks and constraint set.

The robot model uses the Rigid Body Dynamics Library (RBDL) [42] for providing the robot's kinematic and dynamic properties, such as inertia matrix and bias forces [41]. The RBDL software itself, provides efficient algorithms for both forward and inverse dynamics: Recursive Newton Euler (RNE), Composite Rigid Body (CRBA) and Articulated Body (ABA) [9], [42].

The set of prioritized tasks includes goal configurations of posture and desired operational tasks. On another side, a constraint set does not define a command, but it defines null spaces in which prioritized tasks are allowed to be executed. The library can work with two types of constraints: transmission and contact. Here, transmission constraints defines those imposed when a single motor rotates multiple joints [43]. In order to include a new robot platform, a user should specify a URDF model of the robot and write task and constraint plugins. The remaining components of the library are already predefined and platform-independent. This methodology imposes more flexible modification requirements on the software, for including new robot configurations, compared with previous WBOSC software frameworks such as Stanford-WBC and UTA-WBC.

Furthermore, the design of *ControlIt!* provides the ability for working with many

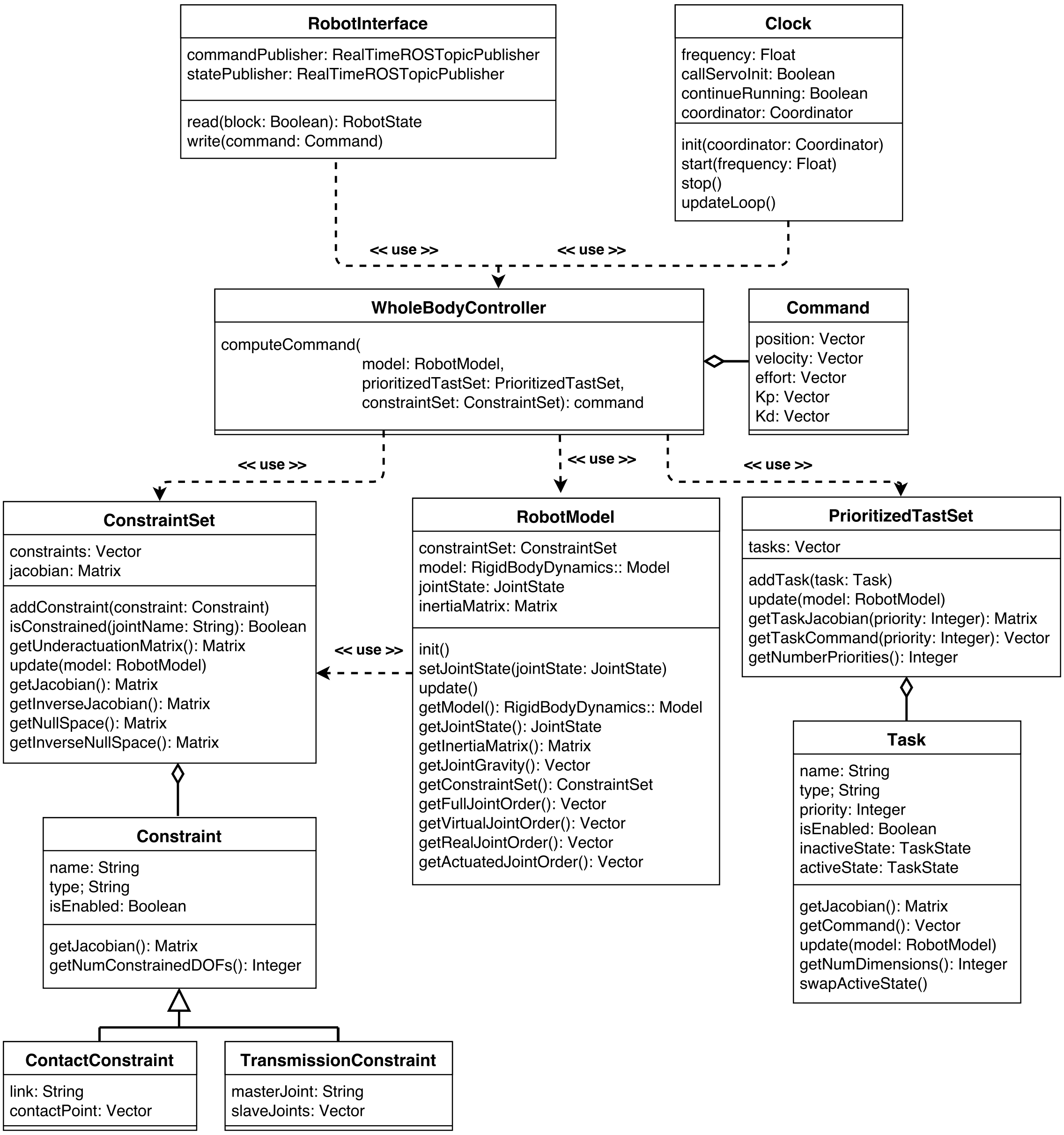
RobotInterface
commandPublisher: RealTimeROSTopicPublisher
statePublisher: RealTimeROSTopicPublisher
read(block: Boolean): RobotState
write(command: Command)
Clock
frequency: Float
callServoInit: Boolean
continueRunning: Boolean
coordinator: Coordinator
init(coordinator: Coordinator)
start(frequency: Float)
stop()
updateLoop()
<< use >>
<< use >>
WholeBodyController
computeCommand(
model: RobotModel,
prioritizedTastSet: PrioritizedTastSet,
constraintSet: ConstraintSet): command
Command
position: Vector
velocity: Vector
effort: Vector
Kp: Vector
Kd: Vector
<< use >>
<< use >>
<< use >>
ConstraintSet
constraints: Vector
jacobian: Matrix
addConstraint(constraint: Constraint)
isConstrained(jointName: String): Boolean
getUnderactuationMatrix(): Matrix
update(model: RobotModel)
getJacobian(): Matrix
getInverseJacobian(): Matrix
getNullSpace(): Matrix
getInverseNullSpace(): Matrix
<< use >>
RobotModel
constraintSet: ConstraintSet
model: RigidBodyDynamics:: Model
jointState: JointState
inertiaMatrix: Matrix
init()
setJointState(jointState: JointState)
update()
getModel(): RigidBodyDynamics:: Model
getJointState(): JointState
getInertiaMatrix(): Matrix
getJointGravity(): Vector
getConstraintSet(): ConstraintSet
getFullJointOrder(): Vector
getVirtualJointOrder(): Vector
getRealJointOrder(): Vector
getActuatedJointOrder(): Vector
PrioritizedTastSet
tasks: Vector
addTask(task: Task)
update(model: RobotModel)
getTaskJacobian(priority: Integer): Matrix
getTaskCommand(priority: Integer): Vector
getNumberPriorities(): Integer
Task
name: String
type; String
priority: Integer
isEnabled: Boolean
inactiveState: TaskState
activeState: TaskState
getJacobian(): Matrix
getCommand(): Vector
update(model: RobotModel)
getNumDimensions(): Integer
swapActiveState()
Constraint
name: String
type; String
isEnabled: Boolean
getJacobian(): Matrix
getNumConstrainedDOFs(): Integer
ContactConstraint
link: String
contactPoint: Vector
TransmissionConstraint
masterJoint: String
slaveJoints: Vector

Figure 2.1: Class diagram for software structure of ControlIt! framework (figure based on [41])

different types of robot's software components [41]. The framework can use abstract tasks descriptions from libraries such as iTaSC [44], or even preplanned motion trajectories from planning libraries such as MoveIt! [45]. Beside its ability to work with many other high level libraries, the framework is also designed to work with lower level software components which provide hardware drivers, in order to ensure realization of complete software package for controlling the robots.
The existing limitation of this software library is the fact that it does not include definitions of inequality constraints in its components. Furthermore, it does not include implementations of other WBC solvers, which can resolve motions based on inequality constraints, by using optimization techniques.

2.4.3 *Stack of Tasks* - Theory

The task-based control scheme called *Stack of Tasks* (SoT) was introduced in [33], [46] and extended in [47]–[51]. The framework is used for hierarchical control of redundant manipulators and humanoid robots, where the motion can be specified with both equality and inequality constraints [52]. It can be used for both types of control, based on kinematic and also on dynamic properties of the robots. In order to enable hierarchical stack of tasks, authors follow the methodology where the lower priority tasks are recursively projected in the remaining motion space of a higher priority tasks.

In contrast to the WBOSC framework described in previous section, the SoT framework have introduced intermediate control level for computing motion errors. In this approach the task is formulated by the error function, for which controllers must ensure that converges to 0. The error function is defined as:

$$e = s - s^* \tag{2.15}$$

Where s^* represent desired feature and s represent its current value. All tasks are mapped to equality constraints and following this approach, motions are computed by resolving them.

The examples of tasks or equality constraints can be desired values for the control of positions/velocities/accelerations of the certain robot part in the joint or operational space. On another side, for inequality constraints examples are avoidance of singularities and collisions, joint limits and visual servoing tasks.

The initial control algorithms used in the *Stack of Tasks* framework were based on analytical solvers introduces in: Generalized Inverted Kinematics (GIK) approach [53], [54] and also Operational Space [32] approach described in previous section, for kinematic and dynamic based control, respectively. In order to address the common issue with both aforementioned approaches, the problem of computing desired motion based on imposed inequality constraints, Mansard et. al [33] developed analytical method based on activation matrices. Activation matrix activates each task based on an inequality constraint value in a certain time step, but this approach cannot cope with constraints imposed by the contact forces.

Later on, they introduced a new, more generic approach for prioritization of tasks and resolving both types of constraints. Approach is based on hierarchical quadratic problem (HQP) were constraints are defined as quadratic programs in priority order sequence [48]. A typical quadratic program consists of a cost function, which should be minimized, and which is subject to task-specific or physical constraints [50]. In order to solve recursively each minimization problem, they use a domain independent HQP active search algorithm [51], [55]. Additionally for computing solutions they introduced slack variables in the constraint equations, in order to transform from inequality to equality form of constraints. The hierarchy in each quadratic program is defined as [50]:

$$\min_{u_t,\zeta_t} ||\zeta_t||^2 \tag{2.16}$$

$$\begin{aligned} \textit{subject to}: \quad & Q_{l,t-1} \leq Y_{t-1}u_t + \zeta^*_{t-1} \leq Q_{h,t-1} \\ & Q_{l,t} \leq \quad Y_t u_t + \zeta_t \quad \leq Q_{h,t} \end{aligned}$$

Where ζ represent slack variables and ζ^* a constant value of a slack variable computed in the previous iteration for higher level constraint(task). Q_l and Q_h are lower and upper limits of the constraints, respectively. The vector u represent the control input, which itself consists of joint torque or acceleration values computed by the QP solver. Matrix Y is used to express the residual part of the constraint equation, in such way that together with limits Q_h and Q_l represents a complete constraint formulation. The subscripts t and $t-1$ define level of quadratic program in hierarchy solved by the algorithm, where the order is defined as: $(1, ..., t-1, K)$. The 1 represents index of highest level and K index of the lowest level in hierarchy.

This generic formulation allows a user to define equality constraints, by setting both upper and lower limits of the equation to be equal, $Q_h = Q_l$.

2.4.4 *Stack of Tasks* - Implementation

The implementation of the SoT control framework is provided as open-source library [46], [52], which can used with the ROS middleware and a CORBA communication system.

The software library consists of entities, where each entity is receiving input signals and sending output signals. An input signal can be used for requesting a data from the entity or to call some of the entity's methods, in such a way that certain computations are then performed. Output signals are used for sending information (data) to other entities.

The software solution includes different types of entities, such as: *Feature*, *Task*, *Dynamic* and *Solver*. The first type of entity called *Feature* provides environment and robot states, as feature vectors s and s^*. The *task* entity defines an error function $e = s - s^*$ based on the feature vectors provided by the *Feature* entity. Additionally it is also used for defining constraints imposed on the robot motion. *Dynamic* entity is provided for defining kinematic chains of robots, based on data from the file. It also used for computation of forward kinematics and Jacobians of end-effectors, center of mass (COM) positions and inertial matrix. Additionally this type of entity uses external libraries such as Pinocchio [19] and JRL-RBDyn [20] for incorporating inverse dynamics algorithms: Recursive Newton Euler (RNE) and Articulated Body (ABA). Finally, a *Solver* type of entity consists of hierarchical task solver for computing motion commands based on the imposed tasks/constraints in priority order.

The software provides the *Factory of entities* for a user to load a predefined and create a new entities, but also to call all methods required for motion computations.

2.4.5 *iTaSC*

The control framework called *iTaSC* was introduced in [34], [56] and later *on* extended in [57], [58], where the name *iTaSC* stands for "instantaneous Task Specification and Control".

In order to control robot systems with multiple sensors, the authors have defined the approach where the task is represented by relative motion, with possible dynamic interaction among objects that are part of robot system or the environment. With this framework, a task programmer describes the task by specifying constraints on forces and relative motions between objects. In another words, a task can be defined by imposing constraints on output variables of the control system. Examples for such variables are: pose of the robot's end-effector with respect to a laser scanner, or the orientation of an object with respect to camera mounted on the robot.

For expressing tasks/constraints, the authors have introduced two types of coordinates in their framework, feature and uncertainty coordinates, both with respect to object and feature frames. Here, feature coordinates are used to represent relative motions or interactions between features on the specific objects. On another side, uncertainty coordinates are used for representing geometric uncertainties, which can be involved due to disturbances in the environment, calibration the errors in the robot arm, etc. The uncertainties, included in this approach, represent major difference in comparisons with other approaches. By using these coordinates, authors have reduced errors in the task execution.

The control scheme developed by the authors is shown in figure 2.2. Here, the environment and the robot system are represented by *Plant* P, and signal u represents the control input (joint positions, velocities or torques). Signal X_u represent various geometric uncertainties, y represent aforementioned output variables of the system and signal z describes measurements from the sensors, such as joint positions, image and laser data, etc.

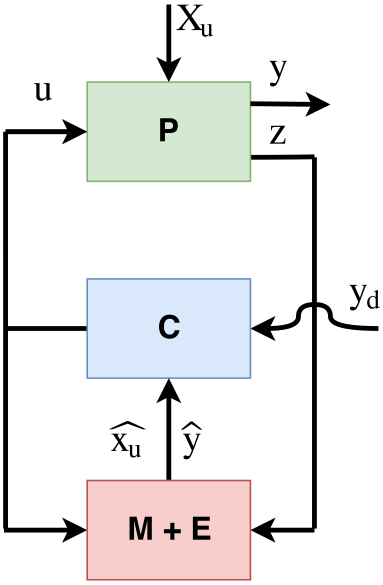

Figure 2.2: Control scheme of *iTaSC* framework (source: [57])

The control input signal u is distributed from the *Control* block C. To produce such signal this block uses data from estimates of disturbances (uncertainties) $\widehat{x_u}$ and outputs $\widehat{y}$, but also desired output values from the signal y_d.
Finally, estimates $\widehat{x_u}$ and $\widehat{y}$ are produced in *Model Update* and *Estimation* block $M + E$ [34], by using data from input signal u and measurement z.
Control laws (equations) for computing desired motions of the robot system are derived in velocity-based manner [57]:

- Robot system equation defines how the robot state is changing given control input u:
$$\dot{x} = s(x, u). \tag{2.17}$$
Here, the x is a vector consisting of joint positions and velocities, $x = (q, \dot{q})^T$

- Equation for the system output is defined as function of joint q and feature coordinates x_f:
$$y = f(q, x_f). \tag{2.18}$$

- The constraint equation is defined as:
$$y = y_d. \tag{2.19}$$
By differentiating this equation with respect to time, resulting equation is:
$$\dot{y} = \dot{y_d} + K(y_d - y). \tag{2.20}$$
Here the K represent feedback gain, which is used to compensate for various disturbances.

- Loop closer equation defines how joint positions q, feature coordinates x_f and geometric uncertainties x_u depend on each other:
$$l(q, x_f, x_u) = 0. \tag{2.21}$$

In initial approach for the *iTaSC* framework, inequality constraints were not included in computations of robot motions, but later on, researches have reformulated approach for computing control values as convex optimization problem. Examples for inequality constraints are joint position and velocity limits, distance of the robot to the object in the environment, etc.
In order to express their approach for computing control input u, authors have presented a generalized formulation of optimization problem [57], [58]:

$$u = arg\ min \quad \delta(\zeta_1, \zeta_2, \zeta_3) \tag{2.22}$$

$$\begin{aligned} subject\ to \quad \dot{x} &= s(x, u) \\ y &= f(q, x_f) \\ 0 &= l(q, x_f, x_u) \\ y &= y_d + \zeta_1 \\ \frac{\mathrm{d}^i y}{\mathrm{d}t^i} &\leq y_{i,max} + \zeta_2 \\ \frac{\mathrm{d}^i y}{\mathrm{d}t^i} &\geq y_{i,min} + \zeta_3 \end{aligned}$$

Here, objective function δ represent any convex function which user defines in order to impose a specific task. For finding the solution of this problem, the authors have introduced the auxiliary(slack) variables ζ_1, ζ_2 and ζ_3 in formulation of the problem. Variables $y_{i,max}$ and $y_{i,min}$ represent upper and lower limits of respective constraints. An overall task can be defined as hierarchical set of sub-tasks, with indexes from 1 to n, in a way that task with index 1 is first on hierarchy. In general, the idea is to compute solution for task $i + 1$, such that it does not introduce errors in execution of the tasks with indexes from the 1 to i.

In order to define the kinematic chain of the robot system, authors use *Kinematics and Dynamics Library* (KDL), and for computing correct estimates $\widehat{x_u}$ and $\widehat{y}$ from sensor data the *Bayesian Filtering Library* (BFL) is used. Both aforementioned libraries and *iTaSC* framework are part of *Orocos* open source project.

As reader can infer, the theoretical part of the *iTaSC* concept does include inequalities constraints in the task definition. However, the implementation itself only includes equality constraints [59].

2.4.6 Final Comparison

The following table represents a comparison of approaches in field of whole body - task based robot control. The comparison is made in terms of:

- Class of the solver which is used for computing required control commands.
- Type of constraints used for task or motion specification.
- Type of control commands computed by the solver.

Table 2.1: Comparison of solvers (approaches) in domain of task-based robot control

	Class	**Constraints**	**Control interface**
WBOSC	Analytical solver	Equality	Torque
Stack Of Tasks	Numerical-based	Equality & Inequality	Velocity & Torque
iTaSC	Numerical-based	Equality & Inequality	Velocity

Chapter 3

Problem formulation and task description

In the research field of friction compensation and for the case of robot systems, many different controller designs have been considered for ensuring correct execution of the desired motion. As presented in section 2.2, the goal in these approaches is the prediction of reaction forces from friction effects and the computation of opposed control inputs. Nevertheless, none of the controllers involve any optimization of control commands in terms of energy consumption.

Existing dynamics solvers such as Recursive Newton-Euler (RNE) or Articulated Body Algorithm (ABA) [6], are computing required or resulting torques/accelerations in joints, based on desired values of these joint quantities and external forces applied on segments. Since they only handle *unconstrained* instantaneous motions, they do not perform any optimization and do not take friction factors into account [6], [25]. On the other hand, different types of dynamics optimization solvers such as domain-specific Popov-Vereshchagin [4], iTaSC [34] and Stack of Tasks [49] algorithms are performing optimization processes for re-solving the imposed motion constraints. However, these algorithms also do not take friction forces into account, while computing the instantaneous robot's motions.

The aim of this project is extending already available Popov-Vereshchagin hybrid dynamics solver [8], by taking static friction effects into account, for computation of energy efficient robot motions. In other words, optimizing instantaneous robot motions based on not only task specifications but also static friction reaction forces.

The existing algorithm is derived from the Gauss' principle of least constraint, and it efficiently (with O(n) complexity) computes a solution to the minimization problem, in order to resolve the imposed constraints on the robot motion. When the Gauss' principle is extended with static friction factors, or more specifically combined with the principle of maximum dissipation, an additional convex optimization (maximization) problem must be solved.

However, the procedure of integrating static friction effects in motion computations, as an extension to the originally formulated Popov-Vereshchagin algorithm, has been briefly outlined by the author of the solver in the paper [4]. Nevertheless, Vereshchagin did not provide a detailed theoretical and experimental elaboration that will show the correctness of the approach.

Finally, the task of this project is the complete and detailed derivation of the procedure for integration of static friction effects in motion computations. Furthermore, the task includes the evaluation of the proposed approach in order to show its feasibility.

Chapter 4

Popov-Vereshchagin hybrid dynamics solver for operational-space control

In the related work section 2.4, the importance of task/constraint based control where complete dynamics properties of the robot are taken in account, has been presented. In order to control a robot in the described manner, it is required to compute control commands which will resolve the constraints imposed by the task definition. Fortunately, in the 1970' researchers [4], [5], [60], [61] have developed a hybrid dynamic algorithm which computes required control commands based on the defined constraints, the robot model, the feedforward joint torque and the external forces from the environment.

4.1 Solver description

The solver is based on a well-known principle of mechanics - *Gauss principle of least constraint* [30] and provides the solution to a hybrid dynamics problem with linear-time, $O(n)$ complexity [8]. The principle states that the true motion (acceleration) of a system/body is defined by the minimum of a convex and even quadratic function that is subject to linear geometric motion constraints [30], [62]. A result from the Gauss function represents the *acceleration energy* of a body, which is defined by the product of its mass and squared distance between its allowed (constrained) acceleration and its free (unconstrained) acceleration [63]. In our case, motion constraints are Cartesian acceleration constraints on the robot's end-effector. This domain-specific solver minimizes the acceleration energy by performing computational (outward and inward) sweeps along robot kinematic chain [8]. Furthermore, by computing the minimum of the Gauss function, the Popov-Vereshchagin solver resolves the problem of kinematic redundancy, when partial control commands are provided [4].

The Gauss function that is minimized by the solver is formulated as:

$$\min_{\ddot{q}} \quad Z(\ddot{q}) = \sum_{i=0}^{N}\{\frac{1}{2}\ddot{X}_i^{\,T} I_i \ddot{X}_i + F_{bias,i}^T \ddot{X}_i\} + \sum_{j=1}^{N}\{\frac{1}{2} d_j \ddot{q}_j^{\,2} - \tau_j \ddot{q}_j\} \tag{4.1}$$

$$\begin{aligned} subject\ to: \qquad & \ddot{X}_{i+1} = {}^{i+1}X_i \ddot{X}_i + \ddot{q}_{i+1} S_{i+1} + \ddot{X}_{bias,i+1} \\ & A_N^T\ \ddot{X}_N \ = \ b_N \end{aligned}$$

where term Z represents acceleration energy of the complete mechanical system [30], and the unit of this quantity is *acceleration times force* $[\frac{Nm}{s^2}]$. Here, $\ddot{X}$ denotes a spatial 6×1 vector of acceleration for particular robot segment (link). This vector contains both linear and angular components defined in Plücker coordinates [6]. A more detailed description of spatial vector representation in Plücker coordinates is presented in appendix section A.1. Matrix I represent rigid-body inertia of a segment and F_{bias} defines a spatial bias force vector for a particular segment. Term d denotes joint rotor inertia, while variable τ represents feed-forward torque, defined by the tasks (e.g. posture control) and/or friction force in the joint. Here, the matrix ${}^{i+1}X_i$ performs the transformation of the coordinates for the previously defined motion vector $\ddot{X}$, from coordinate $\{i\}$ to coordinate $\{i+1\}$ [6]. The term S represents the motion subspace matrix, that defines the motion freedom of a particular joint, based on its physical constraints. Term A_N is a $6 \times m$ matrix, which contains 6×1 unit column vectors, such that each column defines the *direction* of a single constraint force imposed on segment N [8]. Finally, b_N denotes the $m \times 1$ vector of desired acceleration energy for segment N.

In order to compute the required control commands based on the imposed constraints, the original formulation (equation 4.1) of optimization problem was translated to following form [4], [8]:

$$\min_{\ddot{q},\ \nu} \quad Z(\ddot{q}) + \nu^T A_N^T \ddot{X}_N, \tag{4.2}$$

using the method of Lagrange multipliers [64]. Here, the unknown term ν represents a variable called *Lagrange multiplier.*
The approach of Popov and Vereshchagin used for deriving the algorithm includes reformulation of the optimization problem (minimization of the Gauss function), into a discrete optimal control problem [4]. In the next step of the solver derivation, the authors have translated iteratively re-formulated equation 4.2 into a recursive algorithm, using Bellman's principle of optimality [65].
The resulting dynamics solver is computing a *unique motion*, namely *joint accelerations* $\ddot{q}$, as solution to constrained hybrid dynamics problem. Based on derived motion, additional quantities are calculated, namely Cartesian accelerations $\ddot{X}$ and magnitudes of constraint forces (Lagrange multiplier) ν [6]. Furthermore, a complete spatial vector of imposed constraint force F_c can be computed [8], from following relation:

$$F_c = A_N \nu. \tag{4.3}$$

4.2 Algorithm Representation

For computing a solution to the hybrid dynamics problem, the Popov-Vereshchagin solver requires following inputs:

- A *robot model*, defined by kinematic structure of robot system, mass and rigid body inertia parameters of each segment, and inertia parameters of each joint rotor.
- *Joint angles* at current time instant.
- *Joint velocities* at current time instant.
- *Feedforward joint torques.*
- *Spatial acceleration* of robot base segment at current time instant.
- *External forces* applied on robot system.
- *Desired unit constraint forces* defined for end-effector link.
- *Desired acceleration energy* for end-effector link.

In order to compute resulting instantaneous robot motion, based on provided inputs, the algorithm is performing three computational sweeps along kinematic chain [8]:

1. The first, namely *outward* sweep is governing computation of position, velocity, *bias* acceleration and *rigid body bias* force vectors for each robot segment. Here, *bias acceleration* of a link represent a change of link velocity if no external forces are applied on that particular body [6]. In other words, existence of this acceleration is influenced by *Coriolis and centrifugal* effects only. While the *bias force* vector is defined by the Coriolis, centrifugal and external forces imposed on each segment without accounting for gravity forces.
2. The second, namely *inward* sweep is governing computation of *articulated body* quantities, namely inertia and bias force, but also acceleration energy contributions, from *articulated body* bias force, feedforward torque and unit constraint force. Here, articulated body[1] inertia and articulated body bias force of each segment are computed by summing (assembling/composing) its rigid body values with the *apparent* inertias and *apparent* bias forces of its children segments along kinematic chain, respectively [6].
3. The last, also *outward* sweep is governing computations of resulting joint torques and accelerations, and also computations of resulting spatial accelerations for each segment.

Additionally, after completing the recursion in the second (inward) sweep, the solver is computing magnitudes of the imposed constraint force. In more detail,

---

[1] For more detailed explanation on articulated, apparent and rigid body quantities, reader can refer to *Rigid body dynamics algorithms* book, by Roy Featherstone, 2008. [6]

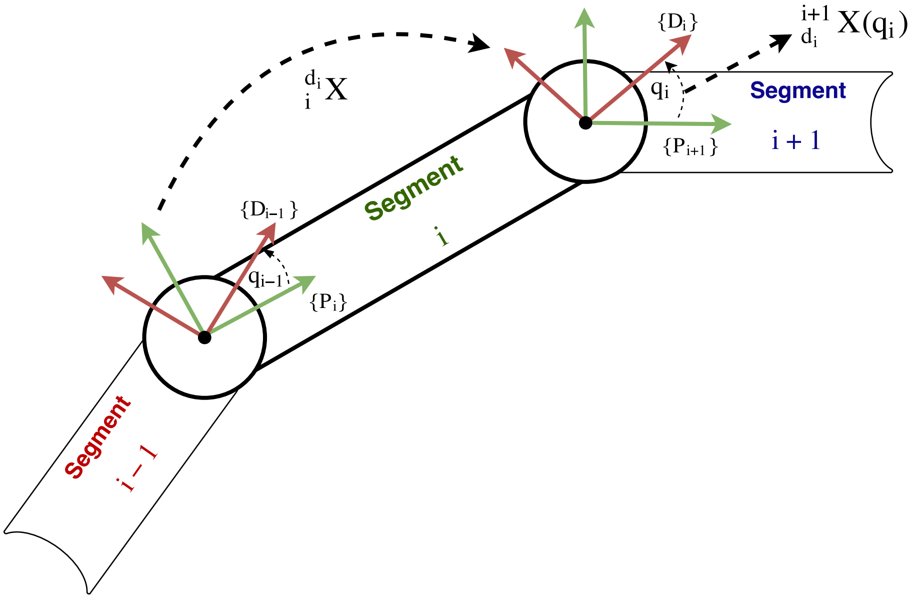

Figure 4.1: Assignment of segment frames and transformations between them for generic kinematic chain.

this operation is performed when the algorithm reaches link $\{0\}$, namely the base segment.

Finally, computations performed in the last sweep represent *final outputs* from the Popov-Vereshchagin solver.

The complete algorithm is formulated as [4], [5], [8]:

Algorithm 1: Constrained Hybrid Dynamics Solver

```
Input  : Robot model, $\mathbf{q}$, $\dot{\mathbf{q}}$, $\tau$, $\ddot{\mathbf{X}}_\mathbf{0}$, $\mathbf{F_{ext}}$, $\mathbf{A_N}$, $\mathbf{b_N}$
Output: $\tau_{\mathbf{control}}$, $\ddot{\mathbf{q}}$, $\ddot{\mathbf{X}}$
begin
    // Outward sweep of position, velocity and bias components
    for i = 0 to N − 1 do
        ${}^{i+1}_{i}X = ({}^{d_i}_{i}X \;\; {}^{i+1}_{d_i}X(q_i))$;
        $\dot{X}_{i+1} = {}^{i+1}X_i\dot{X}_i + S_{i+1}\dot{q}_{i+1}$;
        $\ddot{X}_{bias,i+1} = \dot{X}_{i+1} \times S_{i+1}\dot{q}_{i+1}$;
        $F_{bias,i+1} = \dot{X}_{i+1} \times^* I_{i+1}\dot{X}_{i+1} - {}^{i+1}X_0^* \; F^{ext}_{0,i+1}$;
        $F^{A}_{bias,i+1} = F_{bias,i+1}$;
        $I^{A}_{i+1} = I_{i+1}$;
    end
    // Inward sweep of inertia, force and acceleration energy contributions
    for i = N − 1 to 0 do
        $D_{i+1} = d_{i+1} + S^{T}_{i+1}I^{A}_{i+1}S_{i+1}$;
        $P^{A}_{i+1} = 1 - I^{A}_{i+1}S_{i+1}D^{-1}_{i+1}S^{T}_{i+1}$;
        $I^{a}_{i+1} = P^{A}_{i+1}I^{A}_{i+1}$;
        $I^{A}_{i} = I^{A}_{i} + \sum {}^{i}X^{T}_{i+1}I^{a}_{i+1}\,{}^{i}X_{i+1}$;
        $F^{a}_{bias,i+1} = P^{A}_{i+1}F^{A}_{bias,i+1} + I^{A}_{i+1}S_{i+1}D^{-1}_{i+1}\tau_{i+1} + I^{a}_{i+1}\ddot{X}_{bias,i+1}$;
        $F^{A}_{bias,i} = F^{A}_{bias,i} + \sum {}^{i}X^{*}_{i+1}F^{a}_{bias,i+1}$;
        $A_i = {}^{i}X^{T}_{i+1}P^{A}_{i+1}A_{i+1}$;
        $U_i = U_{i+1} + A^{T}_{i+1}\{\ddot{X}_{bias,i+1} + S_iD^{-1}_{i}(\tau_{i+1} - S^{T}_{i}(F^{A}_{bias,i+1} + I^{a}_{i+1}\ddot{X}_{bias,i+1}))\}$;
        $\mathcal{L}_i = \mathcal{L}_{i+1} - A^{T}_{i+1}S_{i+1}D^{-1}_{i+1}S^{T}_{i+1}A_{i+1}$;
    end
    // Balance of acceleration energy at the base ({0} link)
    // Computation of constraint force magnitudes
    $\nu = \mathcal{L}_0^{-1}(b_N - A_0^T\ddot{X}_0 - U_0)$;
    // Outward sweep of resulting torque and acceleration
    for i = 0 to N − 1 do
        $\ddot{q}_{i+1} = D^{-1}_{i+1}\{\tau_{i+1} - S^{T}_{i+1}(F^{A}_{bias,i+1} + I^{A}_{i+1}({}^{i+1}X_i\ddot{X}_i + \ddot{X}_{bias,i+1}) + A_{i+1}\nu)\}$;
        $\ddot{X}_{i+1} = {}^{i+1}X_i\ddot{X}_i + \ddot{q}_{i+1}S_{i+1} + \ddot{X}_{bias,i+1}$;
    end
end
```

In presented algorithm, the pose of the link $\{i+1\}$ with respect to link $\{i\}$, namely ${}^{i+1}_{i}X$,is computed by the expression defined in line 4. Here, for a pose of particular segment we refer to pose of its proximal frame. This quantity is calculated by composition of 1) transformation ${}^{d_i}_{i}X$ (homogeneous matrix), between proximal and distal frames of segment $\{i\}$ and 2) transformation ${}^{i+1}_{d_i}X$ between distal frame of link $\{i\}$ and proximal frame of link $\{i+1\}$ (see figure 4.1). Latter homogeneous transformation matrix is a function of respective joint position q_i.

Expression in line 5 calculates spatial velocity vector $\dot{X}_{i+1}$ for segment {i+1}, given spatial velocity values of segment {i} and joint velocity {i+1} noted as $\dot{q}_{i+1}$. Here, ${}^{i+1}X_i$ represent matrix which performs transformation of coordinates for motion vector, namely from coordinate $\{i\}$ to coordinate $\{i+1\}$ [6]. A more detailed description of this matrix is presented in appendix section A.2. Term S represents motion subspace matrix, such that it defines motion freedom of a particular joint, based on its physical constraints.

Next equation (line 6) computes *bias* acceleration vector $\ddot{X}_{bias,i+1}$ for link {i+1}, given its velocity vector and rate of joint {i+1}.

In next step (line 7), the solver computes spatial *rigid body bias* force vector $F_{bias,i+1}$ for segment $\{i+1\}$, given its velocity vector $\dot{X}_{i+1}$ and its *rigid body* inertia matrix I_{i+1}. Note, that all *external forces* F^{ext}, imposed on kinematic chain are assumed to be measured in frame $\{i+1\}$, but represented in reference frame of robot base - $\{0\}$. For performing this operation, term ${}^{i+1}X_0^*$ is used and it represent matrix which performs transformation of coordinates for force vector, from coordinate $\{0\}$ to coordinate $\{i+1\}$. A more detailed description of this matrix is presented in appendix section A.2.

For computation of previous quantities (lines 6 and 7), cross product operators in Plücker coordinates $\dot{X}\times$ and $\dot{X}\times^*$ are used. A more detailed description of these terms is presented in appendix section A.3.

The gravity effects are modelled by setting acceleration of the base link ($\ddot{X}_0$) equal to gravitational acceleration [8]. By treating gravity effects in this manner, rather than defining as external forces, a better efficiency of the algorithm is enabled [6].

Expressions in lines 8 and 9 are initializing *articulated body* inertia and *articulated body* bias force variables respectively with *rigid body* values. This procedure is required for following calculations of complete articulated body quantities during the inward sweep of the algorithm. Equation defined in line 13 computes combined inertias of segment {i+1} and its associated joint rotor $\{i+1\}$.

Matrix P_{i+1} defined in line 14, is a projection operator for articulated body inertias and forces [8]. In more details, this matrix projects articulated body inertias and forces of segment {i+1} to its associated joint subspace.

In next two steps of inward sweep (lines 15 and 16), the solver is computing articulated body inertia I^A of segment {i}, where I^a represent apparent inertia assembled recursively from contributions of child segments.

Similarly to the previous computations, an articulated body, as in this case segment {i} (line 18), must also account for bias forces transmitted from its child segments. These forces are computed in one quantity (variable) defined as apparent bias force F^a_{bias} (line 17).

The expression in line 19 computes unit constraint forces that are felt (propagated) on segment {i} due to constraints imposed on the end-effector segment. Note that in this step the magnitude of constraint forces ν is still unknown.

The amount of acceleration energy U_i produced by bias forces, and also by existing feed-forward torque, has been recursively computed by the expression presented in line 20, where $U_N = 0$.

In order to understand the reason for computation of next quantity, a reader should note the following elaboration: "*The inward recursion keeps track of how much of the desired constraint acceleration energy b_N is already generated by the (external and inertial) forces, and by the applied joint torques*" [8]. This means that the aforementioned *contribution* of acceleration energy must not be additionally induced by the constraint forces, which will be computed in following balance equation.

Similarly to the previous computations, the solver also computes constraint *coupling matrix* [8] $\mathcal{L}_i$ in line 21, a term which defines acceleration energy induced by unit constraint forces, where $\mathcal{L}_N = 0$. It is important to note that defined constraint coupling matrix $\mathcal{L}$ can become rank deficient [8]. Conditions in which this situation can occur include cases when a robot has less $DOFs$ available than required by a task (specified via Cartesian acceleration interface in this case). A common example of this situation is the case when a robot is in singular configuration at current time step. To remedy singularity problem, in [8] Shakhimardanov proposed *damped least square* method for finding inverse of constraint coupling matrix $\mathcal{L}$.

The magnitude of constraint forces (at the end-effector) ν can finally be calculated by energy balance equation in line 25.

The solution for the originally formulated problem in equation 4.1 is finally derived during the second (last) outward sweep, where the accelerations and control torques for each joint are computed. The motion $\ddot{q}$ calculated in expression (line) 28 represent true acceleration of the constrained system. From this equation, we can infer three different torque quantities which (combined) represent complete torque required for executing resulted motion of the robot system. Namely, if we reformulate aforementioned expression we can see that resulting (control) torque consists of: feedforward torque that is given as input joint force τ_{i+1} (defined by the task and/or friction forces), bias torque used to accommodate for bias forces and constraint torque used to accommodate for the constraint forces (imposed by a task on the end-effector) felt on each joint (see equation 4.4).

$$\ddot{q}_{i+1} = D_{i+1}^{-1}\{\overbrace{\underbrace{\tau_{i+1}}_{\tau_{ff}} - \underbrace{S_{i+1}^T A_{i+1}\nu}_{\tau_{constraint}} - \underbrace{S_{i+1}^T(F^A_{bias,i+1} + I^A_{i+1}({}^{i+1}X_i\ddot{X}_i + \ddot{X}_{bias,i+1}))}_{\tau_{bias}}}^{\tau_{control}}\} \tag{4.4}$$

The last computation (line 29) in the algorithm deals with finding resulting (complete) spatial acceleration $\ddot{X}$ of each segment in kinematic chain.

4.3 Computation of required joint forces and energy optimal motions

As previously mentioned in section 4.1, the solver is computing robot accelerations $\ddot{q}$ that minimize the acceleration energy [30], [62]. However, in other literature [4], [8], [66] for representing the result of the Gauss function, instead of acceleration, energy the term “Zwang” (degree of constraint) is also used. By following definition of the Gauss principle, the computed *constrained* motion (accelerations) of a system will be the closest possible motion to the *free* (unconstrained) motion [63].

This free motion is characterized (executed) by already existing or applied forces in a system. These forces come under τ_{bias} and τ_{ff} terms in equation 4.4. Here, the τ_{bias} stands for the bias forces that are projected (felt) in the joints. That is, forces that come from external factors, e.g gravity or any other external forces, and additionally stands for the forces that exist due to centrifugal and Coriolis effects. On the other hand, the term τ_{ff} stands for feedforward torques that are determined beforehand and applied in a system to compensate for certain effects.

However, a constrained motion is characterized (executed) by all three force contributions, namely τ_{bias}, τ_{ff} and $\tau_{constraint}$. As defined by the Gauss principle of *least constraints*, the true motion will be the one which is influenced by the *least constraint* - "Zwang". Or in other words, a motion which is affected by the least constraint force. In the case of robot manipulators, this force $F_c(\nu)$ is projected in joint space, and it is defined by $\tau_{constraint}$ term in equation 4.4.

The definition of the Gauss principle give us an insight how the Popov-Vereshchagin solver is deciding on required forces that will execute specified (desired) motion, or in other words, execute a task. Namely, by optimizing acceleration energy (degree of constraint), the solver is minimizing possible magnitudes of constraint forces that will influence the execution of a motion which is as close as possible to the free motion. This means that the Popov-Vereshchagin algorithm is taking advantage of already existing forces (bias and feedforward) in a system when computing motion of a robot. More specifically, it takes advantage in such way that computes the *least*, or in other words, the minimum required constraint forces ($\tau_{constraint}$), along with existing ones (τ_{bias} and τ_{ff}), to ensure correct execution of a desired (constrained) robot motion.

The statement that Popov-Vereshchagin solver is computing energy-optimal robot motions is supported by the following elaboration. As previously described in section 4.2, by minimizing acceleration energy, or in other words the degree of constraint, the solver is minimizing possible magnitudes of constraint forces required to execute desired motion (a task). That is, it takes advantage of existing forces (τ_{bias} and τ_{ff}) in a robot system, while computing motion commands. More specifically, it computes the minimum (least) required *additional* joint forces ($\tau_{constraint}$) to be executed (produced by robot motors), along already existing forces in joints, for correct execution of a task. In terms of robot *motor drives*, this means that controllers need to induce the least required (potential) energy (from any type of source, e.g. electrical) in the motors, which will at the end be converted into mechanical energy, for producing additional constraint torque. This contribution of joint torque that comes from motor drives, together with exiting bias and feedforward joint torques, will correctly execute the desired motion with least energy consumption.

On the other side, a different elaboration about computation of energy optimal motion is presented. In [30], Bruyninckx and Khatib have stated that result of minimizing acceleration energy, defined by the Gauss principle, is equal to minimum of instantaneous kinetic energy of the system. Namely, analytical solution to the constrained optimization problem, defined by the general Gauss principle [67]:

$$\min_{\ddot{q}} \quad Z = [\ddot{q} - \ddot{q}_{free}]^T H(q)[\ddot{q} - \ddot{q}_{free}] \tag{4.5}$$

$$subject\ to: \qquad A\,\ddot{q} \;=\; b,$$

can be found using a weighted generalized inverse approach [30]. The derived result is equivalent to the solution of kinematic redundancy problem, for the inverse velocity and acceleration kinematics type of robot control. More specifically, the equivalence can be seen when the *mass weighted* pseudo inverse of a Jacobian matrix is found using the *minimum of kinetic energy* criterion for redundancy resolution [5], [68]. Namely, the constrained optimization problem for redundancy resolution in terms of minimum kinetic energy is defined as [68]:

$$\min_{\dot{q}} \quad E_{kinetic} = \frac{1}{2}\dot{q}^T H(q)\dot{q}, \tag{4.6}$$

$$subject\ to: \qquad \dot{X} \;=\; J(q)\dot{q},$$

Finally, the obtained solution of both constrained optimization problems (equations 4.5 and 4.6) that is, the same generalized inverse of Jacobian matrix for both cases, takes the following form:

$$J_H^{+} = (JH^{-1}J^T)^{-1}JH^{-1} \tag{4.7}$$

For more detailed derivation of resulted Jacobian matrix, equation 4.7, a reader can refer to following literature [30] and [68].
Moreover, the statement and its elaborations that Popov-Vereshchagin solver is computing energy-optimal robot motions also holds for the extended algorithm, which will be presented in chapter 5.

4.4 Interface for task specification

The important feature of Popov-Vereshchagin hybrid dynamics solver is its ability to take task definitions directly as an input, for computing desired motions. Additionally to the robot model parameters, an input to this solver can be specified by three different task - control specifications:

- Tasks defined by *physical* and *virtual* external forces on each segment via F_{ext}.
- Cartesian acceleration constraint-handling tasks specified for the end-effector via $A_N^T \ddot{X}_N \;=\; b_N$.
- Tasks defined by feedforward joint torques via τ.

Task specification: $\mathbf{F_{ext}}$

The first type of task definition can be used for *Cartesian impedance control* [69]. Here, desired *virtual* forces are modelled (along with existing physical), for producing compliant motion which will ensure safe interaction between robot end-effector and environment [70].

Task specification: $\mathbf{A_N^T \ddot{X}_N = b_N}$

The second type of task specification can used for dealing with *physical* constraints such as contacts with environment [8], or *virtual* constraints that specify desired operational accelerations of end-effector, such as manipulation and/or locomotion. In order to use this part of solver's interface, a user should define A_N, a $6 \times m$ matrix of unit constraint forces and b_N, a $m \times 1$ vector of acceleration energy setpoints. Here, the number of constraints m, or in another words number of unit constraint forces is not required to always be equal to 6, which means that a human programmer can leave some of degrees of freedom unspecified [4], and still produce valid control commands. For example, if we want to constrain motion of the end-effector only in one direction, namely x direction, we can define constraint as [8]:

$$A_N = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad b_N = \begin{bmatrix} 0 \end{bmatrix} \tag{4.8}$$

Note that here, first three rows matrix A_N represent linear elements of unit force and last three elements represent angular elements. By giving zero value to acceleration energy setpoint, we are defining that the end-effector is not allowed to have linear acceleration in x direction. Or in other words, we are restricting a robot from producing any acceleration energy in specified directions.
Another example includes the specification of the constraints in 5 $DOFs$. We can constrain the motion of the robot's gripper, such that it is only allowed to move in linear y direction, without performing angular motions:

$$A_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad b_N = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.9}$$

The last example involves specification of desired motion (accelerations) in all 6

$DOFs$:

$$A_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad b_N = \ddot{X}_N \tag{4.10}$$

In this case, we are can directly assign values (magnitudes) of spatial acceleration 6×1 vector $\ddot{X}_N$ to the 6×1 vector of acceleration energy [8]. Despite the fact that physical dimensions (units) of these two vectors are not the same, the property of matrix A_N (it contains *unit* vectors), permits that we can assign values of desired accelerations to acceleration energy setpoints, in respective directions. Namely, each column of matrix A_N has value of 1 in respective direction in which constraint force works, thus it follows that value of acceleration energy setpoint is the same as value of acceleration, in respective direction.
Note that in [8], Shakhimardanov has extended the original algorithm (1) in order to account for constraints imposed on all segments, not only on last end-effector link.

Task specification: τ

The last type of task definition which can be given as an input to the solver, is via feedforward joint torque τ. This part of interface can be used for posture tasks (e.g. maintaining balance) [8], [37], or for other forward dynamics problems/tasks such as simulations of robot behaviours. Additionally feedforward torque interface can be used for including static friction torques imposed on each joint, as it will be described in following chapter 5.

As already mentioned in section 2.1, a *hybrid* dynamics solver computes initially unknown accelerations and forces, based on already available/given force and acceleration values for particular joints/links. Following this definition, the output interface of Popov-Vereshchagin solver consist of three quantities:

- Control (resulting) torques in the joints → $\tau_{control}$.
- Resulting accelerations of the joints → $\ddot{q}(\tau_{control})$.
- Resulting accelerations of the segments (links)→ $\ddot{X}(\ddot{q})$

Computed forces and motions (accelerations) are used as solutions for two different types of dynamics problem. Namely, resulting joint torques $\tau_{control}$ are used for control purposes, and represent solution to the *inverse* dynamics problem. On the other side, joint and segment accelerations, namely $\ddot{q}$ and $\ddot{X}$ provide solution to the *forward* dynamics problem and these quantities can be used for both control and simulation purposes.

Using Popov-Vereshchagin solver, a user can achieve many types of high-level tasks. In other words, various controllers can be implemented *around* interfaces of the algorithm. Examples can be controllers for hybrid force/position control, impedance [71], join velocity control, etc (see figure 4.2) [70].

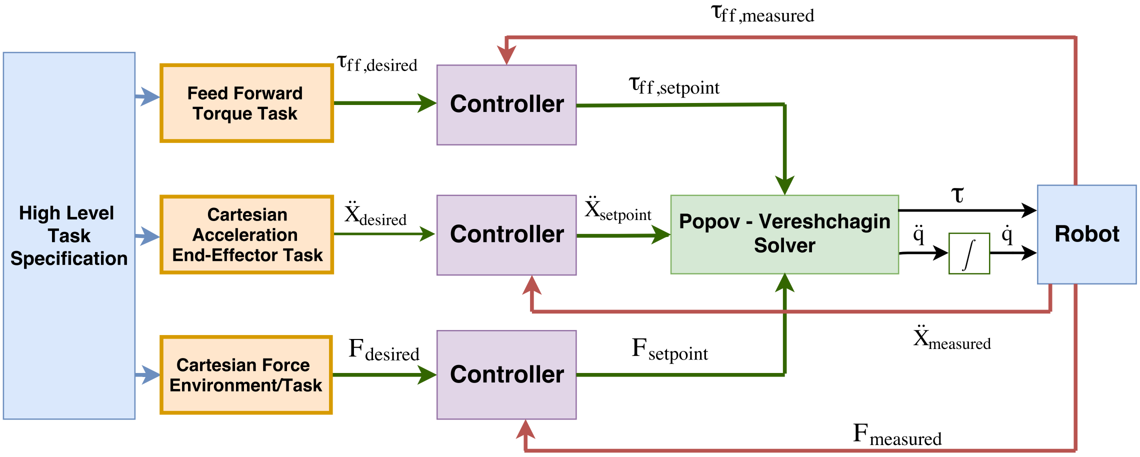

Figure 4.2: Generic control scheme around the original Popov-Vereshchagin solver.

4.5 Implementation and summary

The Popov-Vereshchagin algorithm has been implemented in already mentioned open-source *Kinematics and Dynamics Library* (KDL) [17], by Ruben Smits, Herman Bruyninckx, and Azamat Shakhimardanov. The complete library is created using C++ programming language. Previously mentioned extension to the original solver, for constraints specification on multiple segments, developed by Shakhimardanov [8] has not be implemented. Namely, only originally formulated algorithm created by Popov and Vereshchagin [4], is available in the library.

In order to ensure safe executions of tasks, a user is required to implement an external software *mechanism* for monitoring task feasibility at run-time. Namely, as explained in section 4.2, the constraint coupling matrix $\mathcal{L}$ can become rank-deficient, and in such case the computed control commands can be incorrect. Unfortunately, this type of monitoring mechanism does not exist in KDL library, and a user does not have an information concerning the current state of the matrix $\mathcal{L}$.

Additionally, for usage of resulting spatial vector of link accelerations $\ddot{X}$, a user is required to transform motion vector of each segment to coordinate frame of base segment.

The constrained hybrid dynamic solver developed by Popov and Vereshchagin in [4], [5], [60], and later on extended for hierarchical task specifications by Shakhimardanov in [8], represent a great contribution to robotics community. Not only that the algorithm can compute control commands based on many different task specifications, but it resolves robot redundancy by computing unique and energy optimal motion. In simple words, these properties define a brilliant tool for controlling robots.

Chapter 5

Deriving the integration of dissipative forces in the Popov-Vereshchagin solver

When a certain force induces *dissipation*, or in other words *loss* of energy from a system, than this force is called dissipative or nonconservative force [72], [73]. A well known [74] type of dissipation force is a friction force between two body surfaces in contact. However, friction effects in robot joints imposes additional load on the motors. Actuators need to produce supplementary amount of torque in order to overcome friction factors and move [2]. We cannot eliminate friction, but we can optimize motions to make them more energy efficient, by considering friction effects in motion computations. One of the solutions for overcoming the aforementioned problem is extending already available dynamics solver [4], by taking static friction factors as dissipative forces, into account.

The procedure for integrating dissipative forces, namely reaction forces from static friction effects and joint position limits, as additional conditions in motion computations, has been briefly outlined by Vereshchagin in the paper [4]. However, the author did not provide a detailed theoretical and experimental elaboration in order to show correctness of the approach.

5.1 Modelling of dissipative forces

For correctly modelling dissipative forces, such as reaction forces from physical joint limits and friction effects, common approaches in mechanics and robotics use the *principle of maximum dissipation* [74], [75]. The principle defines that, for a pair of bodies which are in contact, the true reaction force (of all virtual forces) is the one that produce *maximum rate* of energy dissipation [74].

Fortunately, it has been shown [66], [76] that the previously presented Gauss principle of least constraints and maximum dissipation principle can be combined for modelling motion and forces of a same mechanical system, subject to equality and inequality constraints. The augmentation of two principles imposes a dual optimization problem [66]. More specifically, while minimizing acceleration energy, defined by the Gauss

principle, additional conditions or in other words dissipative forces that maximize rate of energy dissipation, must be considered [76].
In [4], Vereshchagin applied this formulation for systems of robot manipulators, in order to extend originally formulated constrained hybrid dynamic solver, presented in chapter 4.

5.2 Formulation of the optimization problem for additional conditions

In order to account for additional factors, namely physical joint limits and static friction, in [4] Vereshchagin formulated an additional optimization problem. The general formulation of the problem for including forces of the afore-mentioned constraints in the computation of motions, involves the maximization of the following function:

$$Z_1(\ddot{q}) = \max_{\mu \in \Lambda}\{\mu^T W \ddot{q}\}, \tag{5.1}$$

Here, Z_1 represent acceleration energy induced by forces of additional factors. Note that in following discussion, we will refer to the original Gauss function (eq. 4.1) with Z_0 notation. The term μ denotes vector of possible reaction forces in robot's joints, generated either by static friction effects or joint position limits. Values of these moments depend on conditions that are accounted in this formulation/concept, and furthermore these values are elements of certain convex set Λ. The term W represents a *selection matrix* that depends on joint velocities or accelerations at the current time step.

In order to compute true motion of a robot, the solver must additionally account for acceleration energy (equation 5.1) induced by these supplementary forces. This means that original optimization problem of minimizing Gauss function, must be extended to account also for additional contributions of acceleration energy [76]. Now, the complete optimization problem is formulated as [4]:

$$\ddot{q}^* = arg \min_{\ddot{q} \in R^N} \max_{\mu \in \Lambda} \{\overbrace{Z_0(\ddot{q}) + \underbrace{\mu^T W \ddot{q}}_{Z_1}}^{Z}\} \tag{5.2}$$

The reason for considering both contributions of acceleration energy (Z_0 and Z_1) under maximization procedure, comes from the fact that computation of these two contributions depend on each other. More specifically, the direction in which friction force work, and produce maximum of acceleration energy, influence resulting motion ($\ddot{q}$) of a robot. That is, directions of joint accelerations depend on directions in which joint friction forces (considered in search for the maximum) operate.

5.3 Application of static friction forces in general formulation

It is well know that friction forces are always present between surfaces of two bodies in contact, and these forces resist relative motion between two bodies [21], [77]. In the simplest case, a model of friction phenomenon consist of two phases, namely *static* friction phase and *kinetic (dynamic)* friction phase. The first phase is characterized by static force which exist until relative motion between two surfaces starts [2] or in other words, until the relative velocity becomes greater then zero at a certain time instant. This time step is called *breakaway* point and at this point value of the static force reaches its maximum (see figure 5.1). When a motion starts, static phase stops existing while dynamic phase starts. This means that in order to account for static friction effects, we need to consider friction torque values of only those joints that have zero velocity. Furthermore, for a joint to move a link, it must first overcome the maximum static friction force, namely breakaway force.

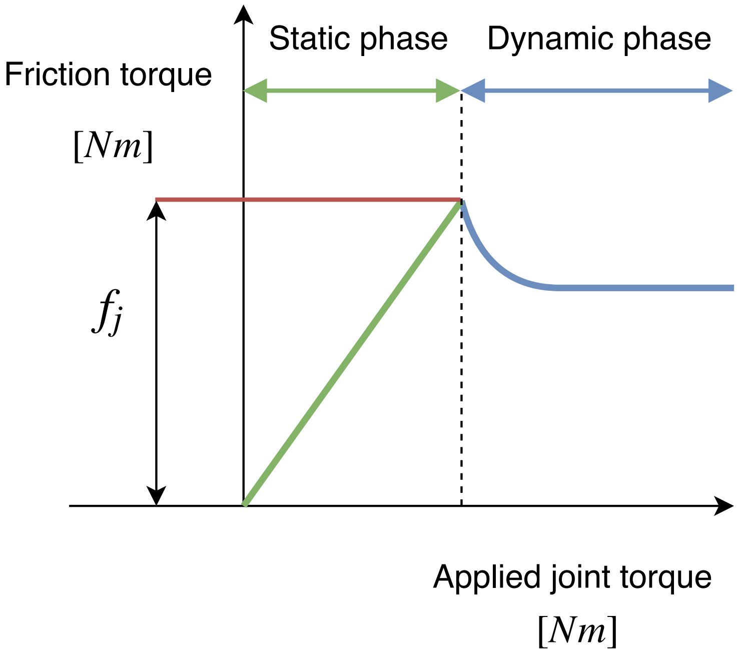

Figure 5.1: Applied joint torque versus reaction friction force (joint reaction torque). Term f_j denotes value of breakaway static friction torque (figure based on [77]).

As described in previous section (5.2), integration of static friction torques in motion computation requires a solution to the additional optimization problem. This is due to supplementary amount of acceleration energy induced by friction forces in robot joints. Namely, in [76] Pozharitskii extended the Gauss principle to account for factors of static friction, in general case of rigid body mechanics. Based on Pozharitskii work, for the case of robot manipulators, in [4] Vereshchagin considered modelling the acceleration energy contributions from static friction effects, in following manner:

$$Z_1(\ddot{q}) = \sum_{j \in \Theta_0} f_j |\ddot{q}_j|, \tag{5.3}$$

Here, term f_j stands for breakaway value of static friction force in joint {j}. The symbol Θ_0 represents a set of joint indices, whose velocities at the current time step are equal to 0, that is $\Theta_0 = \{j : \dot{q}_j = 0\}$ [4]. It is assumed that maximum friction force values f_j are established in advance.

In the same publication [4], Vereshchagin defined that this contribution of acceleration energy can be computed using previously presented formulation for additional conditions. In other words, Vereshchagin have transformed Pozharitskii work in his generic formulation of additional constraints, by means of following relation:

$$Z_1(\ddot{q}) = \sum_{j \in \Theta_0} f_j |\ddot{q}_j| = \max_{\mu \in \Lambda} \{ \sum_{j \in \Theta_0} \mu_j \ddot{q}_j \}, \tag{5.4}$$

In this case term μ defines vector of possible static friction torques in robot's joint. Their range is defined as: $\Lambda = \{-f_j \leq \ \mu_j \ \leq f_j\}$.

5.4 Solution to the optimization problem

After extension with static friction, the Gauss function Z (equation 5.2) loses its quadratic property, but continues being convex [4]. Nevertheless, the latter property ensures the existence of solution to the optimization problem [78].

Fortunately, inertia matrix has a property that is positive definite [6], and it follows that we can reformulate the extended Gauss function in terms of a dual optimization problem [78], [79]. It allows us to swap positions of minimization and maximization problems, such that the complete convex optimization problem can be solved in the following form [4]:

$$\mu^* = arg \ \max_{\mu \in \Lambda} \ \min_{\ddot{q} \in R^N} \ \{\overbrace{Z_0(\ddot{q}) + \mu^T W \ddot{q}}^{Z}\} \tag{5.5}$$

As previously defined in section (5.3), size of a set Θ_0 depends on number of joints that have zero velocity at current time instant, which means that size of a set is not constant. On another side, size of the vector μ is constant, it does not depend on current joint velocity. For that reason, it is necessary to select entries from μ that correspond to non-moving joints, to ensure that only values of these joint friction torques contribute to acceleration energy. Thus, *selection* matrix W was introduced and here it is interpreted as a $N \times N$ diagonal matrix, where its diagonal elements are defined as:

$$w_{j,j} = \begin{cases} 0, & \dot{q}_j \neq 0 \\ 1, & \dot{q}_j = 0 \end{cases} \tag{5.6}$$

As already presented in previous section (4.1), the original solver computes minimum of originally formulated Gauss function (Z_0). Nevertheless, the formulation of extended Gauss function can be adopted (reshaped) such that friction forces can be included via existing *feedforward torque* interface of the original algorithm, in a more convenient way. Namely, friction torques of form $W^T\mu$ can be subtracted directly from existing feed-forward input torques τ, in Gauss function of original shape (equation 4.1). It can be shown that these two formulations are equivalent:

$$\begin{aligned} Z(\ddot{q}) = & \quad Z_0(\ddot{q}) + \mu^T W \ddot{q} \\ = & \sum_{i=0}^{N} \{\frac{1}{2}\ddot{X}_i^T I_i \ddot{X}_i + F_{bias,i}^T \ddot{X}_i\} + \sum_{j=1}^{N} \{\frac{1}{2} d_j \ddot{q}_j^2 - \tau_j \ddot{q}_j\} + \sum_{j \in \Theta_0} \{\mu_j \ddot{q}_j\} \\ = & \sum_{i=0}^{N} \{\frac{1}{2}\ddot{X}_i^T I_i \ddot{X}_i + F_{bias,i}^T \ddot{X}_i\} + \sum_{j=1}^{N} \{\frac{1}{2} d_j \ddot{q}_j^2 - \tau_j \ddot{q}_j + \mu_{j,j \in \Theta_0} \ddot{q}_j\} \\ = & \sum_{i=0}^{N} \{\frac{1}{2}\ddot{X}_i^T I_i \ddot{X}_i + F_{bias,i}^T \ddot{X}_i\} + \sum_{j=1}^{N} \{\frac{1}{2} d_j \ddot{q}_j^2 - (\tau_j - \mu_{j,j \in \Theta_0}) \ddot{q}_j\} \end{aligned} \tag{5.7}$$

By including static friction forces directly via *feedforward torque* interface, the solution to the minimization problem:

$$\ddot{q}^*(\mu) = arg \min_{\ddot{q} \in R^N} \{Z_0(\ddot{q}) + \mu^T W \ddot{q}\}, \tag{5.8}$$

is being computed by the original Popov-Vereshchagin solver, as explained in previous chapter (4). Here, friction torques μ represent constant parts of the function.
The additional part of the optimization problem, imposed by the extension to the original solver, is defined as:

$$\mu^* = arg \max_{\mu \in \Lambda} \{Z_0(\ddot{q}^*(\mu)) + \mu^T W \ddot{q}^*(\mu)\} \tag{5.9}$$

At the end, when the optimum friction torques μ^* are found, "*the accelerations $\ddot{q}(\mu^*)$ will be true accelerations in the system*" [4].

Interpretation of the solution

The presented mathematical derivation of the solution to the problem of including static friction effects in computations of motion, is summarized in following elaboration, in order to provide the interpretation of the approach.
The true motion of a robot is characterized by static friction *reaction* torques in joints, which induce a maximum of the Gauss function (5.9). More specifically, the resulting optimum of this function, defines *maximum* acceleration energy over all possible static friction forces that are in range between *positive* and *negative* breakaway friction values. For each value of static friction force in this range, the original solver is used for computing a motion, or in other words finding a *minimum* of the Gauss function (5.8), over all possible (virtual) joint accelerations. Finally, the *optimum* friction torque μ^*, that induces the *maximum* over all *minimums* of acceleration energy, will produce true robot motion $\ddot{q}(\mu^*)$.
An example of the complete Gauss function, over which max-min optimization

process is performed, is presented in figure 5.2. This particular Gauss function is based on a task specified for one *DOF* robot arm. The red colored polynomial (function), represent a search space over which maximum of acceleration energy is found. Or in other words, a search space over which the true reaction friction force μ^* that exist in the system is found. Each point on this polynomial represent a minimum value of acceleration energy that is function of joint accelerations. A motion (acceleration) for particular value of static friction torque is computed by the original Popov-Vereshchagin solver.

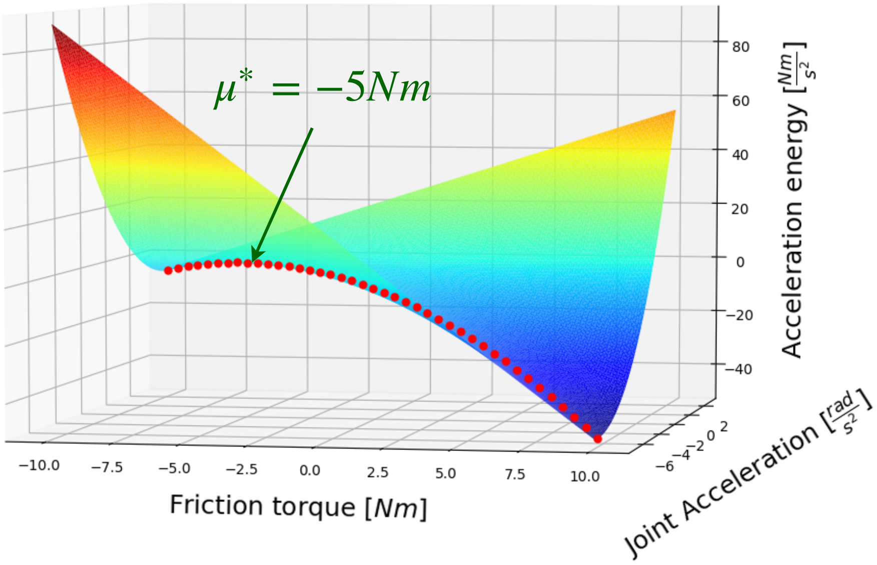

Figure 5.2: Acceleration energy as function of static friction forces and accelerations in the joint, for one DOF robot arm. The red colored polynomial (function), represent a search space over which maximum of acceleration energy is found.

Implementation

A *proof of concept* software implementation of the extension to the original Popov-Vereshchagin algorithm is achieved in C++ language. In order to evaluate proposed and derived approach for including friction effects in computations of motion, the implementation consists of sampling static friction torques in range between *negative* and *positive* breakaway friction values, for each joint specified by a robot model. As previously explained, the complete approach for computing robot motion while accounting for static friction effects depend on the original Popov-Vereshchagin solver. Namely, for each possible set of friction torques, the original algorithm is used for computing minimum of the Gauss function (equation 5.8) or in other words, computing resulting motion for particular vector of friction torques. For each sample of friction reaction forces and computed motion, the Gauss function (equation 5.9) is evaluated. It the end, the software chooses a motion that induces maximum of the Gauss function (5.9), to be the true motion of a system.

For this reason the implementation of the extended algorithm depend on previously described KDL library [17], where the original solver is implemented. Nevertheless, compared to input requirements of the original algorithm specified in section 4.2, for a human programmer to compute robot motion using the extended algorithm, it is only required to additionally provide values of breakaway friction forces as part of a robot model description (see figure 5.3).

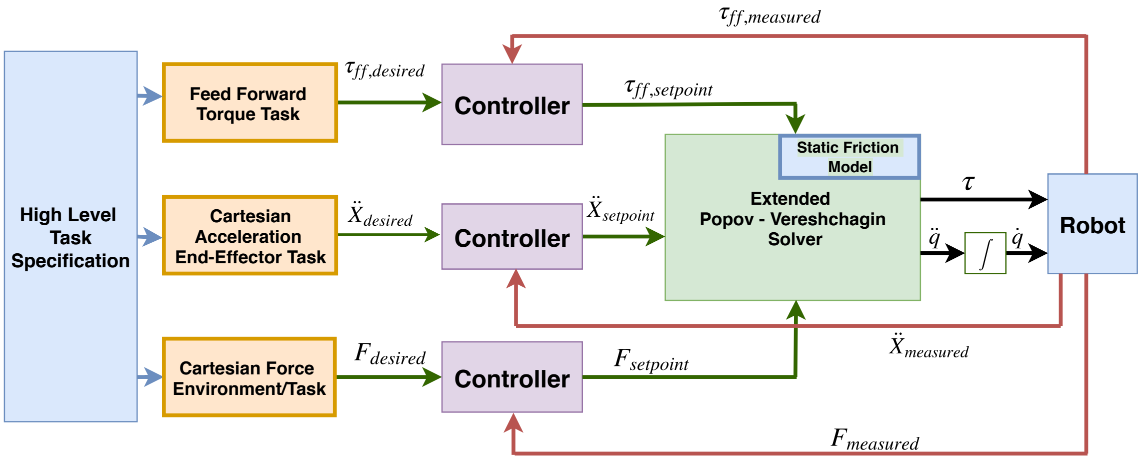

Figure 5.3: Generic control scheme around the extended Popov-Vereshchagin solver.

In [4], Vereshchagin proposed the *convex simplex* method [80] as one of possible techniques for maximizing the 5.9 function. However, the proposed technique is a numerical method, and due to non-constant number of iterations required to be performed in the search for optimum, this method may not be usable for real-time application.

5.5 Improving efficiency of the extended algorithm

As described in section 5.4, the proposed approach for extending the constrained hybrid dynamics solver requires maximization of the extended Gauss function (equation 5.9). Here, for finding a maximum, it was required to use the original Popov-Vereshchagin algorithm to compute resulting accelerations (motion $\ddot{q}^*(\mu)$) of a system. After finalization of the original solver's computations, and based on computed motion, the cost function (5.9) was evaluated. This procedure imposes a high computational load, due to a large number of operations performed by the original Popov-Vereshchagin solver. Nevertheless, for enabling a better performance, or in other words reducing run-time of the extended algorithm, a more convenient method for finding the maximum of the extended Gauss function is proposed. More specifically, in [4] Vereshchagin outlined the idea for improving run-time of approach, but without providing its derivation. Following section is dedicated to deriving the proposed method for improving the efficiency of the extended algorithm.

Method of solution

The equation 5.9 is the function of three variables, namely spatial accelerations $\ddot{X}$, joint accelerations $\ddot{q}$ and feedforward friction torques μ. More specifically, it is the function of two dependent variables $\ddot{X}$ and $\ddot{q}$, while μ is the independent variable. That is, spatial accelerations $\ddot{X}$ are the function of joint accelerations $\ddot{q}$. Moreover, joint accelerations $\ddot{q}$ are itself the function of μ variable. In order to compute joint accelerations $\ddot{q}$ given a vector of feedforward friction torques μ, it is required to evaluate a large number of functions in the original Popov-Vereshchagin solver, as it can be seen from the algorithm representation in section 4.2. However, in order to calculate spatial accelerations $\ddot{X}$ based on already computed joint accelerations $\ddot{q}$, a number of functions required to be evaluated is much smaller (see line 29 of algorithm 1).

Aforementioned properties of the Gauss function allow us to find a characteristic equation or in other words, an ordinary differential equation (ODE) which define how joint accelerations $\ddot{q}$ will change, with respect to possible friction torques μ. The fact that acceleration energy depends directly on $\ddot{q}$ and μ, allows us to compute $Z(\ddot{q}, \mu)$ more efficiently, by using values of joint accelerations computed only from the characteristic equation. In more detail, the method for improving efficiency allows us to compute accelerations (motion) of a robot directly from the characteristic equation. Based on computed motion, the Gauss function can be evaluated. By using this method, it is required to use the original Popov-Vereshchagin algorithm only once, throughout the process of maximizing the Gauss function.

For this problem, a characteristic equation has the form:

$$\ddot{q}^*(\mu) = \ddot{q}^*(0) + \frac{\partial \ddot{q}}{\partial \mu}\mu \tag{5.10}$$

Here, a solution to the characteristic equation is influenced by initial conditions, namely $\ddot{q}^*(0)$. Initial values of acceleration vector are computed by the original solver, and this operation is performed only once. Additionally, while calculating $\ddot{q}^*(0)$ with the original algorithm, we can compute a *partial derivative* of the $\ddot{q}$ function, namely $\frac{\partial \ddot{q}}{\partial \mu}$, directly in *sweeps* of the original Popov-Vereshchagin solver. After computing these constant parts of the characteristic equation, that is $\ddot{q}^*(0)$ and $\frac{\partial \ddot{q}}{\partial \mu}$, then the optimization process involves 1) solving equation 5.10, 2) computing spatial accelerations $\ddot{X}$ and 3) evaluating the Gauss function for each considered joint friction torque vector. Compared with the previous approach, the complexity of proposed method remains the same, but the efficiency is improved, due to the number of functions required to be evaluated, or in other words number of operations to be performed. The presented technique for computing robot motions is formulated based on the *method of characteristics* that is used for solving partial differential equations [81].

An example of the characteristic curve, that can be derived from the Gauss function is presented in figure 5.4. This particular Gauss function is based on a task specified for one *DOF* robot arm. Here, the characteristic equation (curve) is visualized based on the data computed while performing the optimization process, that is finding the maximum of the red polynomial (Gauss) function. The presented

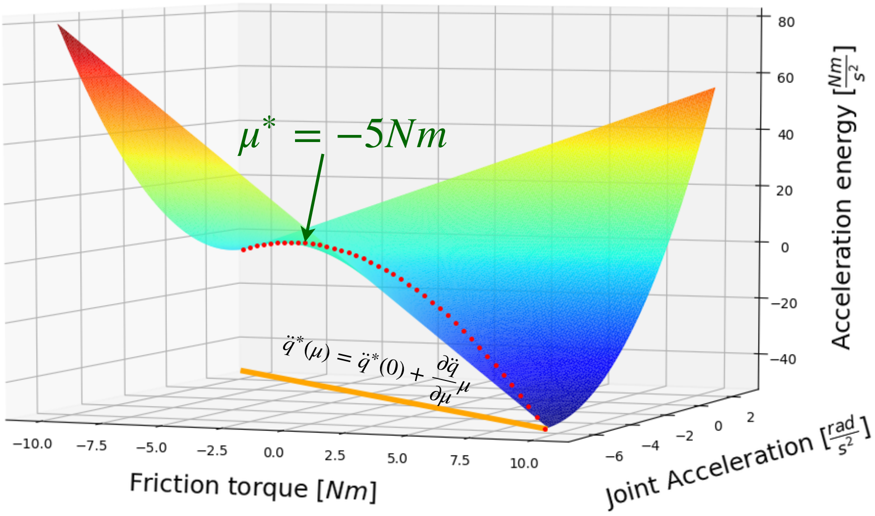

Figure 5.4: Acceleration energy as function of static friction forces and accelerations in the joint for case of a one *DOF* robot. Here, the orange line represent characteristic curve of the Gauss function.

characteristic curve defines the relation (dependency) between joint accelerations and static friction reaction torques. That is, it defines how joint accelerations change with respect to different friction reaction forces in the joint, throughout the optimization (maximization) process.

5.6 Lessons learned

As previously defined in this chapter, when the Gauss' principle is extended with static friction, an additional convex optimization (maximization) problem must be solved, in order to compute the true motion and true reaction friction forces of a system. Linear constraints, namely bounded friction torques described in section 5.3, define a polytope of feasible solution [82]. Furthermore, as described in section 5.4, the cost function 5.9 to be optimized is convex [4].

Initial approach, considered for finding the maximum of acceleration energy induced by these reaction forces, was based on a hypothesis which states that the optimum (maximum) of the Gauss function can be found by iterating over vertices. Here, a vertex is defined by a set of friction torques with breakaway magnitudes, in each joint. However, this hypothesis does not always hold. Namely, when a motion of a robot is completely constrained, i.e. when a number of *DOF* of a task is equal to the number of *DOF* of a robot, then the maximum of the Gauss function can be found by iterating over vertices. This follows from the fact that the Gauss function becomes linear in this scenario (see figure 5.5 for an example) or in other words a hyperplane. Nevertheless, in a case when a robot is partially constrained then the Gauss function becomes quadratic, which shows that initial hypothesis does not hold. An example of the latter scenario is presented in figure 5.2. However, a set of experiments was conducted in order to evaluate this hypothesis. Despite the

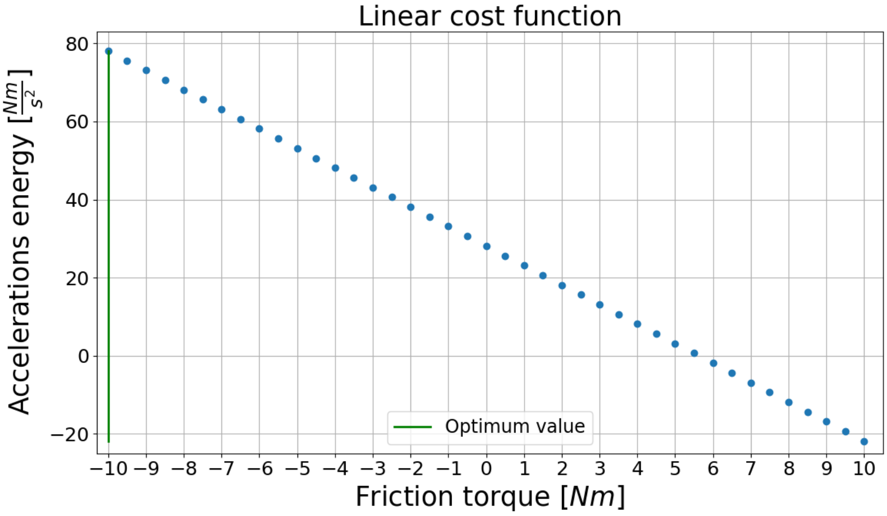

Figure 5.5: Acceleration energy as function of static friction forces in the joint, for case of fully constrained one DOF robot. The function is visualized based on data derived from optimization (maximization) procedure. The imposed task in this example defined a desired angular acceleration of the end-effector, with magnitude of 5 $\frac{rad}{s^2}$. The maximum of this Gauss function is on the left vertex, where static friction reaction force has the negative breakaway value (i.e. -10 Nm).

fact that the experiments have shown non-correctness of this approach, they have resulted in several important lessons.

The conducted tests helped us to understand optimization problem more clearly, i.e. where does the optimum of the Gauss function exist for different task specifications. Additionally, this resulted in a better interpretation of different outputs from the original Popov-Vereshchagin solver. That is, how the computed control commands can be used for different types of low-level robot controllers (e.g. a PID controller), that are designed for ensuring correct executions of desired joint forces. Furthermore, this situation led us to a better understanding on how the original solver is computing required joint torques for execution of a task, and even to understand how the solver is computing energy optimal motions, based on the Gauss principle of least constraint.

Chapter 6

Evaluation and results

This chapter presents an evaluation of the *proof of concept* implementation for the proposed extension to the Popov-Vereshchagin algorithm, presented in chapter 5. All tests discussed in following sections are performed in a simulation environment. Following standalone evaluation was performed with artificially generated values. As mentioned in section 4.5, constraint coupling matrix $\mathcal{L}$ can be rank-deficient depending on task specification and available $DOFs$ of a robot. In following experiments the singularities are tested and defined motions are feasible.

6.1 Experimental setup

Following test setup was created in order to show the feasibility of proposed and derived approach for including static friction joint forces in computations of motion. The experimental setup involves scenarios, i.e. modeling of a robot environment in which gravity effects do not exist. Additionally, artificial static friction joint torques and artificial external Cartesian forces are generated. Furthermore, in both parts of the experiment, a motion of the robot is not constrained.

The reason for designing this type of setup is a necessity for ensuring a simple physical interpretation and straightforward prediction of correct results, from computations of true robot motions using the extended Popov-Vereshchagin algorithm.

In following experiments, a simple model of one degree of freedom robot arm is used (see figure 6.1). Namely, the mass of the robot's segment 1 is 1 kg, while its length is 1 m. Furthermore, the center of mass and the principal axis of inertia of this segment, are located at the same point, that is at 0.5 m from the origin of the segment's proximal frame.

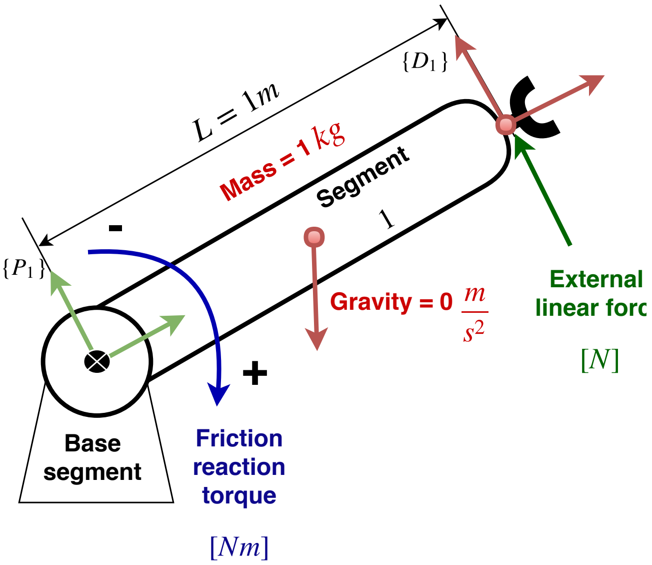

Figure 6.1: Model of one degree of freedom robot arm and forces influencing its motion.

6.2 First scenario

In the first part of the experiment, an external and linear force equal to -5 N is applied on the end-effector of the robot, or more specifically on the origin of segment $\{1\}$ distal frame (see figure 6.1). This means that the torque felt in robot's joint, due to influence of aforementioned external force is -5 Nm. On the other side, breakaway value of the static friction force that exist in this joint is 10 Nm in positive, that is in clockwise direction of rotation, while in negative direction breakaway value is -10 Nm. This means that magnitude of the join torque that is generated by the external force applied on this robot, is smaller than breakaway value of existing static friction torque.

This setup was created in order to show that the external force applied on the robot's segment will not produce motion of the robot joint. In other words, by following formulation presented in chapter 5, in this part of experiment the optimum reaction friction force must ensure zero accelerations of the robot arm.

Following figures describe optimization process that is performed in order to compute true motion of this robot. Namely, in figure 6.2 values of acceleration energy considered throughout optimization (maximization) procedure, are presented. For computing these values, the Gauss function (equation 5.9) has been evaluated.

As defined in section 5.4, static friction force that induces maximum acceleration energy, will be the true reaction force in a system. Following this definition, and for case of this part of experiment, the true reaction friction force in robot joint μ^* is equal to 5 Nm, as it can be seen from figure 6.2. The magnitude of this force is equivalent to the magnitude of joint torque produced by the external force. As these two forces work in opposite directions, the resulting force in the joint (i.e. $\tau_{control}$) will be equal to zero. Thus, it follows that resulting and true motion (i.e. $\ddot{q}(\mu^*)$) of the robot is zero accelerations.

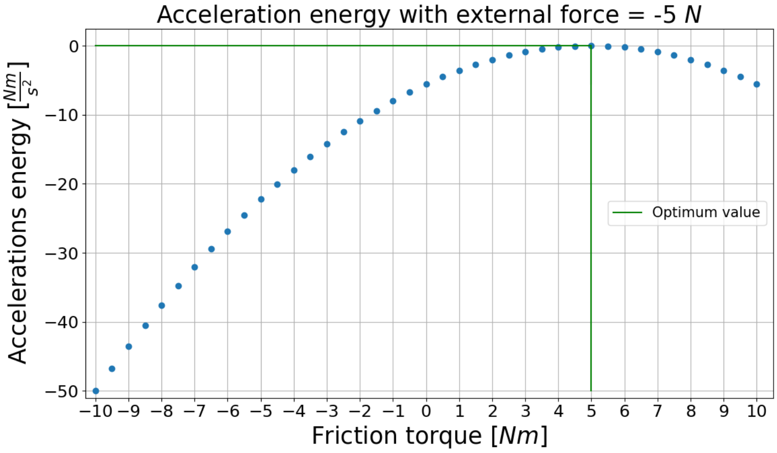

Figure 6.2: Acceleration energy as function of static friction reaction forces in robot joint. The reaction friction torque that induced maximum rate of dissipation is equal to 5 Nm.

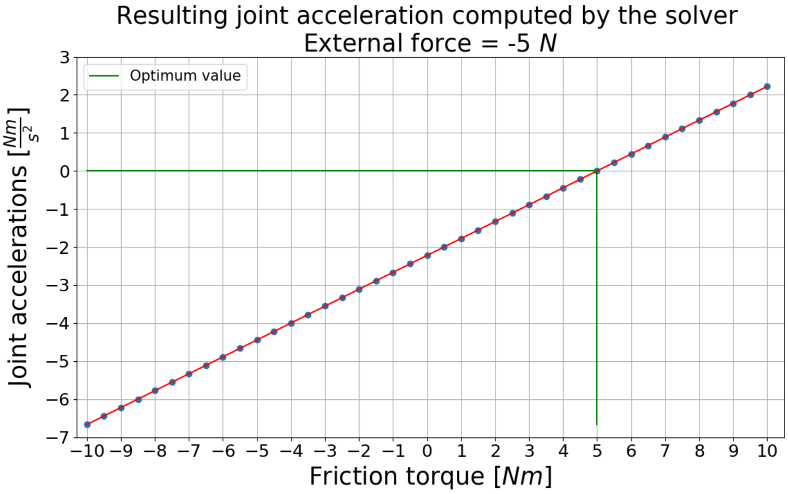

Figure 6.3: Resulting accelerations as function of friction reaction forces in robot joint, throughout optimization (maximization) process. External force is equal to -5N.

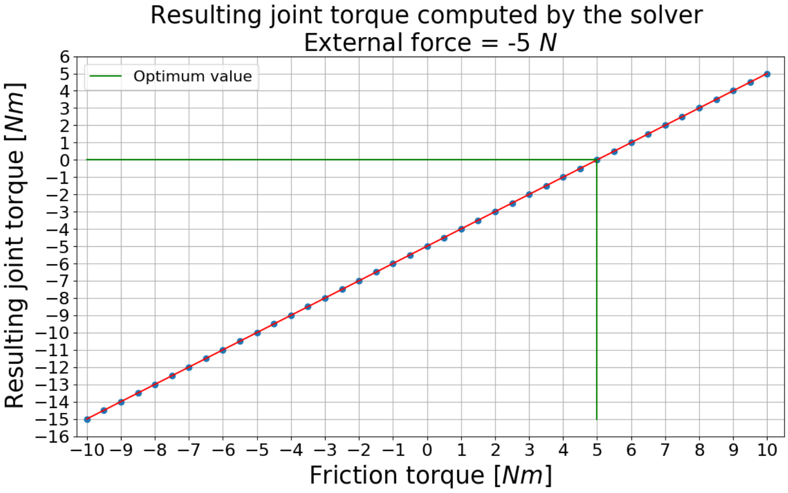

Figure 6.4: Resulting torques as function of friction reaction forces in robot joint, throughout optimization (maximization) process. External force is equal to -5N.

Furthermore, the graphical representation for aforementioned computation of the robot motion, in terms of joint accelerations $\ddot{q}$, is presented in figure 6.3. On the other hand, the graphical representation for computation of the forces resulted in the joint, is presented in figure 6.4.

6.3 Second scenario

In the second part of this experiment, the external force is increased to -11 N. This means that external force will produce -11 Nm of torque in the robot's joint. Hence, the magnitude of this joint force is higher than breakaway value (10 Nm) of the static friction force that exist in the joint.

This setup was created in order to show that the external force applied on the robot arm will produce motion in robot joint. That is, by following formulation presented in chapter 5, the optimum friction reaction force in this part of experiment must not stop the arm from accelerating, and furthermore magnitude of this reaction friction torque must be equal to the breakaway friction value.

The true reaction force μ^* in robot joint, or in other words static friction torque that induces maximum rate of dissipation is equal to 10Nm, as it can be seen from figure 6.5. It follows that the resulting torque in robot's joint is -1 Nm and based on this resulting joint force, the solver has computed joint accelerations of $-0.44\frac{rad}{s^2}$.

The graphical representation for aforementioned computation of the robot motion, in terms of joint accelerations $\ddot{q}$, is presented in figure 6.6. On the other hand, the graphical representation for computation of the forces resulted in the joint, is presented in figure 6.7.

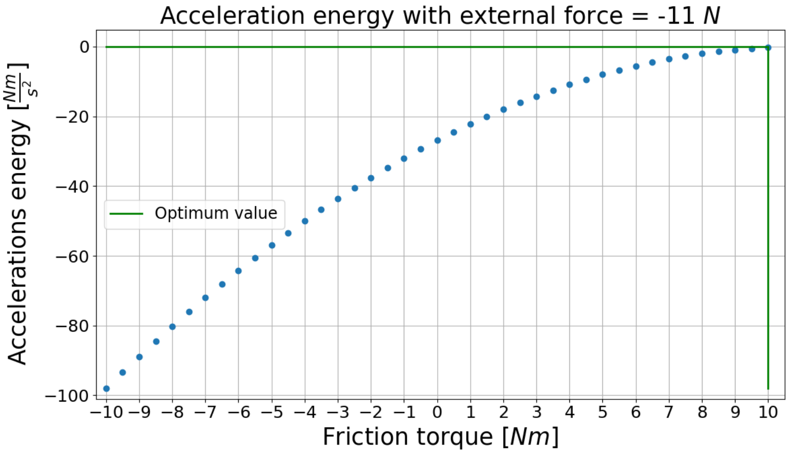

Figure 6.5: Acceleration energy as function of friction reaction forces in robot joint. The reaction friction torque that induced maximum of acceleration energy is equal to 10 Nm.

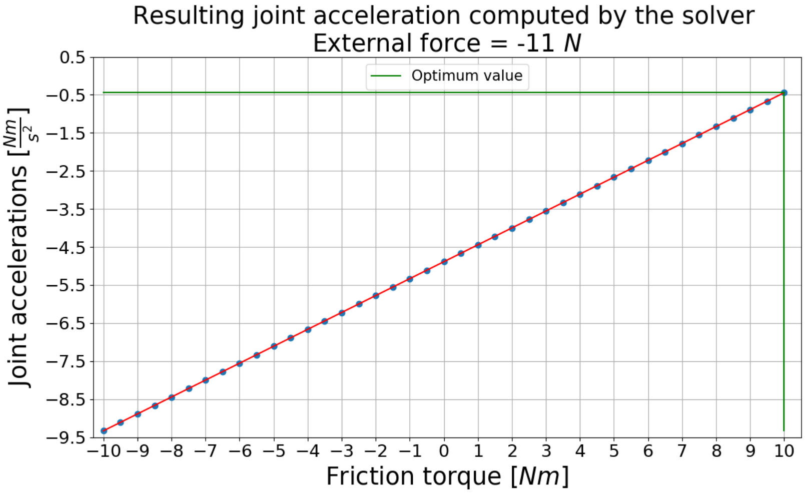

Figure 6.6: Resulting accelerations as function of friction reaction forces in robot joint, throughout optimization (maximization) process. External force is equal to -11N.

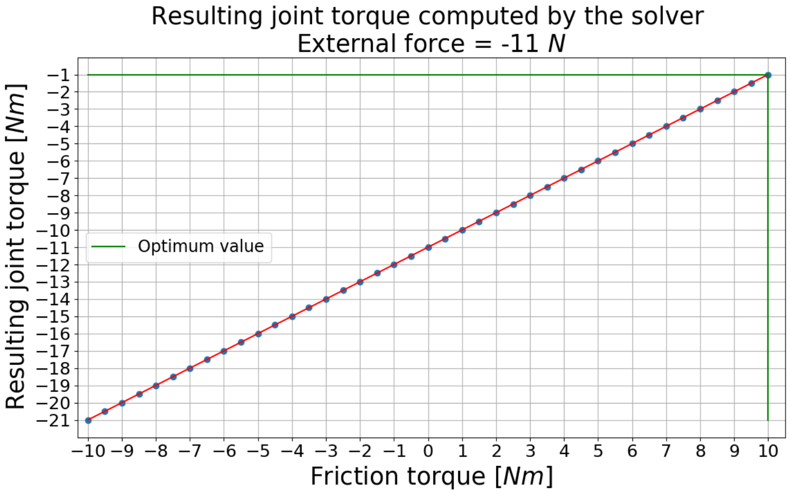

Figure 6.7: Resulting torques as function of friction reaction forces in robot joint, throughout optimization (maximization) process. External force is equal to -11N.

Chapter 7

Conclusion and future work

In this project, the extension for an already existing constrained hybrid dynamics solver, namely Popov-Vereshchagin algorithm, has been presented. The approach for including dissipative forces from static friction factors, in computations of energy optimal instantaneous motions, has been completely derived.

The proof of concept evaluation in chapter 6 has shown the feasibility of the approach proposed by Vereshchagin [4] and derived in this project. More specifically, the performed tests in a simulation environment with one *DOF* robot arm, have shown that the approach for including static friction effects in computations of motions, provides correct results. Additionally, the experiments conducted using the extended algorithm, have shown that the Gauss function remains convex after extension with friction forces. However, the function can take two forms depending on a task (i.e. constraints) specified. That is, the cost function becomes linear if a robot is fully constrained, else, if a robot is partially constrained then the Gauss function to be maximized takes quadratic form. This means that we require a generic method for maximizing the extended Gauss function (equation 5.9). In other words, we require an optimization technique that can be used for optimizing both linear and quadratic functions, depending on the constraints specified in run-time of a robot system.

The presented approach requires the use of the original Popov-Vereshchagin solver, for computing the resulting motion and further on evaluating the Gauss function, for each set of the friction reaction forces considered throughout maximization process. This procedure imposes a high computation load and may not be real-time feasible. For that reason, the research in this project was conducted in order to derive the (briefly outlined by Vereshchagin in [4]) method for improving the efficiency of the proposed extension to the original algorithm. More specifically, to reduce the number of operations (computations) required for finding the true reaction forces and true motion of a system.

Aims of the future research in this field, involves the implementation of the *convex simplex* method, in order to enable the use of the extended algorithm in the computation of motions, for the case of robot systems with an arbitrary number of degrees of freedom. This future plan includes investigation of the real-time feasibility of the complete approach when the convex simplex method is implemented.
Furthermore, future work would include implementation of the derived method for improving computational efficiency, which may remedy the issue of real-time application of the complete approach.

Additionally, the future goal is to include dissipative forces induced by joint position limits, in already derived general optimization procedure for additional conditions, presented in section 5.2.

Finally, future work would concentrate on implementing the extended Popov-Vereshchagin algorithm on a real robot system. Furthermore, investigating additional requirements that may arise when the static friction effects are considered in computations of energy optimal motions, in practical or in other words real robot environment.

Appendix A

Appendix

A.1 Plücker coordinates for spatial vectors

By using convention specified in terms of Plücker coordinates, we can define spatial vectors for both motions and forces. As Featherstone stated in [6], spatial vectors are constructed by combining both angular and linear components of forces or motions in a single vector. This type of 6D vector representation enables more efficient computation of rigid-body dynamic quantities, compared to traditional 3D vector representation [6].

Generic representation of a spatial *motion* vector in Plücker coordinates is defined in the following form [6]:

$$M = \begin{bmatrix} m_{L,x} \\ m_{L,y} \\ m_{L,z} \\ m_{A,x} \\ m_{A,y} \\ m_{A,z} \end{bmatrix}$$

When spatial *velocity* and *acceleration* vectors are defined using this vector representation, they take following form:

$$\dot{X} = \begin{bmatrix} \upsilon_x \\ \upsilon_y \\ \upsilon_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \qquad \ddot{X} = \begin{bmatrix} \dot{\upsilon}_x \\ \dot{\upsilon}_y \\ \dot{\upsilon}_z \\ \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix}$$

Here, symbols υ and ω represent linear and angular velocity components of a body, respectively. While symbols $\dot{\upsilon}$ and $\dot{\omega}$ represent linear and angular acceleration components of a body, respectively.

On another side, representation of spatial *force* vector in Plücker coordinates is defined in the following form [6]:

$$F = \begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \end{bmatrix}$$

Here, symbols f and n represent linear force and angular force (moment) components acting on a body, respectively.

A.2 Coordinate transformation for motion and force vectors

Compared to general 4×4 homogeneous transformation matrix ${}_{i}^{i+1}X$ [83], a coordinate transformation 6×6 matrices for motion and force vectors are constructed differently. Namely, matrix ${}^{i+1}X_i$ (used for motion vectors) has be defined as [6]:

$$ {}^{i+1}X_i = \begin{bmatrix} E & 0 \\ -E\, r\times & E \end{bmatrix} \tag{A.1}$$

where term E represents traditional 3×3 rotation matrix and term $\times$ denotes an operators that maps a 3×1 position vector r to a 3×3 skew-symmetric matrix of form:

$$ r\times = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \tag{A.2}$$

Matrices for transformations of motion and force vectors are related in following manner: ${}^{i+1}X_i^* = {}^{i+1}X_i^{-T}$, and we can infer that ${}^{i+1}X_i^*$ is defined as:

$$ {}^{i+1}X_i^* = \begin{bmatrix} E & -E\, r\times \\ 0 & E \end{bmatrix} \tag{A.3}$$

A.3 Cross product operators

Cross product operators in Plücker coordinates $\dot{X}\times$ and $\dot{X}\times^*$ represent 6×6 matrices which are defined as [6]:

$$\dot{X}\times = \begin{bmatrix}\omega \\ \upsilon\end{bmatrix}\times = \begin{bmatrix}\omega\times & 0 \\ \upsilon\times & \omega\times\end{bmatrix} \tag{A.4}$$

$$\dot{X}\times^* = \begin{bmatrix}\omega \\ \upsilon\end{bmatrix}\times^* = \begin{bmatrix}\omega\times & \upsilon\times \\ 0 & \omega\times\end{bmatrix} = -(\dot{X}\times)^T \tag{A.5}$$

where skew-symmetric matrices $\omega\times$ and $\upsilon\times$ are defined as:

$$\omega\times = \begin{bmatrix}0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0\end{bmatrix} \tag{A.6}$$

$$\upsilon\times = \begin{bmatrix}0 & -\upsilon_z & \upsilon_y \\ \upsilon_z & 0 & -\upsilon_x \\ -\upsilon_y & \upsilon_x & 0\end{bmatrix} \tag{A.7}$$

References

[1] M. Brossog, M. Bornschlegl, J. Franke, *et al.*, “Reducing the energy consumption of industrial robots in manufacturing systems.”, *International Journal of Advanced Manufacturing Technology*, vol. 78, pp: 2346–2353, 2015.

[2] B. Bona and M. Indri, “Friction compensation in robotics: An overview”, in *Decision and Control and European Control Conference IEEE, Seville, Spain*, 2005.

[3] C. Hansen, J. Öltjen, D. Meike, and T. Ortmaier, “Enhanced approach for energy-efficient trajectory generation of industrial robots”, in *IEEE International Conference on Automation Science and Engineering*, 2012.

[4] A. F. Vereshchagin, “Modelling and control of motion of manipulation robots”, *Soviet Journal of Computer and Systems Sciences*, vol. 27, pp. 29–38, 1989.

[5] E. P. Popov, A. F. Vereshchagin, and S. L. Zenkevich, “Manipulyatsionnye roboty: Dinamika i algoritmy”, *Nauka, Moscow*, 1978.

[6] R. Featherstone, *Rigid body dynamics algorithms.* Springer, 2008.

[7] R. Featherstone, “Robot dynamics”, *Scholarpedia*, vol. 2, p. 3829, 2007.

[8] A. Shakhimardanov, “Composable robot motion stack: Implementing constrained hybrid dynamics using semantic models of kinematic chains”, PhD thesis, KU Leuven, 2015.

[9] M. L. Felis, “RBDL: an efficient rigid-body dynamics library using recursive algorithms”, *Autonomous Robots*, vol. 41, pp. 495–511, 2017.

[10] P. Lötstedt, “Mechanical systems of rigid bodies subject to unilateral constraints”, *SIAM Journal on Applied Mathematics*, vol. 42, pp. 281–296, 1982.

[11] S. Ivaldi, V. Padois, and F. Nori, “Tools for dynamics simulation of robots: A survey based on user feedback”, *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2014.

[12] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[13] R. Smith, *ODE physics engine*, Accessed 2017-05-21, [Online] Available: `http://ode.org/`.

[14] Georgia, Institute of Technology, *DART physics engine*, Accessed: 2017-05-21, [Online] Available: `http://dartsim.github.io/`.

[15] Erwin Coumans, *Bullet physics engine*, Accessed: 2017-05-21, [Online] Available: `http://bulletphysics.org/`.

[16] M. A. Sherman, A. Seth, and S. L. Delp, “Simbody: Multibody dynamics for biomedical research”, *Procedia Iutam*, vol. 2, pp. 241–261, 2011.

[17] R. Smits, E. Aertbelien, H. Bruyninckx, and A. Shakhimardanov, *Kinematics and Dynamics Library (KDL)*, Accessed: 2017-05-30, [Online] Available: `http://www.orocos.org/kdl`.

[18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System”, in *ICRA workshop on open source software*, 2009.

[19] N. Mansard and F. Valenza, *Pinocchio library*, Accessed: 2017-07-25, [Online] Available: `https://stack-of-tasks.github.io/pinocchio/`.

[20] Joint Japanese French Robotics Lab, *JRL-RBDyn library*, Accessed: 2017-07-25, [Online] Available: `https://jrl-umi3218.github.io/RBDyn/doxygen/HEAD/index.html`.

[21] B. Armstrong-Hélouvry, P. Dupont, and C. C. De Wit, “A survey of models, analysis tools and compensation methods for the control of machines with friction”, *Automatica*, vol. 30, 1994.

[22] J. Moreno, R. Kelly, and R. Campa, “On manipulator velocity control using friction compensation”, in *IEEE International Conference on Robotics and Automation*, 2002.

[23] A. A. Ata, “Optimal trajectory planning of manipulators: A review”, *Journal of Engineering Science and Technology*, vol. 2, pp. 32–54, 2007.

[24] S.-H. Lee, J. Kim, F. C. Park, M. Kim, and J. E. Bobrow, “Newton-type algorithms for dynamics-based robot movement optimization”, *IEEE Transactions on Robotics*, vol. 21, pp. 657–667, 2005.

[25] J. Kim, J. Baek, and F. C. Park, “Newton-type algorithms for robot motion optimization”, in *International Conference on Intelligent Robots and Systems*, 1999.

[26] B. J. Martin and J. E. Bobrow, “Minimum-effort motions for open-chain manipulators with task-dependent end-effector constraints”, *The International Journal of Robotics Research*, vol. 18, pp. 213–224, 1999.

[27] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl, “Time-energy optimal path tracking for robots: A numerically efficient optimization approach”, in *IEEE International Workshop on Advanced Motion Control*, 2008.

[28] G. S. Sharma, M. Singh, and T. Singh, “Optimization of energy in robotic arm using genetic algorithm”, *International Journal of Computer Science and Technology*, vol. 2, pp. 315–317, 2011.

[29] D. P. Garg and M. Kumar, “Optimization techniques applied to multiple manipulators for path planning and torque minimization”, *Engineering Applications of Artificial Intelligence*, vol. 15, pp. 241 –252, 2002.

[30] H. Bruyninckx and O. Khatib, "Gauss' principle and the dynamics of redundant and constrained manipulators", in *IEEE International Conference on Robotics and Automation*, 2000.

[31] O. Khatib, L. Sentis, J. Park, and J. Warren, "Whole-body dynamic behavior and control of human-like robots", *International Journal of Humanoid Robotics*, vol. 1, 2004.

[32] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation", *IEEE Journal on Robotics and Automation*, vol. 3, pp. 43–53, 1987.

[33] N. Mansard, O. Khatib, and A. Kheddar, "A unified approach to integrate unilateral constraints in the stack of tasks", *IEEE Transactions on Robotics*, vol. 25, 2009.

[34] R. Smits, T. De Laet, K. Claes, H. Bruyninckx, and J. De Schutter, "iTaSC: A Tool for Multi-Sensor Integration in Robot Manipulation", in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems 2008.*

[35] K.-S. Chang and O. Khatib, "Operational space dynamics: Efficient algorithms for modeling and control of branching mechanisms", in *IEEE International Conference on Robotics and Automation.*, 2000.

[36] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments", in *Robotics and Automation, IEEE International Conference on*, 2006.

[37] O. Khatib and L. Sentis, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives", *International Journal of Humanoid Robotics*, vol. 2, pp. 505–518, 2005.

[38] O. Khatib, L. Sentis, and J.-H. Park, "A unified framework for whole-body humanoid robot control with multiple constraints and contacts", in *European Robotics Symposium 2008*, pp. 303–312.

[39] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison", *The International Journal of Robotics Research*, vol. 27, pp. 737–757, 2008.

[40] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation.* CRC press, 1994.

[41] C.-L. Fok, G. Johnson, J. D. Yamokoski, A. Mok, and L. Sentis, "Controlit! a software framework for whole-body operational space control", *International Journal of Humanoid Robotics*, vol. 13, 2016.

[42] M. Felis, *Rigid body dynamics library (RBDL)*, Accessed: 2017-06-20, [Online] Available: `https://rbdl.bitbucket.io/`.

[43] F Furrer, M Burri, M Achtelik, and R Siegwart, "Robot Operating System (ROS): The Complete Reference (Volume 1)", *By A. Koubaa. Cham: Springer International Publishing*, 2016.

[44] D. Vanthienen, T. De Laet, W. Decré, R. Smits, M. Klotzbücher, K. Buys, S. Bellens, L. Gherardi, H. Bruyninckx, and J. De Schutter, “iTaSC as a unified framework for task specification, control, and coordination, demonstrated on the PR2”, in *IEEE International Conference on Mechatronics and Robotics*, 2011.

[45] I. A. Sucan and S. Chitta, *MoveIt!*, Accessed: 2017-06-18, [Online] Available: `http://moveit.ros.org`.

[46] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, “A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks”, in *International Conference on Advanced Robotics*, 2009.

[47] L. Saab, N. Mansard, F. Keith, J. Y. Fourquet, and P. Soueres, “Generation of dynamic motion for anthropomorphic systems under prioritized equality and inequality constraints”, in *IEEE International Conference on Robotics and Automation*, 2011.

[48] L. Saab, O. Ramos, N. Mansard, P. Soures, and J. Y. Fourquet, “Generic dynamic motion generation with multiple unilateral constraints”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011.

[49] N. Mansard, “A dedicated solver for fast operational-space inverse dynamics”, in *IEEE International Conference on Robotics and Automation*, 2012.

[50] L. Saab, O. E. Ramos, F. Keith, N. Mansard, P. Soueres, and J.-Y. Fourquet, “Dynamic whole-body motion generation under rigid contacts and other unilateral constraints”, *IEEE Transactions on Robotics*, vol. 29, 2013.

[51] A. Escande, N. Mansard, and P.-B. Wieber, “Hierarchical quadratic programming: Fast online humanoid-robot motion generation”, *The International Journal of Robotics Research*, vol. 33, 2014.

[52] Lamiraux, Florent and Stasse, Olivier and Mansard, Nicolas, *Stack of Tasks*, Accessed: 2017-05-30, [Online] Available: `http://stack-of-tasks.github.io/`.

[53] Y. Nakamura and H. Hanafusa, “Optimal redundancy control of robot manipulators”, *The International Journal of Robotics Research*, vol. 6, pp. 32–42, 1987.

[54] S. B. Slotine, “A general framework for managing multiple tasks in highly redundant robotic systems”, in *proceeding of 5th International Conference on Advanced Robotics*, 1991.

[55] R. Fletcher, “A general quadratic programming algorithm”, *IMA Journal of Applied Mathematics*, vol. 7, pp. 76–91, 1971.

[56] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, “Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty”, *The International Journal of Robotics Research*, vol. 26, pp. 433–455, 2007.

[57] W. Decre, R. Smits, H. Bruyninckx, and J. D. Schutter, “Extending itasc to support inequality constraints and non-instantaneous task specification”, in *2009 IEEE International Conference on Robotics and Automation*.

[58] W. Decr, H. Bruyninckx, and J. D. Schutter, "Extending the itasc constraint-based robot task specification framework to time-independent trajectories and user-configurable task horizons", in *IEEE International Conference on Robotics and Automation*, 2013.

[59] R. Smits, H. Bruyninckx, and J. D. Schutter, *"iTaSC (instantaneous Task Specification using Constraints)"*, Accessed: 2017-09-27, [Online] Available: `http://www.orocos.org/wiki/orocos/itasc-wiki`.

[60] A. F. Vereshchagin, "Computer simulation of the dynamics of complicated mechanisms of robot-manipulators", *Engineering Cybernetics, 12(6)*, pp. 65–70, 1974.

[61] E. P. Popov, "Control of robots-manipulators", *Engineering Cybernetics*, 1974.

[62] C. F. Gauß, "Über ein neues allgemeines Grundgesetz der Mechanik.", *Journal für die reine und angewandte Mathematik*, vol. 4, pp. 232–235, 1829.

[63] E. Ramm, "Principles of least action and of least constraint", *GAMM-Mitteilungen*, vol. 34, pp. 164–182, 2011.

[64] J. L. Lagrange, "Mécanique analytique", *Mallet-Bachelier*, vol. 1, 1853.

[65] R. Bellman, "On the theory of dynamic programming", *Proceedings of the National Academy of Sciences*, vol. 38, pp. 716–719, 1952.

[66] K. Yunt, "On the relation of the principle of maximum dissipation to the principles of jourdain and gauss for rigid body systems", *Journal of Computational and Nonlinear Dynamics*, vol. 9, p. 031 017, 2014.

[67] F. E. Udwadia and R. E. Kalaba, "A new perspective on constrained motion", *Proceedings: Mathematical and Physical Sciences*, pp. 407–410, 1992.

[68] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control.* Cambridge University Press, 2017.

[69] A. Albu-Schäffer and G. Hirzinger, "Cartesian impedance control techniques for torque controlled light-weight robots", in *IEEE International Conference on Robotics and Automation*, 2002.

[70] B. Siciliano and O. Khatib, *Springer handbook of robotics.* Springer, 2016.

[71] J. De Schutter, D. Torfs, H. Bruyninckx, and S. Dutré, "Invariant hybrid force/position control of a velocity controlled robot with compliant end effector using modal decoupling", *The International Journal of Robotics Research*, vol. 16, pp. 340–356, 1997.

[72] E. Berger, "Friction modeling for dynamic system simulation", *Applied Mechanics Reviews*, vol. 55, pp. 535–577, 2002.

[73] J. S. Eck and W. Thompson, "Dissipative forces and quantum mechanics", *American Journal of Physics*, vol. 45, pp. 161–163, 1977.

[74] D. E. Stewart, "Rigid-body dynamics with friction and impact", *SIAM review*, vol. 42, pp. 3–39, 2000.

[75] E. Drumwright and D. A. Shell, “Modeling contact friction and joint friction in dynamic robotic simulation using the principle of maximum dissipation”, *Algorithmic foundations of robotics IX*, pp. 249–266, 2010.

[76] G. Pozharitskii, “Extension of the principle of gauss to systems with dry (coulomb) friction”, *Journal of Applied Mathematics and Mechanics*, vol. 25, pp. 586 –607, 1961.

[77] H. Olsson, M. Gäfvert, and P. Lischinsky, “Friction models and friction compensation”, *European Journal of Control*, vol. 4, pp. 176–195, 1998.

[78] S. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.

[79] F. Alizadeh, “Interior point methods in semidefinite programming with applications to combinatorial optimization”, *SIAM journal on Optimization*, vol. 5, pp. 13–51, 1995.

[80] W. I. Zangwill, “The convex simplex method”, *Management Science*, vol. 14, pp. 221–238, 1967.

[81] A. D. Polyanin, W. E. Schiesser, and A. I. Zhurov, “Partial differential equation”, *Scholarpedia*, vol. 3, 2008.

[82] A. Brondsted, *An introduction to convex polytopes.* Springer Science & Business Media, vol. 90, 2012.

[83] J. J. Craig, *Introduction to robotics: mechanics and control.* Pearson Prentice Hall Upper Saddle River, vol. 3, 2005.