Efficient Kinematic and Algorithmic Singularity Resolution for Multi-Contact and Multi-Level Constrained Dynamic Robot Control

Kai Pfeiffer

THÈSE

POUR OBTENIR LE GRADE DE

DOCTEUR

Délivrée par **l'Université de Montpellier**

Préparée au sein de l'école doctorale I2S - Information, Structure, Systèmes
Et de l'unité de recherche CNRS-AIST UMI2318/JRL

Spécialité : **SYAM - Systèmes Automatiques et Microélectroniques**

Présentée par : **Kai PFEIFFER**

Efficient Kinematic and Algorithmic Singularity Resolution for Multi-Contact and Multi-Level Constrained Dynamic Robot Control

Soutenue le 4 Décembre 2019 devant le jury composé de

Nicolas MANSARD	Directeur de Recherche, LAAS-CNRS	Rapporteur
Ludovic RIGHETTI	Associate Professor, New York University	Rapporteur
Vincent PADOIS	Associate Professor, Paris-Sorbonne Univ.	Examinateur, Président du jury
Ronan BOULIC	Senior Scientist, MER, Leader, IIG	Examinateur
Daniele PUCCI	Head of Research, DIC, IIT	Examinateur
Adrien ESCANDE	Senior Researcher, CNRS-AIST JRL	Codirecteur de thèse
Abderrahmane KHEDDAR	Directeur de Recherche, CNRS	Directeur de thèse

Acknowledgements

I would like to thank my Ph.D. supervisor Abderrahmane Kheddar for having given me the great chance of pursuing the doctorate at his renowned laboratory. His valuable advice and guidance throughout these three years contributed hugely to my work and my path in academia.

Without the brilliant knowledge and the guidance of my co-supervisor Adrien Escande this work would have not been possible. I am very thankful that I got the opportunity to work with him and for all the things I learned from him.

I am deeply thankful towards the AIST and Eiichi Yoshida for having hosted me in this beautiful country that is Japan.

I am very honoured to have my thesis reviewed by Nicolas Mansard, Ludovic Righetti, Ronan Boulic, Vincent Padois and Daniele Pucci with their great robotics knowledge and expertise.

Parts of chapter 4 are the result of fruitful discussions with Pierre-Brice Wieber. I am very thankful for the input he has given.

I want to thank both Gentiane Venture and Vincent Bonnet who guided me on my first steps in researching humanoid robotics.

I am thankful for having had such great colleagues during my time here. Special thanks goes to Pierre Gergondet who always spared his time to help me with the robot experiments.

I want to thank my friends who accompanied me throughout these years and made my time here unforgettable.

Last but not least, I am grateful for the endless support of my family both in Japan and Germany. I want to thank my aunt for her undemanding support, be it in sending food packages or going to the city hall with me. Finally, I want to thank my parents for always motivating me, believing in me and being there for me despite the distance.

K. P., September 2019, Tsukuba, Japan

Title: Efficient Kinematic and Algorithmic Singularity Resolution for Multi-Contact and Multi-Level Constrained Dynamic Robotic Control

Abstract: The introduction of the latest solvers for least-squares programs allows very fast solving of multi-level constrained robotic control problems. However, seldom are all these constraints perfectly achievable at the same time. This leads to kinematic and algorithmic singularities, an issue that has concerned the research of many roboticists. With this thesis, we build on this already available knowledge and introduce new methods for singularity resolution tailored to such hierarchically constrained least-squares programs for kinematic structures like humanoid robots.

In previous works the connection between the Gauss-Newton algorithm, Newton's method and the Levenberg-Marquardt algorithm has been shown. All have their origin in the second order Taylor expansion of the non-linear function of the quadratic error norm. It is expressible in least-squares form and suitable for our hierarchical least-squares solvers. The Gauss-Newton algorithm, which neglects Taylor second order terms, corresponds to the standard control algorithm exhibiting numerical instabilities when close to singularities. The Levenberg-Marquardt algorithm, which can be considered a classical measure of singularity resolution, approximates the Taylor second-order terms with a weighted identity matrix. Newton's method uses the analytic expression. Based on this circumstance, we assume that Newton's method is a valid approach for singularity resolution.

In a first step, we formulate the Lagrangian gradient and Hessian of the constrained optimization problem. The former enables positive definite updates of the Hessian by the Broyden-Fletcher-Goldfarb-Shanno algorithm which then can be used in a Quasi-Newton method. The latter one can be directly used for Newton's method of multi-level constrained optimization. It also inspires another approximation method based on the Symmetric Rank 1 method. We introduce a method to switch from the Gauss-Newton algorithm to Newton's method when in the vicinity of singularities. We validate our methods in simulation on a set of different kinematic problems going in and out of singularities while being numerically stable and exceeding Levenberg-Marquardt based methods in terms of error reduction.

Next, we make the step from pure optimization to robot control which requires the introduction of a suitable control time-step. This allows the formulation of controllers which imitate acceleration-based control in the velocity domain as it is required for Newton's method. By doing so, we are able to incorporate the robot dynamics in the form of the equation of motion into our control framework. Validation is conducted with real robot experiments on the HRP-2Kai.

We conclude this work by giving some hints on how to leverage bound constraints in hierarchical least-squares solvers. Our implementation is tailored to the LexLSI solver and allows computational speed-ups for problems dominated by bound constraints.

Keywords: Constrained optimization, Least-squares, Kinematic singularities, Algorithmic singularities, Legged Robotics, Humanoid robotics, Redundant robots

Discipline : Systèmes Avancés et Microélectronique

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
UMR 5506 CNRS/Université de Montpellier
Bâtiment 5 - 860 rue de Saint Priest

Résumé de la thèse

L'usage en robotique des tous derniers solveurs de problèmes de moindres carrés linéaires hiérarchiques permet de résoudre très rapidement des problèmes de contrôle multi-niveaux. Cependant, il est rare que les contraintes puissent être toutes satisfaites en même temps. Cela introduit des singularités cinématiques et algorithmiques, un problème qui a fait l'objet de nombreuses études.

Dans cette thèse, nous nous basons sur l'état de l'art et introduisons de nouvelles méthodes de résolution des singularités dédiées à ces problèmes de moindres carrés hiérarchiques pour des structures cinématiques telles que des robots humanoïdes.

Au voisinage des deux types de singularités, la linéarisation du modèle géométrique non-linéaire qui décrit des structures cinématiques comme les robots, typiquement utilisée en contrôle, n'approxime pas suffisamment bien la fonction originale. Cela se traduit par un mauvais conditionnement de la matrice Jacobienne, ce qui entraine des changements rapides de la configuration du robot. Comme cela peut amener à des comportements dangereux pour le robot ou son environnement, une telle situation doit être évitée.

Résolution des singularités dans la cinématique inverse avec priorité

Une approche classique pour la résolution des singularités pour du contrôle en vitesse consiste à introduire une régularisation ou un terme d'amortissement. Cela correspond à l'algorithme de Levenberg-Marquardt, communément utilisé pour trouver la solution de problèmes des moindres carrés. Un inconvénient important est qu'il n'existe pas de recette parfaite pour choisir l'amplitude de la régularisation. Cela peut conduire à des instabilités si la régularisation est trop faible, ou à une mauvaise vitesse de convergence si elle est trop forte.

Dans l'état de l'art, la connexion a été faite entre les méthodes de Newton, Gauss-Newton et Levenberg-Marquard. Toutes prennent leurs origines dans l'expansion de Taylor au second ordre de l'erreur quadratique des fonctions non-linéaires. Cette expansion peut s'approximer comme un problème de moindres carrés linéaire qui peut se

résoudre avec un solveur hiérarchique. Les trois méthodes se différencient comme suit:

- L'approche Gauss-Newton néglige une partie des termes du second ordre dans l'expansion de Taylor. C'est l'approche classique en contrôle, soumise à des instabilités numériques à proximité des singularités.
- L'algorithme de Levenberg-Marquardt approxime ces mêmes termes via un multiple de l'identité
- La méthode de Newton utilise l'expression analytique complète des termes du second ordre.

Du fait de ces caractéristiques, la méthode de Newton semble être un bon candidat pour résoudre les singularités. Elle est cependant pensée pour l'optimisation sans contrainte et il faut l'étendre au contrôle avec priorité.

L'optimisation avec contrainte et le contrôle avec priorité sont intimement liés. En optimisation avec contraintes, le but est de trouver un pas qui optimise (i.e. minimise ou maximise) un objectif tout en respectant les contraintes. Pour le contrôle avec priorité, l'objectif est de trouver un nouvel état (i.e. les vitesses articulaires dans un contrôleur en vitesse) qui minimise l'erreur à chaque niveau sans augmenter les erreurs des niveaux de plus hautes priorités. L'optimisation et le contrôle peuvent être connectés par la durée d'un cycle de contrôle.

Dans un premier temps, nous formulons un problème de contrôle avec priorité basé sur l'optimisation, avec une durée de cycle de 1 seconde. Cela nous permet d'utiliser les outils de la théorie de l'optimisation, en particulier la notion de lagrangien d'un problème avec contraintes. Le gradient et la matrice hessienne correspondant au lagrangien peuvent être utilisés de deux façons:

- Le gradient permet d'obtenir une approximation définie positive de la matrice hessienne grâce à la méthode BFGS, ouvrant la porte à l'utilisation la méthode quasi-Newton
- La matrice hessienne du lagrangien peut être utilisée directement avec la méthode de Newton généralisée à l'optimisation hiérarchique. Son calcul inspire aussi une autre approximation basée sur la méthode SR1. Comme la matrice peut ne pas être définie positive, une régularisation couteuse basée sur une décomposition SVD est nécessaire pour retourner à une forme de moindres carrés linéaires et l'utilisation d'un solveur hiérarchique.

Un problème de la méthode de Newton hiérarchique est que la matrice hessienne ou son approximation est de rang plein. Cela implique que toutes les articulations sur la chaîne cinématique d'une tâche sont utilisées et ne peuvent pas servir pour réaliser des tâches de plus basses priorités. Nous présentons une méthode pour passer, de manière fiable, de l'approche Gauss-Newton à la méthode de Newton seulement au voisinage des singularités. Cela assure que les articulations de la chaîne cinématique concernée sont complètement utilisées quand la tâche correspondante est singulière, mais qu'elles sont autrement disponibles pour les tâches de plus basses priorités.

Nous présentons également une méthode d'adaptation de la région de confiance adaptée à un contexte de contrôle robotique temps-réel. Elle examine directement la stabilité des vitesses articulaires solutions. Si plusieurs changements de signe consécutifs sont observés sur une même articulation, la taille de la région de confiance est diminuée, uniquement pour cette articulation. Cela diffère des méthodes en optimisation pure qui décroissent la taille de la région pour toutes les variables, et peuvent se permettre de recalculer la solution avec la nouvelle taille, chose impossible en contrôle de part la contrainte de temps réel.

Les trois méthodes (Newton, Quasi-Newton basée BFGS et Quasi-Newton basée SR1) sont validées en simulation sur un simple robot stick en 2 dimensions puis sur le robot humanoïde 3-D HRP-2KAI. Nous montrons que ces méthodes permettent un comportement stable en singularités et au voisinage de celles-ci, tout en permettant une meilleure réduction d'erreur que les méthodes basées sur Levenberg-Marquardt.

Résolution des singularités dans le contrôle dynamique avec priorité

Le contrôle dynamique est formulé en accélération car l'équation de la dynamique est aussi exprimée à ce niveau. Celle-ci connecte les quantités cinématiques (positions, vitesse et accélérations articulaires) et dynamiques (couples articulaires et torseurs de contact), et s'assure de la faisabilité physique en ne permettant une évolution du robot que si les torseurs nécessaires peuvent être appliqués sur l'environnement aux points de contacts.

Dans un premier temps, nous introduisons les méthodes de Gauss-Newton et Newton hiérarchiques pour le contrôle. Cela demande d'adopter un pas de contrôle souvent bien plus faible que le pas unitaire que nous avons pris comme hypothèse dans la dérivation de la méthode de Newton hiérarchique pour l'optimisation.

La résolution des singularités dans le contrôle en vitesse a fait l'objet de nombreuses études, ce qui n'est pas le cas pour le contrôle en accélération. De manière similaire au contrôle en vitesse, un modèle de contrôle linéarisé est défini (cette fois au second ordre, avec comme variables de contrôle les accélérations articulaires), ce qui mène aux mêmes problèmes de pertes de rangs des matrices jacobiennes à l'approche des singularités. Nous montrons comment un terme d'amortissement ou une régularisation exprimés directement en accélération a une influence négative sur la convergence des contrôleurs de second ordre. La méthode de Newton hiérarchique amènerait le problème car elle est très liée à l'amortissement présent dans Levenberg-Marquard. Cela nous sert de motivation pour exprimer le contrôle dynamique, y compris l'équation de la dynamique et les contrôleurs de mouvement, dans le domaine des vitesses en utilisant une intégration explicite. Ainsi, la méthode de Newton hiérarchique, qui est supérieure à celle de Levenberg-Marquardt en termes de convergence et de facilité d'utilisation (car il n'y a pas de gain d'amortissement à régler), est bien définie.

Nous validons en simulation le caractère reproductible du contrôle basé accélération

et exprimé en vitesse. Nous montrons de plus l'efficacité de la méthode de Newton hiérarchique pour prévenir les comportements singuliers ou instables dans le contrôle dynamique hiérarchique. La supériorité de la convergence obtenue par rapport aux approches par amortissement est aussi confirmée par trois expériences sur robot réel.

Démonstration robotique avec un solveur pour optimisation contrainte

Bien que permettant de faire du contrôle avec un nombre arbitraire de niveaux de priorité, les moindres carrés linéaires hiérarchiques ne se sont pas encore véritablement imposés. De nombreux laboratoires, JRL compris, s'appuient sur des solveurs de programmes quadratiques (QP) classiques pour un contrôle avec deux niveaux: les contraintes et les objectifs. Il n'y a de fait pas de séparation lexicographique claire entre la faisabilité physique, l'intégrité du robot, sa stabilité et les tâches à réaliser. De plus, les contraintes doivent être réalisables à tout instant. Les singularités au niveau des objectifs sont gérées par l'usage d'une tâche de posture qui a pour effet d'introduire une régularisation au niveau des accélérations articulaires et permet de s'approcher des singularités.

Malgré leurs défauts, les approches de contrôle par QP sont puissantes, et nous montrons leurs capacités dans un scénario d'application industrielle: un robot humanoïde serrant des écrous, dont les dimensions sont bien inférieures à celles du robot, en se servant d'informations visuelles et de forces. Le robot est capable de localiser correctement un écrou grâce à des marqueurs visuels, puis d'insérer celui-ci dans la tête d'une clé dynamométrique du commerce qu'il tient à la main. En utilisant les informations d'un capteur de force, le robot peut confirmer si l'insertion s'est bien déroulée et procède alors à serrer l'écrou, tout en maintenant les forces de réaction faible grâce à un contrôle compliant et qui apprend les corrections à apporter. La robustesse du procédé est confirmée par le serrage répété de plusieurs écrous à différents endroits.

Accélérations des calculs pour un solveur des moindres carrés hiérarchiques

Nous concluons ce travail en donnant quelques indications sur la façon de prendre en compte efficacement les contraintes de bornes dans un solveur de moindres carrés hiérarchiques. Notre implémentation est dédiée au solveur LexLSI Dimitrov et al. [2015] et permet d'accélérer les calculs quand le problème à résoudre contient de nombreuses bornes. Nous proposons aussi de ne recommencer que les morceaux de factorisation nécessaires entre deux itérations du solveur, en fonction du niveau de hiérarchie où une contrainte a été activée ou désactivée. De plus, la connaissance de la structure des matrices du problème est utilisée pour réduire le nombre de calculs à effectuer.

Ces améliorations sont évaluées sur un ensemble de 5 tests, chacun sur aspect différent des modifications. Tous les tests montrent des réductions de temps de calculs, et que la prise en compte des informations supplémentaires nécessaires aux améliorations n'induit qu'un surcoût de calcul négligeable.

Conclusion

Grâce aux méthodes proposées, nous sommes capables de contrôler des structure cinématiques avec une approche par priorités au voisinage des singularités. Les problèmes suivants demandent cependant des efforts de recherche supplémentaires:

- Dans des situations hautement dynamiques, des problèmes d'active-set peuvent survenir. Dans de telles situations, un nombre important d'itérations du solveur contrevient au besoin du temps réel. Même si nous avons proposés quelques améliorations au solveur LexLSI, elles ne sont pas toujours suffisantes pour résoudre ce problème. De futurs travaux devront d'abord étudier pourquoi le nombre d'itérations augmente tant dans des situations dynamiques. La seconde étape sera d'améliorer encore le solveur, en implémentant une méthode de mise à jour des décompositions numériques internes plus avancée, ou en s'intéressant à des stratégies plus efficaces pour l'activation/désactivation des contraintes et le démarrage à chaud.
- Il sera nécessaire de trouver des méthodes plus avancées d'adaptation de la taille de la région de confiance, afin de permettre des mouvements très rapides. Le rayon de confiance doit être suffisamment petit à proximité des singularités pour garantir une bonne précision de l'approximation de Taylor au second ordre et assurer la stabilité. Cependant, cela limite la vitesse maximum dans les configurations régulières, qui pourraient avoir besoin d'une augmentation plus rapide du rayon pour permettre des mouvements plus rapides.
- Enfin, dans ce travail nous avons introduit le concept de faisabilité dynamique dans un contrôle en vitesse, en introduisant l'équation de la dynamique et les contraintes dynamiques correspondantes au plus haut niveau de la hiérarchie. Cela force les sorties des niveaux de plus basses priorités à suivre les principes physiques. Cependant, il sera nécessaire de regarder plus en détail comment le contrôle en force (à un niveau de priorité basse) et le concept de cohérence dynamique s'incorporent et interagissent avec notre schéma de résolution des singularités.

Contents

Nomenclature

Bold small letters stand for vectors and bold capital letters for matrices.

Acronyms and abbreviations

acc.	Acceleration
ADLS	Adaptively Damped Least Squares
AH	Analytic Hessian
asc	Active-set change
BFGS	Broyden Fletcher Goldfarb Shanno
CoM	Centre of Mass
ctr.	Constraint
ctrl	Control, controller
DHQP	Damped Hierarchical Quadratic Programming
DoF	Degree of Freedom
ef	End-effector
exp.	Experiment
FFC	Force Field Characteristics
GN	Gauss-Newton
HHT	Householder Transformation
HQP	Hierarchical Quadratic Programming solver Escande et al. [2014]
LexLSAug2AH	Lexicographic Least Squares Augmented with AH
LexLSAug2BFGS	Lexicographic Least Squares Augmented with BFGS
LexLSAug2SR1	Lexicographic Least Squares Augmented with SR1
LexLSI	Lexicographic Least Squares solver for inequality problems Dimitrov et al. [2015]
LM	Levenberg-Marquardt
LSP	Linear Least-Squares Problem
LSSOL	Quadratic programming solver presented in Gill et al. [1986]
mod	Modified
obj.	Objective

ODE	Ordinary Differential Equation
optim	Optimization
ori	Orientation
P	Proportional
PD	Proportional Derivative
pos	Position
QP	Quadratic Programming
SR1	Symmetric Rank 1
SVD	Singular Value Decomposition
T	Test
vel.	Velocity
WLS	Weighted Least Squares
ZMP	Zero Moment Point

List of symbols

$(k-1), (k), (k+1)$	Indices for the previous, current and next control iteration
Δ, $\boldsymbol{\Delta}$	Trust region radius with $\boldsymbol{\Delta} = \Delta \boldsymbol{I}$
Ψ	Control integration k with $\boldsymbol{\Delta a} \approx \boldsymbol{0}$
Σ	Sum of the amplitude of all the sign changes in the vector $\boldsymbol{\Delta a}$
Ξ	Normed integrated error of equality with highest priority level
$\boldsymbol{A}^{+}$	Moore-Penrose pseudo-inverse (or generalized inverse) of $\boldsymbol{A}$
$\boldsymbol{A}^{\ddagger}$	Hierarchical pseudo-inverse of $\boldsymbol{A}$ Escande et al. [2014]
$\boldsymbol{H}$	Hessian of scalar function
$\boldsymbol{I}_l^*$	Identity matrix with entries only on joints corresponding to the kinematic chains of levels $\leq l$
$\boldsymbol{I}$	Identity matrix
$\boldsymbol{J}$	Jacobian matrix
$\boldsymbol{\Delta a}$	Difference of $\boldsymbol{a}$ between two iterations $k+1$ and k
$\boldsymbol{\Lambda}$	Matrix containing all Lagrange multipliers
$\dot{\boldsymbol{a}}$, $\ddot{\boldsymbol{a}}$	First and second time derivative of vector $\boldsymbol{a}(t)$
$\boldsymbol{\gamma}$	Generalized contact wrenches
$\hat{\boldsymbol{H}}_l$	Lagrangian Hessian of level l
$\hat{\boldsymbol{I}}$	Unordered identity matrix
$\boldsymbol{\lambda}_{i,l}$	Lagrange multipliers, indicating conflict of level l with constraint on level i. Corresponds to $\boldsymbol{\Lambda}(i,l)$.
$\boldsymbol{\tau}$	Joint torques
$\underline{\boldsymbol{A}}_l$	Stacked matrix containing the matrices $\boldsymbol{A}_1$ to $\boldsymbol{A}_l$
$\underline{\boldsymbol{a}}_l$	Stacked vector containing the vectors $\boldsymbol{a}_1$ to $\boldsymbol{a}_l$
$\boldsymbol{v}$	Matrix containing the column lengths of Jacobians of all levels
$\boldsymbol{a}^{(k-1),(k),(k+1)}$	Value of $\boldsymbol{a}$ at iterations $k-1$, k or $k+1$
$\boldsymbol{a}_d$	Desired value of vector $\boldsymbol{a}$

$\boldsymbol{q}$	Joint angles
$\boldsymbol{w}$	Slack variable relaxing infeasible constraints
ϵ	Quadratic norm of the slack $\boldsymbol{w}$
$\leqq$	Symbol gathering both equality and inequality constraints
$\mathcal{A}_l$	Active-set containing the active constraints of level l
$\mathcal{L}_l$	Lagrangian function of the constrained optimization problem of level l
ν	Threshold of quadratic norm of slack for switching method
ξ	BFGS threshold for positive definite curvature
ζ	SR1 and BFGS error norm threshold for initialization
$c_{\mathcal{A}}$	Level of highest priority of all the levels where active-set changes occurred during a control iteration
k_p	Proportional gain, spring stiffness
k_v	Derivative gain, damping value
l_{asc}	Level where the active-set change occurred during one iteration of the active set search
m_l	Row dimension / number of constraints of Jacobian of level l
n	Number of variables
p	Number of priority levels
min.	Minimize

Introduction

Over the course of time humans have unearthed more and more ways of describing the surrounding world. From early cave drawings to human language we – in cosmic dimensions – recently have found means of mathematically expressing the observable regularities of our universe by 'universal laws':

> "If a certain regularity is observed at all times and all places, without exception, then the regularity is expressed in the form of a 'universal law' ".
>
> —Rudolph Carnap. "The Value of Laws: Explanation and Prediction.", 1966

Laws as such have manifested themselves for example in *Hubble's law* describing the expansion of the universe. While such a law has little prevalence on a day-to-day basis, *Newton's law of universal gravitation* and *Newton's Three Laws of Motion* are directly describing the forces and moments any body experiences through gravitational and inertial effects. Us humans instinctively adapt our movements to the given circumstances through experience gained from trial and error. With the rise of robots however, and especially of legged humanoid robots, we need to peculiarly design the robot's motion to enable physical stability without falling. The motion thereby needs to be governed by the given laws of motion or physics. Or expressed mathematically:

$$\text{Physical feasibility} \succ \text{Physical stability}$$

We have now formulated a *hierarchy* where the robot is aware of the laws of physics, i.e. it is *physically feasible*, and within the boundaries of these laws designs a motion which enables for example walking without falling, i.e. *physical stability* is achieved. We consider these two goals to be *lexicographically ordered* and separated on two different lexicographical *priority levels*.

We refer to the law of physical feasibility as *constraint* (ctr.) and the goal of physical stability as *objective* (obj.). This terminology is borrowed from *constrained optimization*. It allows to *optimize* (i.e. minimize or maximize) an objective by finding a suitable robot state which is within the limits that are given by the constraints.

However, besides physical feasibility and stability there might be further constraints we want to respect and objectives we want to optimize when defining a robotic problem.

Such constraints could be related to the robot's own *safety* like obeying joint movement limits, joint torque limits or self-collision constraints. Furthermore, contacts with the environment need to be established, requiring the contacting body parts like feet or hands of the robot to be at specific positions. Other objectives might include to rotate the robot's head to a point of interest or follow a moving target with the hand.

At this point, the terminology of what can be considered a constraint and what an objective starts to become blurry. This dilemma becomes especially relevant when using the classical approach of a two level robot control hierarchy. All the constraints are put on the first level and all the objectives are put on the second level. In the presence of multiple constraints and multiple objectives they can be weighted against each other according to their relative importance. It becomes obvious that this is not an exact image of the reality since the ultimate constraint of physical feasibility is not lexicographically separated from physical stability and robot safety constraints.

It has been proposed however to introduce a hierarchy with n levels of the form

$$\text{Ctr. } 1 \succ \text{Ctr./Obj. } 2 \succ \cdots \succ \text{Ctr./Obj. } p-1 \succ \text{Obj. } p\,. \tag{1}$$

Any level between the first and the last one can be interpreted as a constraint and an objective at the same time. In such a hierarchy, any level l is only optimized to the extent of not influencing the optimality of previous levels $1, \ldots, l-1$. We can therefore formulate a lexicographical problem exactly representing the 'needs' of the robot:

$$\text{Physical feasibility} \succ \text{Robot safety} \succ \text{Physical Stability} \succ \text{Obj. } 1 \succ \cdots \succ \text{Obj. } p\,.$$

In the following, we use the term 'constraints' and 'objectives' interchangeably.

Constraints are formulated as the linearized representation of the non-linear geometric description of the robot. This linear approximation fails in certain cases and no movement can be generated to bring the constraint closer to its goal. This is typically the case if constraints become *infeasible* due to *conflict* between each other. We call such a state a *kinematic* singularity. If one constraint is of lower priority with respect to the other, the lower priority constraint is said to be in *algorithmic singularity* with the higher priority one. In both cases numerical instabilities arise with large changes of the joint values which can bring harm to the robot structure.

Resolving these kinematic and algorithmic singularities in an efficient manner is the main topic of this thesis. We derive, implement and validate a method which improves state of the art resolution methods in terms of convergence and easiness of use by reducing tuning efforts.

This thesis is then divided as follows:

- First, we give some background on research that has been conducted so far on hierarchical constrained optimization problems and handling of kinematic and algorithmic singularities in kinematic and dynamic control scenarios. At the same time we outline the problem at hand and our contributions (see chapter 1).
- In chapter 2 we proceed to give some details on robotic control and constrained least-squares programs.

- The obtained knowledge is then applied in a robot demonstration conducting a task typically seen in industrial manufacturing with a two level control hierarchy. The disadvantages of this type of control architecture are detailed (see chapter 3).
- The drawbacks of the two level hierarchical control framework are overcome by using state of the art hierarchical least-squares solvers. We proceed to tackle the issue of kinematic and algorithmic singularities for kinematic control (see chapter 4).
- We extend the methods of singularity resolution in kinematic control to dynamically feasible kinematic control (see chapter 5).
- We conclude this thesis by giving some hints on how to leverage bound constraints in the used hierarchical least-squares solver in order to improve the computational speed (see chapter 6).

CHAPTER 1

State of the art, problem formulation and contributions

A *robot* can be simply interpreted as a *kinematic structure* with certain points of interest. These points, that we call *end-effectors*, can be located in *operational* or *task-space* by a combination of non-linear geometric functions describing the *kinematic relationship* of *links* and *joints*. An example might be a humanoid robot (kinematic structure) standing in a laboratory (3-D operational space) and moving its CoM and grippers (end-effectors) by rotating its motors (joints as rotating bearings) which are connected to each other (kinematic relationship) through rigid (or possibly flexible) materials (links).

1.1 Constrained optimization

It is now often desired to minimize the *error* of an end-effector between its actual and targeted value. At this point, both the world of robotics and *constrained optimization* start to merge. A constrained optimization problem takes the general form

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) \tag{1.1}$$

$$\text{subject to (s.t.)} \quad \boldsymbol{c}_E(\boldsymbol{x}) = \boldsymbol{0} \tag{1.2}$$

$$\boldsymbol{c}_I(\boldsymbol{x}) \geq \boldsymbol{0} \tag{1.3}$$

where a scalar (for example error) function $f(\boldsymbol{x})$ depending on the state vector $\boldsymbol{x}$ is supposed to be minimized (or maximized) while not violating an array of equality $\boldsymbol{c}_E(\boldsymbol{x})$ and / or inequality constraints $\boldsymbol{c}_I(\boldsymbol{x})$.

The use of *Lagrange multipliers* Bertsekas [1982] allows to solve equality constrained optimization problems by finding the stationary points of the Lagrangian function

$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}_E) = f(\boldsymbol{x}) + \boldsymbol{\lambda}_E^T \boldsymbol{c}_E(\boldsymbol{x})$ with the Lagrange multipliers $\boldsymbol{\lambda}_E$. The corresponding first order optimality condition can be formulated as $\nabla \mathcal{L}_{\boldsymbol{x},\boldsymbol{\lambda}_E}(\boldsymbol{x}, \boldsymbol{\lambda}_E) = \mathbf{0}$.

In the presence of inequality constraints, the Lagrangian function becomes $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}_E^T \boldsymbol{c}_E(\boldsymbol{x}) + \boldsymbol{\lambda}_I^T \boldsymbol{c}_I(\boldsymbol{x})$ and the first order optimality conditions from above extend to the Karush-Kuhn-Tucker (KKT) Karush [1939]; Kuhn et Tucker [1951]; Nocedal et Wright [2006] conditions:

$$\nabla_{\boldsymbol{x}} \mathcal{L} = \mathbf{0} \qquad \text{(stationarity)} \tag{1.4}$$

$$\boldsymbol{c}_E(\boldsymbol{x}) = \mathbf{0}, \quad \boldsymbol{c}_I(\boldsymbol{x}) \geq \mathbf{0} \qquad \text{(primal feasilibity)} \tag{1.5}$$

$$\boldsymbol{\lambda}_I \geq \mathbf{0} \qquad \text{(dual feasilibity)} \tag{1.6}$$

$$\lambda_i c_i = 0 \quad \text{for } i \in I \qquad \text{(complementary condition / slackness)}\,. \tag{1.7}$$

An important statement here is the *complementary condition* $\lambda_i c_i = 0$ which states that an inequality constraint $i \in I$ is either *active* with $c_i = 0$ and $\lambda_i > 0$ or is *inactive* with $c_i > 0$ and the corresponding Lagrange multiplier (or KKT multiplier) are zero $\lambda_i = 0$. *Violated* or *infeasible* constraints $c_i < 0$ are not allowed.

There are many forms of optimization problems, differing in the structure of the constraint and the objective at hand. If both the constraint and the objective are non-linear *Sequential Quadratic Programming* (SQP) Boggs et W. Tolle [1995] is a possible solving method. SQP repeatedly applies Newton-steps to the first-order optimality condition until the problem converges.

If all the constraints as well as the objectives are linear with $\boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ and $f(\boldsymbol{x}) = \boldsymbol{d}^T \boldsymbol{x}$ we speak of *Linear Programming* and corresponding solvers like the *Simplex Method* Dantzig [1991] can be used.

If the objective is a *linear least squares problem* (LSP) $f(\boldsymbol{x}) = \|\boldsymbol{A}_2\boldsymbol{x} - \boldsymbol{b}_2\|_2^2$ and the constraints are linear $\boldsymbol{c}(\boldsymbol{x}) = \boldsymbol{A}_1\boldsymbol{x} - \boldsymbol{b}_1$ we can apply *Quadratic Programming* (QP) Frank et Wolfe [1956]; Gill et al. [1986] methods, for example the range-space or null-space methods.

Problem formulation and contributions

We use such a constrained LSP solver to conduct a high precision industrial task with a humanoid robot. However, such a two level hierarchical LSP formulation comes with certain disadvantages which are outlined in chapter 3.

Those disadvantages are overcome in later chapters with a special form of constrained LSP's as given in Dimitrov et al. [2015]. Here, a cascade of LSP's is introduced to realize any number of priority levels as depicted in (1), see next section 1.2 for details. While the solver called LexLSI solves these problems very efficiently, given structures of the problem can be taken advantage of in order to save computational cost. We explore those and modify the hierarchical least-squares solver accordingly, see chapter 6.

1.2 Prioritized kinematic control

In previous works least-squares solvers based on the null-space method with an unlimited number of constraint levels have been developed Siciliano et Slotine [1991]. They are designed for the *inverse kinematics* control of kinematically redundant robots, i.e. robots which have more degrees of freedom (DoF) than required to fulfil a given task Chiaverini [1997]. For that, a *linearized control model* of the non-linear error function is solved in a least-squares way. If there are remaining DoF's, a second level can then be solved at best in its projection onto the null-space of the first priority level without interfering with its optimality.

The above approach was developed for robotic problems with equality constraints. The seminal work of Kanoun et al. [2011] resolved this limitation and enabled the formulation of inequality constraints at any priority level. Here, a LSP for each level is stated which are constrained such that the optimality of the LSP's of the previous levels is maintained. This resembles a *cascade* of constrained LSP's. Thereby, the notion of the slack variable $\boldsymbol{w}$ allows the violation of constraints $c_i(\boldsymbol{x}) \leqq 0$ by introducing a relaxation $c_i(\boldsymbol{x}) + w_i \leqq 0$. The downside of this approach is a relatively high computation cost because each constrained LSP is redoing some work of the previous ones. Additionally, similarities of the active-sets in consecutive control iterations can not be leveraged by warm-starting each hierarchical LSP.

The work of Escande et al. [2014] improved the computation time significantly by solving each level only once by considering the special structure of the constrained LSP's. Thereby, each level's LSP is smaller than the previous one, enabling further computational speed-up. Additional improvement was achieved in Dimitrov et al. [2015]. The special structure of non-orthogonal null-space bases enables cheap matrix products and computationally efficient blocking methods.

However, the conjunction of a linear control model and a hierarchical least-squares resolution is failing when approaching *singularities*: the constraint matrix (or Jacobian) of one of the tasks (*kinematic singularities*) or its projection onto the null-space of higher-priority tasks (*algorithmic singularities*) is becoming nearly rank-deficient Chiaverini [1997]. Besides the general problem of inverting such a matrix, the high condition number of the Jacobian amplifies numerical noise due to limited machine precision which leads to numerical instabilities with very high joint velocities and accelerations.

Kinematic singularities can be predicted and prevented with an analytical robot workspace analysis Shamir [1990]; Tourassis et Marcelo H. Ang [1992]; Khalil et Dombre [2002]. Algorithmic singularities however depend on the conflict with higher priority tasks at the current robot configuration and therefore are harder to analyse Chiaverini [1997]. Furthermore, hierarchical solvers are intended to be employed on complex robots in any industrial setting like Airbus airplane assembly halls. Engineers without deeper understanding of these issues should be able to set up robot problems safely and easily, even if for example sensor readings give targets that are outside of the feasible workspace of the robot.

Directly limiting the magnitude of the solution is achieved by *damping* which cor-

responds to the *Levenberg-Marquardt* (LM) algorithm Wampler et Leifer [1988]; Chiaverini et al. [1994]; Baerlocher et Boulic [2004]. For this, a weighted term minimizing the solution norm is added to the least-squares. Thereby, this weight can be either fixed or can be automatically adapted, for example with the so-called manipulability factor Nakamura et Hanafusa [1986] or some estimate of the minimum singular value Maciejewski et Klein [1988]; Chiaverini et al. [1991]. Harish et al. [2016] find an optimal damping value by parallelly solving the same control problem with different damping values and choosing the one which leads to the largest error reduction. Sugihara [2011] proposes to use the quadratic norm of the task error plus some small weight adapted to the kinematic structure. The so-called truncated SVD decomposition discards singular values in the calculation of the Jacobian's pseudo-inverse if they are below a certain threshold Hansen [1987]. Another approach prohibits infeasibility by clamping end-effector targets and introduces damping for the robot's joints distinctively by considering all the Jacobian's singular values Buss et Kim [2005]. The Jacobian transpose method avoids the Jacobian inversion altogether by using its transpose instead Wolovich et Elliott [1984]. However, this negatively influences the convergence behaviour. In Nenchev et al. [2000], singularities in tracking of parametrized trajectories are handled by re-parametrization.

These singularity resolution methods are either designed for equality only problems or are not extended for multi-level constrained hierarchies including inequalities as handled by the solvers in Escande et al. [2014]; Dimitrov et al. [2015]. Consequently, these solvers' evaluation was confined to simulation or well-calibrated experiments despite their efficiency. To the best of our knowledge, singularities in multi-level hierarchies have only been considered in Herzog et al. [2016] where the authors propose an efficient cascade of constrained LSP's with fixed damping on each level. The damping values need to be tuned by hand and might only be appropriate for a certain set of robotic scenarios but might lead to numerical instabilities in other situations.

Problem formulation and contributions

Approximating non-linear functions by simpler models is an approach rooted in optimization and it certainly is interesting to draw parallels between control and non-linear least-squares optimization methods Deo et Walker [1993]; Wieber et al. [2017]: Damping the joint velocities can be interpreted as approximating the second order derivatives of the Taylor expansion of the quadratic task error norm Dennis et al. [1981]; Deo et Walker [1993] as a weighted identity matrix while the *Gauss-Newton* (GN) algorithm simply ignores it. Now, what if numerical instabilities caused by singularities are just a sign of insufficient modelling? This is the point where we try to tackle the problem: improve the model in the vicinity of singularities by incorporating second-order information since ignoring it does not seem to suffice. Newton's method does so by incorporating the analytical second order derivatives. These are usually costly to compute, but methods have been developed to approximate them Dennis et al. [1981]; L. Toint [1987], the most widely used being the BFGS algorithm independently intro-

duced by Broyden Broyden [1970], Fletcher Fletcher [1970], Goldfarb Goldfarb [1970] and Shanno Shanno [1970]. Throughout this thesis we refer to such Quasi-Newton methods as well as Newton's method simply as Newton's method.

Newton's method is not specifically designed for multi-level hierarchies. An approach using BFGS updates for the second order information weights its objectives to establish a non-strict hierarchy Zhao et Badler [1994]. Others Sugihara [2011]; Chiaverini [1997] proposed to incorporate the LM algorithm into strict hierarchical schemes by damping each objective of a hierarchical least-squares solver.

In this thesis (see chapter 4), we combine strictly prioritized task-space control with the high model accuracy provided by Newton's method. Based on Lagrangian derivations of the constrained optimization problem we formulate the hierarchical Newton's method. It extends to any number of hierarchy levels consisting equality and inequality constraints, may they be feasible or not, while encompassing numerical stability. A method is presented which reliably detects singular configurations and switches from the GN algorithm to Newton's method.

We validate our methods with a simulation test bench with a simple 2-D stick robot and a complex 3-D humanoid robot. They outperform current state of the art methods of singularity robust kinematic control in terms of error convergence while being computationally competitive.

1.3 Prioritized dynamic control

The second order differential equation of motion connects the entities of kinematics (joint positions, velocities and accelerations) and dynamics (joint torques and contact wrenches). Derived from the Newtonian laws of motion (other derivations from the D'Alembert's principle, Euler-Lagrange equations or Hamilton's equations exist), it ensures *physical feasibility* by allowing a change of momentum only if the necessary wrench can be exerted on the environment through contact points. In general, the equation of motion can be considered full rank if the inertia matrix is physically consistent and therefore positive definite Udwadia et E. Kalaba [1992]; Udwadia et Schutte [2010].

While kinematic control of robots can be sufficient for fixed base robots Caccavale et al. [1997]; Wang et al. [2010] this does not hold for legged robots with an un-actuated free-flyer base and unilateral friction contacts established with its environment. Maintaining a *physically stable* posture is one of the main concerns when it comes to their control. Different approaches have been proposed and applied: keeping the static center of mass (CoM) Wieber [2002]; Bonnet et al. [2018] or the dynamic zero moment point (ZMP) Vukobratovi et Stepanenko [1972] within the convex hull of the support area is suitable for horizontal grounds, while extensions have been made for uneven terrain for the CoM Bretl et Lall [2008] and the ZMP Sardain et Bessonnet [2004]. The Foot-Rotation Indicator Goswami [1999] or the Capture Point Pratt et al. [2006] are other possibilities to quantify the degree of physical stability of a moving robot.

Model predictive control on the other hand can give the robot the foresight to provide the required contact wrenches and motor torques to tackle dynamic situations. It has been applied on the ZMP of an inverted pendulum model Kajita et al. [2003]; Wieber [2006], the linear and angular momentum around the CoM Audren et al. [2014] or the CoM accelerations Caron et Kheddar [2016].

Consequently, the aim for physical stability governs the robot control within the bounds given by the robot's physical feasibility. Put into lexicographic order this can be expressed as

$$\text{Physical Feasiblity} \succ \text{Physical Stability}. \tag{1.8}$$

Such a prioritization is naturally respected in the context of prioritized torque control or prioritized *inverse dynamics*. Khatib [1987] develop a controller for a fixed base robot which compensates the static and dynamic forces. Prioritized operational forces are realized with a special null-space projection which dynamically decouples the different levels. This concept is later coined as *dynamical consistency* Khatib [1995]. In this work simultaneous force and motion control is realized in orthogonal directions of each other. Further work of the same group expands the prioritization to any number of levels Sentis et Khatib [2004] and to free-floating robots Sentis et Khatib [2005]. Righetti et al. [2011] map the equation of motion into the dynamically consistent null-space of the contact constraints. They then proceed to formulate computationally more efficient bases derived from simple orthogonal decompositions of the constraint matrix. Prioritized inverse dynamics controllers are then devised.

A strategy for torque control in the neighbourhood of kinematic singularities was devised in Kyong-Sok Chang et Khatib [1995]. In order to overcome motion locks due to zero commanded torque, the rate of the singular values associated with singular directions is controlled via a potential function. Null-space motion is realized by rotating Jacobians in such a way that the identified singular directions are eliminated. Dietrich et al. [2012] implement a prioritized torque controller with statically consistent Dietrich et al. [2015] null-space bases. These bases are calculated using a truncated pseudo-inverse with smooth transitions in order to handle singular cases. Dynamical consistency and singularity robustness is realized by damping kinematic null-space bases which are then transformed back into the dynamic domain Dietrich et al. [2015].

However, such a projector based approach is generally not suitable for problems with inequality constraints. Another way of realizing such kind of physically feasible hierarchy is by defining a constrained optimization problem where the equation of motion and all the corresponding dynamics constraints are put on the constraints level, while the robot control is put on the objective levels. Such a formulation is not necessarily limited to torque control but also allows kinematic control since joint accelerations (*forward dynamics*) are computed as well. These two level constrained optimization problems do not allow a strict prioritization between physical feasibility (equation of motion), safety (joint limits) and physical stability (CoM) constraints but they are all put on the first level. Handling of inequality constraints is possible. Abe et al. [2007] implement such a hierarchy for a robot with an un-actuated base and

frictional contacts with the environment. The contacts' acceleration in operational space is asked to be zero and the contact forces are required to be within the linearized friction cone. Several objectives including a CoM regulator are defined on the second level and are weighted against each other. In Collette et al. [2007], two QP's are solved: the static one computes a goal position for the CoM which is in accordance with desired contact forces at the friction contacts. It is then fed to the dynamic QP returning the corresponding joint torques to reach the goal. Bouyarmane et Kheddar [2011] use such a QP formulation to realize motions between static postures prescribed by a multi-contact stances planner. Contacts are added and removed on-the-go. Further examples are given in the context of humanoid robot walking Feng et al. [2013]; Kuindersma et al. [2014] and ladder climbing Vaillant et al. [2016].

Saab et al. [2013] extends the hierarchy to any number of levels. The equation of motion on the first level realizes physically feasible control of the motion controllers on some lower priority level. Mansard [2012] notes that simultaneously solving the forward and inverse dynamics in such a hierarchy can lead to numerical instabilities due to the coupling of motion (joint accelerations) and actuation (joint torques). An appropriate decoupling of the dynamics by using null-space bases of the contact constraints is presented. Herzog et al. [2016] likewise implements such a hierarchy. A momentum regulator controls either the accelerations or the contact forces. Another example for prioritized force control is given in Sherikov et al. [2015]. The contact forces on the hand are minimized on some low priority level by using model predictive control during a reaching task.

Problem formulation and contributions

Dynamic control is usually formulated in the acceleration domain as both the equation of motion and linearized motion controllers are of second order. Singularity resolution in velocity-based control has been extensively treated while this is not the case for acceleration-based control. Herzog et al. [2016] introduce a small regularization or damping term but not with singularities in mind. The same holds for Vaillant et al. [2016] who include a regularizing posture task in their objective formulation which enforces the robot to stay as close as possible to a reference posture as it executes the tasks. This allows approaching singularities but is not specifically mentioned in the work.

In the second part of this thesis (see chapter 5) we first introduce the hierarchical GN algorithm of control and the hierarchical Newton's method of control. This requires the adoption of a control time step which is usually much smaller than the unit time step we assume in the derivation of the hierarchical Newton's method of optimization.

We then direct our attention towards singularities in acceleration-based control. Similarly to velocity-based control a linearized control model is defined which leads to the same rank issues of the task Jacobian around singularities. We show how regularization or damping terms directly on the accelerations negatively influence the convergence behaviour of the second order motion controllers. The hierarchical New-

ton's method, which we develop in the first part, causes the same diverging behaviour as its formulation is closely related to damping through the LM algorithm. This motivates us to express dynamic control including the equation of motion and the motion controllers in the velocity domain by using forward integration. Here the hierarchical Newton's method, which is superior to the LM algorithm in terms of convergence and easiness of use due to the lack of damping tuning, is well defined. We evaluate the reproducibility of acceleration-based control in the velocity domain in simulation. Furthermore, we show the effectiveness of the hierarchical Newton's method in terms of preventing singular and numerically unstable behaviour in hierarchical dynamic control. Three real robot experiments confirm superior convergence rates compared to damping based approaches.

CHAPTER 2

Preliminaries

With this chapter we first introduce possible error functions which often happen to be non-linear. We therefore find certain abstractions which enable linear control formulations that can be solved by classical solvers for *linear least-squares problems* (LSP) with *linear constraints*. We give a quick overview of the workings of these solvers before we step on to the next chapter 3 where we apply our obtained knowledge in a robotic demonstration.

2.1 From a non-linear robot to quadratic objectives with linear constraints

2.1.1 Kinematic control

We have mentioned so-called *error* functions $\boldsymbol{e}(\boldsymbol{q}) \in \mathbb{R}^m$

$$\boldsymbol{e}(\boldsymbol{q}) = \boldsymbol{f}_d(t) - \boldsymbol{f}(\boldsymbol{q}) \tag{2.1}$$

where $\boldsymbol{f}_d$ is the desired value and $\boldsymbol{f}(\boldsymbol{q}(t))$ is the actual value of an end-effector. They are dependent of the state vector $\boldsymbol{q} = \begin{bmatrix}\boldsymbol{t}^T & \boldsymbol{p}^T\end{bmatrix}^T \in \mathbb{R}^n$ where $\boldsymbol{p} \in \mathbb{R}^{n-6}$ denotes the joint positions while $\boldsymbol{t} \in \mathrm{SE}(3)$ describes the configuration of the free-flyer base with respect to an inertial frame (usually the fixed world frame).

In the following, we refer to these error functions as *tasks*. A few typical examples are given below Vaillant et al. [2016]:

- Posture task to maintain a desired posture $\boldsymbol{q}_d$: $\boldsymbol{e}_{\text{posture}} = \boldsymbol{q}_d - \boldsymbol{q} = \boldsymbol{0}$;
- Joint limit task: $\boldsymbol{e}_{\text{sup}} = \overline{\boldsymbol{q}} - \boldsymbol{q} \geq \boldsymbol{0}$ and $\boldsymbol{e}_{\text{inf}} = \boldsymbol{q} - \underline{\boldsymbol{q}} \geq \boldsymbol{0}$;
- CoM task: $\boldsymbol{e}_{\text{CoM}} = \boldsymbol{x}_{\text{CoM},d} - \boldsymbol{x}_{\text{CoM}}(\boldsymbol{q}) = \boldsymbol{0}$;

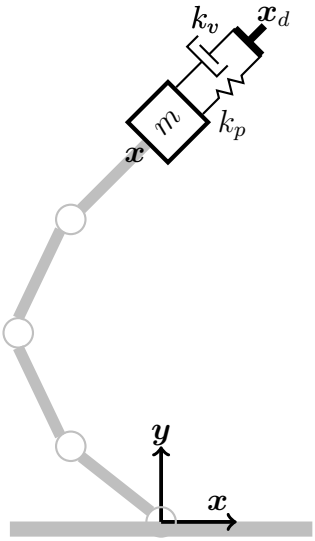


Fig. 2.1. *Schematic of a robotic manipulator in 2-D with gravity neglected. The square box indicates the end-effector of mass m with current position $\boldsymbol{x}$. Its movement is determined by the kinematic structure given in light grey (which is assumed to be massless) with fixed base at the origin. The robot is pulled towards the desired position $\boldsymbol{x}_d$ with a spring-damper system with spring stiffness k_p and damping ratio k_v.*

- Body i position task: $\boldsymbol{e}_{\text{pos}} = \boldsymbol{x}_{d,i} - \boldsymbol{x}_i(\boldsymbol{q}) = \boldsymbol{0}$;
- Body i orientation task: $\boldsymbol{e}_{\text{ori}} = \log(\boldsymbol{R}_i(\boldsymbol{q})^T \boldsymbol{R}_{d,i}) = \boldsymbol{0}$;
- Collision avoidance $\boldsymbol{e}_{\text{col}} = \boldsymbol{\delta}_{i,j}(\boldsymbol{q}) \geq \boldsymbol{0}$.

The superscript d denotes a desired value specified by the user. $\underline{\boldsymbol{q}}$ and $\overline{\boldsymbol{q}}$ are the lower and upper joint limits of $\boldsymbol{q}$. $\boldsymbol{x}_i$ and $\boldsymbol{R}_i$ are the position and orientation of the i-th robot body. log is the logarithm over SO(3) (see Murray et al. [1994]). $\boldsymbol{\delta}_{i,j}$ denotes the distance between body i and j (one of which can be part of the environment).

Our goal is to drive these error functions to zero $\boldsymbol{e} = \boldsymbol{0}$ with a certain *control* behaviour. A typical control behaviour for a position task is realized by a virtual 3-D massless ($m = 0$) robot in space connected with its environment by a spring and a damper. The actual and desired translational positions are $\boldsymbol{f} = \boldsymbol{x}$ and $\boldsymbol{f}_d = \boldsymbol{x}_d$ such that the error function is $\boldsymbol{e} = \boldsymbol{x}_d - \boldsymbol{x}$, $\dot{\boldsymbol{e}} = -\dot{\boldsymbol{x}}$, $\ddot{\boldsymbol{e}} = -\ddot{\boldsymbol{x}}$. The position of the robot is determined by the gray kinematic structure (see fig. 2.1).

The motion of such a system can be deducted from the free-body diagram fig. 2.2 and leads to the differential equation

$$k_v \dot{\boldsymbol{e}} + k_p \boldsymbol{e} = -k_v \dot{\boldsymbol{x}} + k_p \boldsymbol{e} = -k_v \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} - \dot{\boldsymbol{e}}^{\text{ctrl}} = \boldsymbol{0} \tag{2.2}$$

with the simplified choice $k_v = 1$ for the isotropic coefficient for the damping value. k_p is the isotropic coefficient for the spring stiffness. $\dot{\boldsymbol{x}}$ is the first order time derivative

$$\frac{d}{dt}\boldsymbol{x} = \dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{2.3}$$

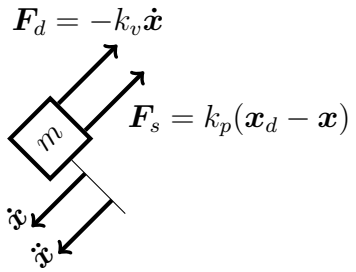


Fig. 2.2. *Free body diagram of the mass-damper-spring system. Applying for example D'Alembert's principle yields the inertial forces and corresponding equation of motion of the system. Note that the kinematic structure only determines the position $\boldsymbol{x}$ of the end-effector as a holonomic constraint but does not exert any forces on it. Thus, reaction forces only occur at the point $\boldsymbol{x}_d$ which is connected to the environment able to sustain infinite reaction forces.*

with the Jacobian $\boldsymbol{J}(\boldsymbol{q}) = \partial \boldsymbol{f}(\boldsymbol{q})/\partial \boldsymbol{q} = (\nabla_{\boldsymbol{q}} \boldsymbol{f}(\boldsymbol{q}))^T \in \mathbb{R}^{m,n}$.

$$\dot{\boldsymbol{e}}^{\text{ctrl}} \stackrel{\text{def}}{=} -k_p \boldsymbol{e} \tag{2.4}$$

is a typical first order control law for velocity-based control we refer to as *Proportional controller*.

For small changes $\boldsymbol{\Delta q}$ of the robot state $\boldsymbol{q}$ in every control iteration k, we can adapt the linearisation

$$\boldsymbol{\Delta x} \approx \boldsymbol{J} \boldsymbol{\Delta q} \tag{2.5}$$

which maps changes of the joint configuration into changes in the task-space.

This permits the omission of the direct integration of the coupled differential equation (2.2) over time. Rather, a new velocity is calculated in a least squares sense

$$\min_{\dot{\boldsymbol{q}}^{(k)}}. \boldsymbol{o} = \min_{\dot{\boldsymbol{q}}^{(k)}}. \|\boldsymbol{J}\dot{\boldsymbol{q}}^{(k)} + \dot{\boldsymbol{e}}^{\text{ctrl}}\|_2^2 \tag{2.6}$$

using the relation (2.2). It is then used for the forward integration to the new joint angle configuration

$$\boldsymbol{q}^{(k+1)} = \boldsymbol{q}^{(k)} + \Delta t \dot{\boldsymbol{q}}^{(k)} = \boldsymbol{q}^{(k)} + \boldsymbol{\Delta q}^{(k)} \tag{2.7}$$

or the free-flyer quaternion by

$$\boldsymbol{q}^{(k+1)} = \exp(\dot{\boldsymbol{q}}^{(k)} \Delta t) \boldsymbol{q}^{(k)} \,. \tag{2.8}$$

Δt is the time difference between two control steps. Throughout this work, we omit the index k for better readability if it is not indispensable for understanding and only keep the indices $k-1$ and $k+1$.

A second order control law

$$\ddot{\boldsymbol{e}}^{\text{ctrl}} \stackrel{\text{def}}{=} -k_p \boldsymbol{e} - k_v \dot{\boldsymbol{e}} \,, \tag{2.9}$$

commonly referred to as a *Proportional derivative controller*, can be derived by describing the behaviour of a spring-damper system attached to a unit mass point robot, see fig. 2.2 and fig. 2.1, now with $m = 1\text{kg}$ but in a gravity free environment. Exponential convergence can be achieved if the mass-spring-damper system is *critically damped* with $k_v = 2\sqrt{mk_p}$. The differential equation is

$$m\ddot{\boldsymbol{e}} + k_v\dot{\boldsymbol{e}} + k_p\boldsymbol{e} = -\ddot{\boldsymbol{x}} + k_v\dot{\boldsymbol{e}} + k_p\boldsymbol{e} = -\boldsymbol{J}\ddot{\boldsymbol{q}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} - \ddot{\boldsymbol{e}}^{\text{ctrl}} = \boldsymbol{0} \tag{2.10}$$

based on the derivative

$$\frac{d^2\boldsymbol{e}}{dt^2} = \ddot{\boldsymbol{e}} = -\ddot{\boldsymbol{e}} = -\boldsymbol{J}\ddot{\boldsymbol{q}} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}\,. \tag{2.11}$$

$\dot{\boldsymbol{J}} \in \mathbb{R}^{m,n}$ is the time derivative of the Jacobian $\dot{\boldsymbol{J}} = \frac{d}{dt}\boldsymbol{J}$.

Similarly to our velocity-based control law, we calculate the new acceleration by solving the least-squares problem

$$\underset{\ddot{\boldsymbol{q}}^{(k)}}{\text{min.}}\, \boldsymbol{o} = \underset{\ddot{\boldsymbol{q}}^{(k)}}{\text{min.}}\, \|\boldsymbol{J}\ddot{\boldsymbol{q}}^{(k)} + \ddot{\boldsymbol{e}}^{\text{ctrl}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}\|_2^2\,. \tag{2.12}$$

Again, the Euler-method is applied to integrate the joint angles by

$$\boldsymbol{q}^{(k+1)} = \boldsymbol{q} + \Delta t\dot{\boldsymbol{q}} + \frac{\Delta t^2}{2}\ddot{\boldsymbol{q}}^{(k)}\,. \tag{2.13}$$

The free-flyer quaternion is integrated by

$$\boldsymbol{q}^{(k+1)} = \exp\left(\frac{1}{2}\boldsymbol{\Omega}\right)\boldsymbol{q} \tag{2.14}$$

where $\boldsymbol{\Omega} = \boldsymbol{f}(\Delta t, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}^{(k)})$ is the Magnus expansion of third order (see Blanes et al. [2009] for details).

2.1.2 Dynamic control

While the robot tries to achieve above control objectives it is indispensable that it obeys the laws of physics. Mathematically we can express this law with the so-called equation of motion Walker et Orin [1982]

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \boldsymbol{S}^T\boldsymbol{\tau} + \boldsymbol{J}_c^T\boldsymbol{K}\boldsymbol{\gamma}\,. \tag{2.15}$$

The equation is non-linear in $\boldsymbol{q} \in \mathbb{R}^n$ and $\dot{\boldsymbol{q}} \in \mathbb{R}^n$ but linear in the joint accelerations $\ddot{\boldsymbol{q}} \in \mathbb{R}^n$, joint torques $\boldsymbol{\tau} \in \mathbb{R}^n$ and the generalized contact wrenches $\boldsymbol{\gamma}$. The relationship $\boldsymbol{f} = \boldsymbol{K}\boldsymbol{\gamma}$ with the external contact forces $\boldsymbol{f}$ holds where $\boldsymbol{K}$ is the discretized friction cone matrix Vaillant et al. [2016]; Stewart [2000]. $\boldsymbol{M} \in \mathbb{R}^{n,n}$ denotes the generalized inertia matrix of the robot, $\boldsymbol{N} \in \mathbb{R}^n$ gathers the centrifugal, Coriolis and gravity effects, $\boldsymbol{S} \in \mathbb{R}^{n,n}$ is a selection matrix accounting for the underactuation of the robot base and $\boldsymbol{J}_c(\boldsymbol{q})$ is the Jacobian matrix of the contact points.

We now define following constrained optimization problem Abe et al. [2007]

$$\min_{\ddot{\boldsymbol{q}},\boldsymbol{\tau},\boldsymbol{\gamma}} \sum_i \omega_i \boldsymbol{o}_i(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}}) + \omega_p\|\boldsymbol{p}\|^2 + \omega_f\|\boldsymbol{\gamma}\|^2 + \omega_\tau\|\boldsymbol{\tau}\|^2 \tag{2.16}$$

$$\text{s.t. } \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \boldsymbol{S}\boldsymbol{\tau} + \boldsymbol{J}_c^T\boldsymbol{K}\boldsymbol{\gamma} \tag{2.17}$$

$$\underline{\boldsymbol{\tau}} \leq \boldsymbol{\tau} \leq \overline{\boldsymbol{\tau}} \tag{2.18}$$

$$\boldsymbol{\gamma} \geq \boldsymbol{0} \tag{2.19}$$

$$-\boldsymbol{J}_i\ddot{\boldsymbol{q}} - \dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}} \leqq \ddot{\boldsymbol{e}}_i^{\text{ctrl}} \quad \forall i\,. \tag{2.20}$$

By virtue of the equation of motion (2.17) we ensure that the required accelerations from the objectives (2.16) are physically feasible, i.e. the joint torques $\boldsymbol{\tau}$ (2.18) are within their limits and the generalized contact wrenches are within the discretized version of the Coulomb friction cones $\boldsymbol{\gamma}$ (2.19) such that the equation of motion is satisfied. The constraints also consist of motion controllers for example for geometric contact constraints of end-effectors (2.20).

The objective is composed as the sum of tasks $\boldsymbol{c}$ weighted with a corresponding scalar ω depending on the task's relative importance and desired accuracy. Additionally, a posture reference task $\boldsymbol{p}$ is introduced which can be interpreted as a regularization / damping term on the joint accelerations. It allows to approach kinematic singularities or tasks to be in conflict with constraints. Furthermore, regularization terms on the generalized contact wrenches $\boldsymbol{\gamma}$ and on the joint torques $\boldsymbol{\tau}$ are formulated in order to yield a fully determined problem.

2.2 Solving unconstrained LSP's

We first solve level 1 of the hierarchy (2.16)

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{A}_1\boldsymbol{x} - \boldsymbol{b}_1\|_2^2 = \min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}_1^T\boldsymbol{A}_1\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{A}_1^T\boldsymbol{b}_1 = \min_{\boldsymbol{x}} f_1 \tag{2.21}$$

with $\boldsymbol{A}_1 \in \mathbb{R}^{m_1,n}$, $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{b}_1 \in \mathbb{R}^n$. The solution of this unconstrained LSP corresponds to the stationary points $\nabla_{\boldsymbol{x}} f_1 = \boldsymbol{0}$ of the scalar function f_1. This corresponds to the pseudo-inverse solution

$$\boldsymbol{x} = \boldsymbol{A}_1^+\boldsymbol{b}_1\,. \tag{2.22}$$

Computing the pseudo-inverse $\boldsymbol{A}_1^+$ explicitly can be achieved with the Singular-Value-Decomposition (SVD) but is expensive with a computational complexity of $\mathcal{O}(4m^2n + 22n^3)$, Golub et Van Loan [1996] chapter 5. There are cheaper approaches where solvers rely on decompositions including orthogonal transformations with $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^T$ and a triangular matrix whose inverse consists of a simple forward or backward substitution.

Such decompositions include for example the *QR decomposition* (LexLSI [Dimitrov et al. 2015], see chapter 6), the *Complete Orthogonal Decomposition* (HQP, [Escande et al. 2014]) or the *TQ Decomposition* (LSSOL solver [Gill et al. 1986]). The overall complexity of these decompositions is approximately $\mathcal{O}(2n^3)$ Escande et al. [2013].

2.3 Solving feasible hierarchical LSP's with linear equality constraints

We now step to our actual goal of solving constrained minimization problems as in (2.16). We consider the case of a hierarchy with p levels. Each level l is then stated as a constrained LSP with the LSP's of the previous levels $< l$ as linear constraints:

$$\min_{\boldsymbol{x}} \quad \frac{1}{2}\left\|\boldsymbol{A}_l\boldsymbol{x}-\boldsymbol{b}_l\right\|_2^2 = \min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}_l^T\boldsymbol{A}_l\boldsymbol{x}-\boldsymbol{x}^T\boldsymbol{A}_l^T\boldsymbol{b}_l \qquad l=1,\ldots,p \tag{2.23}$$

$$\text{s.t.} \quad \underline{\boldsymbol{A}}_{l-1}\boldsymbol{x}=\underline{\boldsymbol{b}}_{l-1}\,. \tag{2.24}$$

The underlined vector $\underline{\boldsymbol{b}}_{l-1}$ or matrix $\underline{\boldsymbol{A}}_{l-1}$ is the stacked vector $\underline{\boldsymbol{b}}_{l-1}=\begin{bmatrix}\boldsymbol{b}_1^T & \cdots & \boldsymbol{b}_{l-1}^T\end{bmatrix}^T$ or the stacked matrix $\underline{\boldsymbol{A}}_{l-1}=\begin{bmatrix}\boldsymbol{A}_1^T & \cdots & \boldsymbol{A}_{l-1}^T\end{bmatrix}^T$.

We start with $p=2$. The *Lagrangian function* of this constrained optimization problem then writes as

$$\boldsymbol{\mathcal{L}}_2=\frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}_2^T\boldsymbol{A}_2\boldsymbol{x}-\boldsymbol{x}^T\boldsymbol{A}_2^T\boldsymbol{b}_2+(\boldsymbol{A}_1\boldsymbol{x}-\boldsymbol{b}_1)^T\boldsymbol{\lambda}_{1,2}\,. \tag{2.25}$$

where $\boldsymbol{\lambda}_{1,2}$ are the *Lagrange multipliers* indicating the conflict of the objective on level 2 with the constraints on level 1. A positive high value shows that a further minimization of level 2 would lead to a great violation of the constraints. The Lagrange multipliers can therefore also be referred to the *sensitivity* between the constraints' and objectives' optimality Nocedal et Wright [2006].

At the optimum, we need to fulfil following first-order necessary optimality conditions

$$\nabla_{\boldsymbol{x}}\boldsymbol{\mathcal{L}}_2=\boldsymbol{A}_2^T\boldsymbol{A}_2\boldsymbol{x}-\boldsymbol{A}_2^T\boldsymbol{b}_2+\boldsymbol{A}_1^T\boldsymbol{\lambda}_{1,2}=\boldsymbol{0} \tag{2.26}$$

$$\nabla_{\boldsymbol{\lambda}_{1,2}}\boldsymbol{\mathcal{L}}_2=\boldsymbol{A}_1\boldsymbol{x}-\boldsymbol{b}_1=\boldsymbol{0}\,. \tag{2.27}$$

A possible way of solving this system is the so-called *null-space method*. It introduces a change of basis

$$\boldsymbol{x}=\boldsymbol{Y}_1\boldsymbol{x}_{1,Y}+\boldsymbol{Z}_1\boldsymbol{x}_{1,Z}\,. \tag{2.28}$$

$\boldsymbol{Z}_1$ is a basis of the null-space of $\boldsymbol{A}_1$. $\boldsymbol{A}_1\boldsymbol{Z}_1=\boldsymbol{0}$ holds.

After we have obtained $\boldsymbol{x}_{1,Y}^*$ by solving (2.27) we can find the solution of the objective without influencing the optimality of the constraints by inserting $\boldsymbol{x}=\boldsymbol{Y}_1\boldsymbol{x}_{1,Y}^*+\boldsymbol{Z}_1\boldsymbol{x}_{1,Z}$ into (2.23)

$$\min_{\boldsymbol{x}_{1,Z}} \quad \left\|\boldsymbol{A}_2\boldsymbol{Z}_1\boldsymbol{x}_{1,Z}+\boldsymbol{A}_2\boldsymbol{Y}_1\boldsymbol{x}_{1,Y}^*-\boldsymbol{b}_2\right\|_2^2 \tag{2.29}$$

and then solving for $\boldsymbol{x}_{1,Z}$.

(2.26) lets us calculate the Lagrange multipliers after multiplying it from the left by $\boldsymbol{Y}_1^T$

$$(\boldsymbol{A}_1\boldsymbol{Y}_1)^T\boldsymbol{\lambda}_{1,2}=\boldsymbol{Y}_1^T(\boldsymbol{A}_2\boldsymbol{b}_2-\boldsymbol{A}_2^T\boldsymbol{A}_2\boldsymbol{x}^*)\,. \tag{2.30}$$

The procedure above can be repeated for any number of constraint levels, introducing consecutively variable changes $\boldsymbol{x}_{l-1,Z} = \boldsymbol{Y}_l \boldsymbol{x}_{l,Y} + \boldsymbol{Z}_l \boldsymbol{x}_{l,Z}$. This leads to a minimization problem of the form

$$\min_{\boldsymbol{x}_{l,Z}} \quad \|\boldsymbol{A}_l \boldsymbol{N}_{l-1} \boldsymbol{x}_{l,Z} + \boldsymbol{A}_l \sum_{k=1}^{l-1} (\boldsymbol{N}_{k-1} \boldsymbol{Y}_k \boldsymbol{x}^*_{k,Y}) - \boldsymbol{b}_l\|_2^2 \tag{2.31}$$

for each level $l = 1, \ldots, p$. The summation is not conducted if $l-1 < 1$. $\boldsymbol{N}_i = \boldsymbol{Z}_1 \ldots \boldsymbol{Z}_i$ is the accumulation of null-spaces from level 1 to i with $\boldsymbol{N}_0 = \boldsymbol{I}$.

The solution is then given by

$$\boldsymbol{x}^* = \boldsymbol{Y}_1 \boldsymbol{x}^*_{1,Y} + \boldsymbol{Z}_1(\boldsymbol{Y}_2 \boldsymbol{x}^*_{2,Y} + \boldsymbol{Z}_2(\boldsymbol{Y}_3 \boldsymbol{x}^*_{3,Y} + \boldsymbol{Z}_3(\ldots))) = \sum_{i=1}^{p} \boldsymbol{N}_{i-1} \boldsymbol{Y}_i \boldsymbol{x}^*_i \,. \tag{2.32}$$

and the Lagrange multipliers by

$$\boldsymbol{Y}_{l-1}^T \ldots \boldsymbol{Y}_1^T \sum_{k=1}^{l-1} \boldsymbol{A}_k^T \boldsymbol{\lambda}_{k,l} = \boldsymbol{Y}_{l-1}^T \ldots \boldsymbol{Y}_1^T (\boldsymbol{A}_l \boldsymbol{b}_l - \boldsymbol{A}_l^T \boldsymbol{A}_l \boldsymbol{x}^*) \,. \tag{2.33}$$

Details are given in Dimitrov et al. [2015].

2.4 Solving infeasible hierarchical LSP's with linear inequality constraints

In the above derivation we assumed that the objectives and constraints are perfectly achievable without any conflicts between constraints on the same or different levels. However, in robotic problems this is barely the case and we either have infeasible constraints (for example a target is out of reach) or an objective is in conflict with constraints (for example the high priority CoM constraint prevents the robot to move its hand any further to the front). In this case we can introduce the notion of the so-called slack variable $\boldsymbol{w}$ which relaxes a level in case of infeasibility or conflict with a higher priority task. It also allows the handling of inequality constraints Kanoun et al. [2011].

$$\min_{\boldsymbol{x}, \boldsymbol{w}_l} \quad \frac{1}{2} \|\boldsymbol{w}_l\|^2 \qquad\qquad l = 1, \ldots, p \tag{2.34}$$

$$\text{s.t.} \quad \boldsymbol{A}_l \boldsymbol{x} - \boldsymbol{b}_l \leqq \boldsymbol{w}_l \tag{2.35}$$

$$\underline{\boldsymbol{A}}_{l-1} \boldsymbol{x} - \underline{\boldsymbol{b}}_{l-1} \leqq \underline{\boldsymbol{w}}^*_{l-1} \,. \tag{2.36}$$

The symbol $\leqq$ gathers both equality ($=$) and inequality ($\leq$) constraints. Thereby, lower bounds $\hat{\boldsymbol{A}}\boldsymbol{x} - \hat{\boldsymbol{b}} \geq \hat{\boldsymbol{w}}$ are included by simply writing $-\hat{\boldsymbol{A}}\boldsymbol{x} + \hat{\boldsymbol{b}} \leq -\hat{\boldsymbol{w}} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq \boldsymbol{w}$.

Such problems are solved very efficiently by the solvers presented in Escande et al. [2014] and Dimitrov et al. [2015].

At each level l we want to minimize the relaxation of the objective $\boldsymbol{w}_l$ while keeping the optimal relaxation already found for previous levels $\boldsymbol{w}_i^*$, $i = 1, \dots, l-1$ untouched. This can be achieved by sequentially minimizing the residual of every level $l = 1, \dots, p$ by (2.31) where $\boldsymbol{w}_l$ is given implicitly.

The Lagrangian function for every of the p constrained optimization problems can be written as

$$\mathcal{L}_l = \frac{1}{2}\boldsymbol{w}_l^T\boldsymbol{w}_l + (\boldsymbol{A}_l\boldsymbol{x} - \boldsymbol{b}_l - \boldsymbol{w}_l)^T\boldsymbol{\lambda}_{l,l} + (\underline{\boldsymbol{A}}_{l-1}\boldsymbol{x} - \underline{\boldsymbol{b}}_{l-1} - \underline{\boldsymbol{w}}_{l-1})^T\underline{\boldsymbol{\lambda}}_{l-1,l}\,. \tag{2.37}$$

The corresponding first-order optimality conditions are given as

$$\nabla_{\boldsymbol{x}}\mathcal{L}_l = \boldsymbol{A}_l^T\boldsymbol{\lambda}_{l,l} + \underline{\boldsymbol{A}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l} = \boldsymbol{0} \tag{2.38}$$

$$\nabla_{\boldsymbol{w}_l}\mathcal{L}_l = \boldsymbol{w}_l - \boldsymbol{\lambda}_{l,l} = \boldsymbol{0} \tag{2.39}$$

$$\nabla_{\boldsymbol{\lambda}_l}\mathcal{L}_l = \boldsymbol{A}_l\boldsymbol{x} - \boldsymbol{b}_l - \boldsymbol{w}_l = \boldsymbol{0} \tag{2.40}$$

$$\nabla_{\underline{\boldsymbol{\lambda}}_{l-1}}\mathcal{L}_l = \underline{\boldsymbol{A}}_{l-1}\boldsymbol{x} - \underline{\boldsymbol{b}}_{l-1} - \underline{\boldsymbol{w}}_{l-1}^* = \boldsymbol{0} \tag{2.41}$$

with $\boldsymbol{w}_l = \boldsymbol{\lambda}_{l,l}$. The corresponding matrix of Lagrange multipliers then takes following structure

$$\boldsymbol{\Lambda} = \begin{bmatrix} \boldsymbol{w}_1 & \boldsymbol{\lambda}_{1,2} & \boldsymbol{\lambda}_{1,3} & \dots & \boldsymbol{\lambda}_{1,p-1} & \boldsymbol{\lambda}_{1,p} \\ \boldsymbol{0} & \boldsymbol{w}_2 & \boldsymbol{\lambda}_{2,3} & \dots & \boldsymbol{\lambda}_{2,p-1} & \boldsymbol{\lambda}_{2,p} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{w}_3 & \dots & \boldsymbol{\lambda}_{3,p-1} & \boldsymbol{\lambda}_{3,p} \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{w}_{p-1} & \boldsymbol{\lambda}_{p-1,p} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{w}_p \end{bmatrix} \tag{2.42}$$

where each column l contains the relaxation of infeasibility $\boldsymbol{w}_l$ and the conflict with higher priority levels $\boldsymbol{\lambda}_{i,l}$, $i = 1, \dots, l-1$. Due to the notion of strict hierarchy, higher priority constraints are never in conflict with lower priority constraints, rendering $\boldsymbol{\lambda}_{i,l} = \boldsymbol{0}$ for $i > l$ and leading to the triangular structure. If $\boldsymbol{w}_l = \boldsymbol{\lambda}_{l,l} = \boldsymbol{0}$ then also $\boldsymbol{\lambda}_{1:l-1,l} = \boldsymbol{0}$ because of (2.33), meaning that a feasible objective l is not in conflict with any higher priority constraint $i < l$. If a constraint is inactive we have both $\boldsymbol{\lambda}_{1:l-1,l} = \boldsymbol{0}$ and $\boldsymbol{\lambda}_{l,l:p} = \boldsymbol{0}$.

The so called *active-set method* determines which inequality constraints need to be active at the optimal solution $\boldsymbol{x}^*, \boldsymbol{w}^*$. At every iteration s of the active-set search the problem (2.34) composed of all equality and all active inequality constraints is solved. A line search $\boldsymbol{x}^{(s+1)} = \boldsymbol{x}^{(s)} + \alpha(\hat{\boldsymbol{x}}^{(s+1)} - \boldsymbol{x}^{(s)})$ from the current iterate $\boldsymbol{x}^{(s)}$ to the solution of the subproblem $\hat{\boldsymbol{x}}^{(s+1)}$ is then conducted. $\alpha = [0, 1]$ is chosen in such a way that it leads to the saturation of the closest constraint (independent of priority level). This constraint is then added to the active-set. If no constraint needs to be activated but the Lagrange multipliers corresponding to a constraint are lexicographically negative (i.e. the first non-zero element of the row of $\boldsymbol{\Lambda}^{(s)}$ corresponding to the constraint is

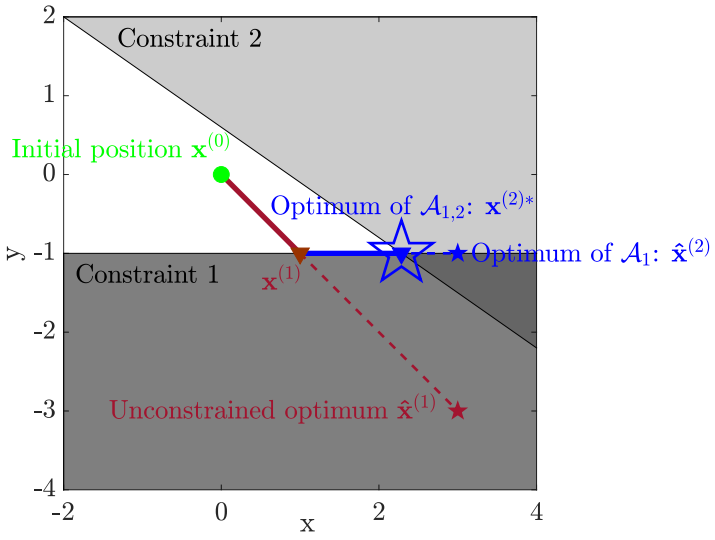


Fig. 2.3. *Two steps of the active-set search with initial feasible point* $\boldsymbol{x}^{(0)}$.

negative), the constraint prevents moving into a descend direction and therefore needs to be removed from the active-set. If several constraints are associated to lexicographically negative Lagrange multipliers, the one of highest priority (or with the largest absolute value if the constraints are on the same level) is chosen for deactivation. The search converges once a full step $\alpha = 1$ can be taken and no lexicographically negative Lagrange multipliers are left.

Figure 2.3 gives a graphical overview of an active-set search with two stages. In the beginning, no constraints are violated. If this is not the case the active-set algorithm conducts a series of constraint activations with full step $\alpha = 1$ until all violated constraints are activated.

In the first stage, a full step to the unconstrained optimum $\hat{\boldsymbol{x}}^{(1)}$ is prevented by constraint 1 which is consequently added to the active-set $\mathcal{A}_1$. $\alpha < 1$ is chosen in such a way that the current iterate $\boldsymbol{x}^{(1)}$ saturates constraint 1. In the second stage, constraint 2 is added to the active-set $\mathcal{A}_{1,2}$ since it prevents moving to the optimum $\hat{\boldsymbol{x}}^{(2)}$ associated with the active-set $\mathcal{A}_1$. At this point $\boldsymbol{x}^{(2)*}$ (again with some $\alpha < 1$), no constraints need to be activated or deactivated and the active-set search finishes.

For more details on the hierarchical active-set search see Escande et al. [2014].

CHAPTER 3

Nut fastening with a humanoid robot

In the previous chapter we outlined means to solve constrained LSP's. We now want to proceed with a direct application of our obtained knowledge. For this, we implement a robot demonstration with the Kawada HRP-2Kai robot conducting the industrial task of nut fastening[1]. It shows the robot's capabilities in terms of accuracy with the current robot framework which is based on the LSSOL solver Gill et al. [1986]. LSSOL solves LSP's being subject to a set of linear constraints

$$\min_{\boldsymbol{x}} \quad \left\|\boldsymbol{W}^{1/2}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})\right\|_2^2 \tag{3.1}$$

$$\text{s.t.} \quad \boldsymbol{l} < \boldsymbol{C}\boldsymbol{x} < \boldsymbol{u}\,. \tag{3.2}$$

$\boldsymbol{W}$ is a positive and diagonal matrix weighting the objectives against each other. $\boldsymbol{l}$ and $\boldsymbol{u}$ are the lower and upper bounds of the constraints. In this chapter, the constraints and the regularization terms on the reference posture and contact wrenches on the objective level are given according to (2.16). The remaining objectives' formulation is concerned with the movement of the end-effectors and is described later in this chapter, see sec. 3.3. This solver is referred to as WLS (**W**eighted **L**east **S**quares) in chapter 5.

We can directly identify some of the weaknesses of this formulation which builds up our narrative of introducing multi-level constrained LSP's (2.34) in later chapters 4 and 5:

- **Feasibility of the constraints**
  Unlike the hierarchical leas-squares solvers HQP Escande et al. [2014] and LexLSI Dimitrov et al. [2015], LSSOL does not introduce the notion of constraint relaxation

[1] The accompanying video can be found at https://youtu.be/r_U3LHHGAdY

through slack variables. While infeasible constraints are rare in practice they lead to solver (and therefore robot control) failure. A possible scenario could be a self-collision constraint that is in conflict for example with the CoM constraint. Also, the constraint matrix is assumed to be always full rank, meaning that there is no handling of rank deficiency and the numerical instabilities that come with such singular cases.

- **Notion of a soft or weighted hierarchy**
  A 'soft' or weighted hierarchy prevents clear decoupling of constraints which are 'existential' for the robot (equation of motion) and ensure its own *safety* and *physical stability*.
- **Automatic singularity handling**
  Even for a two level hierarchy, like it is given in the constrained least-squares formulation (2.16), automatic handling of singularities is advantageous. While in the current framework damping is introduced in the form of a reference posture task, it is attributed with a certain weight relative to other objectives on the objective level. Depending on the level of damping this allows to approach kinematic or algorithmic singularities to a certain degree. However, the level of damping needs to be chosen by hand and might be insufficient for certain singular configurations. In other situations it might be too large which leads to slow robot behaviour and reduces the amount of objective error minimization.

Despite the problems which come with such a formulation, such an approach has been widely applied Abe et al. [2007]; Collette et al. [2007]; Bouyarmane et Kheddar [2011]; Feng et al. [2013]; Kuindersma et al. [2014]; Vaillant et al. [2016] and allows an array of applications of which one we are eager to show in the following.

3.1 Motivation and problem formulation

In this experiment we let the humanoid robot conduct the common task of nut fastening. In a typical industrial scenario like aircraft assembly, nut fastening is among many other repetitive tasks like drilling and riveting. They require high precision over large structures like the fuselage and wings in order to live up to the high quality and safety standards necessary in aviation. Due to their repetitive and physically exhausting nature (for example when working over head or in hardly accessible places) they bring the danger of human workers suffering from concentration loss leading to costly errors. Automation represents an interesting option in order to prevent these errors.

Mounted robots are adversely affected by their immobility if assembly work over large structures is required. Legged humanoid robots overcome this aspect due to their mobile base and could be a feasible vector of automation in the aircraft assembly industry. In previous works (walking on uneven terrains Kim et Oh [2007], climbing ladders Vaillant et al. [2016], driving cars Paolillo et al. [2014]) robots already have been proven to be able to conduct tasks usually performed by humans.

The operation of nut onto bolt fastening can be considered a high precision industrial scenario due to the small size of the nut compared to the robot's dimensions. While LSP based controllers Vaillant et al. [2016] have been used in various simple tasks and multi-contact scenarios (see Abe et al. [2007]; Collette et al. [2007]; Bouyarmane et Kheddar [2011]; Feng et al. [2013]; Kuindersma et al. [2014]; Vaillant et al. [2016] to name a few), employing a humanoid robot not necessarily designed for high-precision purposes such as nut-fastening is a new research direction.

In this chapter we aim to fasten a nut onto a bolt with the human sized humanoid robot HRP-2KAI. Following conditions and assumptions are given:

- We use a commercially available wrench tool typically used in industrial settings like Airbus manufacturing halls. In particular it allows reverse motion without unfastening the nut.
- The nut and bolt used in the experimental evaluation (sec. 3.4) have diameters of 10mm and 5mm, respectively and correspond to dimensions typically found for example in aircraft assembly.
- Currently available visual-tracking libraries and the used robot camera are not capable of reliably and accurately locating small and reflecting metallic nuts. Instead, we use the marker based visual feedback library `whycon` Nitsche et al. [2015]. The position of the nuts relatively to the visual markers is known.
- The robot is placed in such a way that the nut is within the robot's workspace and the fastening can be performed without changing contacts (the robot's relative position to the nut is unknown however).
- The tool is placed into the robot's hand before the demonstration.
- The nut is already partly fastened onto the bolt.

Despite these simplifications there remain several challenges. First and foremost, the correct insertion of the tool-tip onto the nut poses the biggest challenge. It requires high precision in position as well as in orientation which is not necessarily given by the vision system. Since the tool occludes the nut a force sensing approach has to be employed for the final stages of the insertion. Secondly, there are several uncertainties given for example by the position of the tool in the robot hand which is not perfectly known. Furthermore, the robot does not necessarily perform the fastening motion perfectly due to the weighted LSP formulation, noise inflicted sensor readings and modelling errors. With the nut actually being inserted this generates internal forces which might be harmful to the robot structure and therefore need to be avoided.

In this chapter we design several dedicated tasks in our LSP formulation which in their sum enable the humanoid robot to fasten nuts onto bolts. First, we propose a quick and reliable method to confirm tool onto nut insertion (sec. 3.2). We then present the trajectory design for the fastening movement which keeps the reaction forces with the environment low. (sec. 3.3). Experimental validation with a reliability confirmation is then conducted in sec. 3.4.

3.2 Detection of correct nut in tool insertion

The insertion detection is based on the property that any movement in the wall plane leads to reaction forces $\boldsymbol{F}$ pointing towards the nut if the nut is correctly inserted into the tool. Mathematically this can be expressed by

$$\frac{x^{(k)}_{\boldsymbol{y}^w} - x^{\text{nut}}_{\boldsymbol{y}^w}}{x^{(k)}_{\boldsymbol{x}^w} - x^{\text{nut}}_{\boldsymbol{x}^w}} = \frac{F^{(k)}_{\boldsymbol{y}^w}}{F^{(k)}_{\boldsymbol{x}^w}} \text{ for } |F^{(k)}_{\boldsymbol{x}^w}| >= |F^{(k)}_{\boldsymbol{y}^w}|,\ |F^{(k)}_{\boldsymbol{y}^w}| > 0\,. \tag{3.3}$$

which expresses the directional alignment of the reaction force $\boldsymbol{F}^{(k)}$ and the displacement vector $\boldsymbol{x}^{\text{nut}} - \boldsymbol{x}^{(k)}$ between the tool position of the current control iteration and the yet unknown position of the nut $\boldsymbol{x}^{\text{nut}}$ (see fig. 3.2). The $\boldsymbol{x}^w$- and $\boldsymbol{y}^w$- axes define the plane of the wall from which the bolt is perpendicularly sticking out. The detection method consists of moving the tool-tip (exploration movement) and verifying that this is indeed the case. Depending on which force component is larger and in order to avoid singularities the inverse of the above equation (3.3) can be used:

$$\frac{x^{(k)}_{\boldsymbol{x}^w} - x^{\text{nut}}_{\boldsymbol{x}^w}}{x^{(k)}_{\boldsymbol{y}^w} - x^{\text{nut}}_{\boldsymbol{y}^w}} = \frac{F^{(k)}_{\boldsymbol{x}^w}}{F^{(k)}_{\boldsymbol{y}^w}} \text{ for } |F^{(k)}_{\boldsymbol{x}^w}| < |F^{(k)}_{\boldsymbol{y}^w}|\,. \tag{3.4}$$

(3.3) and (3.4) are rewritten to

$$\begin{bmatrix} \frac{F^{(k)}_{\boldsymbol{y}^w}}{F^{(k)}_{\boldsymbol{x}^w}} & -1 \end{bmatrix} \begin{bmatrix} x^{\text{nut}}_{\boldsymbol{x}^w} \\ x^{\text{nut}}_{\boldsymbol{y}^w} \end{bmatrix} = \frac{F^{(k)}_{\boldsymbol{y}^w}}{F^{(k)}_{\boldsymbol{x}^w}} x^{(k)}_{\boldsymbol{x}^w} - x^{(k)}_{\boldsymbol{y}^w} \quad \text{for} \quad |F^{(k)}_{\boldsymbol{x}^w}| \geq |F^{(k)}_{\boldsymbol{y}^w}| > 0 \tag{3.5}$$

$$\begin{bmatrix} 1 & -\frac{F^{(k)}_{\boldsymbol{x}^w}}{F^{(k)}_{\boldsymbol{y}^w}} \end{bmatrix} \begin{bmatrix} x^{\text{nut}}_{\boldsymbol{x}^w} \\ x^{\text{nut}}_{\boldsymbol{y}^w} \end{bmatrix} = x^{(k)}_{\boldsymbol{x}^w} - \frac{F^{(k)}_{\boldsymbol{x}^w}}{F^{(k)}_{\boldsymbol{y}^w}} x^{(k)}_{\boldsymbol{y}^w} \quad \text{for} \quad |F^{(k)}_{\boldsymbol{x}^w}| < |F^{(k)}_{\boldsymbol{y}^w}| \tag{3.6}$$

and then sampled over time ($k = 1, \ldots, n$) in a test movement recording the tool tip position $\boldsymbol{x}$ and force $\boldsymbol{F}$. This enables us to solve for the unknown $\boldsymbol{x}^{\text{nut}}$ in a least-squares sense $\boldsymbol{x}^{\text{nut}} = \boldsymbol{A}^{+}\boldsymbol{b}$ where $\boldsymbol{A}$ is the regressor and $\boldsymbol{b}$ is the right hand side of the sampled equations (3.5) or (3.6).

The information about the force either pointing to or pointing away from the now known nut position is not contained in (3.5) or (3.6). In order to learn the structure of the force field we restore this information with a scalar value we call force field characteristic (FFC)

$$\text{FFC} = \frac{1}{n} \sum_{k=1}^{n} \frac{\boldsymbol{x}^{\text{nut}} - \boldsymbol{x}^{(k)}}{\|\boldsymbol{x}^{\text{nut}} - \boldsymbol{x}^{(k)}\|} \cdot \frac{\boldsymbol{F}^{(k)}}{\|\boldsymbol{F}^{(k)}\|}\,. \tag{3.7}$$

The FFC can take values between -1.0 (source, totally repelling) and 1.0 (sink, totally attracting). In the case of a correct tool insertion we demand that the gross of the recorded reaction forces point towards the nut. We deem FFC ≥ 0.8 to be a good

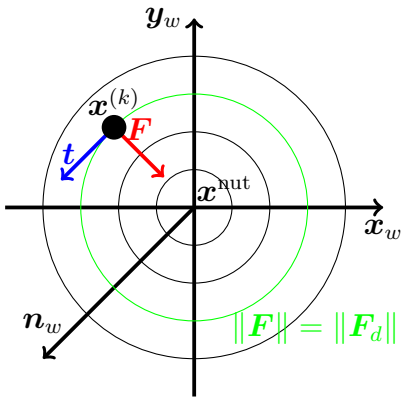


Fig. 3.1. *Force field around the nut center: At the nut center itself $\|\boldsymbol{F}\| = 0N$ is assumed. Ideally the robot end-effector would move along the green contour line where a desired constant force norm is given.*

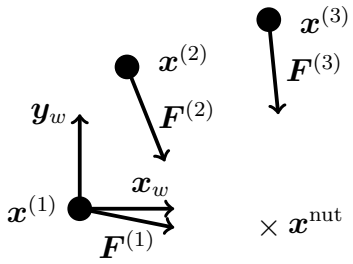


Fig. 3.2. *Schematic drawing of the three recorded forces $\boldsymbol{F}$ and positions $\boldsymbol{x}$ used for the nut position regression problem $\boldsymbol{x}^{nut}$ and the calculation of the force field characteristic FFC. The origin of the wall reference system is put at the position of the first measurement $\boldsymbol{x}^{(1)}$.*

threshold for a correct insertion when accounting for friction when moving along the exploration trajectory and dealing with noisy and offset force sensors.

In order to increase the reliability of the insertion detection method we define further decision criteria based on our obtained measurement data:

- The *condition number* κ of the regressor $\boldsymbol{A}$ is high if the exploration movement and the corresponding recorded reaction forces do not contain enough information for a good identification of $\boldsymbol{x}^{\text{nut}}$. This is the case for example for a straight line motion with some offset on the force sensor. In this work a value of $\kappa < 2.0$ is the threshold for a correct insertion.
- A *drifting exploration movement* is an indicator for an incorrect tool insertion. For a correct insertion we request that the robot hand may only move within the range $\|\boldsymbol{x}^{(1)} - \boldsymbol{x}^{(k)}\| \leq 0.015$ m from the initial tool-tip position $\boldsymbol{x}^{(1)}$. This threshold considers both robot elasticity and clearance between the tool and the nut.
- In order to obtain reasonable results only recorded forces with a norm above the threshold of 1.0 N are considered for the least squares regression. If more than 50% of the points recorded during the exploration movement resulted in reaction forces below the threshold we assume that the insertion failed.

3.2.1 Exploration movement

While sampling the regression problem in (3.5) or (3.6) the robot should be able to explore the force field autonomously rather than moving along a predefined trajectory

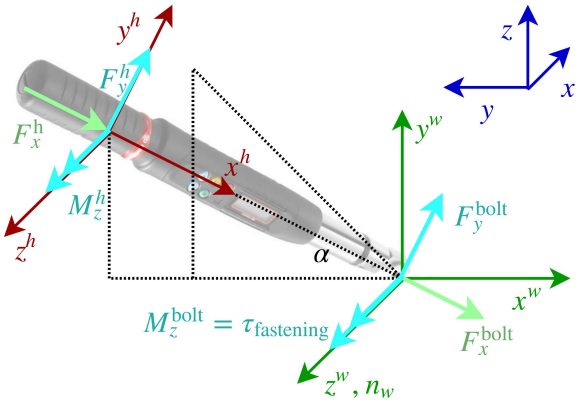


Fig. 3.3. *Free body diagram of the wrench tool: world (blue), wall (w, green) and hand (h, red) reference systems. Following relationships between the reaction torques and forces acting on the tool and the nut/bolt hold:* $F_x^h = -F_x^{bolt}$ *(light green),* $F_y^h = -F_y^{bolt}$ *(turqoise),* $M_z^h = -\tau_{fastening} - L_{wrench}F_y^{bolt}$ *(turqoise).*

Ahmad et Lee [1990] Mi et Jia [2004]. An exploration motion should be rich in information such that it leads to a well-conditioned regressor. Also, we want to record reaction forces as low as possible to limit the load on the robot structure.

The first goal can be achieved by moving along a tangent of the force field around the nut

$$\boldsymbol{t} = \boldsymbol{n}_w \times \boldsymbol{F}\,. \tag{3.8}$$

$\boldsymbol{n}_w$ is the wall normal and $\boldsymbol{F}$ is the currently measured external force. This leads to a circulating movement around the nut if a perfectly circular 2-D force field is assumed.

Low reaction forces can be achieved with a radial component along the direction of the gradient of the force field $\boldsymbol{g} = \boldsymbol{F}/\|\boldsymbol{F}\|$ along which we want to control the force error $\epsilon_F = \|\boldsymbol{F}\| - \|\boldsymbol{F}\|_d$. This keeps the robot end-effector on a single contour line of the force field with a desired constant force norm $\|\boldsymbol{F}\|_d$, see fig. 3.1.

The superposition of the tangential and radial component gives us the overall change of motion which is then added to the current end-effector position $\boldsymbol{x}^{(k)}$

$$\boldsymbol{x}_d^{(k+1)} = \boldsymbol{x}^{(k)} + h_{\text{tangent}}\boldsymbol{t} + h_{\text{force}}\epsilon_F\boldsymbol{g} \tag{3.9}$$

h are the respective control gains with $h_{\text{tangent}} \approx 3h_{\text{force}}$.

3.3 Fastening trajectory

The robot continues to fasten the nut once the insertion of the nut into the wrench tool is confirmed. The fastening trajectory is thereby composed of three components:

- **Basic trajectory**: It is designed as a circular motion $\begin{bmatrix}-L_{\text{wrench}}\cos(\alpha) & L_{\text{wrench}}\sin(\alpha)\end{bmatrix}$ in the wall plane $\boldsymbol{x}_w - \boldsymbol{y}_w$. L_{wrench} is the length of the wrench tool. The nut position is the circle center and the wrench length is its radius. The hand is orientated

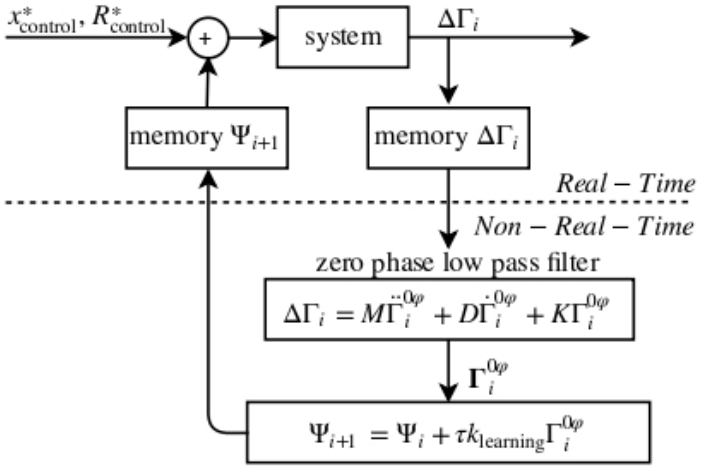


Fig. 3.4. *The trajectory learning control diagram: The system is unknown and outlines the robot's dynamics and its interaction with the environment. Its input are the desired trajectories for the end-effector $\boldsymbol{x}^d$ and $\boldsymbol{R}^d$ and its output is the measured wrench error $\boldsymbol{\Delta\Gamma}$.*

around the wall normal axis $\boldsymbol{z}_w$ by $\boldsymbol{R}^{\boldsymbol{z}_w}(\alpha)$ in such a way that it points towards the nut at all times (see fig. 3.3). α is a minimum jerk parametrization Flash et Hogan [1985].

- **Admittance control**: This enables safe interaction between the robot and its environment which it is quasi rigidly connected to by the tool-nut connection. The admittance control law connects the external (ext) reaction forces $\boldsymbol{F}_{\text{ext}}$ and moments $\boldsymbol{T}_{\text{ext}}$ measured by the 6-D wrench sensor at the hand (h) with the corresponding translational and rotational displacements $\boldsymbol{\Delta x}^h$ and $\boldsymbol{\Delta\phi}^h$ as follows:

$$\Delta\boldsymbol{F}^h_{\text{ext}} = \boldsymbol{F}^{\text{measured},h}_{\text{ext}} - \boldsymbol{F}^h_{\text{ext},d} = \boldsymbol{M}\boldsymbol{\Delta}\ddot{\boldsymbol{x}}^h + \boldsymbol{D}\boldsymbol{\Delta}\dot{\boldsymbol{x}}^h + \boldsymbol{K}\boldsymbol{\Delta x}^h \tag{3.10}$$

$$\Delta\boldsymbol{T}^h_{\text{ext}} = \boldsymbol{T}^{\text{measured},h}_{\text{ext}} - \boldsymbol{T}^h_{\text{ext},d} = \boldsymbol{J}\boldsymbol{\Delta}\ddot{\boldsymbol{\phi}}^h + \boldsymbol{L}\boldsymbol{\Delta}\dot{\boldsymbol{\phi}}^h + \boldsymbol{C}\boldsymbol{\Delta\phi}^h\,. \tag{3.11}$$

All gain matrices are chosen to be diagonal which allows solving the decoupled second order ODE system for $\Delta\boldsymbol{x}^h$ and $\Delta\boldsymbol{\phi}^h$ by the finite difference method. In this work we choose the gains on the diagonals as $M = 2000$ kg, $D = 1600\frac{\text{kg}}{\text{s}}$, $K = 20\frac{\text{kg}}{\text{s}^2}$, $J = 800\ \text{kg}\cdot\text{m}$, $L = 600\frac{\text{kg}\cdot\text{m}}{\text{s}}$ and $C = 13\frac{\text{kg}\cdot\text{m}}{\text{s}^2}$.

- **Trajectory learning control**: Admittance control is only suitable for slow deviations from the desired value due to its low pass filter characteristics. For occurrences of rapid reaction wrenches we apply a phase-free learned trajectory $\boldsymbol{\Psi}$ (Fig. 3.4) Bien et Xu [1998]. The recorded reaction wrenches $\Delta\boldsymbol{\Gamma}_i$ in each turning motion i are zero-phase low-pass filtered to $\Delta\boldsymbol{\Gamma}_i^{0\varphi}$ in an offline process. The low-pass filter is designed as a mass-damper-spring system with gains $M = 1.0$ kg, $D = 1.0$ kg/s, $K = 1.0\ \text{kg/s}^2$, $J = 1\ \text{kg}\cdot\text{m}$, $L = 1\frac{\text{kg}\cdot\text{m}}{\text{s}}$ and $C = 1\frac{\text{kg}\cdot\text{m}}{\text{s}^2}$. The learned wrenches are convoluted by a Tukey window τ Harris [1978] which sets the learned trajectory to zero amplitude on both ends $t = 0$ s and $t = T$:

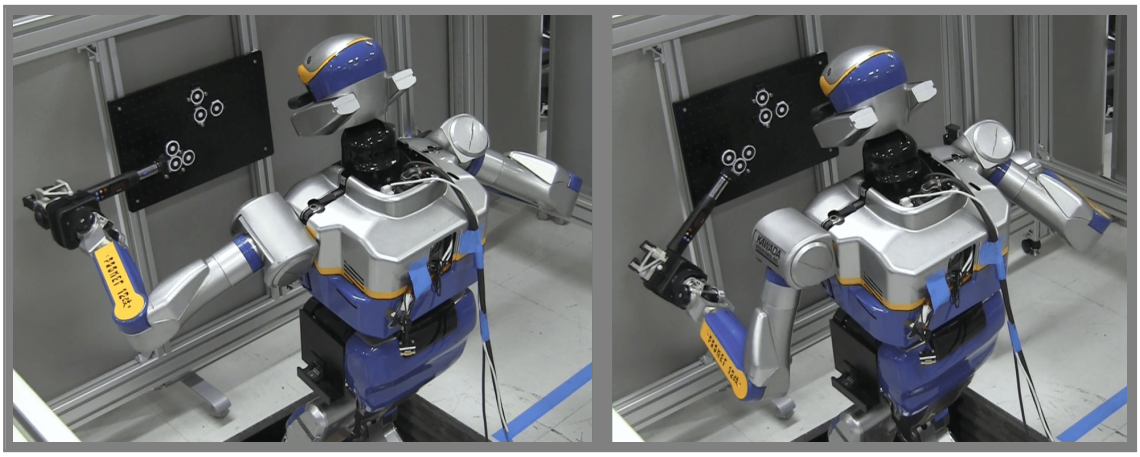


Fig. 3.5. *Robot during nut insertion and nut fastening*

$$\boldsymbol{\Psi}_{i+1} = \boldsymbol{\Psi}_i + \tau k_{\text{learning}} \Delta \boldsymbol{\Gamma}_i^{0\varphi} \,. \tag{3.12}$$

This ensures that the wrench characteristics are repeatable by resetting the initial conditions of the system after every trial. k_{learning} is a learning gain chosen in the range between 0 and 1.

These three components are then expressed in the world frame and composed to the desired hand position $\boldsymbol{x}_d$ and rotation $\boldsymbol{R}_d$.

3.4 Experimental validation

For the experimental validation (see fig. 3.5) a Kawada HRP-2Kai humanoid robot with 32 DoF's (plus 6 DoF's for the unactuated free-flyer) and 6-D wrench sensors on its hands is used. Since the fastening movements are slow we deem a purely static gravity compensation as sufficient. We choose the FACOM E.306-30D weighing 1.52 kg as our torque wrench. In order to allow a firm grasp of the tool we 3-D printed an adapter for the tool handle for a better fit into the robot gripper. The hexagonal nut is pre-fastened onto a M5 bolt sticking 1 cm out of a metal plate. The plate is perpendicularly fixed in a solid metal structure of dimensions $2 \times 2 \times 2$ m^3. The robot is equipped with a Xtion RGB-D camera. The marker tracking is done by the open source software `whycon` Nitsche et al. [2015] with the 3-D marker positions in image frame as output. A single marker only gives the position but the orientation can be easily obtained by aligning three markers into a L-shape.

3.4.1 Visual servoing and tool insertion

In order to obtain accurate knowledge of the relative position between the nut and the tool both the tool and the wall are equipped with `whycon` markers. The offset

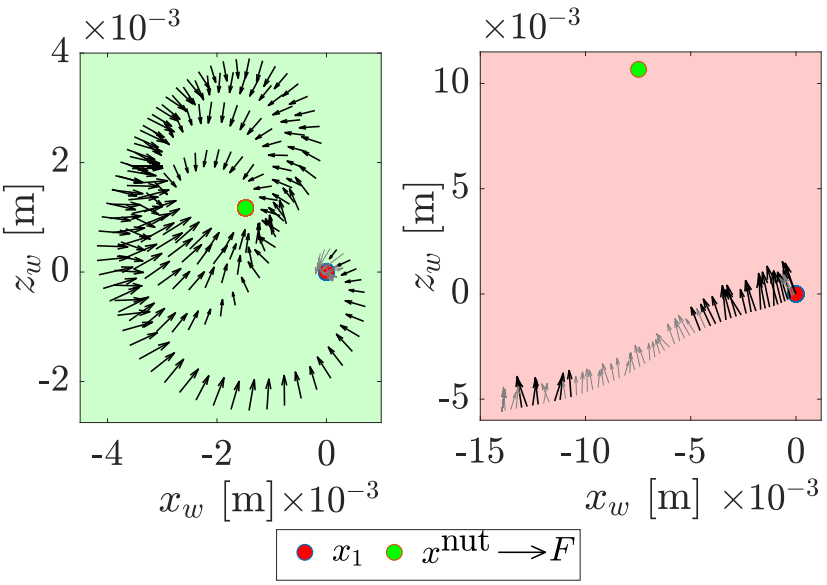


Fig. 3.6. *Sensed force field for inserted (left, with green background) and non-inserted tool (right, with red background); forces with $\|\boldsymbol{F}\| < 1$ N are coloured gray.*

between the nut to the wall marker is known. First, the tool marker is approximately moved into the field of view. The visual servoing controller then steers the tool-tip in front of the nut with a certain offset in wall normal direction before slowly moving towards the nut until a certain force threshold is surpassed in wall normal direction. At this point the hexagonal nut and the hexagonal tooltip are most likely misaligned. In order to overcome this misalignment the robot commences the circular fastening motion around the nut while pressing slightly along the nut axis. This way a full insertion can be achieved within usually two turns.

3.4.2 Insertion detection

The tool and nut connection is now occluded by the tool and requires tactile sensing in order to determine whether the insertion was conducted successfully. For this we use the method described in section 3.2 which enables to reliably detect both cases of inserted and non-inserted nuts.

The recorded exploration movement with recorded reaction forces for the case of a correct insertion of the tool onto the nut is depicted in the left graph of fig. 3.6. The robot performs a circular movement around the point which is later identified as the nut position $\boldsymbol{x}^{\text{nut}}$. This 10 s and 0.054 m long exploration movement resulted in FFC $= 0.90$ and $\kappa = 1.10$. Additionally, 95% of the data points are deemed valid with $\|\boldsymbol{F}\| \geq 1.0$ N, correctly indicating a successful tool insertion. The peak reaction force was $\|\boldsymbol{F}\|_{\max} = 3.33$ N and is only slightly above the desired one $\|\boldsymbol{F}\|_d = 2$ N. The low reaction forces occurring at the beginning of the exploration can be explained by the clearance between the wrench tool and the tool-adapter and between the tool-adapter

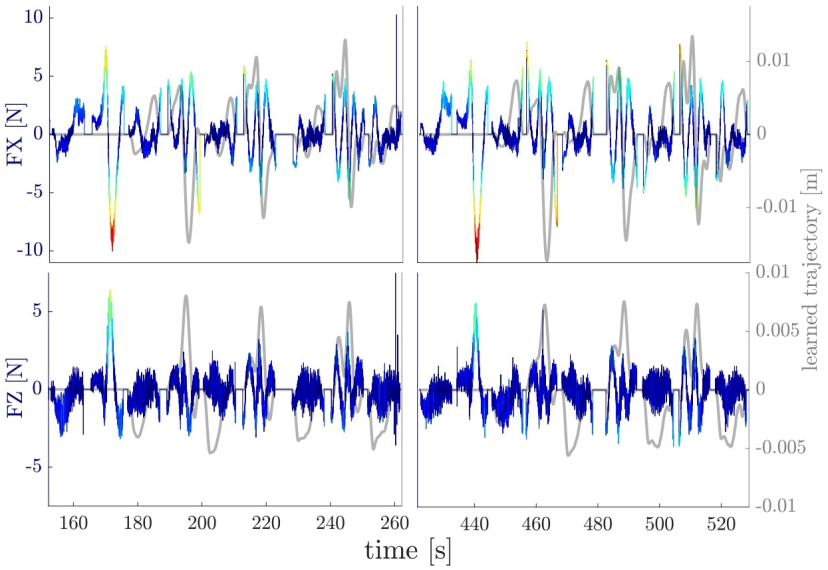


Fig. 3.7. *Force deviations F_x^h and F_z^h from their desired values $F_{x,d}^h = 0$ N and $F_{z,d}^h = 3.5$ N (in color) and applied learned trajectory (in gray) for the two fastening processes. The force error peaks are decreasing with time as the trajectory is refined by the learning control.*

and the nut.

The case of an unsuccessful insertion is depicted in the right graph of fig. 3.6. Here, the end-effector clearly drifts away from the starting point and stops after the travelled distance exceeds the threshold of $\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^{(k)}\| \leq 0.015$ m. The FFC is close to one FFC $= 0.96$ but the high condition number $\kappa = 4.93$ due to the almost linear motion correctly identifies the incorrect insertion. Additionally, only 16% of the recorded forces were above the threshold.

The corresponding gains for the exploration motion are chosen relatively small $h_{\text{tangent}} = 0.006$ and $h_{\text{force}} = 0.00225$ such that it results in a quasi static exploration movement with a speed of ≈ 0.54 cm/s.

3.4.3 Fastening process

The fastening motion is designed as described in sec. 3.3. One movement up or down covers an angle of 90° within 10 s. The wall plane is approximately 0.9 m in front of the robot and makes the desired trajectory achievable well within the workspace of the robot.

The tool and wall axial directions of the end-effector are subject to admittance control in order to enable safe interaction between the robot and its environment. By prescribing the desired force $F_{z,d}^h = 3.5$ N, the robot presses the end-effector against the nut in order to prevent breaking contact during the execution of the fastening movement. Low reaction forces are achieved by reducing force components in radial direction along the tool axis $F_{x,d}^h = 0$ N and the torque about the nut axis $M_{z,d}^h = 0$ Nm.

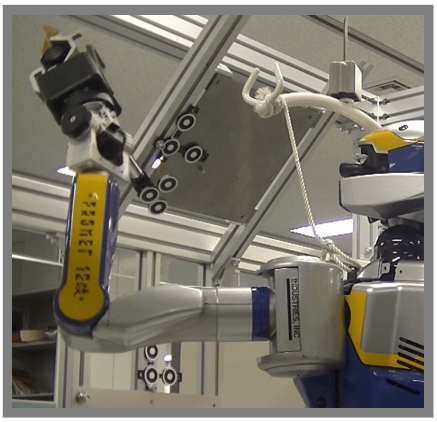


Fig. 3.8. *HRP-2Kai fastening the nut on the upper inclined wall.*

Figure 3.7 shows how the trajectory learning reduces the quickly occurring and vanishing high reaction force peaks by approximately 50%. Such peaks up to 10 N for F_x^h and F_z^h can be observed at around 175 s but decrease to 5 N at around 245 s. The same behaviour can be observed for a second trial from 420 s to 530 s (with the learned trajectory from the first trial being reset beforehand).

3.4.4 Process Reliability

In order to test the reliability of the fastening process we set up an experiment where one nut needs to be fastened onto a M8 bolt fixed on a 45° inclined wall. The robot then proceeds to fasten a second nut on a perpendicular metal wall (see fig. 3.8). The current fastening torque is calculated by (see fig. 3.3)

$$\tau_{\text{fastening}} = -M_z^h + L_{\text{wrench}} F_y^h \tag{3.13}$$

in order to identify the desired fastening torque of 7 Nm.

The overall process of fastening two nuts was repeated nine times with robot restarts between the repetitions. Five trials were successful while for three attempts some issue occurred (see table 3.1). In one case the tool adapter fell off from the wrench during the removal of the tool from the first nut. The rest of the times several insertion attempts were necessary for the first or second nut. In one trial the insertion of the second nut failed completely such that it had to be aborted.

The fastening torque exceeds the desired value in all the trials since the robot is not able to stop its movement exactly in the instance of reaching the desired threshold.

Since the trials with some issue take more time the overall trial duration has quite a large standard deviation of 58 s. Also, the degree of pre-fastening of the nuts influenced the number of fastening movements #turns necessary to tighten the nut. The fastening of the first nut on the inclined wall was conducted with a smaller value of $\alpha = 70°$ in order to avoid collision with the environment. This increases the number of necessary fastening motions.

full success	5	56%	$\tau_{\text{fastening}}$	8.14 ± 0.27 Nm
with some issues	3	33%	duration/trial	6:31 ± 0:58 min
failure	1	11%	1st nut: #turns	17.1 ± 2.6
$\sum$	9	100%	2nd nut: #turns	11.8 ± 1.6

Table 3.1 – Experimental results

3.5 Conclusion

With this nut fastening demonstration we showed the high precision capabilities of the HRP-2Kai humanoid robot with the corresponding control framework centered around the LSSOL quadratic programming solver. For this we implemented specific tasks and validated the reliability of our approach experimentally. The high precision capabilities were shown with the nut insertion and the low reaction forces during the fastening process.

Sensors play a vital part in the achievement of such precise tasks. Force sensors are subject to noise and offsets which need to be accounted for. The vision system used here is marker based and is only precise to a certain degree depending on the distance between the camera and the markers. However, with fusing both those measurements we are able to reliably locate, approach and insert the nut into the tool.

In order to achieve precise fastening movements the motion speed is limited. Besides further developments in the control domain changes, the kinematic structure itself could contribute positively to further improvements. Thinkable are dedicated robots with specialized tools. An example would be Kawada's Nextage robot which can change its end-effector on the fly.

Lastly, by adding further contacts between the robot and its environment the quality of the fastening motion can be improved through increased physical stability and rigidity of the robot structure.

CHAPTER 4

Singularity resolution for kinematic control problems

In the previous chapter we used the LSP formulation (2.16) in order to tackle a typical industrial task with a humanoid robot. It delivered high accuracy despite the weighted formulation of tasks on the objective level. However, we imposed two big assumptions:

- the constraints are always fully feasible without any conflict between them and kinematic singularities for example of the geometric contact constraints are avoided
- the objective tasks approach kinematic and algorithmic singularities only to such a degree as it is safeguarded by the damping introduced by the reference posture task.

Additionally, the notion of a soft or weighted hierarchy prevents clear distinction between safety, security and optimality concerned tasks.

This motivates us in this chapter to first introduce the notion of hierarchical LSP's of the form (2.34)

$$\min_{\boldsymbol{x},\boldsymbol{w}_l} \quad \frac{1}{2}\left\|\boldsymbol{w}_l\right\|^2 \qquad l = 1,\ldots,p \tag{4.1}$$

$$\text{s.t.} \quad \boldsymbol{A}_l\boldsymbol{x} - \boldsymbol{b}_l \leqq \boldsymbol{w}_l \tag{4.2}$$

$$\underline{\boldsymbol{A}}_{l-1}\boldsymbol{x} - \underline{\boldsymbol{b}}_{l-1} \leqq \underline{\boldsymbol{w}}^*_{l-1}\,. \tag{4.3}$$

Solving these LSP's in a very efficient manner was first proposed in Escande et al. [2014] and then further refined in Dimitrov et al. [2015].

This sort of multi-level constrained optimization problems with constraint relaxation overcomes the issue of the need for always feasible constraints. Additionally, the notion of strict hierarchies allows clear distinction between safety, security and optimality concerned tasks.

The main purpose of this chapter is now to design, implement and validate methods to handle kinematic and algorithmic singularities in multi-level hierarchies for kinematics based control problems. While not only enabling tuning free and numerically stable behaviour over a large task-space we also supersede classical damping approaches in terms of accuracy.

The first section 4.1 is dedicated to exploring the problem of kinematic singularities in kinematic control problems from a Taylor model point of view. We do this for a single level and observe how singularities negatively influence the robot control. A resolution method based on Newton's method of optimization is then proposed (sec. 4.1.1). The method is extended to multi-level constrained optimization problems (sec. 4.1.2). In a last step inequality constraints are incorporated into our scheme (sec. 4.1.3).

In the next section 4.2 technical details are given for the calculation of the second order information for Newton's method. This can be achieved either analytically (sec. 4.2.1.1) or by approximation with the SR1 method (sec. 4.2.1.2) and the BFGS algorithm (sec. 4.2.1.3).

Section 4.3 presents the implemented switching strategy between the GN algorithm and Newton's method. In sec. 4.4, a trust region adaptation method for the case of multi-level constrained hierarchies is introduced. Finally, the proposed methods are evaluated on a simulation test bench with a 2-D and a 3-D robot in sec. 4.5.

To test 1 of this test bench we refer to throughout this chapter (namely sections 4.2.1.2 and 4.3) so we shortly introduce it. Here, a 2-D 4 DoF robot with a translating base and three links and three revolute joints (uniform link length of 1 m) follows two targets oscillating cross-diagonally with its two hierarchically ordered end-effector (ef) tasks. The targets are just reachable in the corner case (the robot needs to stretch into kinematic singularity in order to be able to reach the high priority blue target, see fig. 4.1).

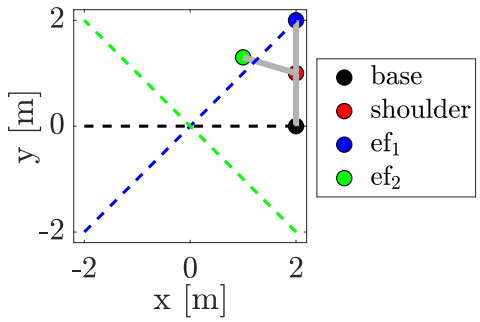


Fig. 4.1. *Schematics of test 1. The high priority ef 1 follows the blue dashed line while the lower priority ef 2 tries to reach the green dashed line.*

More details including the exact task hierarchy (fig. 4.7) are given in sec. 4.5.1.

4.1 Kinematic and algorithmic singularities in kinematic control and their resolution

4.1.1 Single level with equality constraints - A model point of view

In the previous chapter 2.1 we have stated the first order controller

$$-\dot{\boldsymbol{e}}^{\text{ctrl}} - \boldsymbol{J}\dot{\boldsymbol{q}}^{(k)} = k_p\boldsymbol{e} - \boldsymbol{J}\dot{\boldsymbol{q}}^{(k)} = \boldsymbol{e} - \boldsymbol{J}\boldsymbol{\Delta q}^{(k)} = \boldsymbol{0}, \tag{4.4}$$

which drives the task error $\boldsymbol{e} \in \mathbb{R}^m \rightarrow \boldsymbol{0}$ to zero, see (2.2). In this chapter the proportional gain is set to $k_p = 1$ for simplicity. Additionally, the control time step is chosen as $\Delta t = 1$ s which leads to the trivial relation $\dot{\boldsymbol{q}} = \boldsymbol{\Delta q} \in \mathbb{R}^n$.

We can now look for a small change $\boldsymbol{\Delta q}$ of the robot's configuration $\boldsymbol{q}$ with which we update the new one to $\boldsymbol{q}^{(k+1)} = \boldsymbol{q} + \boldsymbol{\Delta q}$. The required error decrease $\dot{\boldsymbol{e}}^{\text{ctrl}}$ may not be achievable so we solve the above *at best* using the linear LSP

$$\min_{\boldsymbol{\Delta q}} \quad \frac{1}{2}\|\boldsymbol{J}\boldsymbol{\Delta q} - \boldsymbol{e}\|_2^2 . \tag{4.5}$$

This form is akin to the *Gauss-Newton (GN) algorithm* and can be solved with the help of the pseudo-inverse $\boldsymbol{J}^+$ such that $\boldsymbol{\Delta q} = \boldsymbol{J}^+\boldsymbol{e}$. This approach is not robust in the vicinity of singularities: $\boldsymbol{J}$ is almost loosing at least one rank and $\boldsymbol{\Delta q}$ becomes very large. This can be seen by

$$\boldsymbol{J}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^T = \sum_{i=1}^{r} \frac{1}{\sigma_i}\boldsymbol{v}_i\boldsymbol{u}_i^T . \tag{4.6}$$

where the Singular Value Decomposition (SVD) of $\boldsymbol{J} = \boldsymbol{U}^T\boldsymbol{\Sigma}\boldsymbol{V}$ is used to calculate the pseudo-inverse. If the current robot configuration $\boldsymbol{q}$ is far away from singularities numerically stable results are obtained. However, if singularities are approached at least one singular value tends towards zero $\sigma \rightarrow 0$. This causes numerically high changes $\boldsymbol{\Delta q}$ in the joint configuration which can damage the robot's structure and needs to be prevented.

An example is given in the following with a two DoF robot with link length 1 m. The robot's initial end-effector's position is at $[1, 1]$ m from which it reaches to the desired position at $[\sqrt{2}/2, \sqrt{2}/2]$ m, see fig. 4.2. From the control iteration 10, the robot is close to kinematic singularity with the Jacobian being almost rank deficient with low minimum singular value. Slight numerical instabilities are seen on the joint level ($\sim 1^{-5}$ m) and the minimum singular value (see fig. 4.3).

If the target is slightly out of reach at $[\sqrt{2}/2 + 0.01, \sqrt{2}/2 + 0.01]$ m, the robot behaves clearly numerically unstable with high joint velocities, see Fig. 4.4. In this case the infeasibility acts as a disturbance of the right hand side $\delta\boldsymbol{e} = 0.01$ m. It leads to a significant perturbation of the end-effector position by $\delta\boldsymbol{x} = \boldsymbol{J}^+\delta\boldsymbol{e}$ as it is

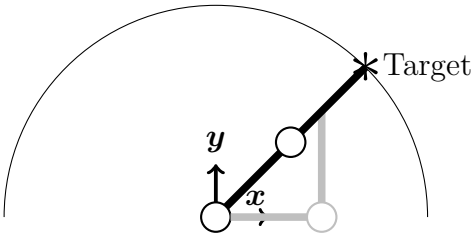


Fig. 4.2. *2-D, 2 DoF robot with link length 2 m. The initial configuration is depicted in gray, the desired configuration with the target ∗ is given in black. The workspace of the robot is the area within the circle.*

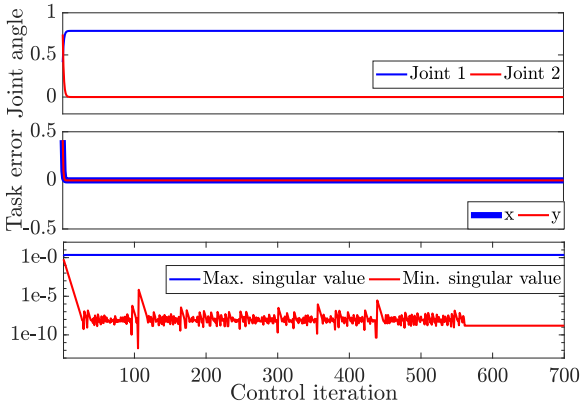


Fig. 4.3. *Joint angles (rad), task error (m) and largest and smallest singular value while approaching the singular configuration. The target is just in reach at* $[\sqrt{2}/2, \sqrt{2}/2]$ *m. The task-space behaviour is numerically stable but there is some noise on the joint level (upper graph) and the smallest singular value (lower graph).*

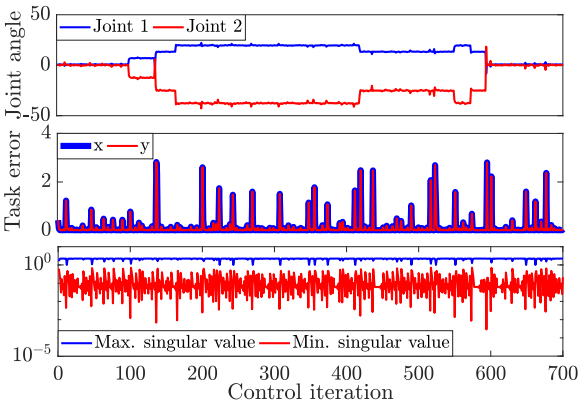


Fig. 4.4. *Joint angles (rad), task error (m) and largest and smallest singular value while approaching the singular configuration. The target is just out of reach at* $[\sqrt{2}/2 + 0.01, \sqrt{2}/2 + 0.01]$ *m. The robot is numerically unstable with large changes in the joint angles (upper graph) and heavily oscillating end-effector (middle graph).*

magnified by the pseudo-inverse of the ill-conditioned Jacobian $\boldsymbol{J}$ (see the perturbation analysis for least-squares solutions, Björck [1996], p. 28).

If the robot is exactly at the singularity, the corresponding singular value is zero and the Jacobian is rank deficient. In the calculation of the pseudo-inverse this value is skipped and a well defined solution can be obtained. The so-called truncated SVD decomposition implements a similar behaviour of discarding singular values in the pseudo-inverse calculation if they are below a certain threshold Hansen [1987].

To further outline the GN algorithm's lack of robustness if close to singularities we now formulate another closely related LSP. For this, let us define the scalar target function Dennis et al. [1981]; Deo et Walker [1993]:

$$\Phi(\boldsymbol{q}) = \frac{1}{2}\|\boldsymbol{f}_d - \boldsymbol{f}(\boldsymbol{q})\|_2^2 = \frac{1}{2}\|\boldsymbol{e}(\boldsymbol{q})\|_2^2 = \frac{1}{2}\boldsymbol{e}^T\boldsymbol{e}\,. \tag{4.7}$$

This non-linear function can be approximated by a second order Taylor series

$$\Phi(\boldsymbol{q}+\boldsymbol{\Delta q}) \approx \Phi(\boldsymbol{q}) + \boldsymbol{\Delta q}^T \nabla\Phi + \frac{1}{2}\boldsymbol{\Delta q}^T \nabla^2\Phi \boldsymbol{\Delta q} \tag{4.8}$$

$$= \Phi(\boldsymbol{q}) - \boldsymbol{\Delta q}^T \boldsymbol{J}^T \boldsymbol{e} + \frac{1}{2}\boldsymbol{\Delta q}^T (\boldsymbol{J}^T\boldsymbol{J} + \boldsymbol{H})\boldsymbol{\Delta q}\,. \tag{4.9}$$

$\boldsymbol{H} \in \mathbb{R}^{n,n}$ assembles the second order derivatives (or Hessians) of $\boldsymbol{f}(\boldsymbol{q})$. This *model* can only be 'trusted' to accurately represent the original function Φ in a small neighbourhood $\boldsymbol{\Delta}$ of the current point $\boldsymbol{q}$. It is commonly referred to as *trust region*. Further details are given in sec. 4.4. The corresponding gradient is given by

$$\nabla\Phi(\boldsymbol{q}) = -\boldsymbol{J}^T\boldsymbol{e} \tag{4.10}$$

and the corresponding second order derivative is given by

$$\nabla^2\Phi(\boldsymbol{q}) = \boldsymbol{J}^T\boldsymbol{J} + \sum_{i=1}^{m} \boldsymbol{e}_i \nabla^2 \boldsymbol{f}_i = \boldsymbol{J}^T\boldsymbol{J} + \boldsymbol{H}\,. \tag{4.11}$$

$\boldsymbol{\Delta q}$ can then be computed by the minimization problem

$$\min_{\boldsymbol{\Delta q}} \quad \Phi(\boldsymbol{q}+\boldsymbol{\Delta q}) = \min_{\boldsymbol{\Delta q}} \quad \frac{1}{2}\boldsymbol{e}^T\boldsymbol{e} - \boldsymbol{\Delta q}^T\boldsymbol{J}^T\boldsymbol{e} + \frac{1}{2}\boldsymbol{\Delta q}^T(\boldsymbol{J}^T\boldsymbol{J}+\boldsymbol{H})\boldsymbol{\Delta q}\,. \tag{4.12}$$

This minimization problem corresponds to *Newton's method* applied to non-linear least-squares Nocedal et Wright [2006]. In case of positive definiteness of the second order information $\boldsymbol{H}$, the Cholesky decomposition $\boldsymbol{H} = \boldsymbol{R}^T\boldsymbol{R}$ can be obtained. Now above can be reformulated to the following least-squares program

$$\min_{\boldsymbol{\Delta q}} \quad \frac{1}{2}\left\| \begin{bmatrix} \boldsymbol{J} \\ \boldsymbol{R} \end{bmatrix} \boldsymbol{\Delta q} - \begin{bmatrix} \boldsymbol{e} \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 = \min_{\boldsymbol{\Delta q}} \quad \frac{1}{2}\|\boldsymbol{J}\boldsymbol{\Delta q} - \boldsymbol{e}\|_2^2 + \frac{1}{2}\|\boldsymbol{R}\boldsymbol{\Delta q}\|_2^2\,. \tag{4.13}$$

It can be seen that the GN algorithm is a degraded version of this form where the second order term $\boldsymbol{H}$ (or $\boldsymbol{R}$) has been discarded. This increases the model error of the second-order Taylor-series. While a truncated Taylor approximation represents the original non-linear function sufficiently away from singularities this does not hold when approaching them. In this case the matrix $\boldsymbol{R}$ helps to keep full rank as the rank of $\boldsymbol{J}$ decreases. In appendix B we illustrate such an augmentation with a simple 2-D robot.

Above form (4.13) is very similar to a classical method of singularity resolution in kinematic control problems: Damped least-squares or the LM algorithm regularizes the solution vector $\boldsymbol{\Delta q}$ by

$$\min_{\Delta q} \frac{1}{2}\|\boldsymbol{J}\boldsymbol{\Delta q} - \boldsymbol{e}\|_2^2 + \frac{1}{2}\alpha^2\|\boldsymbol{I}\boldsymbol{\Delta q}\|_2^2\,. \tag{4.14}$$

This leads to Chiaverini [1997]

$$\boldsymbol{J}^* = \boldsymbol{J}^T(\boldsymbol{J}\boldsymbol{J}^T + \alpha^2\boldsymbol{I})^{-1} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \alpha^2}\boldsymbol{v}_i\boldsymbol{u}_i^T\,, \tag{4.15}$$

compare with (4.6). If the damping factor α is chosen small with $\alpha \ll \sigma_i$ then the factor $\sigma_i/(\sigma_i^2 + \alpha^2) \approx 1/\sigma_i$ is close to the one of the un-damped pseudo-inverse. However, if $\sigma_i \to 0$ then the factor is still well defined with $\sigma_i/(\sigma_i^2 + \alpha^2) \to 1/\alpha$ instead of $1/\sigma_i \to \infty$.

Interestingly, the LM algorithm corresponds to approximating the second order information $\boldsymbol{H}$ of the Taylor expansion as a multiple of the identity. While this leads to numerical stability it does not necessarily lead to the best representation of the original function.

Instead, we propose in this thesis to use the true second order information or its approximation by the SR1 method or the BFGS algorithm. If we are approaching singularities we switch from the GN algorithm to Newton's method. A switching method is proposed in sec. (4.3).

Our reasoning is that neglecting the second order information in the Taylor expansion marks an insufficient model of the original non-linear function around singularities. At the same time the Hessian can be interpreted as an 'optimal damping' term due to the close relationship of Newton's method with the LM algorithm. Instead of approximating the second order information by a weighted identity matrix, Newton's method uses the true Hessian. We show throughout this and the next chapter that this approach not only leads to numerically stable robot behaviour but also exceeds the classical damping approach in terms of error convergence.

In the following we refer to Newton's method as being the 'augmented' (as in augmented with second order information) version of the GN algorithm Dennis et al. [1981].

4.1.2 Hierarchies with equality constraints

In the previous section 4.1.1 we have seen that linear control problems can be cast into an optimization based formulation thanks to our choice of the time step parameter $\Delta t = 1$ s. Both the GN algorithm and Newton's method of optimization can be conveniently expressed in least-squares form which makes them eligible to be solved with the constrained QP solvers previously presented in chapter 2. We then can lexicographically order control objectives from level 1 to l in the following way:

$$\begin{aligned}
&\text{For } l = 1, \ldots, p: \\
&\qquad \min_{\boldsymbol{\Delta q}, \boldsymbol{w}_l^{(k+1)}} \quad \frac{1}{2}\|\boldsymbol{w}_l^{(k+1)}\|_2^2 && (4.16)\\
&\qquad \text{s.t.} \quad \boldsymbol{e}_l - \boldsymbol{J}_l\boldsymbol{\Delta q} = \boldsymbol{w}_l^{(k+1)} && (4.17)\\
&\qquad\qquad \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1}\boldsymbol{\Delta q} = \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)}\,. && (4.18)
\end{aligned}$$

Unlike in the previous section 4.1.1, the relaxation $\boldsymbol{w}$ is introduced on each level (also in the case of a single level) and we keep this formulation throughout this thesis.

The corresponding Lagrangian function of this discretized problem at time step $k+1$ can be formulated as follows (the index $k+1$ of $\boldsymbol{w}^{(k+1)}$, $\boldsymbol{w}^{*,(k+1)}$ and $\boldsymbol{\lambda}^{(k+1)}$ is omitted for better readability):

$$\begin{aligned}\mathcal{L}_l &= \frac{1}{2}\boldsymbol{w}_l^T\boldsymbol{w}_l + \boldsymbol{\lambda}_{l,l}^T(\boldsymbol{w}_l - \boldsymbol{e}_l + \boldsymbol{J}_l\boldsymbol{\Delta q}) + \underline{\boldsymbol{\lambda}}_{l-1,l}^T\left(\underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1} + \underline{\boldsymbol{J}}_{l-1}\boldsymbol{\Delta q}\right)\\ &= \frac{1}{2}\boldsymbol{w}_l^T\boldsymbol{w}_l + \boldsymbol{\lambda}_{l,l}^T\left(\boldsymbol{w}_l - \boldsymbol{e}_l\right) + \underline{\boldsymbol{\lambda}}_{l-1,l}^T\left(\underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1}\right) + \boldsymbol{\Delta q}^T\boldsymbol{J}_l^T\boldsymbol{\lambda}_{l,l} + \boldsymbol{\Delta q}^T\underline{\boldsymbol{J}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l}\,.\end{aligned} \tag{4.19}$$

$\boldsymbol{\lambda}_{l,l}$ are the Lagrange multipliers associated to (4.17) and $\underline{\boldsymbol{\lambda}}_{l-1,l}$ are the ones associated to (4.18). $\boldsymbol{\lambda}_{i,l}$ indicates the conflict of level l with level i.

We can now find the solution of this constrained optimization problem by determining the stationary points of the Lagrangian function $\nabla_{\boldsymbol{\Delta q},\boldsymbol{w}_l,\boldsymbol{\lambda}_{l,l},\underline{\boldsymbol{\lambda}}_{l-1,l}}\mathcal{L}_l = \boldsymbol{0}$. These first order optimality conditions are stated as

$$\nabla_{\boldsymbol{\Delta q},\boldsymbol{w}_l,\boldsymbol{\lambda}_{l,l},\underline{\boldsymbol{\lambda}}_{l-1,l}}\mathcal{L}_l = \boldsymbol{K}_l(\boldsymbol{\Delta q},\boldsymbol{w}_l,\boldsymbol{\lambda}_{l,l},\underline{\boldsymbol{\lambda}}_{l-1,l}) = \begin{bmatrix}\boldsymbol{J}_l^T\boldsymbol{\lambda}_{l,l} + \underline{\boldsymbol{J}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l}\\ \boldsymbol{w}_l + \boldsymbol{\lambda}_{l,l}\\ \boldsymbol{w}_l - \boldsymbol{e}_l + \boldsymbol{J}_l\boldsymbol{\Delta q}\\ \underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1} + \underline{\boldsymbol{J}}_{l-1}\boldsymbol{\Delta q}\end{bmatrix} = \boldsymbol{0} \tag{4.20}$$

or in matrix form

$$\begin{bmatrix}\boldsymbol{0} & \boldsymbol{0} & \boldsymbol{J}_l^T & \underline{\boldsymbol{J}}_{l-1}^T\\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0}\\ \boldsymbol{J}_l & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0}\\ \underline{\boldsymbol{J}}_{l-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{\Delta q}\\ \boldsymbol{w}_l\\ \boldsymbol{\lambda}_{l,l}\\ \underline{\boldsymbol{\lambda}}_{l-1,l}\end{bmatrix} = \begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{0}\\ \boldsymbol{e}_l\\ \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{w}}_{l-1}^*\end{bmatrix}. \tag{4.21}$$

In the above derivation we have formulated our constrained optimization problem with the goal of driving the linearisation of the non-linear task error $\boldsymbol{e}(\boldsymbol{q}) = \boldsymbol{f}_d - \boldsymbol{f}(\boldsymbol{q})$ (with $\nabla_{\boldsymbol{q}}\boldsymbol{e} = -\boldsymbol{J}$)

$$\boldsymbol{e}_l(\boldsymbol{q}+\boldsymbol{\Delta q}) \approx \boldsymbol{e}_l(\boldsymbol{q}) + \nabla_{\boldsymbol{q}}\boldsymbol{e}_l(\boldsymbol{q})\boldsymbol{\Delta q} = \boldsymbol{e}_l(\boldsymbol{q}) - \boldsymbol{J}\boldsymbol{\Delta q} = \boldsymbol{0} \text{ (or } \boldsymbol{w}_l^*) \tag{4.22}$$

to zero 'at best' with some possible relaxation $\boldsymbol{w}_l^*$ in the case of infeasibility. However, this Lagrangian analysis does not result in an expression for the hierarchical second order information since we have $\nabla^2_{\boldsymbol{\Delta q}}\mathcal{L}_l = \boldsymbol{0}$.

Therefore, let's look at the original non-linear optimization problem which is to be solved:

$$\min_{\boldsymbol{q},\boldsymbol{w}_l} \quad \frac{1}{2}\left\|\boldsymbol{w}_l\right\|^2 \qquad l = 1,\ldots,p \tag{4.23}$$

$$\text{s.t.} \quad \boldsymbol{e}_l(\boldsymbol{q}) = \boldsymbol{w}_l \tag{4.24}$$

$$\underline{\boldsymbol{e}}_{l-1}(\boldsymbol{q}) = \underline{\boldsymbol{w}}_{l-1}^*\,. \tag{4.25}$$

The goal now is to drive the constraint violation $\boldsymbol{w}_l$ of each level l to zero 'at best' such that the task error $\boldsymbol{e}_l(\boldsymbol{q}) = \boldsymbol{0}$ (or $\boldsymbol{e}_l(\boldsymbol{q}) = \boldsymbol{w}_l^*$ in the case of infeasibility). Already obtained optimal violations of previous levels $\underline{\boldsymbol{w}}_{l-1}^*$ must stay unchanged. The Lagrangian function of this problem writes as

$$\mathcal{L}_l = \frac{1}{2}\boldsymbol{w}_l^T\boldsymbol{w}_l + \boldsymbol{\lambda}_{l,l}^T(\boldsymbol{w}_l - \boldsymbol{e}_l) + \underline{\boldsymbol{\lambda}}_{l-1,l}^T(\underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1})\,. \tag{4.26}$$

$\boldsymbol{\lambda}_{l,l}$ are the Lagrange multiplier associated to (4.24) and $\underline{\boldsymbol{\lambda}}_{l-1,l}$ are the ones associated to all the constraints (4.25). The first order optimality condition of this problem is

$$\nabla_{\boldsymbol{q},\boldsymbol{w}_l,\boldsymbol{\lambda}_{l,l},\underline{\boldsymbol{\lambda}}_{l-1,l}}\mathcal{L}_l = \boldsymbol{K}_l(\boldsymbol{q},\boldsymbol{w}_l,\boldsymbol{\lambda}_{l,l},\underline{\boldsymbol{\lambda}}_{l-1,l}) = \begin{bmatrix} \boldsymbol{J}_l^T\boldsymbol{\lambda}_{l,l} + \underline{\boldsymbol{J}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l} \\ \boldsymbol{w}_l + \boldsymbol{\lambda}_{l,l} \\ \boldsymbol{w}_l - \boldsymbol{e}_l \\ \underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1} \end{bmatrix}. \tag{4.27}$$

Thereby, the gradient $\nabla_{\boldsymbol{\Delta q}}\mathcal{L}_l = \boldsymbol{J}_l^T\boldsymbol{\lambda}_{l,l}^{(k+1)} + \underline{\boldsymbol{J}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l}^{(k+1)}$ in (4.20) seems to be the discretized version of $\nabla_{\boldsymbol{q}}\mathcal{L}_l = \boldsymbol{J}_l^T\boldsymbol{\lambda}_{l,l} + \underline{\boldsymbol{J}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l}$ at time step $k+1$.

We then apply a Newton step

$$\boldsymbol{K}_l(\boldsymbol{x} + \boldsymbol{\Delta x}) \approx \boldsymbol{K}_l(\boldsymbol{x}) + \nabla\boldsymbol{K}_l(\boldsymbol{x})\boldsymbol{\Delta x} = \boldsymbol{0} \tag{4.28}$$

which leads to the expression

$$\begin{bmatrix} \boldsymbol{J}_l^T\boldsymbol{\lambda}_{l,l} + \underline{\boldsymbol{J}}_{l-1}^T\underline{\boldsymbol{\lambda}}_{l-1,l} \\ \boldsymbol{w}_l + \boldsymbol{\lambda}_{l,l} \\ \boldsymbol{w}_l - \boldsymbol{e}_l \\ \underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1} \end{bmatrix} + \begin{bmatrix} \sum_{i=1}^{l}\sum_{u=1}^{m_i}\lambda_{i,l}^u\boldsymbol{H}_{u,i} & \boldsymbol{0} & \boldsymbol{J}_l^T & \underline{\boldsymbol{J}}_{l-1}^T \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{J}_l & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \underline{\boldsymbol{J}}_{l-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta q} \\ \boldsymbol{\Delta w}_l \\ \boldsymbol{\Delta\lambda}_{l,l} \\ \boldsymbol{\Delta}\underline{\boldsymbol{\lambda}}_{l-1,l} \end{bmatrix} = \boldsymbol{0} \tag{4.29}$$

with $\boldsymbol{x} = \begin{bmatrix}\boldsymbol{q}^T & \boldsymbol{w}_l^T & \boldsymbol{\lambda}_{l,l}^T & \underline{\boldsymbol{\lambda}}_{l-1,l}^T\end{bmatrix}^T$. We define

$$\hat{\boldsymbol{H}}_l = \nabla_{\boldsymbol{q}}^2\mathcal{L}_l = \nabla_{\boldsymbol{q}}\underline{\boldsymbol{J}}_l^T\underline{\boldsymbol{\lambda}}_{l,l} = \sum_{i=1}^{l}\sum_{u=1}^{m_i}\lambda_{i,l}^u\boldsymbol{H}_{u,i} \tag{4.30}$$

which we refer to as the Lagrangian Hessian $\hat{\boldsymbol{H}}_l$ of level l throughout this thesis. It provides us with the second order information for Newton's method of multi-level constrained optimization.

Using the variable changes

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w} + \boldsymbol{\Delta w} \tag{4.31}$$

$$\boldsymbol{\lambda}^{(k+1)} = \boldsymbol{\lambda} + \boldsymbol{\Delta\lambda} \tag{4.32}$$

$$\underline{\boldsymbol{\lambda}}^{(k+1)} = \underline{\boldsymbol{\lambda}} + \boldsymbol{\Delta}\underline{\boldsymbol{\lambda}}\,, \tag{4.33}$$

we get the system (the index $k+1$ is omitted again)

$$\begin{bmatrix} \hat{\boldsymbol{H}}_l & \boldsymbol{0} & \boldsymbol{J}_l^T & \underline{\boldsymbol{J}}_{l-1}^T \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{J}_l & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \underline{\boldsymbol{J}}_{l-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{q} \\ \boldsymbol{w}_l \\ \boldsymbol{\lambda}_{l,l} \\ \underline{\boldsymbol{\lambda}}_{l-1,l} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{0} \\ \boldsymbol{e}_l \\ \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{w}}_{l-1} \end{bmatrix} . \tag{4.34}$$

Solving this system yields the step $\Delta\boldsymbol{q}$ which can be integrated to the new configuration $\boldsymbol{q}^{(k+1)} = \boldsymbol{q} + \Delta\boldsymbol{q}$.

Above is also the optimality condition $\nabla_{\boldsymbol{x}}\mathcal{L}_l = \boldsymbol{0}$ of the quadratic program

$$\min_{\Delta\boldsymbol{q}, \boldsymbol{w}_l^{(k+1)}} \quad \frac{1}{2}\|\boldsymbol{w}_l^{(k+1)}\|^2 + \frac{1}{2}\Delta\boldsymbol{q}^T\hat{\boldsymbol{H}}_l\Delta\boldsymbol{q} \tag{4.35}$$

$$\text{s.t.} \quad \boldsymbol{e}_l - \boldsymbol{J}_l\Delta\boldsymbol{q} = \boldsymbol{w}_l^{(k+1)} \tag{4.36}$$

$$\underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1}\Delta\boldsymbol{q} = \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} \tag{4.37}$$

with the Lagrangian function (the index $k+1$ is again omitted for better readability)

$$\mathcal{L}_l = \frac{1}{2}\boldsymbol{w}_l^T\boldsymbol{w}_l + \frac{1}{2}\Delta\boldsymbol{q}^T\hat{\boldsymbol{H}}_l\Delta\boldsymbol{q} \tag{4.38}$$

$$+ \boldsymbol{\lambda}_{l,l}^T(\boldsymbol{w}_l - \boldsymbol{e}_l + \boldsymbol{J}_l\Delta\boldsymbol{q}) + \underline{\boldsymbol{\lambda}}_{l-1,l}^T(\underline{\boldsymbol{w}}_{l-1}^* - \underline{\boldsymbol{e}}_{l-1} + \underline{\boldsymbol{J}}_{l-1}\Delta\boldsymbol{q}) . \tag{4.39}$$

This form is akin to the *hierarchical Newton's method of optimization*. If $\hat{\boldsymbol{H}}_l$ is positive definite above problem can be rewritten to least squares form

$$\min_{\Delta\boldsymbol{q}} \quad \frac{1}{2}\left\| \begin{bmatrix} \boldsymbol{J}_l \\ \boldsymbol{R}_l \end{bmatrix} \Delta\boldsymbol{q} - \begin{bmatrix} \boldsymbol{e}_l \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 \qquad l = 1, \ldots, p \tag{4.40}$$

$$\text{s.t.} \quad \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1}\Delta\boldsymbol{q} = \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} , \tag{4.41}$$

with the Cholesky decomposition $\hat{\boldsymbol{H}}_l = \boldsymbol{R}_l^T\boldsymbol{R}_l$ and with $\boldsymbol{w}_l^{(k+1)}$ being given implicitly. If the second order information $\boldsymbol{R}_l$ is neglected we get the *hierarchical GN-algorithm*

$$\min_{\Delta\boldsymbol{q}} \quad \frac{1}{2}\|\boldsymbol{J}_l\Delta\boldsymbol{q} - \boldsymbol{e}_l\|_2^2 \qquad l = 1, \ldots, p \tag{4.42}$$

$$\text{s.t.} \quad \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1}\Delta\boldsymbol{q} = \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} . \tag{4.43}$$

Both the hierarchical Newton's method and hierarchical GN algorithm are generalizations of (4.13) for the case of multi-level constrained optimization.

4.1.3 Hierarchies including inequality constraints

In robotic control problems it is often the case that a task value is not necessarily asked to be at a single value but to be within an admissible range. Examples are joint angles to be within their joint limits or the CoM to lay within the support polygon

spanned by the projection of the feet onto the ground. The linearized problem then is formulated as

$$\begin{aligned} &\text{For } l = 1, \dots, p: \\ &\quad \min_{\Delta \boldsymbol{q}, \boldsymbol{w}_l} \quad \frac{1}{2} \|\boldsymbol{w}_l^{(k+1)}\|_2^2 && (4.44) \\ &\quad \text{s.t.} \quad \boldsymbol{e}_l - \boldsymbol{J}_l \Delta \boldsymbol{q} \leqq \boldsymbol{w}_l^{(k+1)} \\ &\qquad\quad \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1} \Delta \boldsymbol{q} \leqq \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} . && (4.45) \end{aligned}$$

The first order optimality conditions developed in the previous section extend to the Karush-Kuhn-Tucker conditions. The according complementary condition provides the basis for the active-set method. The hierarchical GN algorithm and hierarchical Newton's method are then formulated as

$$\min_{\Delta \boldsymbol{q}} \quad \frac{1}{2} \|\boldsymbol{J}_l \Delta \boldsymbol{q} - \boldsymbol{e}_l\|_2^2 \qquad l = 1, \dots, p \tag{4.46}$$

$$\text{s.t.} \quad \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1} \Delta \boldsymbol{q} \leqq \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} \tag{4.47}$$

and

$$\min_{\Delta \boldsymbol{q}} \quad \frac{1}{2} \left\| \begin{bmatrix} \boldsymbol{J}_l \\ \boldsymbol{R}_l \end{bmatrix} \Delta \boldsymbol{q} - \begin{bmatrix} \boldsymbol{e}_l \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 \qquad l = 1, \dots, p \tag{4.48}$$

$$\text{s.t.} \quad \underline{\boldsymbol{e}}_{l-1} - \underline{\boldsymbol{J}}_{l-1} \Delta \boldsymbol{q} \leqq \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} , \tag{4.49}$$

respectively. Both formulations include equality constraints. Inactive constraints l result in $\boldsymbol{w}_l = \boldsymbol{0}$, $\boldsymbol{\lambda}_{l,l+1:p} = \boldsymbol{0}$ so they do not further influence the Lagrangian Hessian

$$\hat{\boldsymbol{H}}_g = \sum_{i=1}^{g} \sum_{j \in \mathcal{A}_i^{(k)}} \lambda_{i,g}^j \boldsymbol{H}_{j,i} = \boldsymbol{R}_g^T \boldsymbol{R}_g \tag{4.50}$$

on some level $g > l$. $\mathcal{A}_i^{(k)}$ is the active-set of the level i at the control iteration k containing m_i tasks.

4.1.4 Algorithm outline

At this stage we have designed a concept of resolving singularities in multi-level constrained hierarchies. The overall resolution method's outline is as follows:

- We state a multi-level hierarchy as in (4.1) with the first level as a trust region constraint due to our considerations from a Taylor model point of view in sec. 4.1.1. More detailed explanations, definitions and a trust region adaptation method are given in sec. 4.4.

- For every level of the hierarchy and in every control iteration we check if a task is about to become singular.
  - If this is not the case the GN algorithm is sufficient and the respective level is formulated as in (4.47).
  - If this is the case the GN algorithm is not sufficient and the respective level is formulated as Newton's method (4.49).

  The process of checking for singularities and the reason for switching back to the GN algorithm in regular configurations are explained in sec. 4.3.
- The hierarchical Newton's method requires the hierarchical Lagrangian Hessian (4.30). Its computation is detailed in the next section 4.2.
- Finally, each level's formulation (either the GN algorithm (4.47) or Newton's method (4.49)) is sent to a hierarchical least squares solver like LexLSI Dimitrov et al. [2015] or HQP Escande et al. [2014] and the whole problem is solved for the new control output $\boldsymbol{\Delta q}$.

4.2 Hessian calculation

In order to augment the GN algorithm to Newton's method in the case of singularities we require the knowledge of the Lagrangian Hessian (4.30) or (4.50) in the case of inequality constraints. While the LM algorithm approximates it by a simple weighted identity matrix we want to gain a better model representation by either providing the true analytic Hessian or an approximation of it by the SR1 method or the BFGS algorithm.

In the following we detail the calculation first for multi-level hierarchies with equalities in sec. 4.2.1. These results also hold for problems with a single level by setting the number of levels to $p = 1$. In sec. 4.2.2 necessary extensions are presented in order to incorporate inequality constraints.

4.2.1 Hierarchies with equality constraints

In the above section 4.1.2 we have formulated the Lagrangian function (4.26) and the corresponding first order optimality condition (4.27) for a multi-level equality constrained optimization problem. This resulted in the formulation for the Hessian of the Lagrangian (4.30). In the following, we give technical details for the calculation of the analytic Hessian and its approximation by the SR1 method or the BFGS algorithm.

4.2.1.1 Analytic Lagrangian Hessian (LexLSAug2AH)

For the analytic Hessian calculation $\hat{\boldsymbol{H}}_l$ of level l (4.30)

$$\hat{\boldsymbol{H}}_l = \sum_{i=1}^{l} \sum_{u=1}^{m_i} \lambda_{i,l}^{u} \boldsymbol{H}_{u,i} \tag{4.51}$$

we need to calculate the second order derivatives

$$\boldsymbol{H}_{u,i} = \nabla_{\boldsymbol{q}}^2 \boldsymbol{f}_{u,i}(\boldsymbol{q}) \quad \text{with } i = 1, \ldots, l \text{ and } u = 1, \ldots, m_i \tag{4.52}$$

of the geometric functions $\boldsymbol{f}_i(\boldsymbol{q}) \in \mathbb{R}^{m_i}$ for all levels $i = 1, \ldots, l$. For this we follow Erleben et Andrews [2017]. While the calculation of a single Hessian is of complexity $\mathcal{O}(n^3)$, the calculation of $\hat{\boldsymbol{H}}_l$ is of order $\mathcal{O}(n^3 \sum_{i=1}^{l} m_i)$

Numerical behaviour can be improved if Hessians $\boldsymbol{H}$ are only added to $\hat{\boldsymbol{H}}_l$ if their corresponding Lagrange multiplier $\lambda > \rho$ with $\rho \approx 1e^{-12}$ being a small numerical value.

Since the Hessian $\hat{\boldsymbol{H}}_l$ can become indefinite a simple Cholesky decomposition can not be applied. Instead we use the regularization proposed in Higham [1988] beforehand to get the closest semi positive definite matrix $\hat{\boldsymbol{H}}_l^{\text{reg}}$ by

$$\hat{\boldsymbol{H}}_l^{\text{reg}} = \frac{1}{2}(\hat{\boldsymbol{H}}_l + \boldsymbol{V}(\boldsymbol{\Sigma} + \iota \boldsymbol{I}^*)\boldsymbol{V}^T)\,. \tag{4.53}$$

The regularization requires a full polar decomposition for example by the SVD decomposition $\hat{\boldsymbol{H}}_l = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ which is computationally expensive with $\mathcal{O}(12n^3)$. An approximation of the polar decomposition based on Newton's method is available Higham [1986] but does not calculate the squared singular values $\boldsymbol{\Sigma}$ explicitly. A cheaper regularization based on the Bunch-Kaufman decomposition Bunch et al. [1976] could also be applied but only gives an approximation to the closest positive definite matrix. In some preliminary results however, we observed worse behaviour and therefore apply the Higham regularization.

$\hat{\boldsymbol{H}}_l^{\text{reg}}$ is the matrix $\hat{\boldsymbol{H}}_l$ with all the negative Eigenvalues being zero. We add a small term $\iota \boldsymbol{I}^*$ with $\iota \ll 1$ in order to make the Hessian $\hat{\boldsymbol{H}}_l^{\text{reg}}$ strictly positive definite in order to be able to apply the Cholesky decomposition. This improves numerical stability and smoothness of the solution. However, the convergence behaviour is influenced negatively. Indeed, the tendency to only converge to suboptimal minima increases with the magnitude of ι.

$\boldsymbol{I}_l^*$ is an identity matrix only occupying diagonal entries corresponding to joints on the kinematic chain of the tasks on all levels $\leq l$. Otherwise lower priority objectives acting in the task's null-space can not make use of these unnecessarily occupied joints. For visualization, a problem is given with two levels where each level has tasks with distinct kinematic chains. For level 1 ($m_1 = 4$) , all joints except joints 9, 10, 13 and 14 are on the tasks' kinematic chains. The corresponding Jacobian $\boldsymbol{J}_1$ with maximum rank $rank(\boldsymbol{J}_1) = 4$ is given below. $\times_i$ is the matrix entry corresponding to the i-th

column. Empty spaces are filled with zeros. Columns without any entries are marked in gray.

Level 1:

$$\boldsymbol{J}_1 = \begin{bmatrix} \times_1 & \times_2 & \times_3 & \times_4 & & & & & & & & \\ \times_1 & \times_2 & \times_3 & \times_4 & \times_5 & \times_6 & & & & & & \\ \times_1 & \times_2 & \times_3 & \times_4 & & & \times_7 & \times_8 & & & & \\ \times_1 & \times_2 & \times_3 & \times_4 & & & \times_7 & \times_8 & & \times_{11} & \times_{12} & \end{bmatrix}. \tag{4.54}$$

The corresponding dotted identity matrix $\boldsymbol{I}^*$ is then

$$\boldsymbol{I}_1^* = \begin{bmatrix} 1_1 & & & & & & & & & & \cdots & 0 \\ & 1_2 & & & & & & & & & & \vdots \\ & & 1_3 & & & & & & & & & \\ & & & 1_4 & & & & & & & & \\ & & & & 1_5 & & & & & & & \\ & & & & & 1_6 & & & & & & \\ & & & & & & 1_7 & & & & & \\ & & & & & & & 1_8 & & & & \\ & & & & & & & & 0_9 & & & \\ & & & & & & & & & 0_{10} & & \\ & & & & & & & & & & 1_{11} & \\ & & & & & & & & & & & 1_{12} \\ \vdots & & & & & & & & & & 0_{13} & \\ 0 & \cdots & & & & & & & & & & 0_{14} \end{bmatrix}. \tag{4.55}$$

In the case of rank deficiency of $\boldsymbol{J}_1$, augmentation of the level with the Cholesky decomposition $\boldsymbol{R}_1$ of the Hessian $\hat{\boldsymbol{H}}_1$ becomes necessary. With the dotted identity matrix $\boldsymbol{I}_1^*$, the rank of $\begin{bmatrix}\boldsymbol{J}_1^T & \boldsymbol{R}_1^T\end{bmatrix}^T$ is 10 and the size of the level 1 null-space is $size(\mathcal{N}_1) = 4$. If $\boldsymbol{I}_1^*$ was a full identity matrix its size would be 14 and the size of the level 1 null-space would be $size(\mathcal{N}_1) = 0$, leaving no variables (or joints) for the tracking of the tasks on the lower priority level 2:

Level 2:

$$\boldsymbol{J}_2 = \begin{bmatrix} \times_1 & & \times_5 & \times_6 & \times_7 & \times_8 & \times_9 & \times_{10} & \times_{11} & \times_{12} & \times_{13} & \times_{14} \end{bmatrix} \tag{4.56}$$

$$\boldsymbol{I}_2^* = \begin{bmatrix} 1_1 & & & & & & & & & & & \cdots & 0 \\ & 1_2 & & & & & & & & & & & \vdots \\ & & 1_3 & & & & & & & & & & \\ & & & 1_4 & & & & & & & & & \\ & & & & 1_5 & & & & & & & & \\ & & & & & 1_6 & & & & & & & \\ & & & & & & 1_7 & & & & & & \\ & & & & & & & 1_8 & & & & & \\ & & & & & & & & 1_9 & & & & \\ & & & & & & & & & 1_{10} & & & \\ & & & & & & & & & & 1_{11} & & \\ & & & & & & & & & & & 1_{12} & \\ & & & & & & & & & & & & 1_{13} \\ \vdots & & & & & & & & & & & & \\ 0 & \cdots & & & & & & & & & & & 1_{14} \end{bmatrix} \tag{4.57}$$

With the given dotted identity matrix $\boldsymbol{I}_1^*$ and the corresponding augmentation $\boldsymbol{R}_1$ on level 1, joints 9, 10, 13 and 14 can still be used for the fulfilment of the task given in $\boldsymbol{J}_2$.

Note that $\boldsymbol{I}_2^*$ includes the joints 2, 3 and 4 which are only on the kinematic chain of the level 1. However, they need to be incorporated due to the formulation of the hierarchical Hessian (4.30) which is the sum of the Hessians of both levels 1 and 2.

In appendix B we illustrate the effect on lower levels of such a dotted augmentation.

The overall structure of the algorithm, which we call *LexLSAug2AH* (**Lex**icographic **L**east-**S**quares **Aug**mentation with **2**nd order information from the **A**nalytic **H**essian, at times abbreviated to '...-AH'), is given in alg. 3 of appendix C.

4.2.1.2 Lagrangian Hessian built from BFGS or SR1 approximations (LexLSAug2SR1)

In the previous section 4.2.1.1 we have used the analytic second order derivatives of the geometric functions $\boldsymbol{f}(\boldsymbol{q}) \in \mathbb{R}^m$ in order to calculate the Hessian $\hat{\boldsymbol{H}}_l$. While computing $\nabla^2_{\boldsymbol{q}} \boldsymbol{f}(\boldsymbol{q})$ is somewhat expensive (the computational complexity is of order $\mathcal{O}(mn^3)$) we can calculate a cheaper approximation of order $\mathcal{O}(mn^2)$. The approximation can be obtained by the BFGS algorithm which was introduced independently by Broyden Broyden [1970], Fletcher Fletcher [1970], Goldfarb Goldfarb [1970] and Shanno Shanno [1970]:

$$\boldsymbol{B}_i^{(k)} = \boldsymbol{B}_i^{(k-1)} + \frac{\boldsymbol{y}_i \boldsymbol{y}_i^T}{\boldsymbol{y}_i^T \boldsymbol{\Delta q}^{(k-1)}} - \frac{\boldsymbol{B}_i^{(k-1)} \boldsymbol{\Delta q}^{(k-1)} \boldsymbol{\Delta q}^{(k-1),T} \boldsymbol{B}_i^{(k-1)}}{\boldsymbol{\Delta q}^{(k-1),T} \boldsymbol{B}_i^{(k-1)} \boldsymbol{\Delta q}^{(k-1)}} \quad \text{with } i = 1, \ldots, m\,. \tag{4.58}$$

We use it to approximate each component $i = 1, \ldots, m$ of the second order derivatives $\boldsymbol{B}_i^{(k)} \approx \nabla^2_{\boldsymbol{q}} \boldsymbol{f}_i(\boldsymbol{q})$ of the function $\boldsymbol{f}(\boldsymbol{q})$. This requires the knowledge of the last update $\boldsymbol{B}_i^{(k-1)}$, the solution from the previous control iteration $\boldsymbol{\Delta q}^{(k-1)}$ and the change of the gradient $\boldsymbol{y}_i$ between the current iteration k and the previous iteration $k-1$. For each component of $\boldsymbol{f}(\boldsymbol{q})$ we set it as the difference of the transposed corresponding row of

$\boldsymbol{J}^{(k)}$ and $\boldsymbol{J}^{(k-1)}$

$$\boldsymbol{y}_i = (\boldsymbol{J}_i^{(k)} - \boldsymbol{J}_i^{(k-1)})^T \quad \text{with } i = 1, \ldots, m\,. \tag{4.59}$$

The update $\boldsymbol{B}_i^{(k)}$ is positive definite for a given positive definite $\boldsymbol{B}_i^{(k-1)}$ and for positive curvature

$$\boldsymbol{y}_i^T \boldsymbol{\Delta q}^{(k-1)} > \xi\,. \tag{4.60}$$

ξ is a small numerical threshold like $\xi = 10^{-12}$. If the curvature condition is not met no update is done but rather the last update is kept $\boldsymbol{B}_i^{(k)} = \boldsymbol{B}_i^{(k-1)}$.

In the beginning, $\boldsymbol{B}_i^{(k-1)} = \mu \boldsymbol{I}^*$ is initialized with a (positive definite) identity matrix. Following the approach in Sugihara [2011] μ is set as

$$\mu = \max(\zeta, \frac{1}{2}\|\boldsymbol{e}\|_2^2)\,. \tag{4.61}$$

$\zeta = 10^{-3}$ is a lower threshold to handle cases when the norm of the error $\|\boldsymbol{e}\|_2^2$ is very small but the task is still required to assemble $\hat{\boldsymbol{H}}_l$.

For the composition of the Hessian of some level l $\hat{\boldsymbol{H}}_l$ (4.30)

$$\hat{\boldsymbol{H}}_l \approx \hat{\boldsymbol{B}}_l = \sum_{i=1}^{l} \sum_{u=1}^{m_i} \lambda_{i,l}^u \boldsymbol{B}_{u,i}^{(k)}. \tag{4.62}$$

we need to conduct $\sum_{i=1}^{l} m_l$ BFGS updates (4.58) in order to obtain the Hessian approximations $\boldsymbol{B}$ for all $\boldsymbol{f}_j(\boldsymbol{q})$, $j = 1, \ldots, l$. The overall complexity is then $\mathcal{O}(n^2 \sum_{i=1}^{l} m_i)$. If augmentations on several levels are necessary it is of course sufficient to calculate the respective Hessian approximations $\boldsymbol{B}^{(k)}$ only once if they are required for the assembly of $\hat{\boldsymbol{B}}$ on several levels.

The BFGS updates $\boldsymbol{B}^{(k)}$ are positive definite but due to the summation (with possibly negative Lagrange multipliers as weights) we again can end up with an indefinite matrix $\hat{\boldsymbol{B}}$ which needs to be regularized by the Higham regularization.

Some preliminary results for this BFGS method showed bad behaviour (i.e. unsmooth joint trajectories). The Symmetric Rank 1 (SR1) method Conn et al. [1988, 1991]; Khalfan et al. [1993]

$$\boldsymbol{B}_i^{(k)} = \boldsymbol{B}_i^{(k-1)} + \frac{(\boldsymbol{y}_i - \boldsymbol{B}_i^{(k-1)} \boldsymbol{\Delta q}^{(k-1)})(\boldsymbol{y}_i - \boldsymbol{B}_i^{(k-1)} \boldsymbol{\Delta q}^{(k-1)})^T}{(\boldsymbol{y}_i - \boldsymbol{B}_i^{(k-1)} \boldsymbol{\Delta q}^{(k-1)})^T \boldsymbol{\Delta q}^{(k-1)}} \quad \text{with } i = 1, \ldots, m\,. \tag{4.63}$$

is an alternative as it allows indefinite updates and therefore gives a good approximation to the indefinite analytic Hessians of $\boldsymbol{f}(\boldsymbol{q})$.

Figure 4.5 shows the joint velocities for test 1 of the evaluation section 4.5. Three different update methods for the Lagrangian Hessian are compared: the BFGS algorithm with the positive curvature condition $\boldsymbol{y}^T \boldsymbol{\Delta q}^{(k-1)} > \xi$ (upper graph) and with a modified curvature condition $|\boldsymbol{y}^T \boldsymbol{\Delta q}^{(k-1)}| > \xi$ (middle graph) and the SR1 method with $|(\boldsymbol{y} - \boldsymbol{B}^{(k-1)} \boldsymbol{\Delta q}^{(k-1)})^T \boldsymbol{\Delta q}^{(k-1)}| > \xi$ (lower graph). Despite the use of the trust

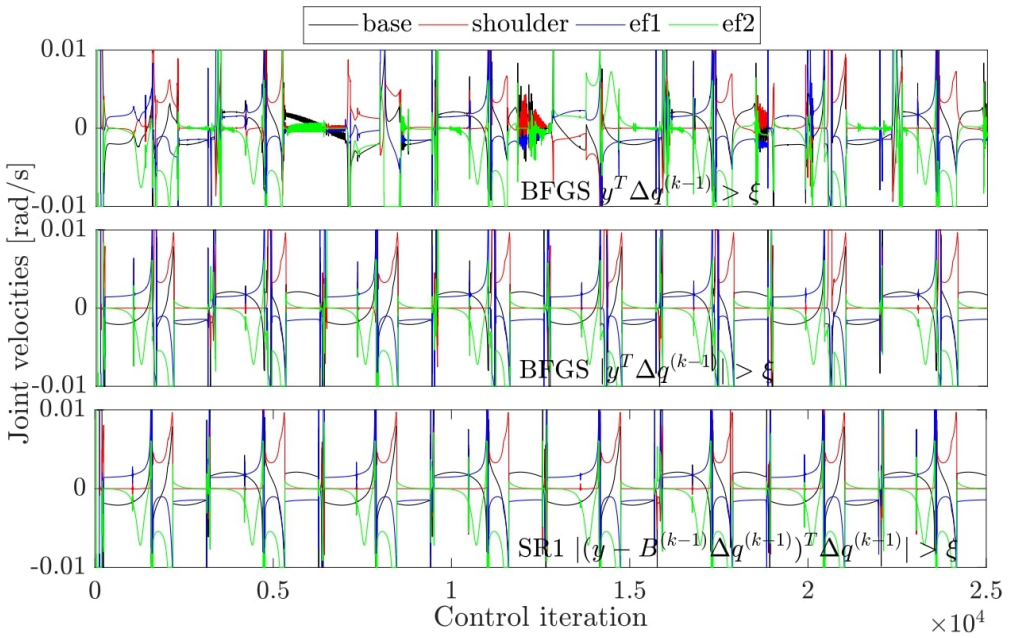


Fig. 4.5. *Joint velocities for test 1. The upper graph shows the numerically unstable behaviour for positive definite BFGS updates of the approximation of the second order derivates. This is in contrast to the indefinite BFGS (middle graph) and SR1 (bottom graph) updates which both result in numerically stable joint velocities.*

region adaptation method (see sec. 4.4) numerical instabilities of large magnitude can be observed for the positive definite BFGS updates (iterations 5000 - 7000, 12000 - 13000 and 18000 - 18500). This is in contrast to the modified BFGS algorithm and the SR1 method which both behave numerically stable despite their indefinite updates.

In light of these results, the Lagrangian Hessian in the algorithm *LexLSAug2SR1* (**Lex**icographic **L**east-**S**quares **Aug**mentation with **2**nd order information from the **S**ymmetric **R**ank **1**, at times abbreviated to '...-SR1') is built from SR1 approximations. Its pseudo-algorithm is given in Alg. 4 (see appendix C).

LexLSAug2SR1 is very similar to LexLSAug2AH in terms of the calculation of the Lagrangian Hessian. While its behaviour is slightly worse than that of LexLSAug2AH (especially in test 20 of the evaluation section 4.5 we observed bad conditioning of the updates which leads to bad numerical behaviour) it only provides little speed up which is negligible compared to the computational effort of the Higham regularization. Therefore we do not evaluate it in the next chapter 5 and prefer the usage of LexLSAug2AH.

4.2.1.3 Direct BFGS approximation of the Lagrangian Hessian (LexLSAug2BFGS)

Above we have seen that the analytic Hessian in sec. 4.2.1.1 (or its composition by SR1 approximations in sec. 4.2.1.2) can become negative definite which requires an

expensive regularization to a positive definite matrix. In the previous section 4.1.2 we have derived the gradient of the Lagrangian function of the constrained optimization problem on level l. This enables the direct and positive definite approximation of the Hessian of the Lagrangian function $\nabla^2_{\boldsymbol{q}}\mathcal{L}_l = \hat{\boldsymbol{H}}_l \approx \hat{\boldsymbol{B}}_l$ with the BFGS algorithm (4.58) by using the Lagrangian gradient $\nabla_{\boldsymbol{q}}\mathcal{L}_l$ of (4.27).

Only one BFGS update needs to be done per augmented level. This means that for the calculation of the Lagrangian Hessian $\hat{\boldsymbol{B}}_l$ on level l the computational effort is $\mathcal{O}(n^2)$ instead of $\mathcal{O}(n^3 \sum_{i=1}^{l} m_i)$ for LexLSAug2AH and $\mathcal{O}(n^2 \sum_{i=1}^{l} m_i)$ for LexLSAug2SR1, respectively.

We only use the Lagrange multipliers of the current iteration in the calculation of $\boldsymbol{y}_l$ Nocedal et Wright [2006]

$$\boldsymbol{y}_l = \nabla_{\boldsymbol{q}}\mathcal{L}_l^{(k)} - \nabla_{\boldsymbol{q}}\mathcal{L}_l^{(k-1)} = \boldsymbol{J}_l^{(k),T}\boldsymbol{\lambda}_{l,l}^{(k)} + \underline{\boldsymbol{J}}_{l-1}^{(k),T}\underline{\boldsymbol{\lambda}}_{l-1,l}^{(k)} - (\boldsymbol{J}_l^{(k-1),T}\boldsymbol{\lambda}_{l,l}^{(k-1)} + \underline{\boldsymbol{J}}_{l-1}^{(k-1),T}\underline{\boldsymbol{\lambda}}_{l-1,l}^{(k-1)}) \tag{4.64}$$

$$\rightarrow \boldsymbol{y}_l = (\boldsymbol{J}_l^{(k)} - \boldsymbol{J}_l^{(k-1)})^T\boldsymbol{\lambda}_{l,l}^{(k)} + (\underline{\boldsymbol{J}}_{l-1}^{(k)} - \underline{\boldsymbol{J}}_{l-1}^{(k-1)})^T\underline{\boldsymbol{\lambda}}_{l-1,l}^{(k)}\,. \tag{4.65}$$

$\hat{\boldsymbol{B}}_l^{(k-1)}$ of level l is initialized as

$$\hat{\boldsymbol{B}}_l^{(k-1)} = \sum_{i=1}^{l} \max(\zeta, \frac{1}{2}\|\boldsymbol{e}_i\|_2^2)\boldsymbol{I}_i^*\,. \tag{4.66}$$

For numerical stability and similarly to the analytic Hessian we conduct the Cholesky decomposition on a strictly positive definite matrix $\hat{\boldsymbol{B}}_l^{(k)} + \iota\boldsymbol{I}_l^*$ with a small value $\iota \ll 1$.

Alg. 5 (see appendix C) gives an overview of the algorithm of calculating the second order augmentation $\hat{\boldsymbol{B}}_l$ by the BFGS algorithm. We name this algorithm *LexLSAug2BFGS* (**Lex**icographical **L**east **S**quares **Aug**mented with **2**$^{\text{nd}}$ order information from the **BFGS** algorithm).

4.2.2 Hierarchies including inequality constraints

At every control iteration k, the optimal active-sets $\mathcal{A}_l^{(k+1)}$ of all levels $l = 1, \ldots, p$ need to be found. It contains the active inequality constraints in addition to all equality constraints. The active-set might differ between control iterations. This needs to be considered if a level i contains a singular task which requires second order augmentation $\boldsymbol{R}_i$. $\boldsymbol{R}_i$ is of full rank and therefore all the joints on the kinematic chains of the tasks are occupied. These joints can not be used any more for the resolution of tasks of lower priority $l > i$.

If a constraint on some level $c_{\mathcal{A}} < i$ ($c_{\mathcal{A}}$ is the level of highest priority from all the levels where an active-set change occurred) is now deactivated, corresponding joints do not need to be augmented any more with $\boldsymbol{R}_i$ (only in case that these joints are not part of the kinematic chains of other still active tasks on levels $l \leq i$). In the given example (4.54), (4.55) and (4.56), if the second task $\boldsymbol{J}_{2,1}$ of level 1 is deactivated, the joints 5 and 6 can now be used for the fulfilment of the level 2 task.

4.2.2.1 Analytic Lagrangian Hessian and Lagrangian Hessian built from BFGS or SR1 approximations (LexLSAug2AH and LexLSAug2SR1)

For both LexLSAug2AH and LexLSAug2SR1 the Lagrangian Hessian is composed as

$$\hat{\boldsymbol{H}}_l = \sum_{i=1}^{l} \sum_{j \in \mathcal{A}_i^{(k)}} \lambda_{i,l}^j \boldsymbol{H}_{j,i} \approx \hat{\boldsymbol{B}}_l = \sum_{i=1}^{l} \sum_{j \in \mathcal{A}_i^{(k)}} \lambda_{i,l}^j \hat{\boldsymbol{B}}_{j,i}^{(k)} \,. \tag{4.67}$$

It can be immediately seen that inactive constraints $j \notin \mathcal{A}_i^{(k)}$ are not added to the Hessian any more (additionally, $\boldsymbol{\lambda}_{l,l+1:p}^j = \boldsymbol{0}$ for inactive constraints). Therefore, no further steps are required.

4.2.2.2 Direct BFGS approximation of Lagrangian Hessian (LexLSAug2BFGS)

With the presence of inequalities, the BFGS gradient of level l is composed by (4.65)

$$\boldsymbol{y}_l = \sum_{i=1}^{l} \sum_{j \in \mathcal{A}_i^{(k)}} (\boldsymbol{J}_{j,i}^{(k)} - \boldsymbol{J}_{j,i}^{(k-1)})^T \boldsymbol{\lambda}_{i,l}^j \,. \tag{4.68}$$

The gradient itself only contains the gradient information of the current active-set $\mathcal{A}_i^{(k)}$. Consequently, only joints corresponding to tasks of the current active-set are occupied and no further adjustments are required.

However, the BFGS algorithm is of iterative nature such that $\hat{\boldsymbol{B}}_l^{(k)}$ is dependent of the Hessian approximation of the last control iteration $\hat{\boldsymbol{B}}_l^{(k-1)}$. $\hat{\boldsymbol{B}}_l^{(k-1)}$ still occupies joints on the kinematic chains of deactivated constraints. Therefore, a reinitialization of the form

$$\hat{\boldsymbol{B}}_l^{(k-1)} = \sum_{i=1}^{l} \sum_{j \in \mathcal{A}_i^{(k)}} \max(\underline{\mu}, \frac{1}{2} \|\boldsymbol{e}_{j,i}\|_2^2) \boldsymbol{I}_{j,i}^* \tag{4.69}$$

is necessary. Inactive constraints and their corresponding dotted identity matrices $\boldsymbol{I}^*$ are neglected in the summation. This needs to be done for all levels $l \geq c_{\mathcal{A}}$.

In our example (4.54), (4.55) and (4.56), $c_{\mathcal{A}} = 1$ and $p = 2$ so the dotted identity matrix becomes

$$\boldsymbol{I}_1^* = \begin{bmatrix} 1_1 & & & & & & & & \cdots & 0 \\ & 1_2 & & & & & & & & \vdots \\ & & 1_3 & & & & & & & \\ & & & 1_4 & & & & & & \\ & & & & \mathbf{0_5} & & & & & \\ & & & & & \mathbf{0_6} & & & & \\ & & & & & & 1_7 & & & \\ & & & & & & & 1_8 & & \\ & & & & & & & & 0_9 & \\ & & & & & & & & & 0_{10} \\ & & & & & & & & 1_{11} & \\ & & & & & & & & & 1_{12} \\ \vdots & & & & & & & & 0_{13} & \\ 0 & \cdots & & & & & & & & 0_{14} \end{bmatrix} \tag{4.70}$$

where the kinematic chain corresponding to the deactivated constraint $\boldsymbol{J}_{2,1}$ (rows and columns coloured in dark gray) is omitted.

Another possibility of reinitializing the BFGS algorithm is to keep the last update $\hat{\boldsymbol{B}}^{(k-1)}$ and only set rows and columns of joints to zero that are not occupied any more by the now inactive constraints. In the case of constraint activation we put a small value on the diagonal of now occupied joints by newly activated constraints. Especially with the occurrence of several on and off switches of the same constraint over several control iterations it seems unreasonable to discard the so far gained second order information. In some other cases however it might be more favourable to fully restart the BFGS algorithm in order to avoid second order artefacts of some inactivated constraints which might decelerate convergence of the current active-set. However, in our simulations we show that the BFGS algorithm possesses quick recovery capabilities which allows second order artefacts to vanish over the course of a few control iterations.

4.3 Switching strategy between GN-algorithm and Newton's method

The analytic expression (LexLSAug2AH) or its approximation by the SR1 method (LexLSAug2SR1) of $\hat{\boldsymbol{H}}_l = \sum_{i=1}^{l} \sum_{u=1}^{m_i} \lambda_{i,l}^u \boldsymbol{H}_{u,i} \approx \hat{\boldsymbol{B}}_l = \sum_{i=1}^{l} \sum_{u=1}^{m_i} \lambda_{i,l}^u \boldsymbol{B}_{u,i}^{(k)}$ becomes nil inherently for a combination of either $\boldsymbol{\lambda} = \mathbf{0}$ or $\boldsymbol{H} = \boldsymbol{B}^{(k)} = \mathbf{0}$. This is not the case for the direct BFGS approximation of the Lagrangian Hessian $\hat{\boldsymbol{B}}_l^{(k)}$ (LexLSAug2BFGS) but has to be explicitly enforced. Similarly to the inequality case, the augmentation unnecessarily occupies joints which then are missing in the fulfilment of lower priority tasks.

The second order Taylor approximation of level l of the current iteration is given

by

$$\Phi_l(\boldsymbol{q} = \boldsymbol{q}^{(k-1)} + \boldsymbol{\Delta q}^{(k-1)}) = \frac{1}{2}\boldsymbol{e}_l^T(\boldsymbol{q})\boldsymbol{e}_l(\boldsymbol{q}) \tag{4.71}$$

$$= \Phi_l(\boldsymbol{q}^{(k-1)}) - \boldsymbol{\Delta q}^{(k-1),T}\boldsymbol{J}_l^{(k-1),T}\boldsymbol{e}_l(\boldsymbol{q}^{(k-1)}) \tag{4.72}$$

$$+ \frac{1}{2}\boldsymbol{\Delta q}^{(k-1),T}(\boldsymbol{J}_l^{(k-1),T}\boldsymbol{J}_l^{(k-1)} + \hat{\boldsymbol{H}}_l^{(k-1)})\boldsymbol{\Delta q}^{(k-1)} + \mathcal{O}_l^{(k)} \tag{4.73}$$

$$= \frac{1}{2}\left\| \begin{bmatrix} \boldsymbol{J}_l^{(k-1)} \\ \boldsymbol{R}_l{}^{(k-1)} \end{bmatrix} \boldsymbol{\Delta q}^{(k-1)} - \begin{bmatrix} \boldsymbol{e}_l(\boldsymbol{q}^{(k-1)}) \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 + \mathcal{O}_l^{(k)} = \frac{1}{2}\left\| \begin{bmatrix} \boldsymbol{w}_l^{J,(k)} \\ \boldsymbol{w}_l^{R,(k)} \end{bmatrix} \right\|_2^2 + \mathcal{O}_l^{(k)} \tag{4.74}$$

which enables us to solve for the model error

$$\mathcal{O}_l^{(k)} = \frac{1}{2}\boldsymbol{e}_l^T(\boldsymbol{q})\boldsymbol{e}_l(\boldsymbol{q}) - \frac{1}{2}\left\| \begin{bmatrix} \boldsymbol{w}_l^{J,(k)} \\ \boldsymbol{w}_l^{R,(k)} \end{bmatrix} \right\|_2^2 . \tag{4.75}$$

The Taylor series of our non-linear model function is limited to second order so the approximation error is at least of order $\mathcal{O}(\boldsymbol{\Delta q}^3)$. However, due to either omitting (GN algorithm) or approximating (Newton's method) the second order information the error is dominated by $\mathcal{O}(\boldsymbol{\Delta q}^2)$.

The value of $\mathcal{O}^{(k)}$ can be both negative and positive and therefore poses difficult to threshold. Instead we propose to observe the sign-free quadratic norm residual $\boldsymbol{w}^J$ of the current control iteration in order to decide whether the use of only the GN algorithm is appropriate

$$\epsilon_l = \frac{1}{2}\|\boldsymbol{w}_l^J\|_2^2 = \frac{1}{2}\|\boldsymbol{J}_l\boldsymbol{\Delta q} - \boldsymbol{e}_l\|_2^2 . \tag{4.76}$$

In the presence of several tasks (for example one end-effector task for the left hand and one for the right hand) on the same level l, the value $\boldsymbol{w}_l^R$ of each respective task is not available. The reason is that on a single level there is only one Hessian matrix $\hat{\boldsymbol{H}}_l$ with corresponding slack $\boldsymbol{w}_l^R$. Therefore, $\boldsymbol{w}_l^R$ is neglected during the calculation of each respective task's ϵ.

ϵ addresses whether we can find a step $\boldsymbol{\Delta q}$ within the scope of our model such that all the constraints $\boldsymbol{J\Delta q} = \boldsymbol{e}$ are fulfilled. The problem is then feasible with a (numerical) zero slack $\boldsymbol{w}^J \approx \boldsymbol{0}$. The measure can be seen as an indicator of numerical accuracy during the least-squares solving.

The GN algorithm is used whenever ϵ is equal or smaller than a certain numerical threshold ν (typically 10^{-12}). Newton's method is used whenever ϵ is larger:

$$\epsilon \leq \nu \qquad \rightarrow \text{GN algorithm} \tag{4.77}$$

$$\epsilon > \nu \qquad \rightarrow \text{Newton's method} . \tag{4.78}$$

Despite our considerations in the beginning of this section, we also use the switching method for the two methods LexLSAug2AH and LexLSAug2SR1. The reason is

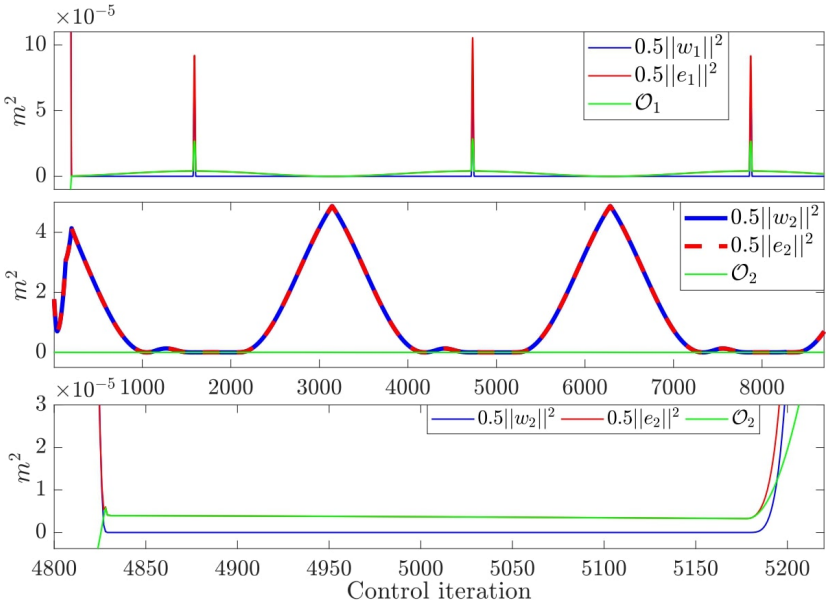


Fig. 4.6. *Test 1. The upper graph shows the squared norms of the slack $\boldsymbol{w}_1$ and the task error $\boldsymbol{e}_1$ and the model error $\mathcal{O}_1$ for level 1. The two lower graphs show the values for level 2 with the bottom graph showing a close-up of level 2 being feasible.*

that due to numerical inaccuracies $\hat{\boldsymbol{H}}_l$ becomes only approximately zero and still unnecessarily occupies joints which are then missing in the fulfilment of lower priority tasks.

For LexLSAug2BFGS in the case of a switch from the GN algorithm to Newton's method on level $c_{\mathrm{GN}\to\mathrm{N}}$, all augmentations from level $c_{\mathrm{GN}\to\mathrm{N}}$ to the last level p are reset. Details with regards to the BFGS reinitialization are discussed in sec. 4.2.2.2.

Behaviour of the different values

The switching method (4.78) is facilitated by the fact that ϵ can easily become numerical zero whenever the least squares problem is feasible. This is in contrast to both $\mathcal{O}$ and $\boldsymbol{e}$ (which also could be considered as a decision variable in the switching method) which hardly become numerical zero which complicates the choice of a threshold. The different values for test 1 of the validation section 4.5 are given in fig. 4.6.

The upper graph shows the values for level 1 while the two lower graphs show the values for level 2 with a close-up in the bottom graph. For level 1 and during iteration 4850 to 5150 also for level 2 (see bottom graph for a close-up), ϵ is numerically zero which triggers the use of the GN algorithm. The other two values $\mathcal{O}$ and $\boldsymbol{e}$ are of magnitude 10^{-5} m^2. The level 1 end-effector task becomes infeasible if the trust region constraint forbids fast enough motions to follow the moving target. This is the case as seen from the three spikes of all three values in the upper graph which requires a switch to Newton's method since $\epsilon_1 > \nu$. The level 2 end-effector task is in conflict with the level 1 task at most times but switches back to the GN algorithm approximately at

iteration 1400, 4850 and 7800 with $\epsilon_2 \leq \nu$.

The trust region radius is chosen as $\Delta = 0.01$ rad for revolute or $\Delta = 0.01$ m for translational joints (see next section 4.4) and consequently $\mathcal{O}(\boldsymbol{\Delta q}) \sim 10^{-4}$ as expected (can be seen especially in the middle graph with $\mathcal{O}_2 \ll 1$ at all times).

4.4 Choosing the Trust Region Radius

The second order Taylor series (4.9) represents the original non-linear function (4.7) only with a given model error which is dependent on the step $\boldsymbol{\Delta q}$. Therefore, the step length $\|\boldsymbol{\Delta q}\|_\infty$ needs to be bounded within a small neighbourhood Δ of the current state $\boldsymbol{q}$ in order to keep the model error small. This neighbourhood is commonly referred to as the trust region with trust region radius Δ. It can be enforced with a constraint of the form

$$\|\boldsymbol{\Delta q}\|_\infty < \Delta \,. \tag{4.79}$$

The trust region constraint needs to be put on the very first level of the hierarchy as $-\underline{\boldsymbol{\Delta}} \leq \boldsymbol{\Delta q} \leq \overline{\boldsymbol{\Delta}}$. $\underline{\boldsymbol{\Delta}}$ and $\overline{\boldsymbol{\Delta}}$ are vectors with n entries where all values are set to Δ.

Choosing the trust region Δ such that the Taylor approximation is well defined requires some attention. Self-tuning methods from unconstrained optimization (for example observing the error reduction behaviour ρ, see more in appendix A) are difficult to transfer to the constrained robot control case:

- It is unclear how to introduce the notion of hierarchy. With good error reduction on high priority levels but bad error reduction on some lower priority levels, to which extent do we interfere with the high priority levels' performance to achieve good performance on the lower priority levels? One approach for a constrained optimization problem was proposed in Fletcher et Leyffer [2002] in the context of a trust region sequential quadratic programming (SQP) solver (see appendix A).
- In SQP, if the trust region is deemed too large it is reduced and the current step is recomputed. This is not an approach we can afford in control due to computation-time constraints (while for now we have $\Delta t = 1\,\mathrm{s}$, in the next chapter 5 we adapt our control time step to $\Delta t \ll 1\mathrm{s}$).
- With redundant robots it seems unnecessary to reduce or increase the trust region radius of all the joints if one of the tasks with a limited kinematic chain yields a bad error reduction.

This argumentation calls for a different approach in designing a trust region adaptation method for Δ. Our idea is to ignore both the hierarchical and kinematic structure of the problem and directly observe the solution variable $\boldsymbol{\Delta q}$.

- Loop through all variables $i = 1, \ldots, n$ of the solution $\boldsymbol{\Delta q}^{(k)}$ and check if a single entry $\boldsymbol{\Delta q}_i^{(k)}$ changed its sign compared to the previous solution $\boldsymbol{\Delta q}_i^{(k-1)}$.
- If so, $\boldsymbol{\eta}_i = \min(\overline{\eta}, \rho_{\mathrm{dec}}^{\boldsymbol{\alpha}_i} \cdot \boldsymbol{\eta}_i)$, and increase $\boldsymbol{\alpha}_i$ by 1.

- If not, $\boldsymbol{\eta}_i = \max(1, 1/\rho_{\text{inc}} \cdot \boldsymbol{\eta}_i)$, and decrease $\boldsymbol{\alpha}_i$ by 1.
- Change the trust region radius for joint i by $\underline{\boldsymbol{\Delta}}_i = -\Delta/\boldsymbol{\eta}_i$ and $\overline{\boldsymbol{\Delta}}_i = \Delta/\boldsymbol{\eta}_i$.

$\boldsymbol{\eta}$ and $\boldsymbol{\alpha}$ are vectors with n entries. $\boldsymbol{\eta}$ and $\boldsymbol{\alpha}$ are initialized with 1's. $\overline{\eta}$ is an upper threshold with for example $\overline{\eta} = 10^6$ which prevents that the trust region radius becomes too small. $\rho_{\text{dec}} = \rho_{\text{inc}} = 1.2$ are the trust region decrease or increase ratios. These values are determined in simulation such that occurring numerical instabilities are effectively suppressed. At the same time quick restoration of the original trust region radius needs to be ensured when the robot motion is numerically stable.

The original trust region radius Δ needs to be chosen in such a way that the model behaves well in a wide array of situations. This might require some tuning by starting with a high value (possibly desired since it corresponds to the joint velocity limits) which is successively reduced until satisfactory (i.e. numerically stable and smooth results especially in singular configurations) results are obtained. From our experience, this value depends on the time step, the task gains and the kinematic structure itself. In this work, we set $\Delta = 0.01$ rad for revolute or $\Delta = 0.01$ m for translational joints.

In addition to the trust region adaptation method we introduce a further adjustment that operates directly on the BFGS Lagrangian Hessian approximation $\hat{\boldsymbol{B}}$ of LexLSAug2BFGS. For this, we observe the joint accelerations. Once we have several (usually two) consecutive sign changes in the joint accelerations we add a small value (ζ) on the diagonal of $\hat{\boldsymbol{B}}^{(k-1)}$ corresponding to the joint where the event occurred. In case that consecutive sign changes in the acceleration $\ddot{q}_j$ of joint j are observed, $\hat{\boldsymbol{B}}^{(k-1)}$ is modified to $\hat{\boldsymbol{B}}^{(k-1)}(j,j) = \hat{\boldsymbol{B}}^{(k-1)}(j,j) + \zeta$. After this the BFGS update (4.58) for $\hat{\boldsymbol{B}}^{(k)}$ is conducted. This way we add additional damping to decelerate this specific joint and avoid potential numerical instability.

4.5 Validation

We assess our methods LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS (which we refer to as 'LexLSAug2 methods' at times) on a simulation test bench[1]. The GN algorithm (4.47) as well as Newton's method (4.49) are solved with the hierarchical least-squares solver LexLSI Dimitrov et al. [2015]). We use two different robots and compare with three solvers that are able to solve multi-level constrained control problems:

1. The hierarchical GN-algorithm.
2. The hierarchical inverse kinematics solver with adaptive damping (ADLS) described in Chiaverini [1997]. It allows any number of levels with equality constraints. For a single level, it corresponds to the LM-algorithm. The maximum damping is set to 2 while the threshold for the minimum singular value is 0.5.

[1]The accompanying video can be found at https://youtu.be/XJzkVOHLIvw

Hierarchy **A** (T1 and T3 to T19) and **B** (T2)

0. 4 trust region limits $\underline{\boldsymbol{\Delta}} \leq \boldsymbol{I}\boldsymbol{\Delta q} \leq \overline{\boldsymbol{\Delta}}$ (GN, LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS) or 4 velocity limits $\underline{\boldsymbol{\Delta}} \leq \boldsymbol{I}\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{\Delta}}$ (DHQP)

0.-1. 2 inequality constraints on shoulder

1. 2 equality constraints on blue end-effector
2. 2 equality constraints on green end-effector
3. Minimal norm solution $\dot{\boldsymbol{q}} = \boldsymbol{\Delta q} = \boldsymbol{0}$

Fig. 4.7. *Hierarchy A (**T1** and **T3** to **T19**) and B (**T2**)*

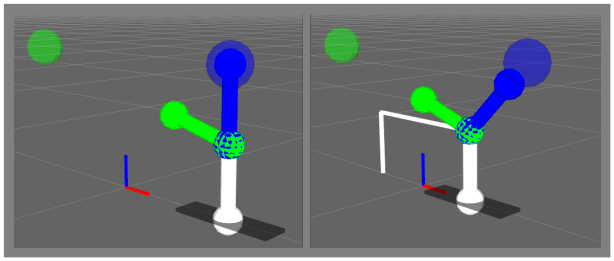


Fig. 4.8. *Robot for **T1** and **T3** to **T19** (left) and **T2** (right) (LexLSAug2BFGS screenshot). The robot can translate along the long axis of the black slider. The other joints are revolute. The green end-effector is tracking the green target while the blue end-effector is tracking the blue target. In both frames, the targets are at their turning points $[2, 2]$ m (blue ball) and $[-2, 2]$ m (green ball). The robot is able to just reach the blue target for T1 while for T2 both targets remain unreachable. This is due to the position of the shoulder being bound inside the white box.*

3. The damped hierarchical quadratic programming (DHQP) described in Herzog et al. [2016]. It handles both equality and inequality constraints. The damping is set to 2.

4.5.1 Test bench

The test bench consists of 20 different test cases with a limited length of 25000 control iterations. The first 19 test cases **T1** to **T19** are performed with a 2-D 4 DoF robot with two end-effectors and a uniform link length of 1 m (see fig. 4.8). A translational DoF at its base allows the robot to move along the $\boldsymbol{x}$ axis of the world reference frame. Furthermore, it possesses three revolute joints, one at the base and two at its 'shoulder' enabling the two end-effectors to rotate freely. The task hierarchy is given in fig. 4.7.

The joint velocity constraint is omitted for ADLS since inequalities can not be handled with this solver. Test T2 adds an additional inequality constraint on the 'shoulder' between level 0 and 1 of hierarchy A.

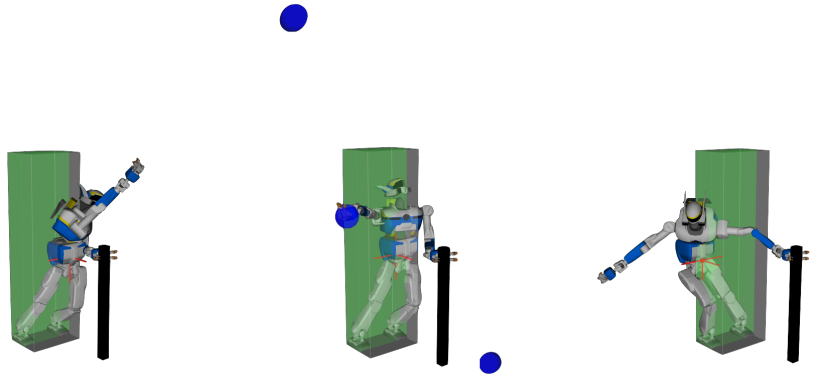


Fig. 4.9. T20 *screenshots for LexLSAug2BFGS, from left to right: HRP2 reaching for the target on its top left (out of reach, both CoM tasks are active), then the target is in reach (the left hand task is switched to the GN-algorithm) and finally the target is out of reach again in HRP2's far right bottom. The robot tries to stay as long as possible in the smaller (green) CoM bounding box on the last level as it is tracking the target (made possible by switching to the GN-algorithm when the target is in reach, and which can not be observed for DHQP).*

The initial robot configuration is $[0, 0]$ m for the free flyer and $[-\pi/2, 0, 0]$ rad for the revolute joints (this corresponds to a fully stretched robot with both end-effectors at $[0, 2]$ m). The end-effectors at the tip of both arms then must follow two targets oscillating diagonally with amplitude $[2, 2]$ m starting from $[0, 0]$ m. For **T1**, the kinematic structure of the robot (stretched length of 2 m) allows to just reach the targets in the corner cases. For **T3**, the targets oscillate with amplitude $[2, 3]$ m which renders the targets to be unreachable in the corner cases. For **T2** an additional inequality constraint (hierarchy **B**) is imposed on the position of the shoulder while the targets move like in T1. The constraint box reaches from $[-1, 0]$ m to $[1, 1]$ m. A schematic drawing of the tests is given in fig. 4.10. The robot links are given in grey in the generic posture 0 m for the base and $[\pi/2, \pi/4, -\pi/2]$ rad for the revolute joints.

T4 to **T8** are characterized by static targets for level 1 and 2 at $[0, y]$ m and $[x, 1]$ m with (x, y) equal to $(1, 2)$, $(1+\epsilon, 2+\epsilon)$, $(1-\epsilon, 2-\epsilon)$, $(1+10, 2+10)$ and $(1-0.25, 2-0.25)$. The three first cases are intended to test the robustness of the switching method by slight variations with $\epsilon = 0.001$ m ($1/1000$ of the link length) so both targets are either just out of reach or just in reach. The last two cases test the behaviour in case that the targets are either well out of reach or fully reachable. A schematic drawing of the tests is given in fig. 4.11.

T9 shows that the second-order approximation can deal with highly noisy targets. For this, both targets change their position randomly within $[\pm 2, \pm 1000]$ m at each of the first 12500 iterations, and then are static at $[-1.9, -422]$ m and $[-1.5, 299]$ m respectively for the rest of the test.

T10 to T19 are intended to show that our method is capable of simultaneously tracking static and dynamic targets with interchanged priority and robustness in the switching method.

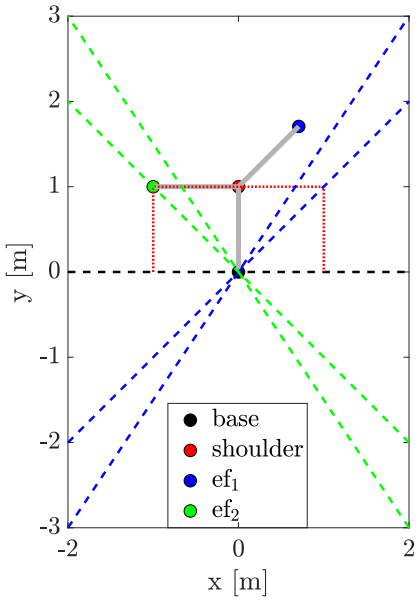


Fig. 4.10. *The schematics of* ***T1*** *to* ***T3****. The robot links are given in grey. The base is kinematically restricted to the black line. The shoulder is bounded within the dotted red box (T2 only) while the end-effectors 1 and 2 try to follow at best the blue and the green dashed lines, respectively.*

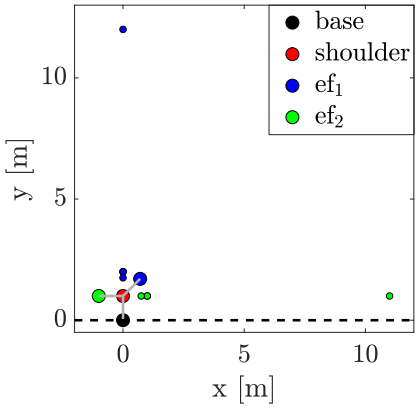


Fig. 4.11. *The schematics of* ***T4*** *to* ***T8****. The robot links are given in grey. The base is kinematically restricted to the black line. The shoulder is unbounded and can move freely. The end-effectors 1 and 2 try to reach at best the blue and the green dots, respectively.*

For **T10** to **T14**, the targets for the first and the second end-effector are at $[t, y]$ m and $[-2, 2]$ m respectively. t increases linearly by 0.001 m with the number of iterations while y has the same value as for T4 to T8. A schematic drawing of the tests is given in fig. 4.12.

For **T15** to **T19**, the first target is at $[0, y]$ m, with y having the same value as for T4 to T8 while the second target oscillates as in T1. A schematic drawing of the tests is given in fig. 4.13.

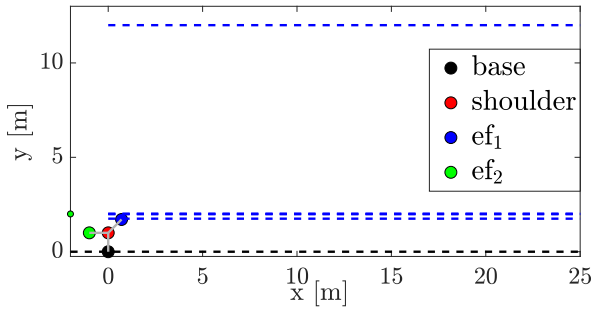


Fig. 4.12. *The schematics of* ***T10*** *to* ***T14****. The robot links are given in grey. The base is kinematically restricted to the black line. The shoulder is unbounded and can move freely. The end-effector 1 tries to follow at best a target moving on one of the blue lines while the end-effector 2 tries to reach the green dot.*

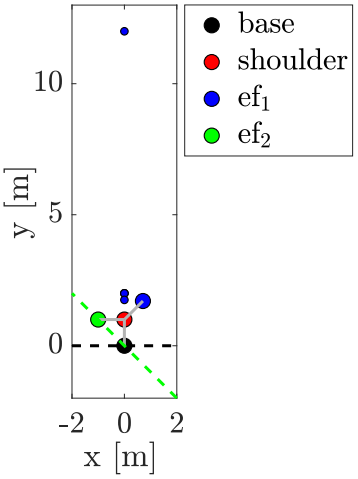


Fig. 4.13. *The schematics of* ***T15*** *to* ***T19****. The robot links are given in grey. The base is kinematically restricted to the black line. The shoulder is unbounded and can move freely. The end-effector 1 tries to reach one of the blue dots while the end-effector 2 tries to follow a target moving on the the green line.*

T20 shows that our LexLSAug2 methods are suitable for robots with any number of DoF. In the experiment, the robot HRP-2Kai with 38 DoF is standing in a cluttered aircraft assembly environment with both feet on the ground and the left hand grabbing a pole in front of the robot. Those contacts are fixed and do not change throughout the course of the simulation. On the next priority level, the center of mass (CoM) projection is bound within a rectangle exceeding the support polygon spanned by the feet a little bit to the front ($[-0.1, -0.275, -]$ m to $[0.3, 0.275, -]$ m, the $\boldsymbol{z}$ direction is unbounded). The right hand on the next hierarchy level then tracks a target swinging from the robot's far top left $[2, 1, 3]$ m to its far right bottom $[-1, -2, 0]$ m. Finally, we constrain the CoM to a smaller polygon corresponding to the feet only ($[-0.1, -0.275, -]$ m to $[0.215, 0.275, -]$ m). The overall hierarchy is given in fig. 4.14.

Hierarchy **C** for T20 with HRP-2Kai

0. 38 trust region limits $\underline{\boldsymbol{\Delta}} \leq \boldsymbol{I}\boldsymbol{\Delta q} \leq \overline{\boldsymbol{\Delta}}$ (GN, LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS) or 38 velocity limits $\underline{\boldsymbol{\Delta}} \leq \boldsymbol{I}\dot{\boldsymbol{q}} \leq \overline{\boldsymbol{\Delta}}$ (DHQP)
1. 38 joint limits
2. 18 in-reach equality constraints (translation, rotation) on left, right foot and left hand
3. 3 inequality constraints on the CoM
4. 3 in/out-of-reach equality constraints (translation, oscillating target in 3-D space) on right hand
5. 3 stricter inequality constraints on the CoM
6. Minimal norm solution $\dot{\boldsymbol{q}} = \boldsymbol{\Delta q} = \boldsymbol{0}$

Fig. 4.14. *Hierarchy C for* ***T20*** *with HRP-2Kai.*

4.5.2 Evaluation criteria

The main problem that arises near a singularity are high joint velocities (possibly to the extent of total solver failure with numerically high values). However, with damping or a trust region constraint present, numerical instabilities occur in the form of high frequency oscillations. We measure the presence of oscillations by tracking the sign changes of each component of the solution vector $\boldsymbol{\Delta q}$ between iterations. The sum of the amplitude of these changes is our evaluation criteria Σ

$$\Sigma = \sum_{k=2}^{25000} \sum_{i=1}^{d} \begin{cases} |\boldsymbol{\Delta q}_i^{(k)}| & \text{if } \operatorname{sgn}(\boldsymbol{\Delta q}_i^{(k)}) \neq \operatorname{sgn}(\boldsymbol{\Delta q}_i^{(k-1)}) \\ 0 & \text{otherwise} \end{cases}$$

A low Σ means few or no oscillations.

Additionally, the normed error of the equality with highest priority (level 1 blue end-effector task for hierarchy A and B, right hand task for hierarchy C) is integrated over all control iterations and then normalized by the smallest value of all solvers for this test case (Ξ). Since bad convergence of a task with high priority can lead to better convergence of a task of lower priority, the performance of lower priority tasks is not considered due to lack of relevance. ADLS is able to converge faster from the initial position to the targets since the joint velocity constraint is omitted. Therefore, its performance is not considered in the normalization of Ξ (corresponding values are coloured in grey). For static test cases, we deem the robot to be converged if every entry of the solution vector becomes smaller than 10^{-6} rad or m. The corresponding iteration is given as Ψ.

	T1	T2	T3	T4	T5
...-AH	0.15 / 1.	**0** / **1**	0.007 / **1.**	**0** / 1.-171	**0** / **1**-160
...-SR1	0.14 / 1.	0. / 1.	**0.005** / 1.	**0** / 1.-172	**0** / 1.-164
...-BFGS	0.3 / **1**	0.06 / 1.1	0.3 / 1.	**0** / 1.-201	**0** / 1.-169
DHQP	**0** / 2.4	0.3 / 1.	284 / 1.1	**0** / 1.3-15k	**0** / 1.4-17k
ADLS	**0** / 0.5	-	$4 \cdot 10^6$ / 10	**0** / 0.3-15k	**0** / 0.5-17k
GN	574 / 1.	155 / 1.2	547 / 1	**0** / **1**-169	10^3 / 1.2
	T6	**T7**	**T8**	**T9**	**T10**
...-AH	**0** / 1.-**163**	**0** / **1**-160	**0** / 1.-**204**	**0.01** / **1**-13k	0.015 / 1.
...-SR1	**0** / 1.-**163**	**0** / **1.04**-166	**0** / 1.-**204**	**0.01** / **1**-13k	**0.005** / 1.
...-BFGS	**0** / **1**-171	**0** / 1.-162	**0** / **1**-210	0.1 / **1**-13k	0.02 / **1**
DHQP	**0** / 1.3-10k	828 / 21	**0** / 1.1-280	470 / 10	808 / 4.8
ADLS	**0** / 0.3-10k	10^5 / 30	**0** / 0.1-29	$3 \cdot 10^3$ / 1.1	10^5 / 600
GN	**0** / **1**-163	10^3 / 21	**0** / **1**-205	551 / 10	912 / 1.02
	T11	**T12**	**T13**	**T14**	**T15**
...-AH	**0** / 1.	**0** / 1.	**0** / 1.	**0** / 1.	**0.010** / 1.
...-SR1	**0** / 1.	0. / 1.	**0** / 1.	**0** / 1.	0.011 / 1.
...-BFGS	**0** / **1**	$\mathbf{10^{-6}}$ / **1**	**0** / **1**	**0** / **1**	0.03 / **1**
DHQP	809 / 4.6	806 / 4.8	743 / 1.	693 / 2.1	7.4 / 1.4
ADLS	10^5 / 600	10^5 / 600	10^5 / 1.4	10^5 / 600	4.6 / 0.4
GN	821 / 1.01	955 / 1	966 / 1.	849 / 1	775 / 1.
	T16	**T17**	**T18**	**T19**	**T20**
...-AH	**0** / 1.	0.3 / 1.	**0** / 1.	0.002 / 1.	**0.003** / **1**
...-SR1	**0** / 1.	0.24 / 1.	**0** / 1.	0.002 / 1.	0.05 / **1**
...-BFGS	**0** / **1**	**0.1** / **1**	**0** / **1**	0.004 / **1**	0.7 / **1**
DHQP	12 / 1.3	1.9 / 1.3	605 / 1.	**0** / 1.	0.2 / 1.1
ADLS	6.5 / 0.5	1.9 / 0.3	10^5 / 1.4	**0** / 0.1	-
GN	696 / 1.	890 / 1.	971 / 1.	516 / 1	2000 / 1.

Table 4.1 – Performance results for the tests T1 to T20. Table entries are of following form: Σ / Ξ - Ψ (Ψ only for static test cases and when smaller than 25000); Red coloured entries mean that the test case was numerically unstable (high Σ). Blue coloured entries mean that the corresponding solver performed best in this category for the corresponding test case. The ADLS values for Σ compared to the others solvers are much higher for numerically unstable tests since the solution is not bound by a velocity constraint.

4.5.3 Evaluation

Table 4.1 shows the evaluation results of all test cases. Our LexLSAug2 methods are numerically stable (low Σ) and perform best in terms of convergence ($\Xi \approx 1$). DHQP and ADLS are numerically unstable in many test cases. This is due to the fact that the damping values found for T1, T2 and T20 enabling numerically stable behaviour and a fair comparison for the convergence Ξ are also applied to all the other tests. Especially for the tests T10 to T15 these damping values are not sufficient but could be tuned in such a way that $\Sigma \approx 0$ rad (possibly requiring very high damping terms leading to slow convergence behaviour).

The results of the tests show the following characteristics of our LexLSAug2 methods:

- Numerical stability in the presence of singularities

  - when the target is far away (T7, T9, T10, T11, T12, T13, T14, T18)
  - when the target is at the border of reachability (T1, T4, T5, T6, T10, T11, T12, T15, T16, T17)

- Fast and good convergence properties
- Numerically stable for all test cases without the need for damping tuning.
- Works for inequalities (T2, T20)
- Scalable, applicable for any number of DoF (T20)
- Strong self-regulating capabilities (T9)
- The hierarchy is not disturbed as it is the case with constant damping (i.e. the damping term does not become zero in non-singular robot configurations) as it is the case for DHQP. It introduces a model discontinuity when the rank of a matrix changes and the damping is not applied in the same null-space (T15, T16) (a damped pseudo-inverse $\begin{bmatrix}\boldsymbol{J}^T & \boldsymbol{I}^T\end{bmatrix}^{T+}$ is applied for the solution of each level but the null-space bases are computed from a SVD of just $\boldsymbol{J}$). While it violates the notion of strict hierarchies it discards the need for a switching strategy. This is because the null-space is always of maximum dimension and does not occupy joints which are then missing in the fulfilment of lower priority tasks (for example if we disable the switching method and constantly apply Newton's method).

The joint velocities for **T1** are given in fig. 4.16 (the joint velocities for LexLSAug2SR1 are given in fig. 4.5, see sec. 4.2.1.2). For LexLSAug2BFGS, slight numerical instabilities of low amplitude $< 10^{-3}$ rad can be observed on the shoulder joint for the level 2 end-effector. The joint velocities for the LexLSAug2 methods change quickly in order to track the targets optimally. This is in contrast to DHQP and ADLS with slow changes of joint velocities. This is behaviour typically seen for the LM algorithm. Consequently, better error norm reduction is achieved for the LexLSAug2 methods as seen in fig. 4.16. The level 1 targets are tracked very well due to the switching to the GN-algorithm by removing the damping characteristic of the second order information (recall that a small damping term is applied in order to render the Hessians strictly positive definite). ADLS shows the same behaviour when its damping term is set to zero by the adaptation method. At times, DHQP and ADLS have lower task error norms for the second end-effector. This is caused by the worse convergence of level 1, enabling the level 2 end-effector to move closer to its prescribed position. Generally, DHQP shows the highest error norm due to the constantly present damping.

The GN-algorithm is numerically unstable with a high $\Sigma = 574$[rad]. It shows that the GN-algorithm alone is not a safe method around singularities.

The adapted trust region radius (symmetric with $\boldsymbol{\Delta} = |\underline{\boldsymbol{\Delta}}| = \overline{\boldsymbol{\Delta}}$) for LexLSAug2BFGS is given in fig. 4.20. The trust region radii of the base translational joint and the shoulder joint for the end-effector 1 are only reduced at a few instances. However, especially during instances when the base rotational joint is in an upright configuration as seen on the left picture of fig. 4.8 (iteration 2200 to 3500 and 5700 to

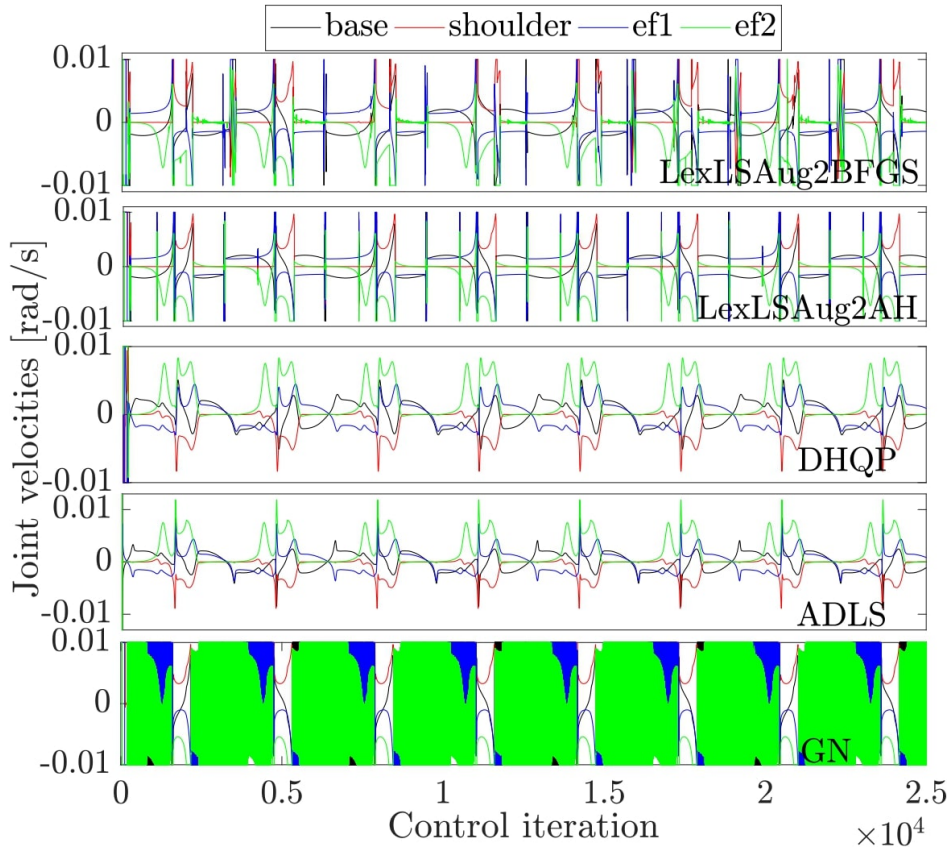


Fig. 4.15. ***T1****, joint velocities seen for the different methods LexLSAug2BFGS, LexLSAug2AH, DHQP, ADLS and pure GN algorithm (from top to bottom).*

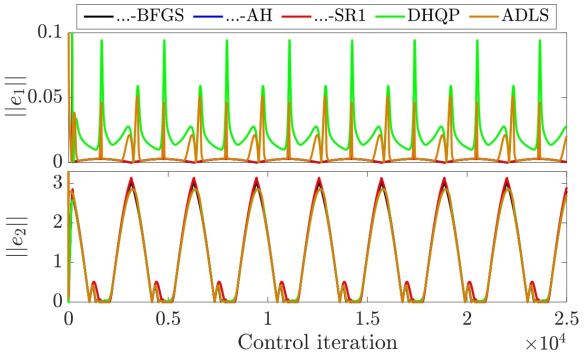


Fig. 4.16. ***T1****, norm of task errors for level 1 and 2 with different methods.*

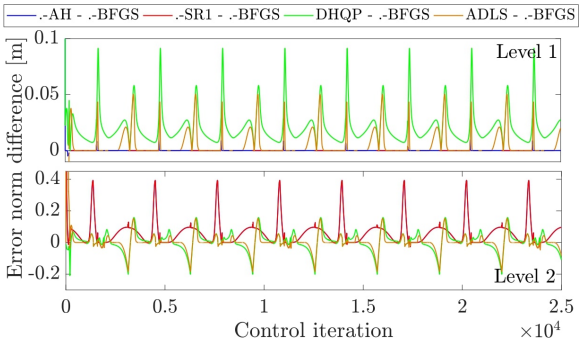


Fig. 4.17. *T1, difference of the norm of task errors for level 1 and 2 between the different methods.*

6600), both the base rotational joint and the shoulder joint of the end-effector 2 are almost completely put to a hold with $\boldsymbol{\Delta}(2) \approx 0$ and $\boldsymbol{\Delta}(4) \approx 0$ in order to prevent numerical instabilities. Thereby, for the shoulder joint of the end-effector 2 back and forth behaviour (between reducing and relaxing the trust region radius) of small amplitude

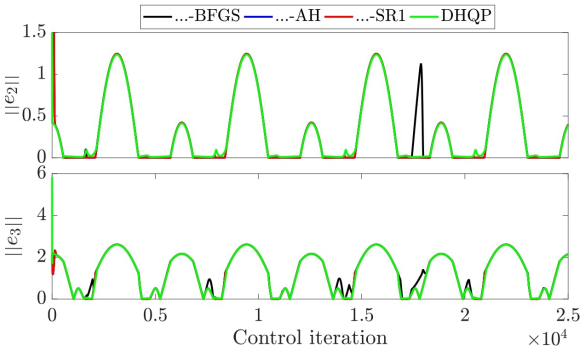


Fig. 4.18. ***T2****, norm of task errors for level 1, 2 and 3 for LexLSAug2AH, LexLSAug2SR1, LexLSAug2BFGS and DHQP.*

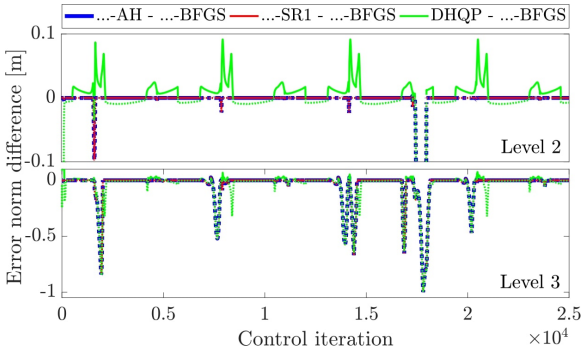


Fig. 4.19. ***T2****, difference of the norm of task errors for level 1, 2 and 3 between LexLSAug2BFGS and DHQP. Positive values indicate that LexLSAug2BFGS converges better than DHQP. DHQP is slightly numerically unstable on the second level due to insufficient damping as can be seen from the jittering in the lower graph (for example iterations 4500 - 5000).*

$< 10^{-3}$ rad can be observed. The development of more advanced trust region adaptation heuristics might be able to prevent such behaviour. Furthermore, a full trust region relaxation might be desired if the robot is in a singularity-free configuration and motion speeds higher than admissible by the trust region constraint are required.

For **T2**, which is omitted for ADLS due to the presence of the inequality task bounding the robot shoulder, the task error norms for the level 2 and level 3 end-effectors are given in fig. 4.18. The inequality constraint on level 1 is tracked to a high degree with only a few violations at a low amplitude $\sim 10^{-5}$ m and is not shown. The LexLSAug2 methods and DHQP track the level 2 target very well. At around iteration 17000, LexLSAug2BFGS looses track especially of the level 2 target due to a Type 2 singularity Kyong-Sok Chang et Khatib [1995] (high velocities are prevented by the trust region constraint). The LexLSAug2 methods behave slightly better in instances when the target is fully reachable. In these cases, the error norm for the LexLSAug2 methods is close to zero while for DHQP some residual of < 0.05 m remains. This again enables DHQP to minimize the error on level 3 to a higher degree than seen for the LexLSAug2 methods.

For the static test cases **T4** to **T8**, the LexLSAug2 methods behave in a numerically stable manner while DHQP becomes numerically unstable when the targets are at $[0, 2+10]$ m and $[1+10, 1]$ m. While the LexLSAug2 methods converge quickly within less than 210 iterations due to the switch to the GN algorithm, DHQP converges in an exponential fashion only after iteration $\geq 10^4$ for T4, T5 and T6. DHQP, ADLS and the GN-algorithm are numerically unstable during T7 due to insufficient damping. All solvers behave well for T8 which poses a feasible problem without any singularities

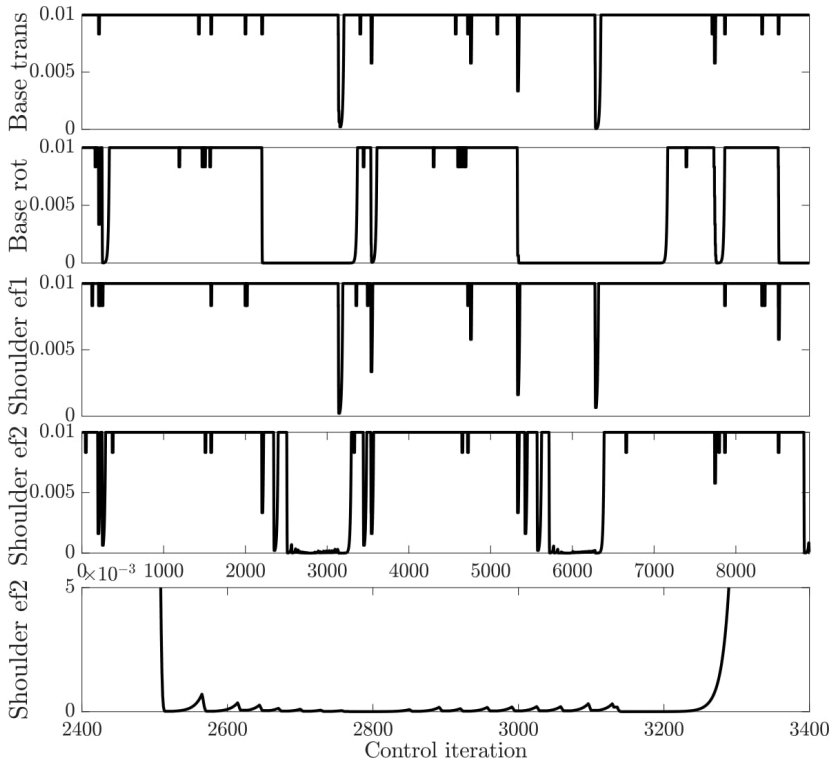


Fig. 4.20. *Test 1, LexLSAug2BFGS, adapted trust region radius for the base's translational (unit: m) and rotational joint and the shoulder joints (unit: rad) for the end-effector 1 and end-effector 2. The original trust region is chosen as 0.01 rad (or m for the translational base joint). The bottom graph shows a section of the shoulder joint for the end-effector 2 where the trust region adaptation method suppresses numerical instabilities of small amplitude.*

present.

For **T9**, the LexLSAug2 methods do not follow the random noise of the targets with a low $\Sigma \leq 0.1$ rad due to the trust region adaptation method. For DHQP, ADLS and the GN algorithm we have $\Sigma = 470$ rad, $\Sigma = 3e^3$ rad and $\Sigma = 551$ rad respectively. The value for ADLS is exceptionally high since the solution is not bounded by a velocity constraint.

For **T10** to **T14**, the LexLSAug2 methods are numerically stable with $\Sigma < 0.02$ rad. DHQP and ADLS are only numerically stable in the beginning of each test and then become increasingly numerically unstable as the robot moves away from the second target. Increasing the damping values would prevent these numerical instabilities but leads to worse convergence behaviour.

For **T15** to **T16** the hierarchy is violated by DHQP due to the damping not being applied in the calculation of the null-spaces. The first end-effector is trying to reach at best the just in reach or just out of reach targets at $[0, 2]$ m and $[0, 2 + 0.001]$ m respectively. For the LexLSAug2 methods, the base and the shoulder joints are solely attributed to the achievement of the first end-effector task. For DHQP however, these joints move as they are involved to a small degree in the achievement of reaching the

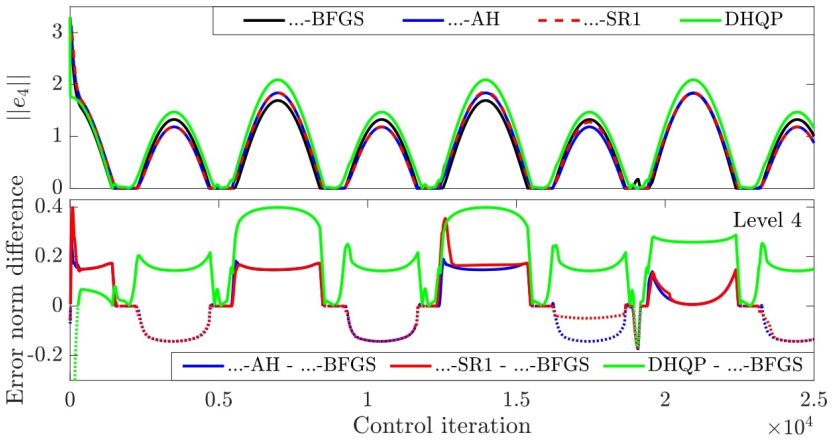


Fig. 4.21. T20*. The upper graph shows the norm of the task error for the right hand on hierarchy level 3. LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS move closer to the target than DHQP. The lower graph shows the difference between the methods.*

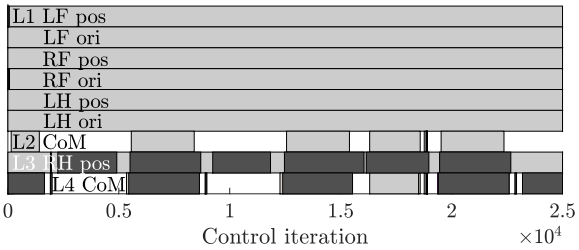


Fig. 4.22. T20*, map of activity (light gray) and Newton's method (dark gray) for LexLSAug2BFGS.*

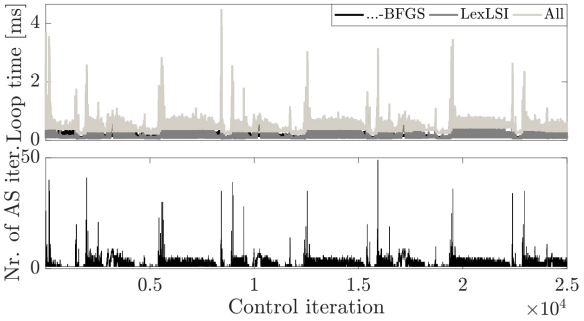


Fig. 4.23. T20*, control loop times in ms. The maximum loop times are 4.5 ms for LexLSAug2BFGS, 4.0 ms for LexLSI and 0.5 ms for LexLSAug2BFGS. The maximum number of active-set iterations in LexLSI is 49.*

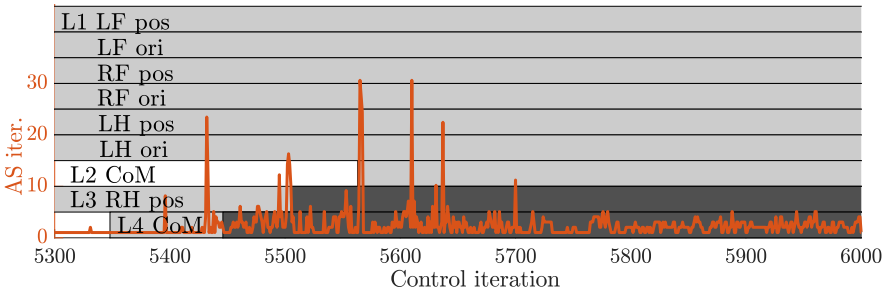


Fig. 4.24. T20*, excerpt of the map of activity (light gray) and Newton's method (dark gray) with overlaid active-set iteration count for LexLSAug2BFGS. The maximum active-set iteration count is 30 at control iteration 5565 and 5610.*

level 2 target with the second end-effector.

For **T20**, the LexLSAug2 methods and DHQP track the target in a numerically stable manner with $\Sigma = 0.003$ rad, $\Sigma = 0.05$ rad, $\Sigma = 0.7$ rad and $\Sigma = 0.2$ rad respectively. Fig. 4.21 shows the better convergence of the LexLSAug2 methods for the level 4 task of the right hand tracking the oscillating target (the error norms of the remaining higher priority tasks are not shown since for all methods they are zero). Especially, if the target is in reach the LexLSAug2 methods show a task error norm close to zero due to switching to the GN-algorithm. DHQP only gets into the proximity of the reachable target. For LexLSAug2SR1, the conditioning of the SR1 updates deteriorates over time with increasingly high condition numbers. While it does not lead to solver failure, first the smoothness of the joint trajectories deteriorates before the augmentation takes very large values on certain joints such that they stop moving. The exact reason for this behaviour requires further investigation.

The activation of the tasks and whether the GN algorithm or Newton's method is used is shown for LexLSAug2BFGS in fig. 4.22. An augmented task is also active.

The computation time for the control iterations are given for LexLSAug2BFGS in fig. 4.23. The computations were done on an Intel Core i7-4720HQ CPU @ 2.60GHz with 8 GB of RAM. Even with several levels being augmented to Newton's method the computation times lay well below 1 ms. However, some peaks up to 4.5 ms (with 35 active-set iterations) can be observed when the number of active-set iterations increases significantly (maximum 49 active-set iterations at control iteration 15932 with a computation time of 3.128 s). Note that the increase in active-set iterations occurs at different instances than the switches between the GN algorithm and Newton's method. An example (see fig. 4.24) would be the CoM task on level 4 which is activated at control iteration 5348 within 1 active-set iteration. The task then switches to Newton's method at control iteration 5446 which requires 2 active-set iterations. Shortly after at control iteration 5505, the level 3 right hand task switches to Newton's method which requires 10 active-set iterations. Finally, the level 2 CoM task is activated at control iteration 5563 with 1 active-set iteration. However, shortly after the active-set iteration count goes up to 30 at control iteration 5565.

Since the computations with DHQP are only done with an unoptimized version of the algorithm we omit its computation times due to the lack of a fair comparison.

4.5.4 Performance comparison

The overall performance of the LexLSAug2 methods (in combination with LexLSI) and DHQP (as a stack of LSP's solvers) and ADLS (uses the SVD for the pseudo-inverse calculation) in the test bench is compiled comprehensively in table 4.2.

The LexLSAug2 methods performed best in the category 'Error convergence'. The performance of DHQP and ADLS is highly dependent of the level of chosen damping. Its behaviour can only be considered acceptable at best since determining 'perfect' damping without proper adaptation methods seems intractable. The same holds for the 'Joint stability' and 'Joint smoothness' of the joint trajectories. If the damping weights are too low, the behaviour is numerically unstable and therefore not smooth.

	...-AH	**...-SR1**	**...-BFGS**	**DHQP**	**ADLS**
Error convergence	+	+	+	o / –	o / –
Joint stability	+	+	+	+ / –	+ / –
Joint smoothness	+ / o	o	o	+ / –	+ / –
Easiness of use	+	+	+	–	–
Computation time	o / –	o / –	+ / –	o	o
Lex. separation	+	+	+	o	+
Handling of infeas. ctr.	+	+	+	+	+
Handling of inequ. ctr.	+	+	+	+	-

Table 4.2 – Comprehensive overview of the test bench performance of LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS (in combination with LexLSI) and DHQP (as a stack of QP solvers) and ADLS (uses the SVD for the pseudo-inverse calculation). The symbols +, o, – indicate best, acceptable and worst performance in the corresponding evaluation criteria. Criteria solely dependent on the characteristics of LexLSI or the corresponding solver are coloured in grey.

If the damping weights are too high the joint trajectories are very smooth but the error convergence is bad.

LexLSAug2AH shows very smooth joint trajectories. LexLSAug2SR1 and LexLSAug2BFGS perform a bit worse with less smooth joint trajectories. Especially LexLSAug2BFGS relies on the trust region adaptation method in order to suppress numerical instabilities at times. These numerical instabilities are not of high amplitude. In T20 we observed bad conditioning of the SR1 updates which led to numerical instabilities for LexLSAug2SR1. For all three methods no damping tuning is required which makes them very easy to use ('Easiness of use').

For T20, LexLSI required an increased number of active-set iterations in some instances. The control loop time was still under 5 ms but might violate the real-time constraint if more active-set iterations are required (see next chapter's section 5.3). LexLSAug2AH and LexLSAug2SR1 require the expensive SVD decomposition for the Higham regularization of the Lagrangian Hessian which makes its computation time only acceptable at best.

All methods allow an unlimited number of hierarchy levels ('lexicographical (lex.) separation') and infeasible constraints ('Handling of infeas. ctr.'). HDHQP does not propagate damping terms onto the null-spaces which violates the notion of strict lexicographic separation to a certain degree.

4.6 Conclusion

With this chapter we have proposed, implemented and validated new robust methods which handle singularities in prioritized inverse kinematics control schemes. Our methods LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS are able to solve robotic

problems with any number of hierarchical levels consisting of equality and inequality constraints. These constraints may be infeasible, in kinematic singularities or in conflict with higher priority constraints without showing signs of numerical instability. Instead, our methods enable smooth control. At the same time, our approach converges to a higher degree than other state-of-the-art methods.

Our approach is inspired by methods of constrained optimization: Newton's method represents the original non-linear task error function accurately around singularities which provides numerically stable and accurate control. Our adaptation method switches reliably to the GN-algorithm whenever a problem becomes feasible. This frees joints which then can be used by lower priority levels. Furthermore, due to the least-squares formulation of the GN-algorithm and Newton's method we are able to solve the hierarchical LSP's by state-of-the-art, fast hierarchical least-squares solvers into our scheme allowing real-time control.

We do not elaborate on the discontinuity that comes with a switch from the GN algorithm to Newton's method. A perturbation analysis for least-squares problems in the sense of Björck [1996] is relevant for full rank problems and gives an upper bound on the discontinuity. However, with our full rank augmentation the rank of the least-squares problem usually changes since $m < n$. In this case only a lower bound can be found [Wei 1992; Wedin 1973]. Putting these results into context of our hierarchical least-squares formulation needs to be addressed in future work.

We introduced methods both for analytically calculating and approximating the Lagrangian Hessian. The Hessian provided by LexLSAug2AH and LexLSAug2SR1 can become indefinite if not close to the solution. This requires an expensive SVD decomposition in order to enforce semi-positive-definiteness of the Hessian using the Higham regularization.

In future work it is desirable to more thoroughly understand the negative definiteness either of the analytic Hessians (LexLSAug2AH) or the SR1 updates (LexLSAug2SR1) on numerical stability and convergence. Cheaper regularization methods based for example on the Bunch-Kaufman decomposition require our attention. Also, occurrences of negative definite curvature during the BFGS updates (LexLSAug2BFGS) are ignored but rather the last positive definite update is kept. This slows down convergence and also might lead to numerical instability and needs to be investigated.

Model inaccuracies are accounted for with a trust region adaptation method which is customized for constrained optimization. Further developments might include a full trust region relaxation method. This is of relevance if the robot is in a singularity-free configuration and motion speeds higher than admissible by the trust region constraint are required.

For LexLSAug2BFGS, it would be desirable to directly update the Cholesky decomposition $\boldsymbol{R}_l$ of $\hat{\boldsymbol{B}}_l$ as realized in Fletcher [2006] in order to save computation time.

The methods were evaluated on a test bench with 20 test cases. As the evaluation criteria for numerical stability we counted consecutive sign changes on joints of the calculated joint velocity vector. While this measure gives a rough idea about the

numerical stability performance of the methods it is desirable to formulate a more universal definition of a numerical instability, for example based on the singular values of the Jacobian or the Lagrangian Hessian.

CHAPTER 5

Dynamically feasible kinematic control with singularity resolution

The previous chapter was concerned with resolving singularities in multi-level constrained kinematic control problems. While kinematic control of robots can be sufficient for fixed base robots Caccavale et al. [1997]; Wang et al. [2010], this does not necessarily hold for legged humanoid robots with an un-actuated free-flyer base and unilateral friction contacts. Only with knowledge of the dynamic forces and torques acting on the robot's body a control can be designed which aims at maintaining a physically stable posture Khatib [1987]. Therefore, we present ways to include the equation of motion and dynamics constraints (torque limits, contact force constraints) into our methods from the previous chapter 4 to realize dynamically feasible motions.

Additionally, the time step was conveniently chosen as $\Delta t = 1$ s which enabled the formulation of kinematic control problems as optimization problems. The application of Newton's method for multi-level constrained optimization problems was then trivial. However, in dynamic control we have to consider the true dynamic robot state including the joint velocities and accelerations without making an assumption about the time step.

This chapter then presents the following new contributions:

- We adapt the GN algorithm and Newton's method, which are both tools from optimization, to control (see sec. 5.1).
- We show how regularization terms like damping negatively influence the exponential convergence of second-order motion controllers (see sec. 5.2.1).
- Second-order motion controllers are then suitably adapted so they can be expressed as the velocity-based Newton's method of control (see sec. 5.2.2).

- The dynamics in form of the equation of motion are adapted accordingly and then integrated into the hierarchical control scheme (see sec. 5.2.3).
- We experimentally assess our developments with the HRP-2Kai humanoid robot (see sec. 5.4).

5.1 From optimization to kinematic control

In the previous chapter 4 we introduced the GN algorithm and Newton's method of constrained optimization to drive a non-linear geometric error function to zero. This is equivalent to defining a quadratic Taylor approximation (4.9) of the quadratic function (4.7) around $\boldsymbol{q}^{(k)}$ and looking for a bounded step $\boldsymbol{\Delta q}^{(k)}$ in a certain neighbourhood (called trust region) of it. Assuming a control time step of $\Delta t = 1$ s with $\boldsymbol{\Delta q}^{(k)} = \Delta t \dot{\boldsymbol{q}}^{(k)*} = \dot{\boldsymbol{q}}^{(k)*}$ we can make a trivial connection between optimization, which aims at making a step $\boldsymbol{\Delta q}^{(k)}$ towards the optimum, and control, which aims at determining the new robot state triple $\left[\boldsymbol{q}^{(k+1)}, \dot{\boldsymbol{q}}^{(k+1)}, \ddot{\boldsymbol{q}}^{(k+1)}\right]$ while minimizing the error function with a certain behaviour. $\dot{\boldsymbol{q}}^{(k)*}$ corresponds to the new velocity $\dot{\boldsymbol{q}}^{(k+1)}$ (in the case of acceleration-based control $\ddot{\boldsymbol{q}}^{(k)*} = \ddot{\boldsymbol{q}}^{(k+1)}$). We introduce this specific notation in order to keep the control formulation consistent with the step $\boldsymbol{\Delta q}^{(k)}$ which is calculated in optimization for the current index k. At the same time we distinguish it from the value $\dot{\boldsymbol{q}}^{(k)}$ which is used to calculate $\dot{\boldsymbol{e}}^{\text{ctrl}}_{\text{PD}}$, see sec. 5.2.2.

However, a robot is usually controlled at a much higher frequency with $\Delta t \ll 1$ s. Especially when considering the robot dynamics one needs to compute the new velocity $\dot{\boldsymbol{q}}^{(k)*}$ at each control step k without making an assumption about the time step Δt. One can look for the new velocity $\dot{\boldsymbol{q}}^{(k)*}$ with the relation $\dot{\boldsymbol{e}}^{\text{ctrl}} = -\boldsymbol{J}\dot{\boldsymbol{q}}^{(k)*}$ (2.2). A solution is given by $\dot{\boldsymbol{q}}^{(k)*} = -\boldsymbol{J}^{+}\dot{\boldsymbol{e}}^{\text{ctrl}}$ using the Moore-Penrose pseudo-inverse $\boldsymbol{J}^{+}$. This formulation corresponds to the GN algorithm.

However, this approach is not suitable in the vicinity of singularities: $\boldsymbol{J}$ is almost loosing at least one rank and $\dot{\boldsymbol{q}}^{(k)*}$ becomes very large due to numerical limitations. In such situations we proposed in the previous chapter 4 to use Newton's method instead. Our claim was that neglecting the second order information $\boldsymbol{H}$ in the second order Taylor approximation (4.7) (as done in the GN algorithm) is only sufficient if away from singularities. Using the second order information close to singularities restores the original accuracy of the second order Taylor approximation and prevents singular behaviour.

In the hierarchy, each non-linear task $\boldsymbol{f}(\boldsymbol{q}) = \boldsymbol{f}_d$ or $\boldsymbol{f}(\boldsymbol{q}) \leq \boldsymbol{f}_d$ is rewritten to the constrained GN algorithm of control

$$\min_{\dot{\boldsymbol{q}}^{(k)*}} \quad \frac{1}{2}\|\boldsymbol{J}_l\dot{\boldsymbol{q}}^{(k)*} + \dot{\boldsymbol{e}}^{\text{ctrl}}_l\|^2_2 \tag{5.1}$$

$$\text{s.t.} \quad -\underline{\dot{\boldsymbol{e}}}^{\text{ctrl}}_{l-1} - \underline{\boldsymbol{J}}_{l-1}\dot{\boldsymbol{q}}^{(k)*} \leqq \underline{\boldsymbol{w}}^{*,(k+1)}_{l-1} \tag{5.2}$$

or the constrained Newton's method of control

$$\min_{\dot{\boldsymbol{q}}^{(k+1)}} \quad \frac{1}{2} \left\| \begin{bmatrix} \boldsymbol{J}_l \\ \boldsymbol{R}_l \end{bmatrix} \dot{\boldsymbol{q}}^{(k)*} + \begin{bmatrix} \dot{\boldsymbol{e}}_l^{\text{ctrl}} \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 \tag{5.3}$$

$$\text{s.t.} \quad -\underline{\dot{\boldsymbol{e}}}_{l-1}^{\text{ctrl}} - \underline{\boldsymbol{J}}_{l-1} \dot{\boldsymbol{q}}^{(k)*} \leqq \underline{\boldsymbol{w}}_{l-1}^{*,(k+1)} \,. \tag{5.4}$$

Unlike in the previous chapter 4 we do not make the assumptions $\Delta t = 1$ s and $\dot{\boldsymbol{e}}^{\text{ctrl}} = -k_p \boldsymbol{e} = -\boldsymbol{e}$ as it was done in (4.4). The set of priority-ordered least-squares problems is then passed to the hierarchical solver Dimitrov et al. [2015].

Δt thereby connects the two entities of 'optimization' (optim) and 'control'

$$\boldsymbol{J} \boldsymbol{\Delta q}^{(k)} + \Delta t \dot{\boldsymbol{e}}^{\text{ctrl}} = \Delta t (\boldsymbol{J} \dot{\boldsymbol{q}}^{(k)*} + \dot{\boldsymbol{e}}^{\text{ctrl}}) \tag{5.5}$$

$$= \boldsymbol{w}^{\text{optim}} = \Delta t \boldsymbol{w}^{\text{ctrl}} \,. \tag{5.6}$$

That is, we calculate a new velocity $\dot{\boldsymbol{q}}^{(k)*}$ but only do a model based step $\boldsymbol{\Delta q}^{(k)} = \Delta t \dot{\boldsymbol{q}}^{(k)*}$ towards the optimum. Consequently, the model needs to be updated with the Lagrangian Hessian (4.26) using $\boldsymbol{w}^{\text{optim}}$ and $\boldsymbol{\lambda}^{\text{optim}}$. Since solving the constrained control problems (5.1) or (5.3) yields $\boldsymbol{w}^{\text{ctrl}}$ and $\boldsymbol{\lambda}^{\text{ctrl}}$, a scaling of the form $\boldsymbol{w}^{\text{optim}} = \Delta t \boldsymbol{w}^{\text{ctrl}}$ according to (5.6) is required. Due to the linear dependency between the slack $\boldsymbol{w}$ and the Lagrange multipliers $\boldsymbol{\lambda}$ (see Dimitrov et al. [2015]) we further get $\boldsymbol{\lambda}^{\text{optim}} = \Delta t \boldsymbol{\lambda}^{\text{ctrl}}$ (or simply $\boldsymbol{\Lambda}^{\text{optim}} = \Delta t \boldsymbol{\Lambda}^{\text{ctrl}}$). In what follows, we write $\boldsymbol{w} = \boldsymbol{w}^{\text{optim}}$ and $\boldsymbol{\lambda} = \boldsymbol{\lambda}^{\text{optim}}$.

Accordingly, the thresholds for the switching method and BFGS algorithm need to be adapted too. ξ is a threshold for $\boldsymbol{y}^T \boldsymbol{\Delta q}^{(k-1)} = (\boldsymbol{J}^T \Delta t \boldsymbol{\lambda}^{\text{ctrl}})^T \Delta t \dot{\boldsymbol{q}}^{(k-1)}$, ζ a threshold for $\Delta t \dot{\boldsymbol{e}}^{\text{ctrl},T} \Delta t \dot{\boldsymbol{e}}^{\text{ctrl}}$ and ν a threshold for $\boldsymbol{w}^{\text{optim},T} \boldsymbol{w}^{\text{optim}} = \Delta t \boldsymbol{w}^{\text{ctrl},T} \Delta t \boldsymbol{w}^{\text{ctrl}}$. Since these values are all quadratically dependent of the time step Δt, we choose $\xi = \Delta t^2 \xi$ as the threshold of the positive curvature condition for the BFGS algorithm, $\zeta = \Delta t^2 \zeta$ for the minimum value in the BFGS initialization and $\nu = \Delta t^2 \nu$ for the switching method.

5.2 Dynamically feasible kinematic control

The equation of motion ensuring physical feasibility is of second order and correspondingly second order motion controllers are defined. However, in the previous section 5.1 we have formulated the GN algorithm and Newton's method only in the velocity domain. In sec. 5.2.1 we argue why Newton's method cannot be extended easily to the acceleration domain. Therefore, our idea for acceleration-based control is to change the right hand side $\dot{\boldsymbol{e}}^{\text{ctrl}}$ from a linear proportional controller to some controller $\dot{\boldsymbol{e}}_{\text{PD}}^{\text{ctrl}}$ that emulates second order PD control $\ddot{\boldsymbol{e}}^{\text{ctrl}}$ in the velocity domain. In sec. 5.2.2 we show how this can be achieved and apply the corresponding necessary adaptations to the equation of motion which we include into our control framework (see sec. 5.2.3).

5.2.1 Damping in acceleration-based control

In the following we show that

$$\min_{\ddot{\boldsymbol{q}}^{(k)*}} \quad \frac{1}{2} \left\| \begin{bmatrix} \boldsymbol{J} \\ \boldsymbol{R} \end{bmatrix} \ddot{\boldsymbol{q}}^{(k)*} + \begin{bmatrix} \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} + \ddot{\boldsymbol{e}}^{\mathrm{ctrl}} \\ \boldsymbol{0} \end{bmatrix} \right\|_2^2 \tag{5.7}$$

leads to low frequency oscillations around a minimum. For simplicity, we assume $\boldsymbol{R} = \mu \boldsymbol{I}$.

Let's take a look at a 1D robot as a point mass ($m = 1$ kg) moving on a line with position $f = x$. Its desired position is $x_d = 0$. The task error is then $e = x_d - x = -x$. The Jacobian of this robot is $J = \frac{df}{dx} = \frac{dx}{dx} = 1$, the time derivative of the Jacobian is $\dot{J} = 0$.

5.2.1.1 Velocity-based control

The control law is usually written as the first order ordinary differential equation (ODE)

$$J\dot{x} + \dot{e}^{\mathrm{ctrl}} = J\dot{x} - k_p e = \dot{x} + k_p x = 0 \tag{5.8}$$

with the solution

$$x = D\mathrm{e}^{-k_p t}\,, \tag{5.9}$$

where $x \to 0$ for $k_p > 0$ and $t \to \infty$. D is a constant of integration.

The numerical integration can be formulated as

$$x^{(k+1)} = x - \Delta t k_p x\,. \tag{5.10}$$

5.2.1.2 Velocity-based control with damping

If we introduce damping we get the following form

$$\begin{bmatrix} 1 \\ \mu \end{bmatrix} \dot{x} + \begin{bmatrix} k_p \\ 0 \end{bmatrix} x = \begin{bmatrix} 0 \\ 0 \end{bmatrix}\,. \tag{5.11}$$

The least squares solution to this problem for $\dot{x}$ is the first order ODE

$$\dot{x} + \frac{k_p}{1 + \mu^2} x = 0 \tag{5.12}$$

and converges exponentially as in sec. 5.2.1.1.

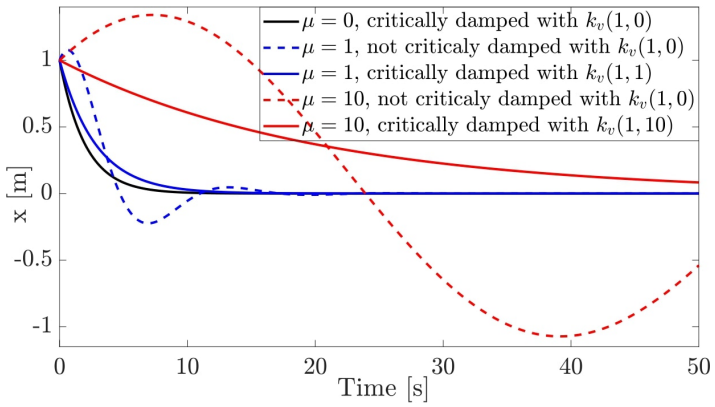


Fig. 5.1. *The second order homogeneous ODE solution (5.15) with k_p^* and k_v^* from (5.17) are plotted for $m = 1$ kg, $k_p = 1$, different μ and $k_v(k_p) = 2\sqrt{mk_p}$ (black line and dashed lines) or $k_v(k_p, \mu) = 2\sqrt{m(1+\mu^2)k_p}$.*

5.2.1.3 Acceleration-based control

The control law for *acceleration-based control* is usually written as (see (2.10))

$$m(J\ddot{x} + \dot{J}\dot{x}) + \ddot{e}^{\text{ctrl}} = m(J\ddot{x} + \dot{J}\dot{x}) - k_p e - k_v \dot{e} = m(J\ddot{x} + \dot{J}\dot{x}) + k_p x + k_v \dot{x} \quad (5.13)$$

$$= m\ddot{x} + k_v \dot{x} + k_p x = 0\,. \quad (5.14)$$

An analytic solution can be found in

$$x = \exp(-\delta t)(A\cos(w_d t) + B\sin(w_d t))\,. \quad (5.15)$$

$\delta = k_v/2/m$ and $w_d = \sqrt{k_p/m - \delta^2}$. Exponential convergence can be achieved if the eigenvalue of the system is chosen as $w_d = 0$. This tuning of gains is called critical damping and corresponds to $k_v = 2\sqrt{mk_p}$.

5.2.1.4 Acceleration-based control with damping

For the augmented system

$$\begin{bmatrix}1\\ \mu\end{bmatrix} m\ddot{x} + \begin{bmatrix}1\\ 0\end{bmatrix} k_v \dot{x} + \begin{bmatrix}1\\ 0\end{bmatrix} k_p x = \begin{bmatrix}0\\ 0\end{bmatrix} \quad (5.16)$$

we get the following ODE as its least-squares solution for $\ddot{x}$:

$$m\ddot{x} + \frac{k_v}{1+\mu^2}\dot{x} + \frac{k_p}{1+\mu^2}x = m\ddot{x} + k_v^*\dot{x} + k_p^* x = 0\,. \quad (5.17)$$

The analytical solution is given by (5.15) with k_p and k_v in δ and w_d replaced by k_p^* and k_v^*. The numerical integration writes as

$$x^{(k+1)} = x + \Delta t\dot{x} + \frac{\Delta t^2}{2m}\left(-\frac{k_v}{1+\mu^2}\dot{x} - \frac{k_p}{1+\mu^2}x\right)\,, \quad (5.18)$$

clearly exposing the influence of the damping μ on the critically damped task gains k_p and k_v. Critical damping can now be achieved with $k_v(k_p, \mu) = 2\sqrt{m(1+\mu^2)k_p}$. Some convergence curves of (5.15) with k_p^* and k_v^* are plotted in fig. 5.1. It can be clearly seen that the damping μ influences the critically damped system negatively (i.e. overshooting) if the gain is chosen according to $k_v(k_p, 0)$ instead of $k_v(k_p, \mu)$.

For a more complicated 3-D robot with more and especially coupled DoF's and a time varying $\boldsymbol{R}$ it seems cumbersome to find the expression for critical damping $k_v(k_p, \boldsymbol{R})$ such that overshooting behaviour can be prevented.

Therefore, we favour to shift the whole problem into the velocity domain and emulate acceleration-based control by adapting the right hand-side accordingly. This way we can easily achieve exponential convergence as shown in the next section 5.2.2.

5.2.2 Acceleration-based control expressed in the velocity domain

In acceleration-based control a new joint acceleration can be obtained by solving

$$\ddot{\boldsymbol{e}}^{\text{ctrl}} = -\boldsymbol{J}\ddot{\boldsymbol{q}}^{(k)*} - \dot{\boldsymbol{J}}\dot{\boldsymbol{q}} \tag{5.19}$$

for $\ddot{\boldsymbol{q}}^{(k)*}$. $\ddot{\boldsymbol{e}}^{\text{ctrl}}$ is defined as a PD controller $\ddot{\boldsymbol{e}}^{\text{ctrl}} \overset{\text{def}}{=} -k_p\boldsymbol{e} - k_v\dot{\boldsymbol{e}}$ (2.9). We extend the formulation to a multi-level constrained control hierarchy where each level is either formulated as the GN algorithm (5.1) or Newton's method (5.3) such that we get for the stacked values $\underline{\boldsymbol{J}}_p$ and $\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}$

$$\underline{\boldsymbol{J}}_p = \begin{bmatrix} \boldsymbol{J}_1 \\ \boldsymbol{R}_1 \\ \cdots \\ \boldsymbol{J}_{p-1} \\ \boldsymbol{R}_{p-1} \\ \boldsymbol{I}_p \end{bmatrix}, \quad \underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}} = \begin{bmatrix} \ddot{\boldsymbol{e}}_1^{\text{ctrl}} + \dot{\boldsymbol{J}}_1\dot{\boldsymbol{q}} \\ \boldsymbol{0}_1 \\ \cdots \\ \ddot{\boldsymbol{e}}_{p-1}^{\text{ctrl}} + \dot{\boldsymbol{J}}_{p-1}\dot{\boldsymbol{q}} \\ \boldsymbol{0}_{p-1} \\ \boldsymbol{p}_p \end{bmatrix}. \tag{5.20}$$

The gray matrices and vectors are only present in case that the respective level l is augmented with second order information $\boldsymbol{R}_l$ due to a switch to Newton's method. The identity matrix on the last level ensures that the problem is fully determined. The corresponding right hand side $\boldsymbol{p}_p$ is defined such that the last level corresponds to the zero velocity task $\boldsymbol{I}\dot{\boldsymbol{q}} = \boldsymbol{0}$ expressed as an acceleration task.

The integration to the new joint configuration takes the form

$$\ddot{\boldsymbol{q}}^{(k+1)} = \ddot{\boldsymbol{q}}^{(k)*} = -\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}} \tag{5.21}$$

$$\dot{\boldsymbol{q}}^{(k+1)} = \dot{\boldsymbol{q}} + \Delta t\ddot{\boldsymbol{q}}^{(k+1)} = \dot{\boldsymbol{q}} - \Delta t\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}, \tag{5.22}$$

$$\boldsymbol{q}^{(k+1)} = \boldsymbol{q} + \Delta t\dot{\boldsymbol{q}} + \frac{\Delta t^2}{2}\ddot{\boldsymbol{q}}^{(k+1)} = \boldsymbol{q} + \Delta t\dot{\boldsymbol{q}} - \frac{\Delta t^2}{2}\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}. \tag{5.23}$$

$\underline{\boldsymbol{J}}_p^\ddagger$ is the hierarchical inverse of $\underline{\boldsymbol{J}}_p$ Escande et al. [2014]. We now want to find a velocity-based controller $\dot{\boldsymbol{e}}_{\text{PD}}^{\text{ctrl}}$ which imitates the acceleration-based PD controller $\ddot{\boldsymbol{e}}^{\text{ctrl}}$ and leads to the same results for $\dot{\boldsymbol{q}}^{(k+1)}$ and $\boldsymbol{q}^{(k+1)}$.

For this lets replace the accelerations in (5.19) by forward differences

$$\ddot{\boldsymbol{q}}^{(k)*} = \frac{\dot{\boldsymbol{q}}^{(k)*} - \dot{\boldsymbol{q}}^{(k)}}{\Delta t} \tag{5.24}$$

such that we get

$$\dot{\boldsymbol{e}}_{\text{PD}}^{\text{ctrl}} = -\boldsymbol{J}\dot{\boldsymbol{q}}^{(k)*}\,. \tag{5.25}$$

$\dot{\boldsymbol{e}}_{\text{PD}}^{\text{ctrl}}$ replaces $\dot{\boldsymbol{e}}^{\text{ctrl}}$ in 5.1 and in 5.3 and is defined as

$$\dot{\boldsymbol{e}}_{\text{PD}}^{\text{ctrl}} \overset{\text{def}}{=} -\boldsymbol{J}\dot{\boldsymbol{q}} + \Delta t(\ddot{\boldsymbol{e}}^{\text{ctrl}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}) \tag{5.26}$$

for the GN algorithm and

$$\dot{\boldsymbol{e}}_{\text{PD}}^{\text{ctrl}} \overset{\text{def}}{=} -\begin{bmatrix}\boldsymbol{J}\\ \boldsymbol{R}\end{bmatrix}\dot{\boldsymbol{q}} + \Delta t\begin{bmatrix}\ddot{\boldsymbol{e}}^{\text{ctrl}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}\\ \boldsymbol{0}\end{bmatrix} \tag{5.27}$$

for Newton's method, respectively. The stacked value $\underline{\ddot{\boldsymbol{e}}}_{\text{PD},p}^{\text{ctrl}}$ then is

$$\underline{\ddot{\boldsymbol{e}}}_{\text{PD},p}^{\text{ctrl}} = -\underline{\boldsymbol{J}}_p\dot{\boldsymbol{q}} + \Delta t\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}\,. \tag{5.28}$$

Note that this forward integration requires the orientation of the robot-base to be expressed in Euclidean space instead of quaternions. Singular cases (gimbal lock) of Euler-angles (in our case we use the $x-y-x$ Euler angles α, β and γ) need to be avoided. For this, in every control iteration the robot's free-flyer base configuration (for example the Denavit-Hartenberg parameters) is updated with the newly calculated Euler angles. After that the Euler angles are reset to zero. The changes of the base orientation between control iterations are assumed to be small with $\alpha, \beta, \gamma \ll \pi/2$ so singular configurations are avoided.

The acceleration-based PD control in velocity gives

$$\dot{\boldsymbol{q}}^{(k+1)} = \dot{\boldsymbol{q}}^{(k)*} = -\underline{\boldsymbol{J}}_p^\ddagger\underline{\ddot{\boldsymbol{e}}}_{\text{PD},p}^{\text{ctrl}} = \underline{\boldsymbol{J}}_p^\ddagger\underline{\boldsymbol{J}}_p\dot{\boldsymbol{q}} - \Delta t\underline{\boldsymbol{J}}_p^\ddagger\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}} \tag{5.29}$$

$$\boldsymbol{q}^{(k+1)} = \boldsymbol{q} + \Delta t\dot{\boldsymbol{q}}^{(k+1)} = \boldsymbol{q} - \Delta t\underline{\boldsymbol{J}}_p^\ddagger\underline{\ddot{\boldsymbol{e}}}_{\text{PD},p}^{\text{ctrl}} = \boldsymbol{q} + \Delta t\underline{\boldsymbol{J}}_p^\ddagger\underline{\boldsymbol{J}}_p\dot{\boldsymbol{q}} - \Delta t^2\underline{\boldsymbol{J}}_p^\ddagger\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}\,. \tag{5.30}$$

If $\text{rank}(\underline{\boldsymbol{J}}_p) = n$ (which it is because of the identity matrix on the last level) then $\underline{\boldsymbol{J}}_p^\ddagger\underline{\boldsymbol{J}}_p = \boldsymbol{I}$ Escande et al. [2014] and the above value of $\dot{\boldsymbol{q}}^{(k+1)}$ in (5.30) is the same as the one obtained with acceleration-based control in (5.23). However, the joint positions are missing the factor 0.5 in front of the third term of (5.30). Therefore, an adjustment in the calculation of the joint positions needs to be made:

$$\boldsymbol{q}_{\text{mod}}^{(k+1)} = \boldsymbol{q}_{\text{mod}} + \Delta t(\frac{1}{2}\dot{\boldsymbol{q}}^{(k+1)} + \frac{1}{2}\dot{\boldsymbol{q}}) \tag{5.31}$$

$$= \boldsymbol{q}_{\text{mod}} + \Delta t(\frac{1}{2}(\dot{\boldsymbol{q}} - \Delta t\underline{\boldsymbol{J}}_p^\ddagger\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}) + \frac{1}{2}\dot{\boldsymbol{q}}) \tag{5.32}$$

$$= \boldsymbol{q}_{\text{mod}} + \Delta t\dot{\boldsymbol{q}} - \frac{\Delta t^2}{2}\underline{\boldsymbol{J}}_p^\ddagger\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}\,. \tag{5.33}$$

The modified joint positions correspond to the ones of the acceleration-based control problem. The joint velocities $\dot{\boldsymbol{q}}^{(k+1)}$ stay untouched since they already correspond to the ones of the acceleration-based PD controller. Accordingly, we need to calculate the step $\boldsymbol{\Delta q}$ for LexLSAug2BFGS by

$$\boldsymbol{\Delta q}_{\text{mod}} = \boldsymbol{q}_{\text{mod}}^{(k+1)} - \boldsymbol{q}_{\text{mod}} \tag{5.34}$$

instead of $\boldsymbol{\Delta q} = \Delta t \dot{\boldsymbol{q}}^{(k)*}$. Both LexLSAug2AH and LexLSAug2BFGS are robust with regard to the inconsistent Lagrange multipliers which correspond to the step $\boldsymbol{\Delta q}$ and not to the true step $\boldsymbol{\Delta q}_{\text{mod}}$.

Another possibility, which is consistent with the Lagrange multipliers, is to use

$$\ddot{\boldsymbol{q}}^{(k)*} = \frac{\dot{\boldsymbol{q}}^{(k)*} - \dot{\boldsymbol{q}}^{(k)}}{0.5\Delta t}\,. \tag{5.35}$$

instead of the forward differences in (5.24) for replacing the accelerations in (5.19). We then get the controller (similar for the case of Newton's method)

$$\dot{\boldsymbol{e}}_{\text{PD},0.5}^{\text{ctrl}} \overset{\text{def}}{=} -\boldsymbol{J}\dot{\boldsymbol{q}} + \frac{\Delta t}{2}(\ddot{\boldsymbol{e}}^{\text{ctrl}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}})\,. \tag{5.36}$$

The stacked value $\underline{\ddot{\boldsymbol{e}}}_{\text{PD},0.5,p}^{\text{ctrl}}$ is

$$\underline{\ddot{\boldsymbol{e}}}_{\text{PD},0.5,p}^{\text{ctrl}} = -\underline{\boldsymbol{J}}_p\dot{\boldsymbol{q}} + \frac{\Delta t}{2}\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}}\,. \tag{5.37}$$

If used in 5.1 and in 5.3 instead of $\dot{\boldsymbol{e}}^{\text{ctrl}}$ it leads to the joint positions and velocities

$$\dot{\boldsymbol{q}}^{(k+1)} = \dot{\boldsymbol{q}}^{(k)*} = -\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_{\text{PD},0.5,p}^{\text{ctrl}} = \underline{\boldsymbol{J}}_p^{\ddagger}\underline{\boldsymbol{J}}_p\dot{\boldsymbol{q}} - \frac{\Delta t}{2}\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}} \tag{5.38}$$

$$\boldsymbol{q}^{(k+1)} = \boldsymbol{q} + \Delta t\dot{\boldsymbol{q}}^{(k+1)} = \boldsymbol{q} - \Delta t\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_{\text{PD},0.5,p}^{\text{ctrl}} = \boldsymbol{q} + \Delta t\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\boldsymbol{J}}_p\dot{\boldsymbol{q}} - \frac{\Delta t^2}{2}\underline{\boldsymbol{J}}_p^{\ddagger}\underline{\ddot{\boldsymbol{e}}}_p^{\text{ctrl}} \tag{5.39}$$

Now the joint positions (5.39) correspond to the ones of the acceleration-based PD controller (5.23) but the joint velocities (5.38) are weighted with the wrong factor 0.5 in front of the second term. They can be corrected by

$$\dot{\boldsymbol{q}}_{\text{mod}}^{(k+1)} = 2\dot{\boldsymbol{q}}^{(k+1)} - \dot{\boldsymbol{q}}_{\text{mod}} = 2\dot{\boldsymbol{q}}_{\text{mod}} - 2\frac{\Delta t}{2}\boldsymbol{J}^{\ddagger}(\ddot{\boldsymbol{e}}^{\text{ctrl}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}_{\text{mod}}) - \dot{\boldsymbol{q}}_{\text{mod}} \tag{5.40}$$

$$= \dot{\boldsymbol{q}}_{\text{mod}} - \Delta t\boldsymbol{J}^{\ddagger}(\ddot{\boldsymbol{e}}^{\text{ctrl}} + \dot{\boldsymbol{J}}\dot{\boldsymbol{q}}_{\text{mod}})\,. \tag{5.41}$$

This of course needs to be done *after* the integration to the joint positions. However, this modification leads to some instabilities as we observed in sec. 5.4.1.1. Especially with direct constraints on the joint velocities, as it is the case for the trust region constraint, we observed severe numerical instabilities on the modified joint velocities.

It is subject to discussion whether consistency with the acceleration-based problem or consistency with the Lagrange multipliers is preferred. In further research the behaviour of solving the problem (5.1) or (5.3) twice with both the integrations (5.24)

and (5.35) could be investigated. The former one gives the correct velocity while the latter one gives the correct joint positions and Lagrange multipliers corresponding to the true step $\boldsymbol{\Delta q}$. Of special interest is whether the active-sets of both solutions are similar or even identical which would greatly facilitate the computational aspect of solving two hierarchical problems.

For the real robot experiments in the validation section 5.4.2 we use (5.24) without the corresponding modification of the joint positions (5.31). This poses a good compromise between consistency with the acceleration-based problem and the optimization theory.

In Flacco et De Luca [2014] it is proposed to simply control a robot in velocity-based control

$$\dot{\boldsymbol{e}}^{\text{ctrl}} = -\boldsymbol{J}\dot{\boldsymbol{q}}^{(k)*} \tag{5.42}$$

with $\dot{\boldsymbol{e}}^{\text{ctrl}} = -k_p\boldsymbol{e}$ since both velocity-based and acceleration-based control are consistent. However, desired behaviours like PD control can not be realized. That is why we choose to implement the above presented method for acceleration-based control in velocity, solving (5.1) or (5.3) with our new right hand side $\dot{\boldsymbol{e}}^{\text{ctrl}}_{\text{PD}}$ (5.26).

To come back to the 1-D robot example from the previous section 5.2.1, we can now see that damping terms do not influence the critically damped system k_p, k_v. The new velocity for a control step k can be calculated by

$$\dot{x}^{(k+1)} = -\dot{e}^{\text{ctrl}}_{\text{PD}} = J^{-1}J\dot{x} - \Delta t J^{-1}(\ddot{e}^{\text{ctrl}} + \dot{J}\dot{x}) = \dot{x} - \Delta t\ddot{e}^{\text{ctrl}} \tag{5.43}$$

with $J = 1$ and $\dot{J} = 0$. The numerical integration can be formulated as

$$x^{(k+1)} = x + \Delta t\dot{x} - \Delta t^2\ddot{e}^{\text{ctrl}}\,. \tag{5.44}$$

Note that for exponential convergence the original system $\ddot{x} + k_v\dot{x} + k_p x = 0$ still needs to be critically damped with $k_v = 2\sqrt{mk_p}$.

The augmented system's

$$\begin{bmatrix}1\\ \mu\end{bmatrix}\dot{x}^{(k+1)} + \begin{bmatrix}\dot{e}^{\text{ctrl}}_{\text{PD}}\\ 0\end{bmatrix} = \begin{bmatrix}0\\ 0\end{bmatrix} \tag{5.45}$$

least-squares solution for $\dot{x}$ is

$$\dot{x}^{(k+1)} = -\frac{1}{1+\mu^2}\dot{e}^{\text{ctrl}}_{\text{PD}} = \frac{1}{1+\mu^2}(J\dot{x} - \Delta t(\ddot{e}^{\text{ctrl}} + \dot{J}\dot{x}))\,. \tag{5.46}$$

The numerical integration can be formulated as

$$x^{(k+1)} = x + \frac{\Delta t}{1+\mu^2}(\dot{x} - \Delta t\ddot{e}^{\text{ctrl}})\,. \tag{5.47}$$

The original critically damped system k_p, k_v is not influenced by the damping μ which ensures exponential convergence.

5.2.3 Including the dynamics constraints

We have formulated our second order motion controllers in the velocity domain. Similarly, the acceleration components of the equation of motion (2.15) are replaced by the forward differences (5.24):

$$\boldsymbol{M}(\boldsymbol{q})\frac{\dot{\boldsymbol{q}}^{(k)*}-\dot{\boldsymbol{q}}^{(k)}}{\Delta t}+\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})=\boldsymbol{S}\boldsymbol{\tau}^{(k+1)}+\boldsymbol{J}_c^T\boldsymbol{\gamma}^{(k+1)}\,, \tag{5.48}$$

or rewritten into matrix form

$$\begin{bmatrix}\dfrac{\boldsymbol{M}(\boldsymbol{q})}{\Delta t} & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{q}}^{(k)*}\\ \boldsymbol{\tau}^{(k+1)}\\ \boldsymbol{\gamma}^{(k+1)}\end{bmatrix}=\boldsymbol{M}\frac{\dot{\boldsymbol{q}}}{\Delta t}-\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})\,. \tag{5.49}$$

However, for numerical robustness it is desirable to keep the conditioning of the system matrix by

$$\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{q}}^{(k)*}\\ \Delta t\boldsymbol{\tau}^{(k+1)}\\ \Delta t\boldsymbol{\gamma}^{(k+1)}\end{bmatrix}=\boldsymbol{M}\dot{\boldsymbol{q}}-\Delta t\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})\,. \tag{5.50}$$

This computes $\dot{\boldsymbol{q}}^{(k)*}$, $\Delta t\boldsymbol{\tau}^{(k+1)}$ and $\Delta t\boldsymbol{\gamma}^{(k+1)}$, so the solution vector has to be changed accordingly to get $\boldsymbol{\tau}^{(k+1)}$ and $\boldsymbol{\gamma}^{(k+1)}$ by dividing $\Delta t\boldsymbol{\tau}^{(k+1)}$ and $\Delta t\boldsymbol{\gamma}^{(k+1)}$ by Δt.

The equation of motion can be considered full rank if the inertia matrix $\boldsymbol{M}$ is physically consistent and therefore positive definite Udwadia et E. Kalaba [1992]; Udwadia et Schutte [2010]. This means that the system matrix of the equation of motion $\begin{bmatrix}\boldsymbol{M} & -\boldsymbol{S}^T & -\boldsymbol{J}_c^T\end{bmatrix}$ is not concerned with kinematic singularities of the contact Jacobians $\boldsymbol{J}_c$.

For the joint velocities we then get

$$\begin{aligned}\dot{\boldsymbol{q}}^{(k+1)}=\dot{\boldsymbol{q}}^{(k)*}&=\{\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}^+\boldsymbol{M}\dot{\boldsymbol{q}}\}_{1:n} && (5.51)\\ &\quad-\Delta t\{\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}^+\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})\}_{1:n} && (5.52)\\ &=\dot{\boldsymbol{q}}-\Delta t\{\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}^+\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})\}_{1:n} && (5.53)\\ \boldsymbol{q}^{(k+1)}&=\boldsymbol{q}+\Delta t\dot{\boldsymbol{q}}^{(k+1)} && (5.54)\\ &=\boldsymbol{q}+\Delta t\dot{\boldsymbol{q}}-\Delta t^2\{\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S}^T & -\boldsymbol{J}_c^T\end{bmatrix}^+\boldsymbol{N}(\boldsymbol{q},\dot{\boldsymbol{q}})\}_{1:n}\,. && (5.55)\end{aligned}$$

$\{\}_{1:n}$ are the first n entries of the vector in the braces with n being the number of joint variables. The same modifications of the joint positions as described in the previous section 5.2.2 need to be applied. Note that

$$\begin{aligned}&\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}^+\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} && (5.56)\\ &=\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}^+\begin{bmatrix}\boldsymbol{M}(\boldsymbol{q}) & -\boldsymbol{S} & -\boldsymbol{J}_c^T\end{bmatrix}\begin{bmatrix}\dot{\boldsymbol{q}}\\ \boldsymbol{0}\\ \boldsymbol{0}\end{bmatrix}=\begin{bmatrix}\dot{\boldsymbol{q}}\\ \boldsymbol{0}\\ \boldsymbol{0}\end{bmatrix} && (5.57)\end{aligned}$$

if the pseudo inverse is a left inverse.

In the hierarchy dynamic constraints including the equation of motion and torque and contact force limits have the highest priority. Furthermore, we do not apply the linearization described in sec. 4.1.2 on the equation of motion. It is already linear in the accelerations (or velocities in case of the forward integration), joint torques and generalized contact wrenches. It therefore fits into our lexicographical problem (2.34) with the system matrix $\boldsymbol{A} = \begin{bmatrix}\boldsymbol{M} & -\boldsymbol{S}^T & -\boldsymbol{J}_c^T\end{bmatrix}$ and the right hand side $\boldsymbol{b} = -\boldsymbol{M}\dot{\boldsymbol{q}} + \Delta t \boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}})$. We consequently do not consider it in the hierarchical gradient and Hessian calculation of the lower level linearized constraints.

5.3 Computational performance

The problem (5.1) or (5.3) is built and solved in a matter of several hundred microseconds. Yet the real computational challenge is the active-set method which might require solving (5.1) or (5.3) several times until the optimal active-set is determined. Especially in dynamically challenging motions, e.g. contact switching, robot falling or being close to falling with almost loosing closure of the friction contacts... we observed cases of above hundred active-set iterations. This is caused by an interplay between the dynamic constraints and the trust region constraint and requires further investigation.

Since making the solver Dimitrov et al. [2015] faster is highly involved (for example updating the QR-decomposition factors after an active-set change or improving the active-set search itself) we resort to a makeshift in which we stop the active-set method once a certain level is optimal enough. Usually this would be the contact constraints since we do not want to compromise on their optimality and risk falling.

After every active-set iteration we decide upon several criteria whether we stop the active-set search:

- Measure the quadratic norm of the slack $\boldsymbol{w}$ of the decision level and check if it is below a certain numerical threshold. We choose it identically to the threshold ν of the switching method between the GN algorithm and Newton's method.
- We have gone through a certain number of iterations in the optimality phase (10 iterations). The optimality phase starts with the first deactivation of a constraint.
- We are above a certain number of overall active-set iterations (30 iterations).

If so, we exit the solver and are satisfied with the current solution. At this point, the robot motion is physically feasible and all the hard constraints like contact wrench, joint torque and joint limits, the trust region constraint and the contact constraints are at their optimum. However, lower level constraints are only solved sub-optimally. In the following control iteration the solver is warm started with the found intermediary active-set and the active-set search is continued. Due to the continuity property Escande et al. [2014] good behaviour is guaranteed in practice.

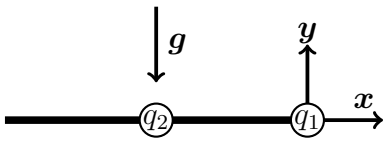


***Fig. 5.2. Example 1**, initial configuration of the two link two joint robot.*

Hierarchy for example 1	
1.	• Equation of motion, acceleration-based or velocity-based with accelerations integrated by 5.24 or by 5.35 • $\boldsymbol{\tau} = \mathbf{0}$
2.	• Regularization term $\dot{\boldsymbol{q}} = \mathbf{0}$ (vel. based), expressed with accelerations by using forward differences (5.24) or (5.35) in case of acceleration-based control

***Fig. 5.3. Example 1**. Hierarchy for swinging pendulum.*

Since cases of high active-set iterations only occur over a limited number of control iterations the optimality of lower priority levels is restored quickly in subsequent control iterations Escande et al. [2014].

We also use a slightly modified version of Dimitrov et al. [2015] where we only restart the decomposition from the level where the active-set change occurred (see chapter 6; the bound handling was not enabled during the experiments).

5.4 Validation

In this section we first conduct three simulations in order to confirm the behaviour of this chapter's derivations, see sec. 5.4.1. Later, we transfer the method from small 2-D robots to real robot experiments with the HRP-2Kai humanoid robot, see sec. 5.4.2[1].

5.4.1 Three examples in simulation

5.4.1.1 Example 1: Freely swinging pendulum

In this example we let a 2-D robot with two links (each link with unit length and unit mass) and two revolute joints and fixed base swing freely. The initial configuration is $[-\pi/2, 0]$ rad with zero velocity (see fig. 5.2). The motion of the pendulum is determined by solving the instantaneous equation of motion in a least squares sense. The joint torques of the two joints are asked to be zero to allow the swinging pendulum motion. We then solve the problem given in fig. 5.3.

[1]The accompanying video can be found at https://youtu.be/6ClRlODUdys

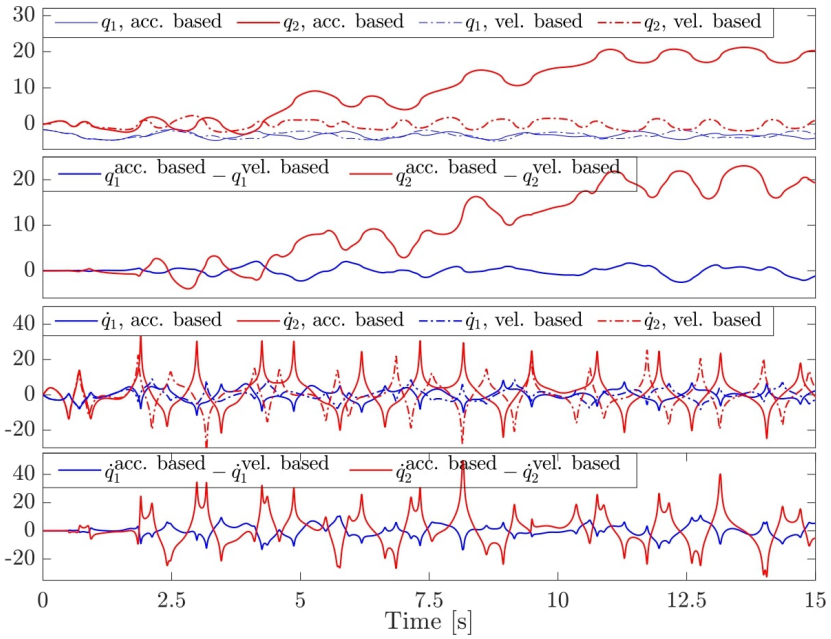


Fig. 5.4. Example 1. *Joint positions and velocities of the swinging pendulum for the acceleration-based ('acc. based') equation of motion and the velocity-based ('vel. based') one where the accelerations are replaced by (5.24). The first and third graphs show the positions / velocities of the acceleration- and velocity-based approach. The second and fourth graphs show the difference between the values. For the first second the values are close but start to diverge from each other afterwards.*

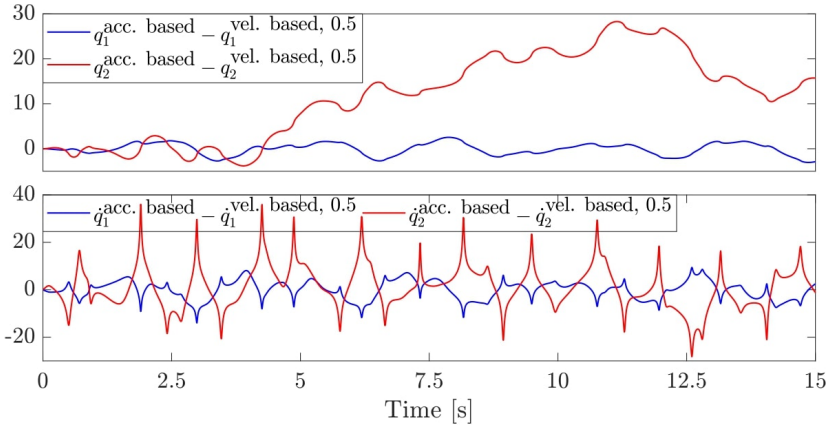


Fig. 5.5. Example 1. *The upper graph shows the difference of the joint positions of the swinging pendulum in the case of the acceleration-based ('acc. based') equation of motion and the velocity-based ('vel. based') one. The accelerations are replaced by (5.35) ('0.5'). The values immediately diverge from each other. The lower graph shows the difference in joint velocity.*

We compare the behaviour of the acceleration-based equation of motion as reference with the velocity-based ones where the acceleration is replaced by forward differences

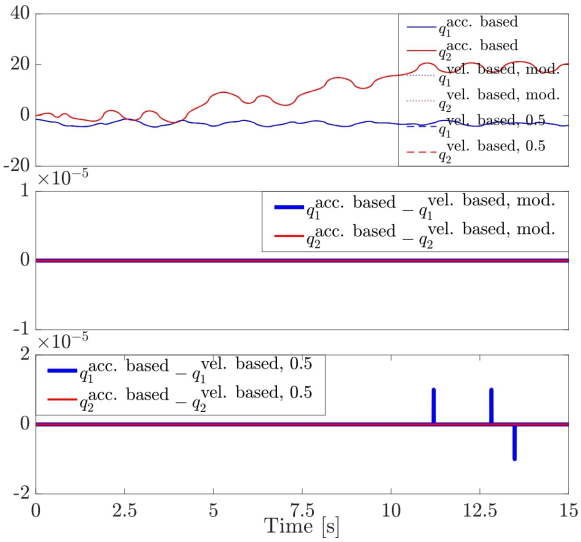


Fig. 5.6. Example 1. *The upper graph shows the joint positions of the swinging pendulum for the acceleration-based ('acc. based') equation of motion and the velocity-based ('acc. based') ones where the accelerations are replaced by either (5.24) with modified ('mod.') joint positions according to (5.31) or replaced by (5.35) ('0.5') with modified joint velocities according to (5.40). The two lower graphs show the difference between the values. A very good match is achieved thanks to the modifications.*

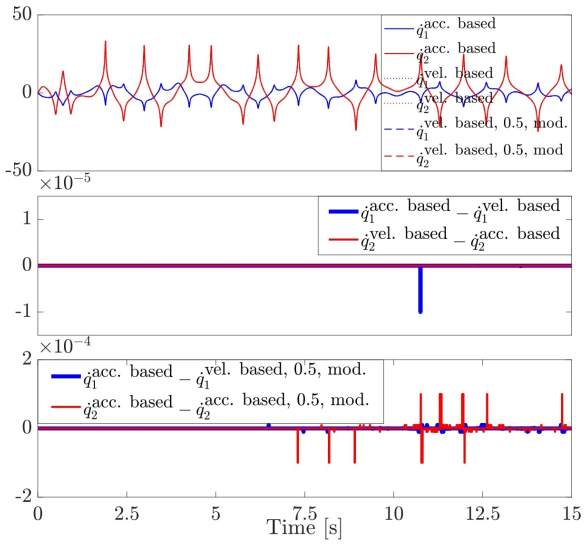


Fig. 5.7. Example 1. *The upper graph shows the joint velocities of the swinging pendulum for the acceleration-based equation of motion ('acc. based') and the velocity-based ('vel. based') ones where the accelerations are replaced by either (5.24) with modified ('mod.') joint positions according to (5.31) or (5.35) ('0.5') with modified joint velocities according to (5.40). The two lower graphs show the difference between the values. A very good match is achieved thanks to the modifications. However, the modification of the joint velocities (5.40) comes with numerical noise with an increasing frequency over time.*

either given in (5.24) or (5.35). For both replacement schemes of the acceleration it can be observed that the joint positions and velocities diverge from the reference ones (see fig. 5.4 for (5.24) and fig. 5.5 for (5.35)) While there is a good match for the first few seconds for (5.24) (see fig. 5.4), the wrong velocities from (5.35) lead to an immediate and strong deviation from the reference due to the wrong calculation of $\boldsymbol{N}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ (see fig. 5.5).

However, if we apply the modifications of the joint positions or velocities according to (5.31) or (5.40) we can perfectly reproduce the behaviour of the acceleration-based equation of motion in the velocity domain (see fig. 5.6 for the joint positions and fig. 5.7 for the joint velocities). Slight numerical instabilities can be observed in the lower graph of fig. 5.7 for the joint velocities modified according to (5.40).

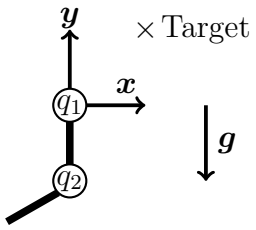


***Fig. 5.8. Example 2**, initial configuration of the two link two joint robot. The desired end-effector position is at the cross.*

Hierarchy for example 2

1. Equation of motion, acceleration-based or velocity-based with accelerations integrated by 5.24
2. Joint velocity limit (corresponds to the trust region constraint), expressed with accelerations by using forward differences (5.24) in the case of acc. based control
3. End-effector task with $\ddot{\boldsymbol{e}}^{\text{ctrl}}$ (acc. based) or with $\dot{\boldsymbol{e}}^{\text{ctrl}}_{\text{PD}}$ (vel. based)
4. $\dot{\boldsymbol{q}} = \mathbf{0}$ (vel. based), expressed with accelerations by using forward differences (5.24) in case of acceleration-based control
5. $\boldsymbol{\tau} = \mathbf{0}$

***Fig. 5.9. Example 2**, hierarchy.*

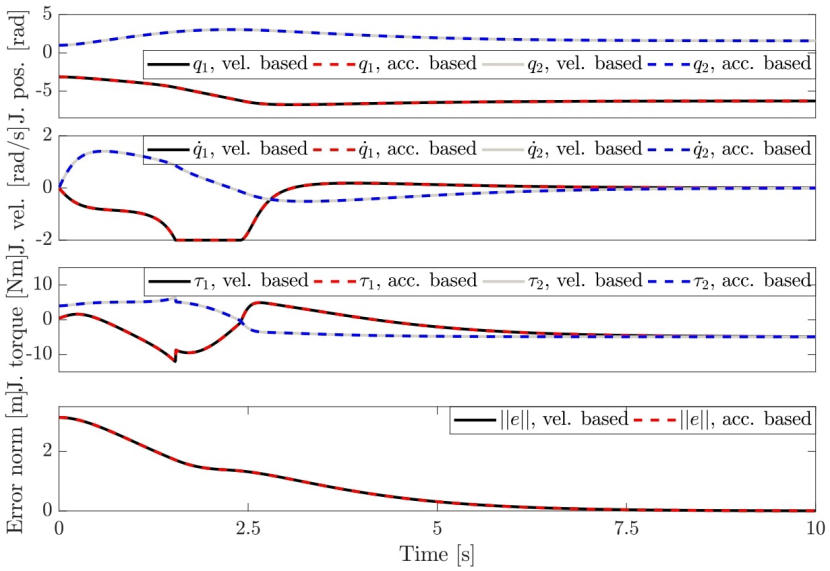


***Fig. 5.10. Example 2**, joint (J.) positions, joint velocities, joint torques and task error norm. They are identical for the acceleration- and velocity-based equation of motion and PD motion controllers $\ddot{\boldsymbol{e}}^{ctrl}$ and $\dot{\boldsymbol{e}}^{ctrl}_{PD}$.*

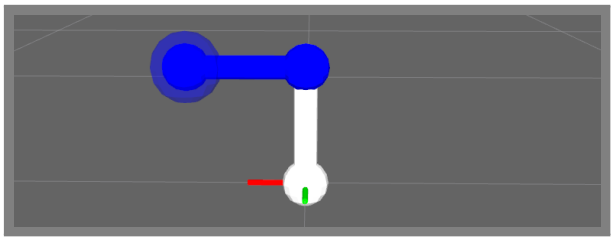


***Fig. 5.11. Example 2**, converged robot posture.*

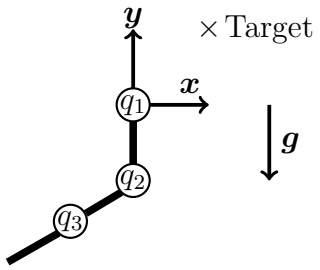


***Fig. 5.12. Example 3**, initial configuration of the three link three joint robot. The desired end-effector position is at the cross.*

5.4.1.2 Example 2: In reach end-effector task

In this simulation we make the robot from the first example 5.4.1.1 reach for the point $[1, 1]$ m. The initial robot posture corresponds to the arbitrary non-singular configuration $[-\pi, 1]$ rad, see fig. 5.8. We define the acceleration- or velocity-based hierarchy in fig. 5.9.

The problem is solved by the GN algorithm and all accelerations are replaced by 5.24 with the modifications of the joint positions by (5.31) in the case of velocity-based control. The joint positions, joint velocities, joint torques and task error norms are given in fig. 5.10. It can be seen that both the acceleration and velocity-based problem lead to identical behaviour. The converged robot posture is given in fig. 5.11.

5.4.1.3 Example 3: Conflict between linearized task constraint and dynamics constraint

In this simulation we want to test the behaviour of the robot if a linearized end-effector task is in algorithmic conflict with a (non-linearized) dynamics constraint. Numerical instabilities due to algorithmic singularities with higher level non-linearized tasks only occur if the higher priority task forces kinematic subchains of the lower priority linearized end-effector task into kinematic singularity in order to minimize the task error. Therefore, we require at least a three link three joint robot. We set its initial configuration to $[-\pi, 1, 0]$ rad, see fig. 5.12. The commanded joint torque of joint 1 is set to zero and therefore conflicts with the end-effector task of reaching the point $[1, 1]$ m. A trust region constraint is introduced (also for the acceleration-based

Hierarchy for example 3

1. • Equation of motion, acceleration-based or velocity-based with accelerations integrated by 5.24
   • $\tau_1 = 0$
2. Trust region constraint, expressed with accelerations by using forward differences (5.24) in the case of acc. based control
3. End-effector task with $\ddot{\boldsymbol{e}}^{\text{ctrl}}$ (acc. based) or with $\dot{\boldsymbol{e}}^{\text{ctrl}}_{\text{PD}}$ (vel. based), solved by the GN algorithm, Newton's method (vel. based) or the LM algorithm (acc. based)
4. $\dot{\boldsymbol{q}} = \boldsymbol{0}$ (vel. based), expressed with accelerations by using forward differences (5.24) in case of acceleration-based control
5. $\boldsymbol{\tau} = \boldsymbol{0}$

***Fig. 5.13. Example 3**, hierarchy.*

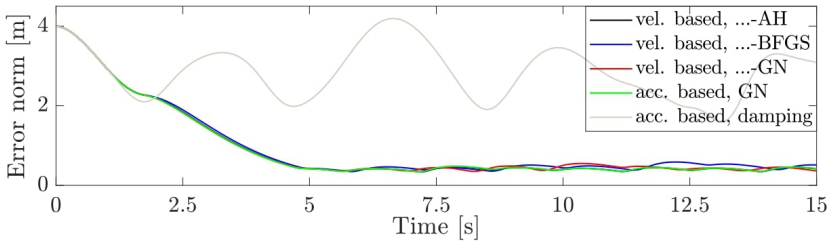


***Fig. 5.14. Example 3**, task error norm for the end-effector task. Damping in the acceleration domain leads to overshooting behaviour and the error is barely minimized (dark gray curve). The velocity-based PD controller $\dot{\boldsymbol{e}}^{ctrl}_{PD}$ prevents this behaviour.*

control) in order to be able to apply Newton's method with second order information either from LexLSAug2AH or LexLSAug2BFGS in the velocity-based control. The hierarchy is given in fig. 5.13.

The norm of the distance of the end-effector from the target is given in fig. 5.14. As we have discussed in sec. 5.2.1, damping in the acceleration domain leads to slowly oscillating behaviour as can be observed from the error curve 'acc. based, damping'. The damping value was set to $\mu = 0.1$. The error norm of the other methods gets minimized fairly well but is disturbed by the slow oscillations of the freely swinging joint 1. The joint torque of joint 1 is zero as can be seen in fig. 5.15.

Figure 5.16 shows the joint velocities of the robot. For the GN algorithm (both acc. and vel. based) numerical instabilities can be observed. This is due to the zero commanded joint torque of joint 1 which conflicts with the aim of getting closer to the target with the end-effector. In order to minimize the task error as much as possible the kinematic subchain consisting of link 2 and link 3 is forced into kinematic singularity .

Newton's method with second order information from both LexLSAug2AH and

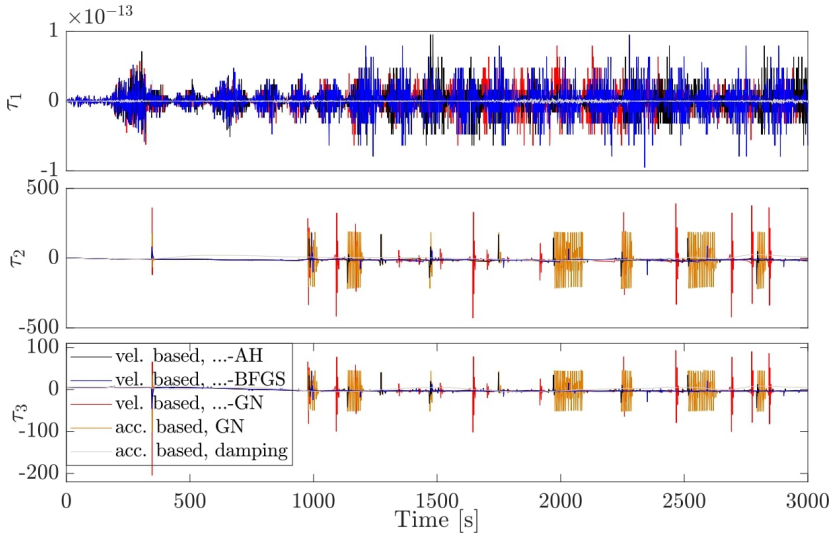


***Fig. 5.15. Example 3**, joint torques. The acc. and vel. based GN algorithm leads to numerical instabilities in the joint torques of joint 2 and 3. LexLSAug2AH and LexLSAug2BFGS are numerically stable.*

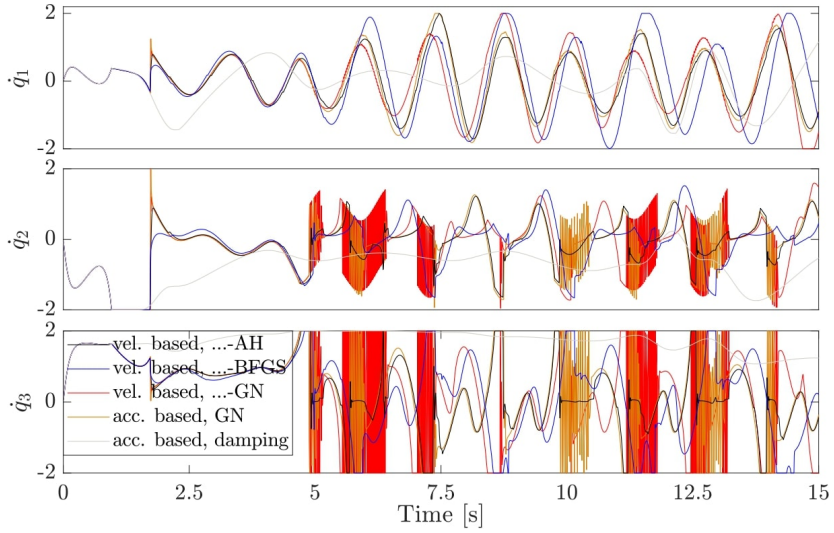


***Fig. 5.16. Example 3**, joint velocities. The acc. and vel. based GN algorithm leads to numerical instabilities in the joint torques of joint 2 and 3. They occur whenever the kinematic subchain consisting of link 2 and link 3 is close to kinematic singularity. LexLSAug2AH and LexLSAug2BFGS are numerically stable. Damping in the acceleration domain is numerically stable but leads to the mentioned overshooting behaviour.*

LexLSAug2BFGS leads to numerically stable joint velocity behaviour without the overshooting end-effector characteristics seen for the damping in acceleration-based control. The robot configuration with low error norm but some remaining swaying motion is given in fig. 5.17. Note that both LexLSAug2AH and LexLSAug2BFGS do not take

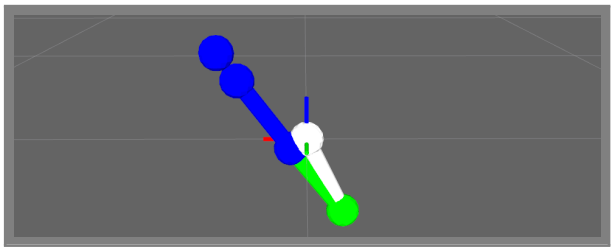


Fig. 5.17. Example 3. *The robot managed to come to the target as close as possible (LexLSAug2BFGS). However, some swaying motion remains due to the freely swinging joint 1 (white ball) with zero commanded torque. Note how the kinematic subchain of the green and blue link is in kinematic singularity.*

Hierarchy for LexLSAug2BFGS and LexLSAug2AH

1. • $4n_c$ (n_c: number of contacts) bounds on generalized contact wrenches: $\boldsymbol{\gamma} > \mathbf{0}$
   • 32 joint limits using a velocity damper Faverjon et Tournassoud [1987]
2. • 38 integrated equations of motion
   • 38 torque limits
3. 38 trust region limits
4. 3 **inequality constraints for self-collision avoidance**
5. 18 (12 for exp. 1) **geometric contact constraints**
6. 1 equality constraint on the head yaw joint to put the vision marker into the field of view, not for exp. 1
7. 3 **inequality constraints on the CoM**
8. • 6 **end-effector equality constraints for left and right hand**
   • 3 **equality constraints to keep the chest orientation upright, not in exp. 1**
9. 3 **stricter inequality constraints on the CoM, not for exp. 1.**
10. 38 constraints to minimize joint velocities: $\dot{\boldsymbol{q}} = \mathbf{0}$
11. $\boldsymbol{4n_c}$ constraints to minimize the generalized contact wrenches: $\boldsymbol{\gamma} = \mathbf{0}$

Fig. 5.18. Exp. 1 to exp. 3*, hierarchy for LexLSAug2BFGS and LexLSAug2AH.*

into account the equation of motion or the dynamics constraints in the hierarchical gradient or Hessian calculation.

Hierarchy for WLS

1. • $4n_c$ bounds on generalized contact wrenches: $\boldsymbol{\gamma} > \mathbf{0}$
   • 32 joint limits using an acceleration damper Faverjon et Tournassoud [1987]
   • 38 equations of motion
   • 38 torque limits
   • 3 inequality constraints for self collision avoidance
   • 18 (12 for exp. 1) geometric contact constraints
   • 3 inequality constraints on the CoM
2. • 6 end-effector equality constraints for left and right hand
   • 32 equality constraints to maintain a reference posture (with the head yaw joint turned towards the vision marker, exp. 2 and exp. 3)
   • $4n_c$ constraints to minimize the generalized contact wrenches: $\boldsymbol{\gamma} = \mathbf{0}$

***Fig. 5.19. Exp. 1** to **exp. 3**, hierarchy for WLS.*

5.4.2 Three real robot experiments

Now that we have validated our derivations from this chapter in simulation we move on to conduct three experiments with the position controlled HRP-2Kai robot with 32 DoF (plus 6 DoF for the un-actuated free-flyer), 1.71 m height and 2.11 m arm span. As our solver we use LexLSI Dimitrov et al. [2015] which solves problem (5.1) or possibly (5.3) with second order augmentation from LexLSAug2AH or LexLSAug2BFGS. LexLSAug2SR1 is not evaluated due to its similar but worse behaviour in terms of numerical stability and smoothness compared to LexLSAug2AH. Its slight advantage in computational speed is negligible (see sec. 4.2.1.2). LexLSI is based on the active-set method and we warm start it at every control iteration with the active-set found in the previous control iteration. As mentioned in sec. 5.2.2, the accelerations are replaced by (5.24) but the joint positions are not modified according to (5.31). We then solve for the variables $\dot{\boldsymbol{q}}^{(k+1)}$, $\Delta t\boldsymbol{\tau}^{(k+1)}$ and $\Delta t\boldsymbol{\gamma}^{(k+1)}$. The resulting velocities are integrated to the joint positions (2.7) which are sent to the robot with a control frequency of 200 Hz ($\Delta t = 5$ ms). The corresponding hierarchy is given in fig. 5.18.

Constraints in bold are considered in the calculation of the hierarchical Hessian. Dynamic constraints and bound constraints are not taken into account. Note that for bound constraints we generally have $\boldsymbol{B} = \mathbf{0}$ and $\boldsymbol{H} = \mathbf{0}$.

The hierarchical separation of the bound constraints on the generalized contact wrenches and the joint limits from the equation of motion allows LexLSI to cheaply handle the variable bounds on the first level.

The trust region constraint is necessary to ensure that the new step $\boldsymbol{\Delta q} = \Delta t\dot{\boldsymbol{q}}^{(k+1)}$

is bounded within a certain neighbourhood Δ of the current state $\boldsymbol{q}$, $\|\boldsymbol{\Delta q}\|_\infty < \Delta$. In this region we 'trust' the second order Taylor approximation of $\Phi(\boldsymbol{q})$ (4.7) to be accurate enough. We set $\Delta = 0.01$ rad or m.

Note that the trust region constraint and the joint velocity minimum norm task include the none actuated free-flyer. Indeed, the free-flyer is numerically relevant as it is present in all the Jacobians. Therefore, we put the constraint only after the equation of motion to avoid constricting the free-flyer velocity for example in the case of a robot free fall.

As comparison we use the least squares solver LSSOL Gill et al. [1986] which is also used in our laboratory's robot control framework. It has a constraint (level 1) and an objective level (level 2). Thereby, inequality constraints are only allowed on level 1. It is also based on the active-set method and we warm start it with the active-set found in the previous control iteration. On both levels a soft hierarchy can be established by weighting tasks against each other (therefore we call this solver **W**eighted **L**east **S**quares - WLS). The constrained robot problem is then defined in fig. 5.19 following previous works Abe et al. [2007]; Collette et al. [2007]; Bouyarmane et Kheddar [2011]; Feng et al. [2013]; Kuindersma et al. [2014]; Vaillant et al. [2016]; Pfeiffer et al. [2017] which make use of control frameworks based on weighted constrained LSP's.

The hierarchy for WLS contains the dynamic constraints (equation of motion, $\boldsymbol{\gamma}$ bounds, $\boldsymbol{\tau}$ bounds), joint limits, self-collision avoidance, CoM task and contact tasks as constraints on level 1. All these tasks have the same priority without weighting. Since the notion of constraint relaxation is not introduced in LSSOL, the feasibility of these constraints has to be guaranteed in order to avoid solver failures. Especially the self-collision avoidance, the contact and the CoM constraints are the source of potential conflict or even of (unresolved) kinematic singularities. This highlights the significance of being able to easily design safe robot problems with LexLSAug2BFGS or LexLSAug2AH.

The reaching task is defined as an objective on level 2. A posture reference task is also added at the objective level. Similarly to the LM algorithm it acts as a velocity damper, allowing to approach singular configurations. The task is added with a low weight $1e^{-3}$ ($5e^{-2}$ for exp 2 and exp 3), which needs to be tuned depending on the task to be performed. For example reaching for very far away targets requires a higher weight. Additionally, another task $\boldsymbol{\gamma} = \mathbf{0}$ on the objective level is added to yield a fully determined and full rank problem. It has a small weight $1e^{-5}$ ($1e^{-4}$ for exp. 2 and exp.3).

We solve for the variables $\ddot{\boldsymbol{q}}^{(k+1)}$, $\boldsymbol{\tau}^{(k+1)}$ and $\boldsymbol{\gamma}^{(k+1)}$ and integrate the accelerations twice to the joint positions (2.13) which are then sent to the robot.

For both solvers we substitute the torques $\boldsymbol{\tau}^{(k+1)}$ with the equation of motion which reduces the number of variables from 108 to 70 for exp. 1 and 112 to 74 for exp. 2 and exp. 3 respectively.

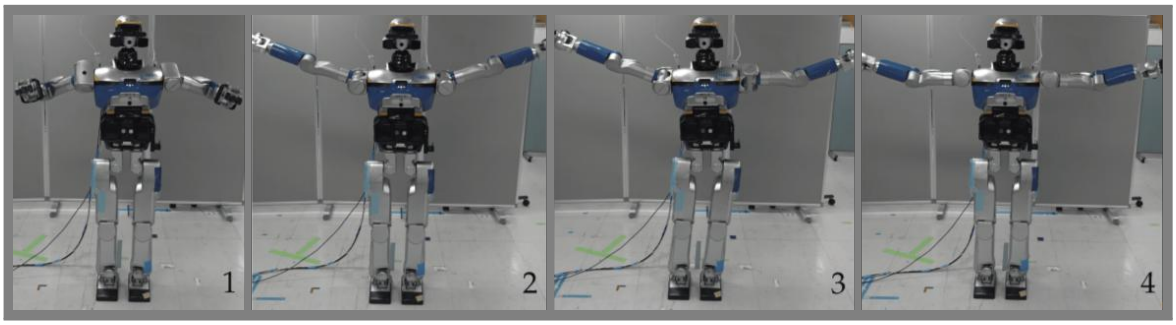


***Fig. 5.20. Exp. 1**, from left to right: 1.: LexLSAug2BFGS, the robot has moved both its hands to the first target, 2.: LexLSAug2BFGS, the robot has converged with both arms and both legs stretched 3.: LexLSAug2AH, the robot has converged with one arm and both legs stretched, 4.: WLS, the robot has converged with only the legs stretched.*

5.4.2.1 Experiment 1: Stretching

In the first experiment (exp. 1, see Fig. 5.20) the robot is controlled to take a simple stretching pose. It shows that LexLSAug2BFGS and LexLSAug2AH enable numerically stable robot behaviour even with the presence of kinematic and algorithmic singularities. At the same time the task error norm is minimized to a higher degree w.r.t WLS.

For this we establish two contacts with the environment on the two feet standing on the ground. Each foot consists of four contact points (overall $n_c = 8$) placed on each corner of the rectangular foot.

The robot then continues to move the left hand to $[0.4, 0.5, 1.3]$ m and the right hand to $[0.4, -0.5, 1.3]$ m which is on each side of the robot's chest. After that it targets $[0.4, -2, 2]$ m and $[0.4, 2, 2]$ m which are both out-of-reach positions on its left top and right top side respectively. This forces the robot to take a stretched configuration along the convergence process. At this point, the end-effector tasks are in algorithmic singularity due to conflict with the geometric contact constraints.

Thereby, LexLSAug2BFGS shows the best minimization of the norm of the Euclidean distance of the left and right hand to their targets (see Fig. 5.21). Both legs and arms are fully stretched. Consequently, the contact Jacobians of the left and right foot are singular. This seemingly does not influence the robot's joint velocity (see Fig. 5.22), joint torque (see Fig. 5.30) and contact wrench (see Fig. 5.32) behaviour due to the full rank property of the equation of motion.

The convergence is the longest out of the three solvers as there is a large period of flat curvature with very small $\boldsymbol{y}^T \boldsymbol{\Delta q}^{(k-1)} \approx 1e^{-16}$ (between ca. 45 s and 90 s). This leads to slow joint motions since the BFGS Hessian approximation is not updated but rather remains as a static damping term similar to the LM algorithm.

The joint velocities are fairly smooth (see Fig. 5.22) and pose no problem on the real robot. If for some end-user the movements are too jerky a weighted and dotted identity matrix $\alpha \boldsymbol{I}^*$ could be added to $\boldsymbol{B}$. However, this has a light side effect on the

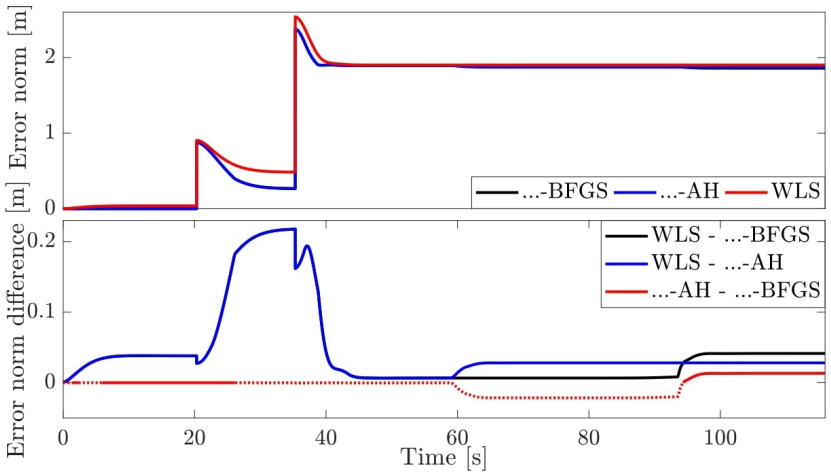


***Fig. 5.21. Exp. 1**, comparison of sum of the error norms of the right and the left hand. LexLSAug2BFGS has a final error of 1.86 m, LexLSAug2AH of 1.87 m and WLS of 1.9 m. The lower graph shows the differences of the error norms of the different methods. The data of LexLSAug2AH and WLS is synchronized with the one of LexLSAug2BFGS.*

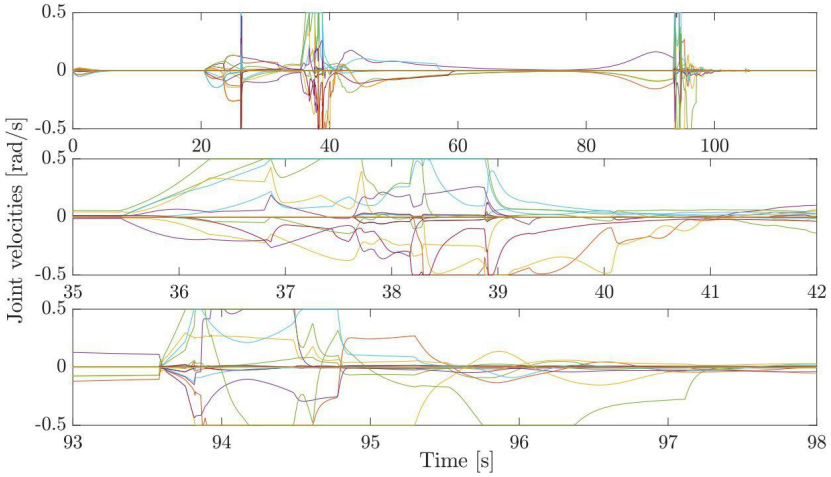


***Fig. 5.22. Exp. 1**, LexLSAug2BFGS, joint velocities.*

convergence.

The switch to Newton's method is taking place on the way to the first way point at around 25 s. There is a quick switch to Newton's method and back at around 105 s on the left and right foot position task but does not further influence the robot behaviour (see Fig. 5.23).

For most of the time there is only one active-set iteration per control iteration, yielding computation times well below 5 ms. However, at approximately 40 s there is a peak in active-set iterations of 17 with a loop time of just below 5 ms (see Fig. 5.24). Other peaks are computational artefacts on the home PC which was used to remote control the robot (PC Intel Core i7-4720HQ CPU @ 2.60GHz with 8 GB of RAM).

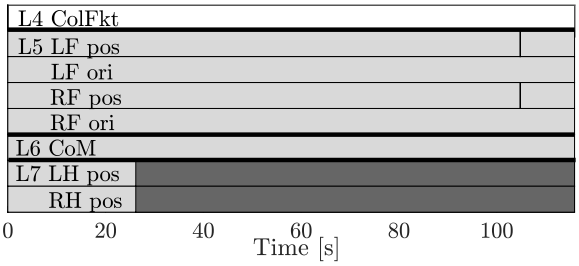


***Fig. 5.23. Exp. 1**, LexLSAug2BFGS, map of activity (light gray) and Newton's method (dark gray).*

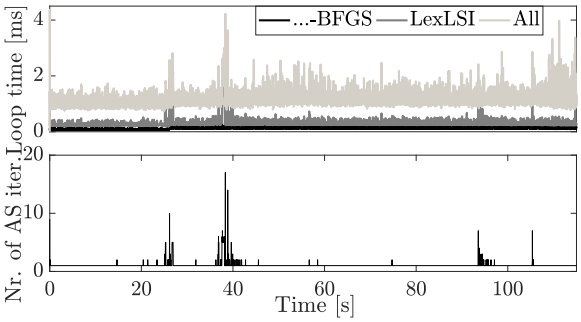


***Fig. 5.24. Exp. 1**, LexLSAug2BFGS, computation times and number of active-set iterations. LexLSAug2BFGS peaks at 376 µs, LexLSI at 3.55 ms and overall at 4.23 ms. The maximum number of active-set iterations is 17 iterations.*

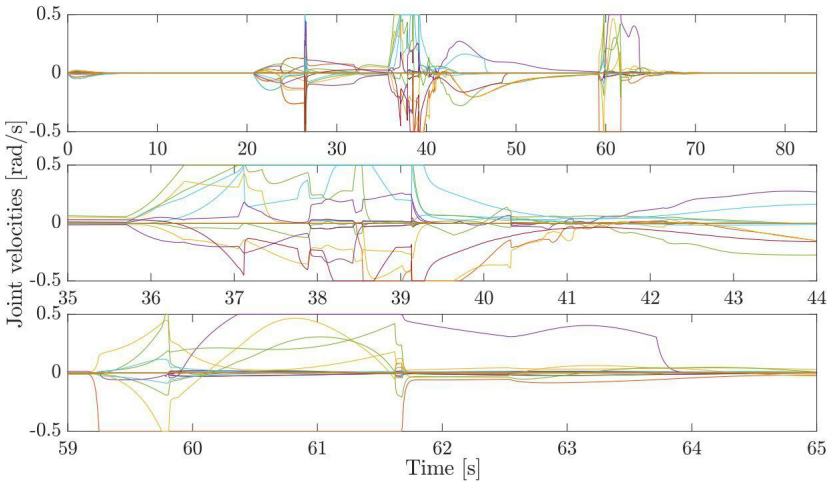


***Fig. 5.25. Exp. 1**, LexLSAug2AH, joint velocities.*

Similar behaviour is seen for LexLSAug2AH but it converges faster to a less optimal minimum (see Fig. 5.21 for the task error norm and Fig. 5.25 for joint behaviour)

In Fig. 5.27 a clear increase in computation times can be seen at around 26 s when the second order augmentation and the corresponding Higham regularization with SVD decomposition start (see Fig. 5.26).

Note that the Hessian is computed all the time even if there is no augmentation. Its computation time is included in 'All'. Computing the CoM Hessian of the 38 DoF HRP-2Kai robot takes about 150 μs and calculating all the Hessians accounts for approximately less than 1 ms.

The WLS method shows the worst convergence (see fig. 5.21). Especially during the first motion the robot fails to lift both its arms up due to conflict with the posture reference task. The joint velocities are very smooth but slow (see fig. 5.28) which is

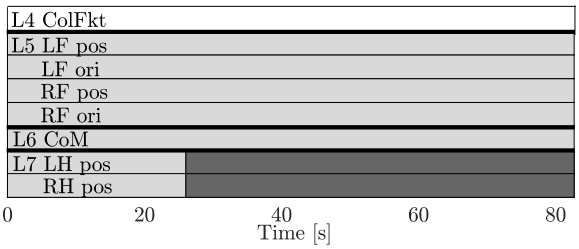


***Fig. 5.26. Exp. 1**, LexLSAug2AH, map of activity (light gray) and Newton's method (dark gray).*

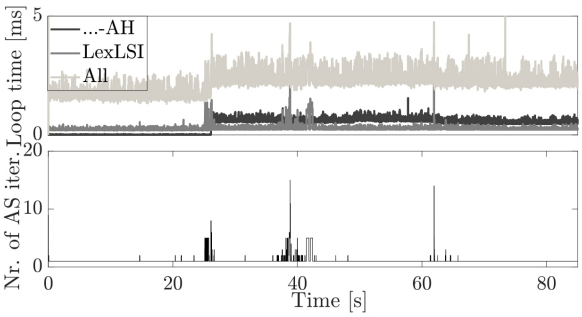


***Fig. 5.27. Exp. 1**, LexLSAug2AH, computation times and number of active-set iterations. LexLSAug2AH peaks at 1.55 ms, LexLSI at 2.96 ms and overall at 4.75 ms. The maximum number of active-set iterations is 15.*

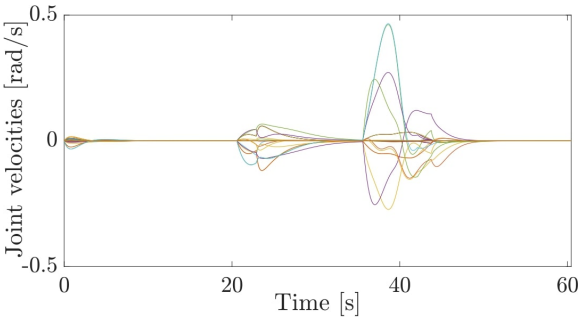


***Fig. 5.28. Exp. 1**, WLS, joint velocities.*

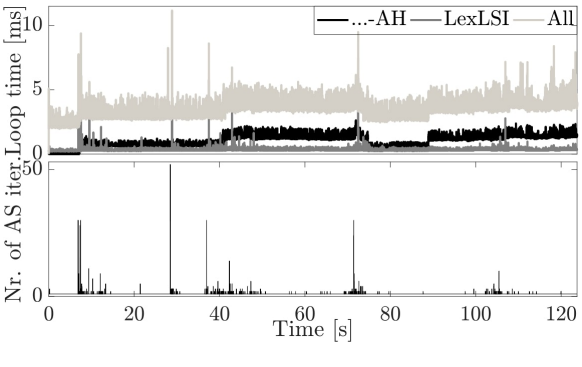


***Fig. 5.29. Exp. 1**, WLS, computation times and number of active-set iterations. LSSOL peaks at 3.76 ms and overall at 4.46 ms with 4 active-set iterations.*

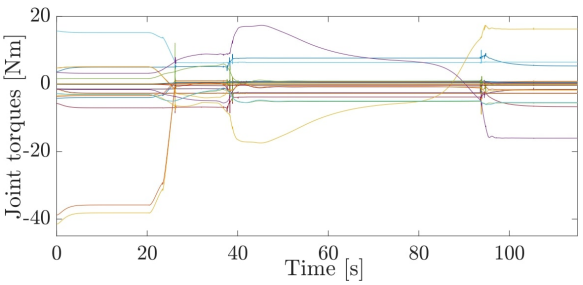


***Fig. 5.30. Exp. 1**, LexLSAug2BFGS, joint torques.*

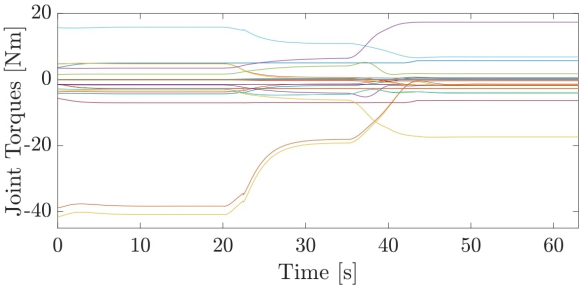


***Fig. 5.31. Exp. 1**, WLS, joint torques.*

behaviour typically seen for the LM algorithm. Consequently, there are less active-set iterations than seen for LexLSAug2BFGS and LexLSAug2AH due to less abrupt

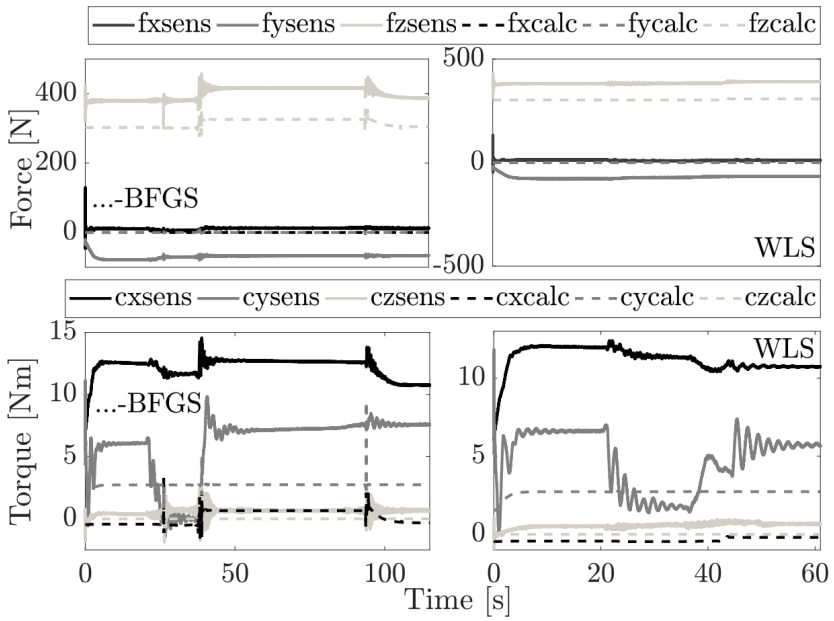


Fig. 5.32. Exp. 1*, comparison of the ground reaction forces (top) and torques (bottom) of the left foot between LexLSAug2BFGS (left column) and WLS (right column), measured (sens) and computed (calc). They are of similar magnitude.*

motion changes.

The computation times for LSSOL for a single iteration are a multiple of the ones of LexLSI. However, it only increases slightly with a higher number of active-set iterations. The reason is that LSSOL cheaply updates decomposition factors for the current active-set which have been computed for the previous one (see Fig. 5.29).

Figures 5.30 and 5.31 show the computed joint torques for LexLSAug2BFGS and WLS. They are approximately of the same magnitude and show the validity of our approach of forward integrating the accelerations in the equation of motion. The joint torques for LexLSAug2AH are very similar to the ones of LexLSAug2BFGS and are therefore not depicted here.

Figure 5.32 shows a comparison of the computed and measured ground reaction forces and torques of the left foot for LexLSAug2BFGS and WLS. They are approximately of the same magnitude and once more show the validity of our approach of forward integrating the accelerations in the equation of motion. The discrepancy between measured and sensed values comes to a large extent from the un-modelled elasticity of the ankle pitch joints but is consistent for LexLSAug2BFGS and WLS. The ground reaction forces and torques for LexLSAug2AH are very similar to the ones of LexLSAug2BFGS and therefore are not depicted here.

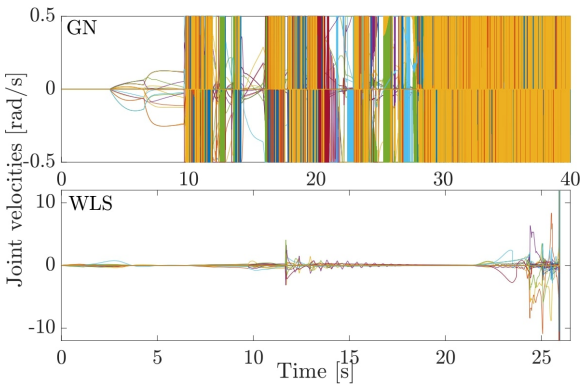


Fig. 5.33. Exp. 1 *without or badly-tuned augmentation, joint velocities. The pure GN algorithm suffers from important joint oscillations while WLS with only 1/1000 of the original weight for the posture reference task fails completely at 25.9 s.*

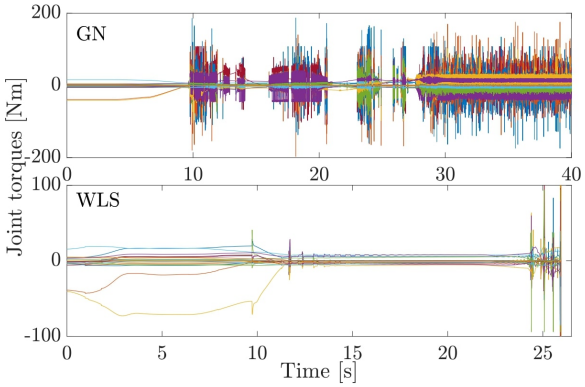


Fig. 5.34. Exp. 1 *without or badly-tuned augmentation, joint torques. Upper graph: the joint oscillations require high motor torques. Lower graph: Joint torques for WLS with a weight of* $1e^{-6}$ *for the posture reference task.*

5.4.2.2 Experiment 1 without or badly-tuned augmentation

For reference we show the robot joint behaviour for exp. 1 in the case of the pure GN algorithm without the switch to Newton's method in case of singularities.

Since singular behaviour would be highly harmful to the robot we conducted this experiment only in simulation.

The robot is clearly numerically unstable on joint level (see Fig. 5.33) and sways heavily. The reason is that joint torque limits are reached repeatedly (see Fig. 5.34) since the joint oscillations require very high motor torques.

Additionally, exp. 1 is conducted with WLS where the reference posture task is weighted with only 1/1000 of its original weight. The weight used is then $1e^{-6}$. While there are no joint oscillations the robot sways slightly until LSSOL fails completely at around 26 s with numerically high joint velocities due to approaching a singular configuration.

5.4.2.3 Experiment 2: Stretching with a third contact

The second experiment (exp. 2, see Fig. 5.35) is designed in such a way that a third contact between the left hand and a rigid pole must be established in order to prevent falling. This contact is not a friction contact but a fixed contact such that we only need to define one contact point here (overall $n_c = 9$). Also, the bound $\boldsymbol{\gamma} > \mathbf{0}$ needs to be omitted for this contact since we allow negative contact forces if the robot is 'hanging' from the pole.

In the following we describe the experiment for LexLSAug2BFGS.

At first, the robot moves its left hand in front of a vertical metal pole which is supposed to be grabbed at approximately $[0.5, 0.25, 1]$ m. The exact pole position

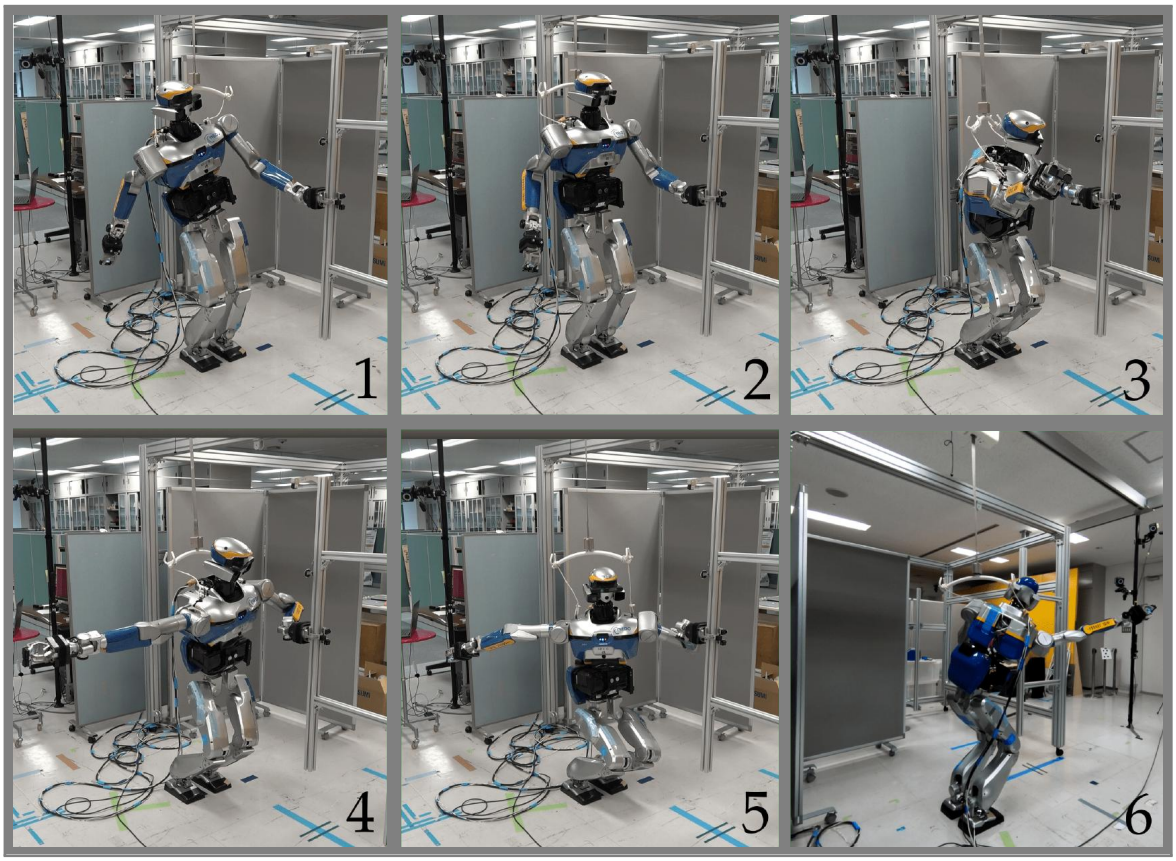


Fig. 5.35. *Pictures of HRP-2Kai performing* ***exp. 2****, LexLSAug2AH, from left to right: 1.: The left hand has grabbed the pole while the right hand task on level 9 is in conflict with the CoM task on level 8. 2.: The CoM on level 8 switched from box 1 to box 2 and is not in conflict with the right hand task any more The right hand task on level 9 is not augmented any more 3.: HRP-2Kai is in full forward stretch. 4.: The robot moves its right hand to the back. 5.: HRP-2Kai is in full backward stretch. 6.: The robot during its second forward stretch.*

is determined via marker based vision provided by the `whycon` library Nitsche et al. [2015]. During the movement, the CoM task on level 8 constrained to the bounding box $[\pm 0.03, \pm 0.1, \pm\infty]$ m gets activated. The right hand on the next level 9, which must remain at its current position, is therefore in conflict and starts to move slightly backwards. This is a purely algorithmic singularity and triggers the switch to Newton's method. Figure 5.38 shows the activation of the CoM task and the augmentation of the right hand position task and the chest orientation task at around 8 s.

The left gripper is closed and the left-hand-to-pole contact is added to the equation of motion. Then we open up the CoM bounding box in x-direction from $[\pm 0.03, \pm 0.1, \pm\infty]$ m to $[\pm 0.2, 0.05 \pm 0.05, \pm\infty]$ m since we have increased the support area of the robot in the sagittal plane due to the additional contact (see Fig. 5.40). However, the robot requires several control iterations with a high number of active-set iterations to adjust

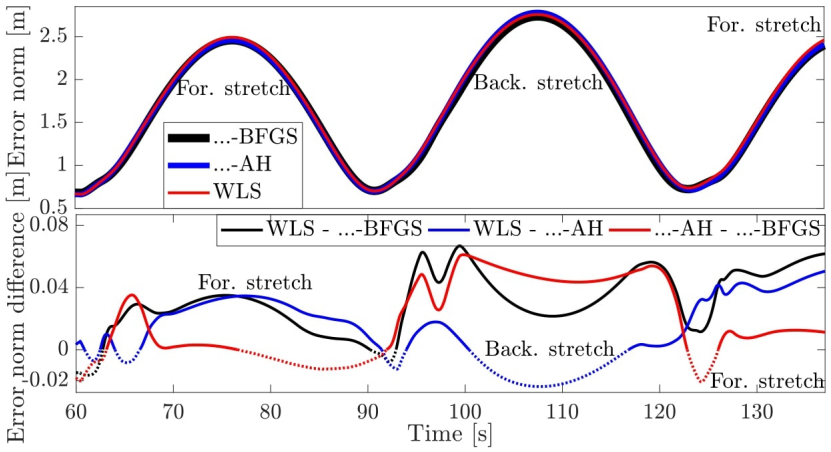


***Fig. 5.36. Exp. 2**, comparison of the error norm of the right hand tracking the swinging target during forward (for.) and backward (back.) stretch. The lower graph shows the differences of the error norms of the different methods. The data of LexLSAug2AH and WLS is synchronized with the one of LexLSAug2BFGS.*

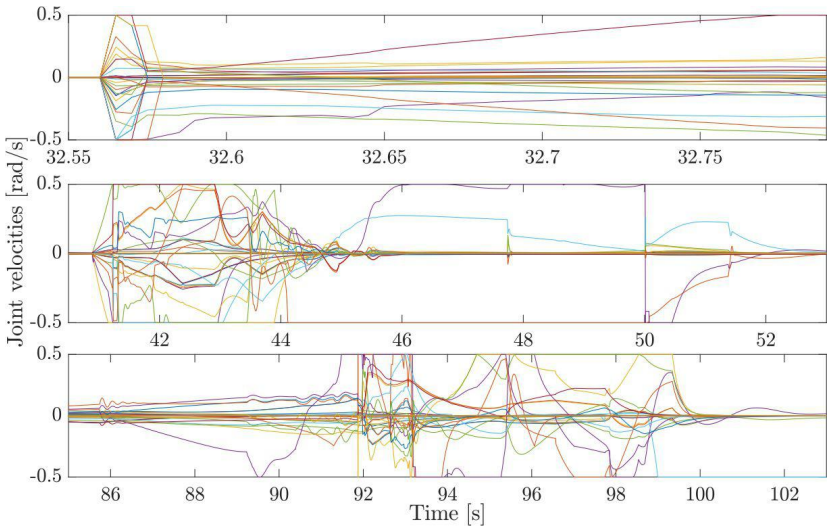


***Fig. 5.37. Exp. 2**, LexLSAug2BFGS, joint velocities. The upper graph shows the moment when the CoM is released and the 52 active-set iterations occur (followed by 5 occurrences of 30 iterations until 32.7 s). The middle one shows the moment when the right hand stretches to the right and starts being augmented. The lower graph shows the joint velocities when the robot stretches to the back and crouches.*

the robot state. In Fig. 5.39 at the 32 s mark, it can be seen that within 28 control iterations, peaks of 1×52 and 5×30 active-set iterations occur. Figure 5.38 shows that at the instant of the CoM release at around 32 s both CoM tasks at level 8 and 10 are activated and shortly after deactivated again. Similar behaviour is seen for the collision constraint between the left elbow and the torso.

Furthermore, the upper graph of Fig. 5.37 shows how the velocity suddenly increases

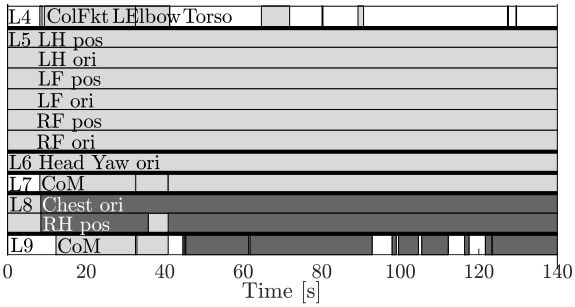


***Fig. 5.38. Exp. 2**, LexLSAug2BFGS, map of activity (light gray) and Newton's method (dark gray)*

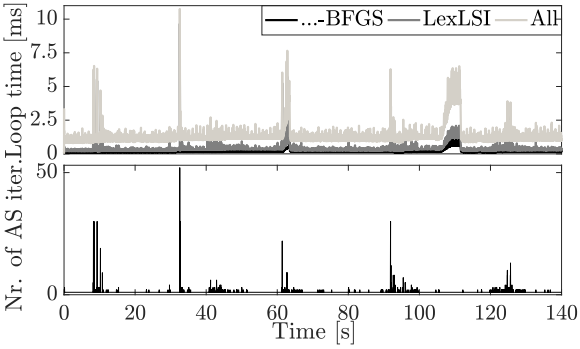


***Fig. 5.39. Exp. 2**, LexLSAug2BFGS, computation times and number of active-set iterations. LexLSAug2BFGS peaks at 1.11 ms, LexLSI at 9.66 ms and overall at 10.75 ms. The maximum number of active-set iterations is 52 iterations. At around 105 s and for the duration of a few seconds there is a computational artefact from the robot controlling PC.*

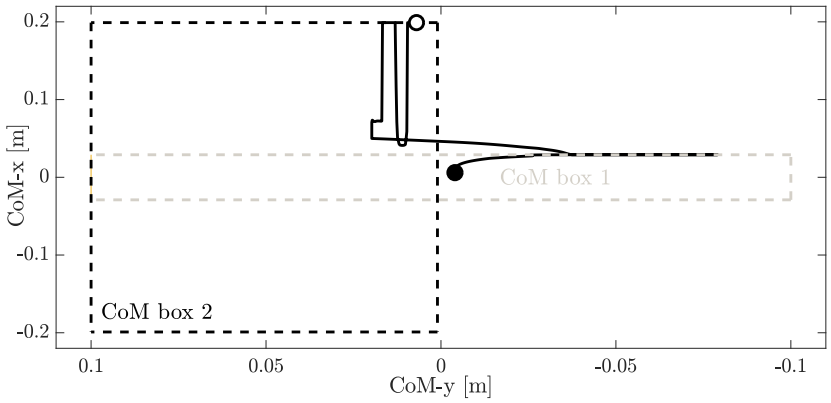


***Fig. 5.40. Exp. 2**, LexLSAug2BFGS, CoM movement. The CoM starts at the black dot and moves until it arrives at the white dot, first being constrained in box 1 and then in box 2.*

from zero to maximum velocity 0.5 s during the CoM release. This requires large joint torques of around 150 Nm as can be seen from the lower graph of Fig. 5.45. The corresponding dynamic effects need to be adjusted for in the trust region and the contact force constraints (with interplay) which leads to the large number of active-set iterations.

Note that without the active-set iteration limitation method presented in section 5.3 the active-set iteration count might easily go up to 200 or more.

At around 36 s the augmentation of the level 9 position task stops as the right hand arrives back at its prescribed position (see Fig. 5.38).

The right hand then first tries to reach $[0, -1.5, 1]$ m and continues to follow a target swinging from the front $[3, -1.5, 2]$ m to the back $[-3, -1.5, 0]$ m of the robot.

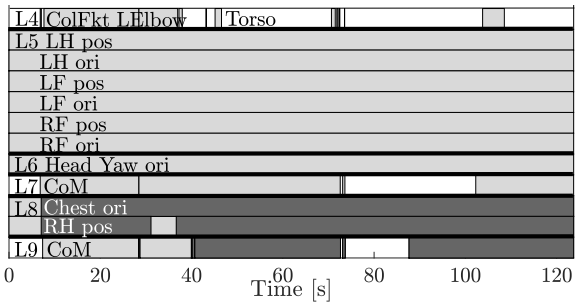


***Fig. 5.41. Exp. 2**, LexLSAug2AH, map of activity (light gray) and Newton's method (dark gray).*

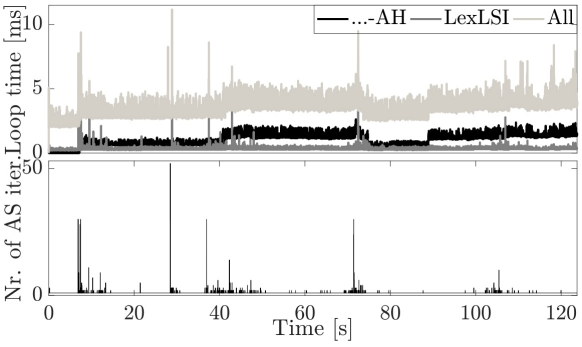


***Fig. 5.42. Exp. 2**, LexLSAug2AH, computation times and number of active-set iterations. LexLSAug2AH peaks at 2.63 ms, LexLSI at 8.03 ms and overall at 11.16 ms. The maximum number of active-set iterations is 52 iterations.*

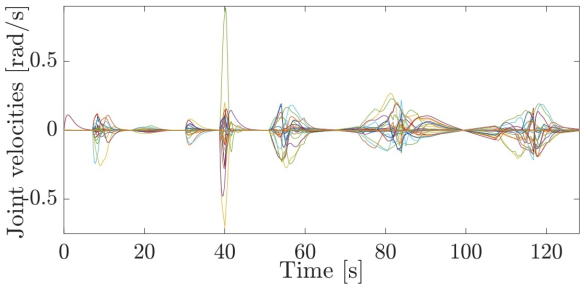


***Fig. 5.43. Exp. 2**, WLS, joint velocities.*

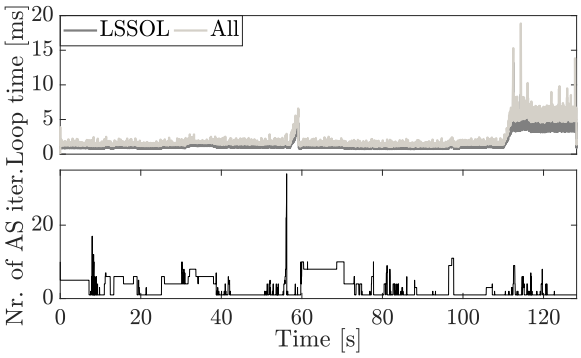


***Fig. 5.44. Exp. 2**, WLS, computation times and number of active-set iterations. In the time segment from 0 s to 56 s (excluding the computational artefact from the controller PC, but including the active-set peak), LSSOL peaks at 2.20 ms and overall at 2.94 ms with 34 active-set iterations. Again, there are some computational artefacts on the robot controlling PC from 56 s and from 110 s.*

This target is always out of reach.

From Fig. 5.47 (top left graph) it can be seen that during the forward stretches from 62 s and from 125 s the robot pushes its left hand against the pole with more than 100 N. This shows the necessity of this contact to prevent falling.

In Fig. 5.40, the CoM moves from the black to the white dot. During the stretch motions it is well outside the support polygon of the feet which covers about $[\pm 0.1, \pm 0.2, \pm\infty]$ m. The CoM at level 8 is first constrained to box 1 and then to box 2. Note that the CoM

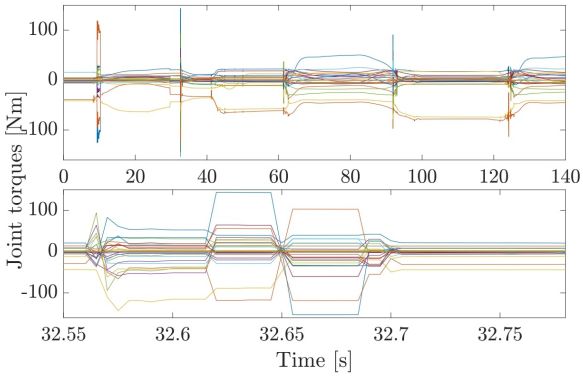


***Fig. 5.45. Exp. 2**, LexLSAug2BFGS, joint torques. The upper graph shows the whole experiment while the lower graph only shows the moment when the CoM is released and the 52 active-set iterations occur (followed by 5 occurrences of 30 iterations until 32.7 s). The maximum torque is -152.8 Nm.*

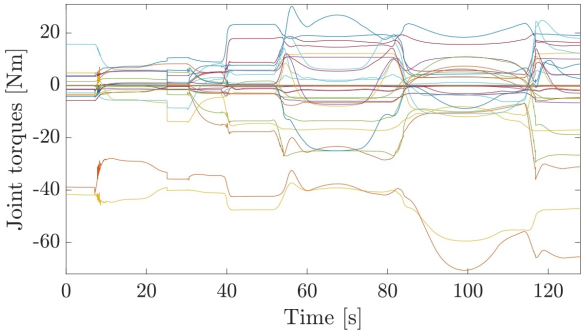


***Fig. 5.46. Exp. 2**, WLS, joint torques. The maximum joint torque is -71 Nm.*

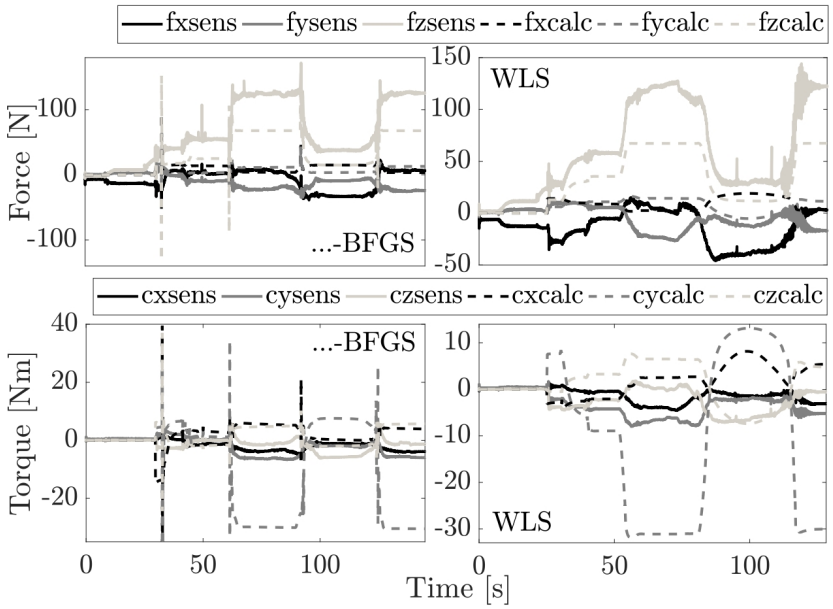


***Fig. 5.47. Exp. 2**, comparison of the reaction forces (top) and torques (bottom) of the left hand grabbing the pole between LexLSAug2BFGS (left column) and WLS (right column), measured (sens) and computed (calc). They are of similar magnitude.*

constraint at the white dot is also activated in y-direction. This is due to the velocity damper which starts to act 0.02 m before the actual bounding box edge. The continuous augmentation of the right hand and the chest task at level 9 occupies all joints. Therefore, the CoM constraint on level 10 has no influence and is therefore not shown here.

Figure 5.36 shows the norms of the tracking error of the right hand during the stretch motions. Especially during the second forward stretch LexLSAug2BFGS gets over 0.05 m closer to the target than WLS due to fully stretching its arm into kinematic singularity. Overall, most of the time the error norm difference $\mathrm{Err}_{\mathrm{WLS}} - \mathrm{Err}_{\mathrm{LexLSAug2BFGS}} > 0$ and shows the better convergence of LexLSAug2BFGS compared to WLS.

While LexLSAug2AH also allows limbs to be fully stretched into kinematic singularities, it has a higher error norm than WLS especially during the backward stretch (see Fig. 5.36). This is despite the fact that the robot crouches more than seen for WLS. The reason is partly due to local minima created by joint limits. However, at forward stretches LexLSAug2AH performs better than WLS ($\mathrm{Err}_{\mathrm{WLS}} - \mathrm{Err}_{\mathrm{LexLSAug2AH}} > 0$).

In Fig. 5.42 it can be seen that the average overall computation time 'All', which includes the calculation time of the analytic Hessian, increases by about 1.5 ms compared to the ones of LexLSAug2BFGS in Fig. 5.39. An additional increase can be observed at 40 s for 'LexLSAug2AH' when the CoM task on level 10 requires augmentation too (see Fig. 5.41). Now two expensive SVD decompositions for the Higham regularizations of the hierarchical analytic Hessian on level 9 and 10 are required.

In general, LexLSAug2AH behaves very similar to LexLSAug2BFGS and shows the capabilities of the BFGS algorithm to provide a valid approximation of the analytic hierarchical Hessian. Therefore, the joint velocities, CoM motion, ground and pole reaction forces and joint torques are not shown since they are very similar to the ones of LexLSAug2BFGS.

As for WLS, it shows again the worst convergence (see Fig. 5.36) with very smooth joint velocities (see Fig. 5.43). Especially during the forward stretches the robot does not fully stretch its right arm. Also during the backward stretch the robot crouches to a lesser extent than seen for LexLSAug2BFGS and LexLSAug2AH.

Since there are no jerky movements the joint torques do not show high peaks (see 5.46). What is quite interesting is that there is still a peak of 32 active-set iterations (see Fig. 5.44) during the first forward stretch at 55 s. However, the computation time only increases by a fraction due to updating factorizations in LSSOL.

5.4.2.4 Experiment 3: Stretching with user defined target

With the last experiment (exp. 3) we show the importance of handling kinematic and algorithmic singularities automatically.

This experiment is set up similarly to exp. 2, except that the robot right hand follows a target tracked by a motion capture system (10 Krestel cameras from Motion Analysis). Typical applications could be human to robot hand-overs. In such situations, and especially for a humanoid robot with multi-level prioritized constraints, it is difficult to determine the robot workspace beforehand. Here our proposed methods LexLSAug2BFGS and LexLSAug2AH allow to only have approximate knowledge or no knowledge at all of the robot workspace while still enabling safe control for the robot. A hand-over could happen at the border of the workspace with the end-effector

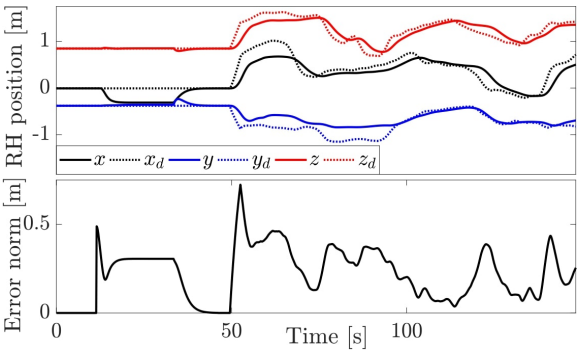


***Fig. 5.48. Exp. 3**, LexLSAug2BFGS, right hand task error. The upper graph shows the actual right hand position in x, y and z direction and the desired ones x_d, y_d and z_d. The lower graph shows the error norm of the right and left hand. The error of the left hand holding the rail can be considered zero once the contact is created at around 20 s.*

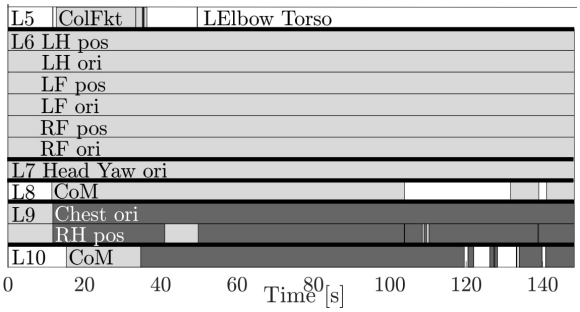


***Fig. 5.49. Exp. 3**, LexLSAug2BFGS, map of activity (light gray) and Newton's method (dark gray).*

being in kinematic singularity or being in algorithmic conflict with a physical stability task like the CoM task on a higher priority level. At the same time, the whole possible workspace is used without restricting it artificially for example with a badly tuned LM algorithm. Thereby, a non-adaptive LM algorithm can always be considered badly tuned. It needs to be tuned with a certain overhead in order to ensure non-singular behaviour over a range of robot configurations. This leads to bad convergence in singularity-free configurations where the robot would be numerically stable without damping.

Since the end-effector target is defined by user input, for the lack of comparability we only conducted the experiment with LexLSAug2BFGS.

In order to prevent too fast motions we set the right hand proportional task gain to $k_p = 0.5$ instead of $k_p = 1$ in the two previous experiments. Additionally, the CoM bounding box 2 is chosen a bit more conservatively as $[\pm 0.1, 0.05 \pm 0.05, \pm\infty]$ m instead of $[\pm 0.2, 0.05 \pm 0.05, \pm\infty]$ m in exp. 2.

The right hand tracking error is given in Fig. 5.48. At around 10 s the left hand moves towards the pole before closing the gripper and adding the contact to the equation of motion. At around 35 s the CoM constraint changes from box 1 to box 2. The right hand stops being in algorithmic conflict with the CoM constraint and moves to its original position (see Fig. 5.49 at around 42 s, the right hand is not augmented any more). From around 50 s onwards, the right hand is following the position of the middle marker of a wand with 3 markers tracked by the motion capture system. The provided position x_d, y_d and z_d is filtered by a 3 s moving average filter (see dotted lines in upper graph of Fig. 5.48). We first move the marker outside of the approximate workspace of the robot (until 100 s). The right arm is in kinematic singularity and

in conflict with the CoM constraint on level 8. We then continue to move the marker within the workspace of the robot. The CoM task at level 8 gets deactivated. However, the right hand switches from Newton's method to the GN algorithm only for a short period from 108.5 s to 109 s and from 109.75 s to 110.25 s before being augmented again. Note that the task error $\boldsymbol{e}$ does not necessarily need to be zero for this to happen. Our switching method only indicates that within the context of our linearized model we can (more or less) numerically stably provide the error velocity prescribed by the PD control emulated in velocity. Hence, there is no conflict with the kinematic restrictions of the robot or other constraints with the same or higher priority. The switching method could be tuned to be less sensitive although this could increase the risk that a singular task might not get augmented in certain situations. A good middle ground needs to be found in tuning the parameter ν of our switching method. Future work could also include the implementation of more accurate switching methods for example by observing the matrix rank of the constraint Jacobians or their projections onto the Jacobians of higher priority tasks.

5.5 Conclusion

In this chapter we extended our method on handling kinematic and algorithmic singularities in kinematic robot control to dynamically feasible kinematic robot control. We showed that it enables the robot to approach and reach singular configurations smoothly and without the high velocities typically seen for unresolved singularities. Thereby, our approach consisting of forward integrating the accelerations proved viable to shift second-order motion controllers and the equation of motion into the velocity domain. Dynamic constraints are respected and joint torques and contact forces are calculated correctly.

While velocity-based dynamic control can exactly represent the acceleration-based one, this requires modifications of the joint positions in an after-processing step. However, this causes inconsistent Lagrange multipliers which deteriorates the quality of the Lagrangian Hessian for Newton's method. We therefore decided to omit the modification of the joint positions. In future work it is desirable to find ways of maintaining consistency of both the acceleration-based control but also the Lagrange multipliers, for example by solving two different problems, see sec. 5.2.2.

Evaluated in three experiments on the position controlled HRP-2Kai robot, our method supersedes classical damping methods in terms of accuracy and error norm reduction. At the same time the notion of hierarchy enables formulating problems with strict safety prioritizations. Some cases of jerky joint movements are handled easily by adding a small damping term.

Computation times of the hierarchical least squares solver are a limiting factor of our approach of augmentation with the BFGS Hessian as well as the analytic Hessian. Without important dynamic effects, the active-set iteration count is limited to a few iterations. However, situations where several torque, trust region and contact force

constraints become active are more challenging for the active-set search. Our future work needs to focus on handling these situations cheaply, for example by factorization updates in the solver by Dimitrov et al. [2015] or a more effective active-set search.

CHAPTER 6

Computational accelerations for LexLSI

While LexLSI Dimitrov et al. [2015] can be considered the state-of-the-art method for solving hierarchical LSP's we saw in the last chapter 5 that it tends to perform quite poorly if a large number of active-set iterations is necessary in order to find the optimal active-set. The reason is that factorizations available from the previous active-set iteration are not cheaply updated as it is the case for example in Escande et al. [2014]. Rather, all factorizations from the first to the last level are computed anew, even if the active-set change occurred on some level $l > 1$. In app. D, a comparison of computational times with the solver Escande et al. [2014] incorporating factorization updates is given for a dynamic control problem.

In this chapter we present some methods to computationally accelerate LexLSI. First some details on the inner workings of LexLSI (sec. 6.1) are given. These are necessary for understanding the mechanisms which allow LexLSI to save computational time. In 6.2 the necessary steps are developed to restart the factorization from the level $l_{\text{asc}} > 1$ where the active-set change occurred (asc: active-set change). Secondly, the decomposition itself is accelerated by leveraging the structure of the constraint matrix (see sec. 6.3) and bound constraints (see sec. 6.4). These changes to LexLSI are then validated in 6.5 on a test bench of five numerical tests showing their capability of enhancing the original algorithm in terms of computational speed.

6.1 Preliminaries

In sec. 2.3 the so-called null-space method was introduced which solves objectives in the projection of the null-space of the constraints. In the case of LexLSI, the bases for the null-space $\boldsymbol{Z}_l$ and its complementary space $\boldsymbol{Y}_l$ of level l are chosen as

$$\boldsymbol{Y}_l = \boldsymbol{\Pi}_l \begin{bmatrix} \boldsymbol{R}_l^{-1} \\ \boldsymbol{0} \end{bmatrix}, \quad \boldsymbol{Z}_l = \boldsymbol{\Pi}_l \begin{bmatrix} -\boldsymbol{R}_l^{-1}\boldsymbol{T}_l \\ \boldsymbol{I} \end{bmatrix} \tag{6.1}$$

with the QR decomposition of $\tilde{\boldsymbol{A}}_l = \boldsymbol{A}_l \boldsymbol{N}_{l-1} \in \mathbb{R}^{m_l, n-r_{1:l-1}}$ (recall that $\boldsymbol{N}_i = \boldsymbol{Z}_1 \dots \boldsymbol{Z}_i$; $r_{1:l-1} = \sum_{i=1}^{l-1} rank(\tilde{\boldsymbol{A}}_i)$)

$$\boldsymbol{Q}_l^T \tilde{\boldsymbol{A}}_l \boldsymbol{\Pi}_l = \begin{bmatrix} \boldsymbol{Q}_l' & \boldsymbol{Q}_l'' \end{bmatrix}^T \tilde{\boldsymbol{A}}_l \boldsymbol{\Pi}_l = \begin{bmatrix} \boldsymbol{R}_l & \boldsymbol{T}_l \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}. \tag{6.2}$$

We have $\boldsymbol{Q}_l' \in \mathbb{R}^{m_l, r_l}$, $\boldsymbol{Q}_l'' \in \mathbb{R}^{m_l, m_l - r_l}$, $\boldsymbol{R}_l \in \mathbb{R}^{r_l, r_l}$ and $\boldsymbol{T}_l \in \mathbb{R}^{r_l, n-r_{1:l}}$. $\boldsymbol{Q}_l^T = \boldsymbol{H}_{r_l} \dots \boldsymbol{H}_1$ assembles the $r_l = rank(\tilde{\boldsymbol{A}}_l)$ orthogonal Householder transformations (HHT). The HHT transforms the vector $\boldsymbol{t}_{pv_i} \in \mathbb{R}^{m_t}$, $m_t = m_l - i + 1$ at stage i of the decomposition as follows:

$$\boldsymbol{H}_i^{2,2} \begin{bmatrix} \boldsymbol{t}_{pv_i} & \boldsymbol{T} \end{bmatrix} = \begin{bmatrix} \alpha & \boldsymbol{H}_i^{2,2}(1,:)\boldsymbol{T} \\ \boldsymbol{0} & \boldsymbol{H}_i^{2,2}(2:m_t,:)\boldsymbol{T} \end{bmatrix} \tag{6.3}$$

$$\text{with} \quad \boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{0} \in \mathbb{R}^{i-1,i-1} & \\ & \boldsymbol{H}_i^{2,2} \in \mathbb{R}^{m_t, m_t} \end{bmatrix}. \tag{6.4}$$

The colon : encompasses every single entry of the row or column while $j:k$ contains every entry from the j-th to the k-th row or column index. The complexity of the operation is $\mathcal{O}(4nm_t)$, see Golub et Van Loan [1996], page 211. In every of the r_l stages of the decomposition, the pivot column $\boldsymbol{t}_{pv_i}$ is chosen from the right bottom block $\boldsymbol{T}_{i-1,l}^2$ (which corresponds to $\begin{bmatrix} t_{pv} & T \end{bmatrix}$ in (6.3)) of the previous iterate $i-1$ of the decomposition

$$\boldsymbol{H}_{i-1} \dots \boldsymbol{H}_1 \tilde{\boldsymbol{A}}_l \boldsymbol{\Pi}_1 \dots \boldsymbol{\Pi}_{i-1} = \begin{bmatrix} \boldsymbol{R}_{i-1,l} \in \mathbb{R}^{i-1,i-1} & \boldsymbol{T}_{i-1,l}^1 \in \mathbb{R}^{i-1, n-r_{1:l-1}-i+1} \\ \boldsymbol{0} & \boldsymbol{T}_{i-1,l}^2 \in \mathbb{R}^{m_t, n-r_{1:l-1}-i+1} \end{bmatrix}. \tag{6.5}$$

The pivot column can be picked in a simple ascending fashion $pv_i = i$ with $i = 1, \dots, r_l$ and $\boldsymbol{\Pi}_l = \boldsymbol{I}$. However, $\tilde{\boldsymbol{A}}_l$ might be rank deficient. By always choosing the column of $\boldsymbol{T}_{i-1,l}^2$ which has the largest quadratic norm ρ, a so-called *rank revealing QR decomposition* is conducted, see Björck [1996], chapter 1. This selection of pivot columns $\rho_{pv_1} > \rho_{pv_2} > \dots > \rho_{pv_{r_l}}$ introduces a sequence of column permutations $\boldsymbol{\Pi}_l = \boldsymbol{\Pi}_{1,l} \dots \boldsymbol{\Pi}_{r_l,l}$ which swap the pivot column pv_i with the column of the current index i of the QR decomposition.

The algorithm stops if the rank of the matrix is revealed by reaching either the maximum rank ($= min(nrRows, nrCols)$) or by revealing rank deficiency with the column norms of all the remaining columns $\boldsymbol{T}_{r_l+1,l}^2$ being below a small numerical threshold ($\approx 1e^{-12}$).

$\boldsymbol{Z}_l$ is a basis of the null-space of $\tilde{\boldsymbol{A}}_l$ as can be seen from

$$\tilde{\boldsymbol{A}}_l \boldsymbol{Z}_l = \boldsymbol{Q}_l \begin{bmatrix} \boldsymbol{R}_l & \boldsymbol{T}_l \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \boldsymbol{\Pi}_l^T \boldsymbol{\Pi}_l \begin{bmatrix} -\boldsymbol{R}_l^{-1}\boldsymbol{T}_l \\ \boldsymbol{I} \end{bmatrix} = \boldsymbol{Q}_l \begin{bmatrix} -\boldsymbol{R}_l\boldsymbol{R}_l^{-1}\boldsymbol{T}_l + \boldsymbol{T}_l \\ \boldsymbol{0} \end{bmatrix} = \boldsymbol{0}. \tag{6.6}$$

The independence of the bases is given by the identity matrix which chooses a set of $n - m$ independent variables for the null-space basis, see chapter 15 of Nocedal et Wright [2006].

The bases are not orthogonal ($\boldsymbol{Y}^T\boldsymbol{Y} \neq \boldsymbol{I}$, $\boldsymbol{Z}^T\boldsymbol{Z} \neq \boldsymbol{I}$) and do not result in minimal norm solutions $\boldsymbol{x}$. However, they enable a simple *variable elimination*

$$\boldsymbol{x}_{l-1} = \boldsymbol{Y}_l\boldsymbol{x}_{l,Y} + \boldsymbol{Z}_l\boldsymbol{x}_{l,Z} = \boldsymbol{\Pi}_l \begin{bmatrix} \boldsymbol{R}_l^{-1}(\boldsymbol{x}_{l,Y} - \boldsymbol{T}_l\boldsymbol{x}_{l,Z}) \\ \boldsymbol{x}_{l,Z} \end{bmatrix}. \tag{6.7}$$

where $\boldsymbol{x}_{l,Z}$ is some simple selection of variables by the permutation matrix $\boldsymbol{\Pi}_l$. The term variable elimination originates from the fact that a certain set of variables (the size corresponds to the rank of the respective constraint matrix) is dedicated to the fulfilment of each level. These are then 'eliminated' and are not available any more for the fulfilment of lower priority levels.

For every level, a least squares problem of the form (2.31)

$$\min_{\boldsymbol{x}_{l,Z}} \quad \|\tilde{\boldsymbol{A}}_l\boldsymbol{x}_{l,Z} + \boldsymbol{y}_l\|_2^2 \quad l = 1, \ldots, p \tag{6.8}$$

with $\boldsymbol{y}_l = \boldsymbol{A}_l \sum_{k=1}^{l-1}(\boldsymbol{N}_{k-1}\boldsymbol{Y}_k\boldsymbol{x}_{k,Y}^*) + \boldsymbol{b}_l$ is solved. LexLSI implements the lexicographic QR decomposition

$$\underline{\boldsymbol{A}}_p = \begin{bmatrix} \boldsymbol{Q}_1' & & & \boldsymbol{Q}_1'' & & \\ \boldsymbol{L}_{21} & \boldsymbol{Q}_2' & & & \ddots & \\ \vdots & \vdots & \ddots & & & \ddots \\ \boldsymbol{L}_{p1} & \boldsymbol{L}_{p2} & \cdots \; \boldsymbol{Q}_p & & & \boldsymbol{Q}_p'' \end{bmatrix} \begin{bmatrix} \boldsymbol{R}_1 & \boldsymbol{T}_{12} & \cdots & \boldsymbol{T}_{1p} \\ & \boldsymbol{R}_2 & \cdots & \boldsymbol{T}_{2p} \\ & & \ddots & \vdots \\ & & & \boldsymbol{R}_p \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \end{bmatrix} \boldsymbol{\Pi}^T \tag{6.9}$$

with the stacked constraint matrix $\underline{\boldsymbol{A}}_p = \begin{bmatrix} \boldsymbol{A}_1^T & \ldots & \boldsymbol{A}_p^T \end{bmatrix}^T$ and the QR decompositions (6.2) of $\tilde{\boldsymbol{A}}_1$ to $\tilde{\boldsymbol{A}}_p$. The components $\boldsymbol{L}$ are a result of the variable elimination and their computation is detailed in (6.10).

This lexicographic decomposition can be conducted in place which overwrites the original stacked constraint matrix $\underline{\boldsymbol{A}}_p$. Its general shape is given in fig. 6.1 (without the modifications due to the constraint addition). Note that the matrix $\underline{\boldsymbol{A}} \in \mathbb{R}^{m_{\underline{\boldsymbol{A}}_p}, n}$ is assigned as a maximum sized matrix $m_{\underline{\boldsymbol{A}}_p} = \sum_{i=1}^{p} m_i$ including all possible inequality constraints in order to avoid memory reassignment in cases of active-set changes. The overall algorithm is described in the following:

- Conduct the rank revealing QR decomposition of $\tilde{\boldsymbol{A}}_l$ in place. Column permutations $\boldsymbol{\Pi}_l$ are applied to the whole matrix $\underline{\boldsymbol{A}}_p$. The orthogonal matrix $\boldsymbol{Q}_l$ can be stored below $\boldsymbol{R}_l$ as Householder vectors $\mathbb{H}_l$ but is omitted in fig. 6.1.
- Conduct the variable elimination in two steps
  - First, form the matrix product

$$\boldsymbol{L}_{l+1:p,l} = \underline{\boldsymbol{A}}_{l+1:p,l}\boldsymbol{R}_l^{-1} \tag{6.10}$$

in place. $\underline{\boldsymbol{A}}_{l+1:p,l}$ is the matrix block below $\boldsymbol{R}_l$ and $\mathbb{H}_l$. The inverse of $\boldsymbol{R}_l$ only requires a cheap backward substitution.

– Secondly, form the matrix

$$\underline{\boldsymbol{A}}_{l+1:p,l+1:p} \leftarrow \underline{\boldsymbol{A}}_{l+1:p,l+1:p} - \underline{\boldsymbol{A}}_{l+1:p,l}\boldsymbol{T}_l \tag{6.11}$$

in place. $\underline{\boldsymbol{A}}_{l+1:p,l+1:p}$ is the matrix block below $\boldsymbol{T}_l$.

- Repeat for each level $l = 1, \ldots, p$

For more details see Dimitrov et al. [2015].

6.2 Factorization from some level l

So far no strategy has been proposed to update the chosen bases in LexLSI. Especially, propagating rank changes from the level l_{asc} where the active-set change occurred to the lower priority levels (i.e. updating the range and null-space bases) seems difficult. Therefore, we look for other means of improving the computational speed of the LexLSI solver. LexLSI redoes the whole factorization even if the active-set change occurred on some low priority level. We identify this circumstance as a possible lever to accelerate the computations. Our goal in this section is to only redo the factorization from l_{asc}.

6.2.1 Constraint addition

Let's assume that the active-set search has determined that the constraint $\boldsymbol{a}$ needs to be activated on level 3. Figure 6.1 depicts the structure of the completed factorization from the previous active-set iteration (see Dimitrov et al. [2015]). In order to insert the new constraint into the constraint matrix, the orange box $\boldsymbol{L}$ is moved one row down and the new constraint row is inserted into the freed space (in the figure 6.1 it has already be done). This partly overwrites data from the red box which can not be reused anyway. Note that the column permutations from the rank revealing QR decompositions of the levels 1 to $l_{\text{asc}} - 1$ are applied to $\boldsymbol{a}$. In the following, the gray patterned box is completely overwritten while the filled orange box and the framed blue box need to undertake a few adjustments.

The columns in the blue box $\boldsymbol{T}$ need to be reversed to the state after the factorization of the second level $l_{\text{asc}} - 1 = 2$. This is done by applying the column permutations $\boldsymbol{T}\boldsymbol{\Pi}_p^T \ldots \boldsymbol{\Pi}_{l_{\text{asc}}}^T$ from the rank revealing QR decompositions of the level l_{asc} to the last level p in reverse order.

The gray patterned box needs to be factorized again from scratch. In order to do so it is filled with the trailing ends of all the constraints that already have been in the old active-set (corresponding to the red dashed boxes $\boldsymbol{T}_3$, $\boldsymbol{T}_4$ and $\boldsymbol{T}_5$). Again, the column permutations of the levels 1 to $l_{\text{asc}} - 1$ are applied (not to $\boldsymbol{a}_{3,T}$).

Next, the variable elimination (6.10), (6.11) needs to be applied to the new row and the dashed red framed boxes. For this we loop from level $i = 1$ to level $l_{\text{asc}} - 1 = 2$. We

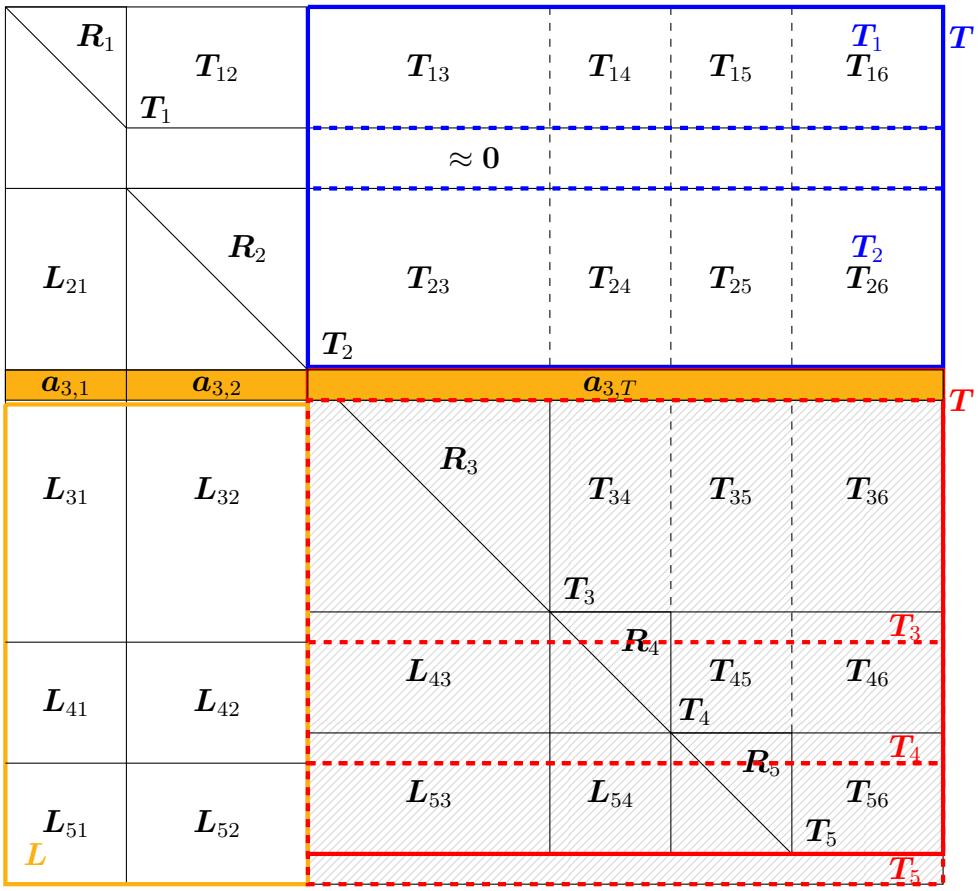


Fig. 6.1. *The structure of the completed factorization from the previous active-set iteration (in black). The new row (filled orange box) has already been inserted after the orange framed box was moved one row down.*

apply the inverse of the upper triangular matrix $\boldsymbol{R}_i^{-1}$ of the current index i from the right to the part of the new row $\boldsymbol{a}_{l_{\text{asc}},i}$ that is below the $\boldsymbol{R}_i$ matrix of the current index i. Then, the second step of the variable elimination with the multiplication from the right with $\boldsymbol{T}_{i,j=i+1:l_{\text{asc}}-1}$ is conducted to the trailing end of the new row $\boldsymbol{a}_{l_{\text{asc}},j=i+1:l_{\text{asc}}-1}$. Note that both $\boldsymbol{T}_{i,j}$ and $\boldsymbol{a}_{l_{\text{asc}},j}$ have the same column order since the permutations of the QR decomposition of the trailing levels are already applied to $\boldsymbol{a}$. With the now updated matrices $\boldsymbol{L}_{l_{\text{asc}},1:l_{\text{asc}}-1}$ we proceed with the standard variable elimination of the remaining trailing blocks $\boldsymbol{T}_{l_{\text{asc}}:p}$ (red dashed framed boxes). The algorithm structure is given in alg. 1. Information about the column lengths and bound constraints can be taken advantage of as described in sec. 6.3 and sec. 6.4.

Note that the added constraint on l_{asc} does not change the pivot column choice of the rank revealing QR decompositions of the previous levels (where the main criteria is to look for the columns with the largest norm of each respective level). Therefore, there is no concern of ill conditioning of the constraint matrices even after several

Algorithm 1 – Pre-variable elimination after active-set change on level l_{asc}

$l_{\text{asc}} = 3,\ p = 5$
for $i = 1 : l_{\text{asc}} - 1$ **do**
  if Add constraint **then**
    $\boldsymbol{a}_{l_{\text{asc}},i} \leftarrow \boldsymbol{a}_{l_{\text{asc}},i}\boldsymbol{R}_{\boldsymbol{i}}^{-1}$
    for $j = i + 1 : l_{\text{asc}} - 1$ **do**
      $\boldsymbol{a}_{l_{\text{asc}},j} \leftarrow \boldsymbol{a}_{l_{\text{asc}},j} - \boldsymbol{a}_{l_{\text{asc}},i}\boldsymbol{T}_{i,j}$
    end for
  end if
  for $k = l_{\text{asc}} : p$ **do**
    $\boldsymbol{T}_k \leftarrow \boldsymbol{T}_k - \boldsymbol{L}_{k,i}\boldsymbol{T}_i$
  end for
end for

constraint additions and removals. The level l_{asc}, where the new (possibly linear dependent) constraint was added, and lower priority levels are subject to rank revealing QR decompositions from scratch. Possible linear dependency is therefore sufficiently handled by the original algorithm.

6.2.2 Constraint removal

In the case of a constraint removal the box $\boldsymbol{L}$ is moved one row up which overwrites the deactivated constraint. Furthermore, in alg. 1 the pre-variable elimination of the new row can be skipped and only the handling of the trailing end needs to be conducted.

6.3 Leveraging column lengths

Another way to save computational cost is by taking advantage of the structure of the constraint matrix $\boldsymbol{A}$. For this, in the very first active-set iteration we measure the length of each column of each level's constraint matrix $\boldsymbol{A}_l$, $l = 1, \ldots, p$. For this the algorithm goes through all columns $\boldsymbol{A}_{l,c}(j)$, $c = 1, \ldots, n$ from the bottom to the top $j = m_l, \ldots, 1$. In case that a value $|\boldsymbol{A}_{l,c}(j)| > \kappa$ is found the current index j is noted in the column length matrix $\boldsymbol{\Upsilon} \in \mathbb{R}^{p,n}$ as

$$\text{column length}(l, c) = \boldsymbol{\Upsilon}(l, c) = j\,. \tag{6.12}$$

The algorithm proceeds with the next column $c = c+1$. $\kappa = 10^{-18}$ is a small numerical value.

Known structure about the problem can be shared with the solver beforehand. In the robotic control case, this could include the kinematic task structure. Joints not on the kinematic chain yield zero columns for the corresponding variable of the task Jacobian. Same holds for the case of incorporated dynamics in the form of the

equation of motion. Jacobians of motion controllers only include the kinematic but not the dynamic variables like joint torques and contact wrenches.

Note that bound constraints are generally ignored in the column length matrix $\boldsymbol{\Upsilon}$ (see sec. 6.4.1). Also, the column permutations $\boldsymbol{\Pi}$ have to be applied to $\boldsymbol{\Upsilon}$ after every step of the QR decomposition. In the case of an active-set change, the permutations from level p to l_{asc} then have to be reversely applied $\boldsymbol{\Upsilon}\boldsymbol{\Pi}_p^T \dots \boldsymbol{\Pi}_{l_{\text{asc}}}^T$ to restore the ordering corresponding to the state of the column permutations after the factorization of the level $l_{\text{asc}} - 1$.

6.3.1 Usage during the QR decomposition and the variable elimination

The column length information enables a shortened HHT during the QR decomposition of the constraint matrix $\tilde{\boldsymbol{A}}_l$ of level l (6.2). If there are bottom zeros on the pivot column $\boldsymbol{t}_{pv_i}$, (6.3) becomes

$$\boldsymbol{H}\begin{bmatrix}\boldsymbol{t}_{pv_i} & \boldsymbol{T}\end{bmatrix} = \boldsymbol{H}\begin{bmatrix}\boldsymbol{t}^1_{pv_i} & \boldsymbol{T}^1\\ \boldsymbol{0} & \boldsymbol{T}^2\end{bmatrix} = \begin{bmatrix}\boldsymbol{H}^{1,1} & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}\begin{bmatrix}\boldsymbol{t}^1_{pv_i} & \boldsymbol{T}^1\\ \boldsymbol{0} & \boldsymbol{T}^2\end{bmatrix} = \begin{bmatrix}\alpha & \boldsymbol{H}^{1,1}(1,:)\boldsymbol{T}^1\\ \boldsymbol{0} & \boldsymbol{H}^{1,1}(2:m_t,:)\boldsymbol{T}^1\\ \boldsymbol{0} & \boldsymbol{T}^2\end{bmatrix} \tag{6.13}$$

where $\boldsymbol{t}^1_{pv_i}$ is of length $m_t = \boldsymbol{\Upsilon}(l, pv_i) - i + 1$. This knowledge enables computational savings of order $\mathcal{O}(4n(m_l - \boldsymbol{\Upsilon}(l, pv_i)))$.

This is especially helpful if the problem is augmented with an upper triangular matrix $\boldsymbol{R}$

$$\begin{bmatrix}\boldsymbol{J}\\ \boldsymbol{R}\end{bmatrix} \tag{6.14}$$

or a level with bounds of the form

$$\begin{bmatrix}\boldsymbol{J}_1 & \boldsymbol{J}_2\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}. \tag{6.15}$$

Furthermore, the knowledge of the column lengths allows to omit bottom zero rows during the variable elimination (6.10)

$$\boldsymbol{L}_{i,l} = \underline{\boldsymbol{A}}_{i,l} = \begin{bmatrix}\underline{\hat{\boldsymbol{A}}}_{i,l}\\ \boldsymbol{0}\end{bmatrix} \leftarrow \underline{\boldsymbol{A}}_{i,l}\boldsymbol{R}_l^{-1} = \begin{bmatrix}\underline{\hat{\boldsymbol{A}}}_{i,l}\\ \boldsymbol{0}\end{bmatrix}\boldsymbol{R}^{-1} = \begin{bmatrix}\underline{\hat{\boldsymbol{A}}}_{i,l}\boldsymbol{R}^{-1}\\ \boldsymbol{0}\end{bmatrix} \tag{6.16}$$

and (6.11)

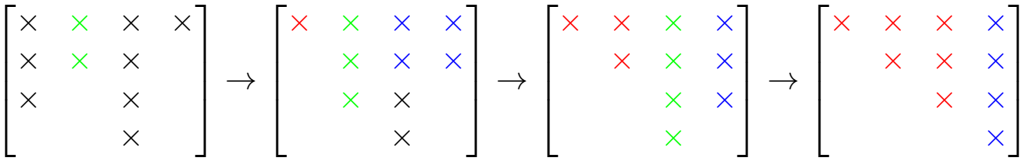
$$\underline{\boldsymbol{A}}_{i,l+1:p} = \begin{bmatrix}\underline{\hat{\boldsymbol{A}}}_{i,l+1:p}\\ \underline{\hat{\hat{\boldsymbol{A}}}}_{i,l+1:p}\end{bmatrix} \leftarrow \underline{\boldsymbol{A}}_{i,l+1:p} - \underline{\boldsymbol{A}}_{i,l}\boldsymbol{T}_l = \begin{bmatrix}\underline{\hat{\boldsymbol{A}}}_{i,l+1:p}\\ \underline{\hat{\hat{\boldsymbol{A}}}}_{i,l+1:p}\end{bmatrix} - \begin{bmatrix}\underline{\hat{\boldsymbol{A}}}_{i,l}\boldsymbol{T}_l\\ \boldsymbol{0}\end{bmatrix} \tag{6.17}$$

for $i = l{+}1, \dots, p$. The length of $\boldsymbol{0}$ (and $\underline{\hat{\hat{\boldsymbol{A}}}}_{i,l+1:p}$) is thereby determined by $m_i - \max(\Upsilon_{i,l}$ where $\max(\Upsilon_{i,l})$ is the length of the longest column in $\underline{\boldsymbol{A}}_{i,l}$.

6.3.2 Updating the column lengths

Constantly determining the column lengths $\boldsymbol{\Upsilon}$ from scratch during every step of the the QR decomposition, the variable elimination or active-set changes is very expensive (and not necessary).

Instead, we keep track of the longest column on the 'left side' $\boldsymbol{v} \in \mathbb{R}^{p,1}$, i.e. the longest column length of all the pivot columns pv that have been chosen so far during the QR decompositions of the levels $1, \dots, l$ (and then were column swapped to the current column index of the QR decomposition - or the 'left side'). A quick example is given below with the QR decomposition of the level 1 constraint matrix $\boldsymbol{A}_1$:



(6.18)

The pivot column of each iteration of the QR decomposition is marked in green. Columns that already have been chosen as pivot columns and have been permuted to the 'left side' are marked in red. Matrix entries that have been affected by the HHT are colored in blue. The HHT of the last column is omitted.

With each chosen pivot column pv we update $\boldsymbol{v}$ by

$$\boldsymbol{v}(i) = \max(\boldsymbol{v}(i), \boldsymbol{\Upsilon}(i, pv)) \quad i = l, \dots, p\,. \tag{6.19}$$

In our example $\boldsymbol{v}$ and $\boldsymbol{\Upsilon}$ are given as

$$\boldsymbol{v} = 0; \boldsymbol{\Upsilon} = \begin{bmatrix} 3 \\ 2 \\ 4 \\ 1 \end{bmatrix}^T \rightarrow \boldsymbol{v} = 2; \boldsymbol{\Upsilon} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}^T \rightarrow \boldsymbol{v} = 3; \boldsymbol{\Upsilon} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}^T \rightarrow \boldsymbol{v} = 4; \boldsymbol{\Upsilon} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 1 \end{bmatrix}^T . \tag{6.20}$$

$\boldsymbol{\Upsilon}$ is only updated with regard to the column permutations.

In multi-level scenarios we need to consider the lengths of the pivot column on all levels $l+1, \dots, p$ after the level l currently subject to the rank revealing QR decomposition. This information can then be used in the variable elimination of the levels $l+1, \dots, p$ once the QR decomposition of level l is finished (and also in the QR decomposition of these lower priority levels). At this point, the current values of $\boldsymbol{v}(i)$ are stored in another matrix $\boldsymbol{C} \in \mathbb{R}^{p,p}$ as $\boldsymbol{C}(i, l) = \boldsymbol{v}(i)$ with $i = l, \dots, p$. With this information $\boldsymbol{v}$ can be properly reset as

$$\boldsymbol{v}(i) = \boldsymbol{C}(i, l_{\text{asc}}) \quad i = l_{\text{asc}}, \dots, p \tag{6.21}$$

in the case of a factorization restart on level l_{asc}, see sec. 6.2.

With the activation or deactivation of a constraint on l_{asc} only the column length of this specific level is influenced. New constraints are inserted at the top of the constraint matrix $\boldsymbol{A}_{l_{\text{asc}}}$. We then loop through all the column lengths of this level $\boldsymbol{\Upsilon}(l_{\text{asc}}, i)$, $i = 1, \dots, n$ and adapt them in the following manner:

- $\boldsymbol{\Upsilon}(l_{\text{asc}}, i) > 0 \rightarrow$ increase column length $\boldsymbol{\Upsilon}(l_{\text{asc}}, i)$ by 1
- $\boldsymbol{\Upsilon}(l_{\text{asc}}, i) = 0$
  - value of current index of the new row $\neq 0 \rightarrow$ increase column length $\boldsymbol{\Upsilon}(l_{\text{asc}}, i)$ by 1
  - value of current index of the new row $== 0 \rightarrow$ do nothing

In the case of the deactivation of constraint d on level l_{asc}, we apply following similar update scheme on the l_{asc}-th row of $\boldsymbol{\Upsilon}$:

- $\boldsymbol{\Upsilon}(l_{\text{asc}}, i) \geq d \rightarrow$ decrease column length $\boldsymbol{\Upsilon}(l_{\text{asc}}, i)$ by 1
- $\boldsymbol{\Upsilon}(l_{\text{asc}}, i) < d \rightarrow$ do nothing.

6.4 Bound handling

While bound constraints on the first level are cheaply handled in LexLSI (actually, only if the first level exclusively consists of bound constraints), there is no special bound treatment on lower levels. However, considering bound constraints especially in the variable elimination has the potential to decrease the computational effort.

6.4.1 Keeping track of bound constraints

Bound constraints are thoroughly kept track of during the composition of the constraint matrix $\underline{\boldsymbol{A}}_p$ at the beginning of every active-set iteration and during every step of the QR decomposition.

During the constraint matrix composition a differentiation between 'general' and 'bound' constraints is made. General constraints are put at the top of each level while bound constraints are put at the bottom. Each bound constraint's row number in the constraint matrix $\underline{\boldsymbol{A}}_p$ is stored.

During the QR decomposition bound constraints require attention if the column corresponding to the bounded variable is chosen as the pivot column. In this case the bound is removed from the bound tracker and turned into a general constraint. On the other hand, if a column corresponding to a bound constraint is permuted with the pivot column (not necessarily containing a bounded variable) the permutation also needs to be applied to the bound tracker (for example we are at iteration 5 of the QR decomposition of level 1; column 5 corresponds to a bounded variable and column 8 was chosen as a pivot column; then the bound tracker is updated such that the newly bounded variable is variable 8 due to the column permutation of column 8 with column 5). Note that bound constraint which not have been pivoted yet are not considered in the column length. Once we pivot a column pv containing bound constraints we update $\boldsymbol{\upsilon}$ by

$$\boldsymbol{\upsilon}(i) = \max(\boldsymbol{\upsilon}(i), \boldsymbol{\Upsilon}(i, pv) + 1) \tag{6.22}$$

with i being every level $i = l, \ldots, p$ where the corresponding variable is bounded.

In the case of active-set changes and usage of the factorization restart method the bound tracker needs to be updated accordingly. Special attention is required if the variable bounded by the added constraint was already used as a pivot column during the decomposition of a level $< l_{\text{asc}}$. If so, the constraint is counted as a general constraint and put at the top of the bound constraints. If not, the bound is added to the bound tracker and put at the bottom of the bound constraints.

6.4.2 Using bounds in the variable elimination

The inverse of a blocked matrix can be calculated in the following way for invertible full rank and square matrices $\boldsymbol{M}$ and $\boldsymbol{P}$ Carlson [1986]:

$$\boldsymbol{R}^{-1} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{O} & \boldsymbol{P} \end{bmatrix}^{-1} \tag{6.23}$$

$$= \begin{bmatrix} (\boldsymbol{M} - \boldsymbol{N}\boldsymbol{P}^{-1}\boldsymbol{O})^{-1} & -(\boldsymbol{M} - \boldsymbol{N}\boldsymbol{P}^{-1}\boldsymbol{O})^{-1}\boldsymbol{N}\boldsymbol{P}^{-1} \\ -\boldsymbol{P}^{-1}\boldsymbol{O}(\boldsymbol{M} - \boldsymbol{N}\boldsymbol{P}^{-1}\boldsymbol{O})^{-1} & \boldsymbol{P}^{-1} + \boldsymbol{P}^{-1}\boldsymbol{O}(\boldsymbol{M} - \boldsymbol{N}\boldsymbol{P}^{-1}\boldsymbol{O})^{-1}\boldsymbol{N}\boldsymbol{P}^{-1} \end{bmatrix}. \tag{6.24}$$

In our case, $\boldsymbol{R}$ is upper triangular so we have $\boldsymbol{O} = \boldsymbol{0}$. With the presence of bound constraints $\boldsymbol{I}$ the QR factorization is designed in such a way that bounds come last, i.e. $\boldsymbol{P} = \boldsymbol{I}$ (see sec. 6.4.3).

The inverse of $\boldsymbol{R}$ then writes as follows:

$$\boldsymbol{R}^{-1} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}^{-1} = \begin{bmatrix} \boldsymbol{M}^{-1} & -\boldsymbol{M}^{-1}\boldsymbol{N} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}. \tag{6.25}$$

Now only the inversion of the smaller upper triangular matrix $\boldsymbol{M}$ instead of $\boldsymbol{R}$ is required. This means for the first step of the variable elimination (6.10) that $\underline{\boldsymbol{A}}_{l+1:p,l}$ is divided into

$$\underline{\boldsymbol{A}}_{l+1:p,l} = \begin{bmatrix} \overline{\underline{\boldsymbol{A}}}_{l+1:p,l} & \overline{\overline{\underline{\boldsymbol{A}}}}_{l+1:p,l} \end{bmatrix} \tag{6.26}$$

so

$$\underline{\boldsymbol{A}}_{l+1:p,l}\boldsymbol{R}_l^{-1} = \begin{bmatrix} \overline{\underline{\boldsymbol{A}}}_{l+1:p,l} & \overline{\overline{\underline{\boldsymbol{A}}}}_{l+1:p,l} \end{bmatrix} \begin{bmatrix} \boldsymbol{M}_l^{-1} & -\boldsymbol{M}_l^{-1}\boldsymbol{N}_l \\ \boldsymbol{0} & \boldsymbol{I}_l \end{bmatrix} \tag{6.27}$$

$$= \begin{bmatrix} \overline{\underline{\boldsymbol{A}}}_{l+1:p,l}\boldsymbol{M}_l^{-1} & \overline{\overline{\underline{\boldsymbol{A}}}}_{l+1:p,l} - \overline{\underline{\boldsymbol{A}}}_{l+1:p,l}\boldsymbol{M}_l^{-1}\boldsymbol{N}_l \end{bmatrix}. \tag{6.28}$$

6.4.3 Special QR factorization

In order to be able to leverage bound constraints in the variable elimination step a special QR decomposition needs to be applied which aims to permute bound constraints to the bottom right corner of the triangular matrix $\boldsymbol{R}$.

The level subject to the QR decomposition is of following form:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \hat{\boldsymbol{I}} \end{bmatrix} \tag{6.29}$$

The full rank matrix $\hat{\boldsymbol{I}}$ is an 'unordered' identity matrix, i.e. every column and row has only a single entry 1 which are not necessarily ordered diagonally. This can happen since during the composition of the constraint matrix $\boldsymbol{A}$ there is no specific ordering. The rank of $\boldsymbol{A}$ is then $rank(\boldsymbol{A}) = rank(\boldsymbol{M}) + rank(\hat{\boldsymbol{I}})$. This can be seen by multiplying $\boldsymbol{A}$ from the left with a full rank square matrix (which does not change the original rank of the (possibly non-square) matrix $\boldsymbol{A}$)

$$rank(\boldsymbol{A}) = rank\left(\begin{bmatrix} \hat{\boldsymbol{I}} & -\boldsymbol{N} \\ \boldsymbol{0} & \hat{\boldsymbol{I}} \end{bmatrix} \boldsymbol{A}\right) = rank\left(\begin{bmatrix} \hat{\boldsymbol{I}} & -\boldsymbol{N} \\ \boldsymbol{0} & \hat{\boldsymbol{I}} \end{bmatrix} \begin{bmatrix} \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{0} & \hat{\boldsymbol{I}} \end{bmatrix}\right) \tag{6.30}$$

$$= rank\left(\begin{bmatrix} \boldsymbol{M} & \boldsymbol{0} \\ \boldsymbol{0} & \hat{\boldsymbol{I}} \end{bmatrix}\right) = rank(\boldsymbol{M}) + rank(\hat{\boldsymbol{I}})\,. \tag{6.31}$$

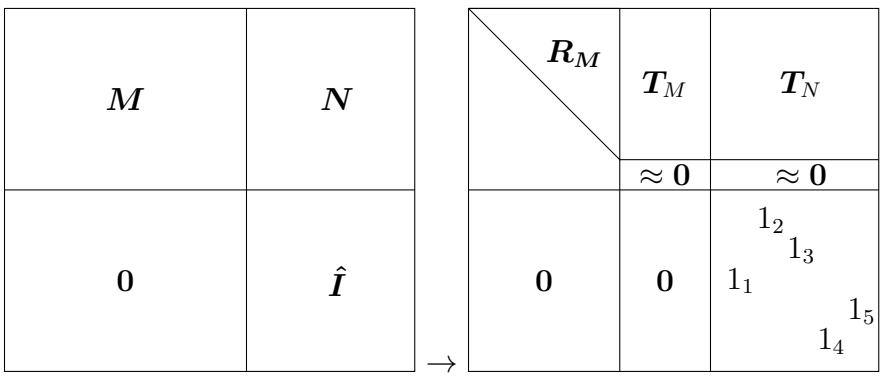
For this it needs to be ensured that a variable is bounded once at most per level.

First, a rank revealing QR decomposition of $\boldsymbol{M}$ is conducted. Once $rank(\boldsymbol{M})$ is revealed pivot columns from $\begin{bmatrix} \boldsymbol{N}^T & \hat{\boldsymbol{I}}^T \end{bmatrix}^T$ are chosen. Since the rank of the remaining columns is already known ($= rank(\hat{\boldsymbol{I}})$), the rank revealing part of the QR decomposition is omitted. Instead, any remaining column containing a bound constraint is chosen without ordering them after their norm. This does not change the condition number of $\boldsymbol{R}$ since orthogonal transformations (or no orthogonal transformation thereof due to the absence of column permutations) can be applied under invariance of the condition number:

$$cond(\boldsymbol{A}) = cond(\boldsymbol{PA}) = cond(\boldsymbol{PQR}) = cond(\boldsymbol{R})\,. \tag{6.32}$$

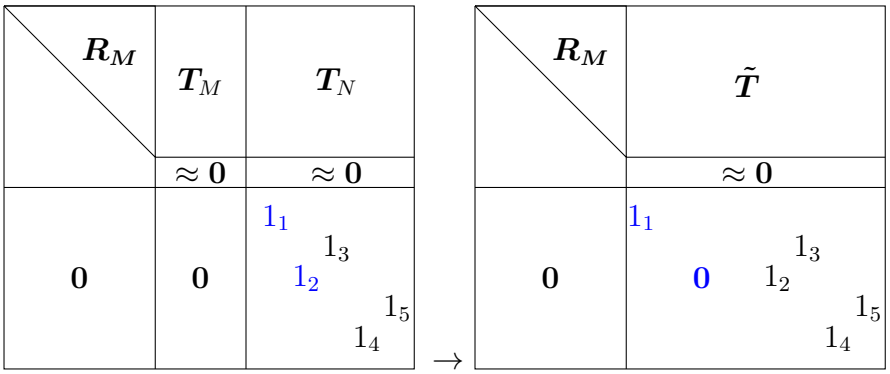
If a pivot column is chosen which represents a bounded variable either on the level currently subject to the QR decomposition or some lower level, it requires removing its entry from the bound tracker.

For visualization of the case of a bounded pivot column, let us take following example with $\boldsymbol{A} \in \mathbb{R}^{10,11}$, $\boldsymbol{M} \in \mathbb{R}^{5,6}$, $\boldsymbol{N} \in \mathbb{R}^{5,5}$ and $\hat{\boldsymbol{I}} \in \mathbb{R}^{5,5}$. The rank of the matrix $\boldsymbol{A}$ is $rank(\boldsymbol{A}) = 9$ with $rank(\boldsymbol{M}) = 4$ and $rank(\hat{\boldsymbol{I}}) = 5$. The algorithm then starts with the QR decomposition of the matrix $\boldsymbol{M}$.



Assume that at this stage the rank of $\boldsymbol{M}$ has been identified as $rank(\boldsymbol{M}) = 4$. The quadratic norms of the remaining columns of $\boldsymbol{M}$ are below the threshold for linear dependency so pivot columns representing bound constraints need to be chosen.

Pivot columns are chosen from left to right, starting with the column that contains 1_1. A row permutation (swapping the two coefficients 1_1 and 1_2 and the corresponding entries of the right hand side $\boldsymbol{b}$ is enough) is then applied. Note that this requires a parallel update of the active-set ordering. 1_1 is then permuted with the column corresponding to the current iteration of the QR decomposition $i = 5$:



Now a shortened HHT can be applied on the leftmost vector

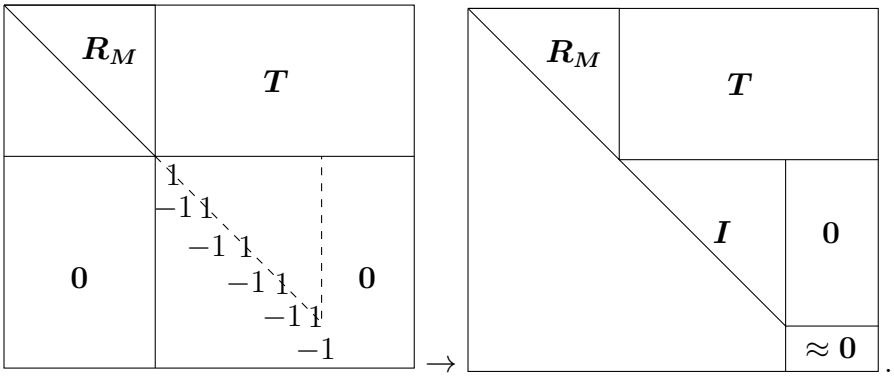
$$\boldsymbol{H} \begin{bmatrix} \mathbf{0} \\ 1_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} \tag{6.33}$$

where $\mathbf{0} \in \mathbb{R}^{rd,1}$ and $rd = 1$ is the rank deficiency of $\boldsymbol{M}$ and $\boldsymbol{A}$ respectively. The Householder vector $\boldsymbol{v}$ for $\boldsymbol{H} = \boldsymbol{I} - \tau \boldsymbol{v}\boldsymbol{v}^T$ can be simply defined as

$$\boldsymbol{v} = \begin{bmatrix} 1 \\ \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} 1 \\ \mathbf{0} \\ -1 \end{bmatrix} . \tag{6.34}$$

$\boldsymbol{s}$ is the essential vector that can be stored cheaply as $\mathbb{H}$ (in the decomposition fig 6.1 below the $\boldsymbol{R}$'s) instead of maintaining a full representation of the $\boldsymbol{Q}$ matrices. $\tau = 1$ and $\mathbf{0} \in \mathbb{R}^{rd-1,1}$. Note that applying $\boldsymbol{H}$ to the trailing end of the matrix only consists of a row swap of the right hand side $\boldsymbol{b}$.

Eventually, the following decomposition is obtained (with $\boldsymbol{T} = \tilde{\boldsymbol{T}}\boldsymbol{\Pi}_{5:}$):



If the matrix $\boldsymbol{M}$ is empty with

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{N} \\ \hat{\boldsymbol{I}} \end{bmatrix} \tag{6.35}$$

then all pivot columns have the form

$$pivot = \begin{bmatrix} \boldsymbol{\beta}^T & \mathbf{0}^T & 1 & \mathbf{0}^T \end{bmatrix}^T . \tag{6.36}$$

$\boldsymbol{\beta}$ is a vector containing non-zero values. In this case a shortened HHT is not applicable. Instead, we first need to swap the bound below $\boldsymbol{\beta}$ and then apply a full HHT on $\begin{bmatrix} \boldsymbol{\beta}^T & 1 \end{bmatrix}^T$.

6.5 Evaluation

We conduct some timing tests comparing the performance of the modified version of LexLSI (v1), which enables factorization restarts after active-set changes and takes advantage of bound constraints and column lengths, to the original one (v0). In the first row of the timing graphs the overall computation times of v0 and v1 are given. Additionally, the timings of some components are given. 'Advance' ('Adv') encompasses the time of the pivot column search, the bound handling and the column length handler updates. 'Householder Transform' ('HHT') and 'Variable elimination' ('VarEl') give the duration of the HHT and the variable elimination respectively. Other components are for example data initialization and data copying (at the beginning of each active-set iteration) but their computation time is not depicted separately since they are of same computational complexity for both v0 and v1.

The second row of the graphs shows how many rows were involved in the HHT while the third row indicates how many rows were involved in the variable elimination. The third row also indicates with 'nrBounds' how many bound constraints could be used for simplified operations during the HHT and the variable elimination.

For all the timings and row counts the percentage of how v1 performed against v0 is given in the right column of the graphs:

$$perf = \frac{v1}{v0} \cdot 100\% . \tag{6.37}$$

6.5.1 Test 1: Factorization restart

In *test 1* the effect of the factorization restart on the computation time is demonstrated. Two full rank matrix $\boldsymbol{A} \in \mathbb{R}^{m_A,n}$ and $\boldsymbol{B} \in \mathbb{R}^{m_B,n}$ with $m_A + m_B = n$ and $n = 100$ are defined. Level 1 consists of a set of equality constraints $\boldsymbol{Ax} = \boldsymbol{b}_A$, level 2 consists of a set of inequality constraints $\boldsymbol{l}_B < \boldsymbol{Bx} < \boldsymbol{u}_B$ and $\boldsymbol{B}$ on level 3 is a set of equality constraints $\boldsymbol{Bx} = \boldsymbol{b}_B$. $\boldsymbol{b}_B$ are chosen as either $\boldsymbol{b}_B < \boldsymbol{l}_B$ or $\boldsymbol{b}_B > \boldsymbol{u}_B$ in order to trigger the activation of the inequality constraints on level 2. Initially, the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are of size $m_A = n - 1$ and $m_B = 1$ respectively. Their sizes are then incrementally

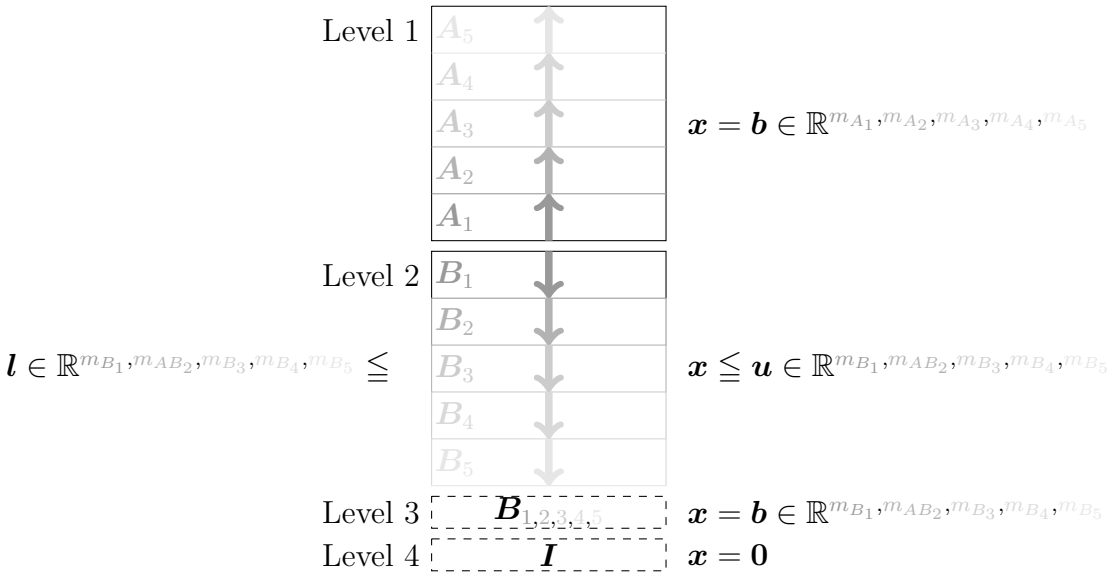


Fig. 6.2. ***Test 1****, schematic of the problem development. The size of* $\boldsymbol{A}$ *is decreased and the size of* $\boldsymbol{B}$ *on the second and third level is increased successively.*

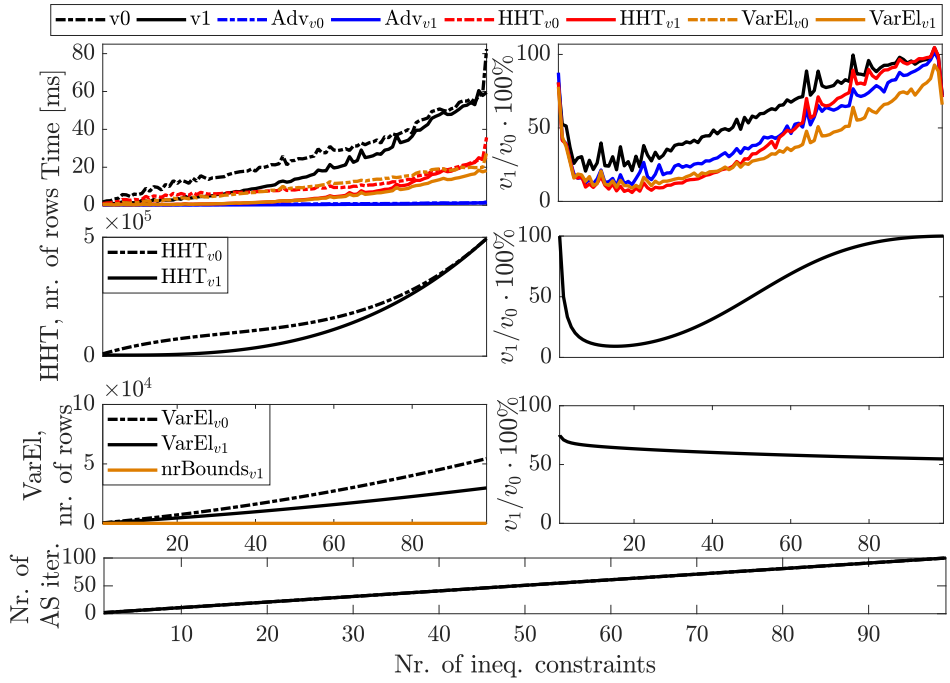


Fig. 6.3. ***Test 1****, evaluation data. The test is characterized by a decreasing number of equality constraints on the first and increasing number of inequalities on the second of four priority levels.*

decreased and increased until $\boldsymbol{A} \in \mathbb{R}^{1,n}$ consists of a single row and $\boldsymbol{B} \in \mathbb{R}^{n-1,n}$. The structure of the evolving problem is given in fig. 6.2.

The speed up accomplished with the factorization restart is heavily dependent on the priority level where the active-set change and therefore the restart is happening, and the size of the factorization of the previous levels. This example however can give a rough idea about the achievable behaviour (see fig. 6.3). The bound handling method is disabled.

With more inequalities added on the second level the number of active-set iterations increases linearly to it. LexLSI does a sequence of full steps to activate all the (violated) inequality constraints on the second level. Thereby, the matrix $\boldsymbol{A}$ is only factorized once in the first iteration. Especially in the beginning, when $\boldsymbol{A}$ still has high row dimensionality, the speed-up is significant. For $m_A = 86$ and $m_b = 14$ there are 15 active-set iterations (the first active-set iteration is without any constraints activated on level 2). The factorization of $\boldsymbol{A} \in \mathbb{R}^{86,50}$ is conducted only once instead of 15 times as it is the case for v0. This leads to v1 only taking about 30% (2.96 ms) of the time seen for v0 (9.73 ms). The reduced number of HHT operations has the biggest influence with a duration of 4.86 ms for v0 and 0.57 ms for v1. With decreasing size of $\boldsymbol{A}$ however, its QR decomposition takes less effort and makes the time savings of the factorization restart decrease as well.

The number of rows involved in the variable elimination sees a steeper increase for v0 than for v1. This is due to sparse structure of the last level of bound constraints where v1 omits bottom zeros with the knowledge of the column lengths.

6.5.2 Test 2: Computational cost of tracking the bound constraints

Test 2 can be considered a worst case scenario, with a large number of bound constraint present but with no possibility of making use of them. Level 1 is structured as a square and full rank constraint matrix $\boldsymbol{A} \in \mathbb{R}^{50,50}$ while increasingly more levels with square bound constraint matrices $\boldsymbol{I} \in \mathbb{R}^{50,50}$ are added until $p = 100$ is reached. All constraints are equality constraints so the problem is always solved within one active-set iteration. LexLSI stops after the decomposition of $\boldsymbol{A}$ since no variables are left for the decomposition of the remaining levels. The schematic of the test is given in fig. 6.4.

For the case $p = 100$, in every step of the QR decomposition of $\boldsymbol{A}$ the bounds of the pivot column need to be swapped 99 times to the bottom of the general constraints of each respective level. Furthermore, each time they need to be removed from the bound tracker. In fig. 6.5 a linear increase of the 'Advance' computation times with the number of pure bound levels can be observed, making it 50 times slower than without bound handling for $p = 100$. However, its absolute computation of 0.25 ms can be considered small, making up only 2% of the overall computation time of 12.89 ms.

The number of rows involved in the HHT stays constant for v0 and v1 since the first level is of full rank so no variables are left for the decomposition of the lower levels. The number of rows involved in the variable elimination increases linearly for both v0 and v1 due to the increasing number of lower priority levels.

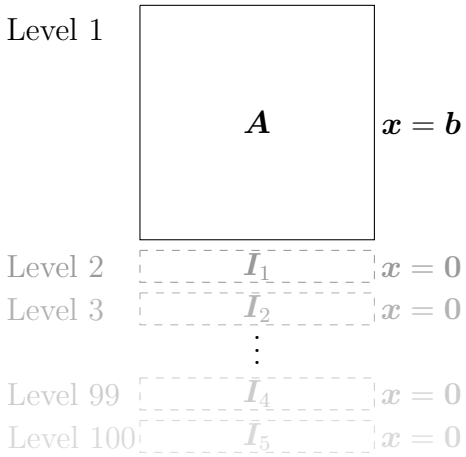


Fig. 6.4. ***Test 2****, schematic of the problem development. More pure bound constraint matrices* $\boldsymbol{I}$ *are added on an increasing number of priority levels.*

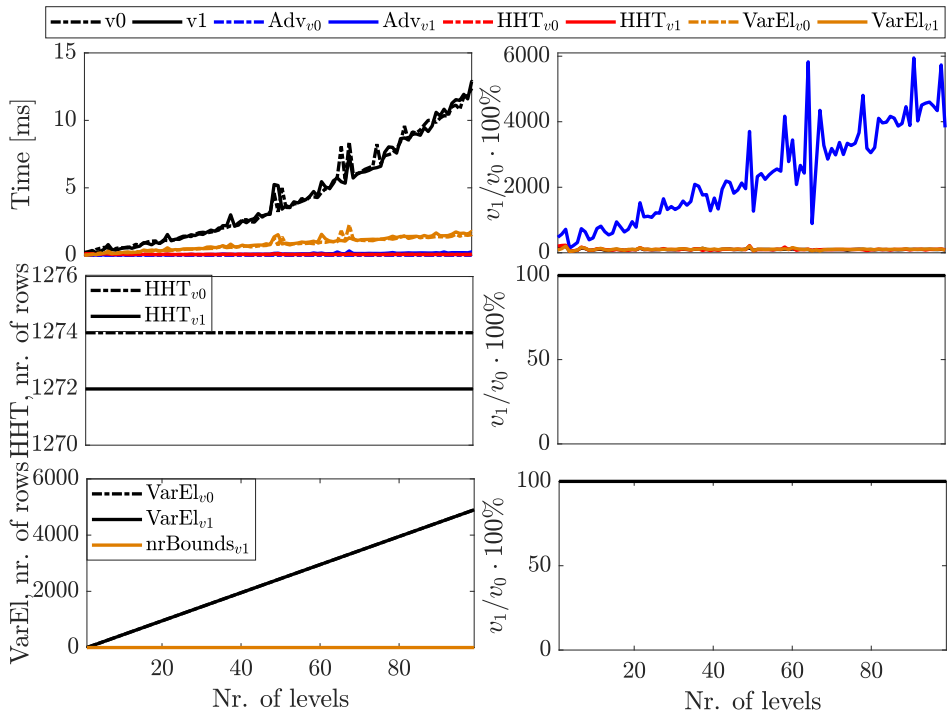


Fig. 6.5. ***Test 2****, evaluation data. The test is characterized by an increasing number of levels constraining all variables.*

6.5.3 Test 3: Taking advantage of bound constraints during the QR decomposition

In *test 3* the general ability of the bound handling to speed up the QR decomposition of a square full rank matrix is shown. The matrix $\boldsymbol{A}$ is of dimensions $\boldsymbol{A} \in \mathbb{R}^{50,50}$. The number of bounds is increased incrementally by 1 such that $\boldsymbol{I} \in \mathbb{R}^{0\to 50,0\to 50}$ and

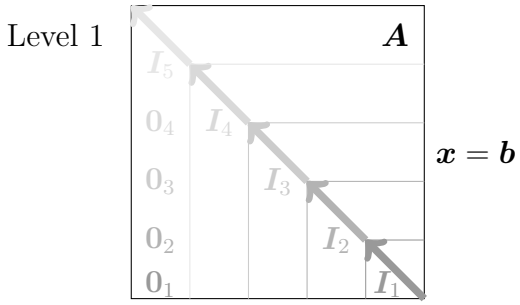


***Fig. 6.6. Test 3**, schematic of the problem development. The number of bound constraints in the matrix **A** is increased successively.*

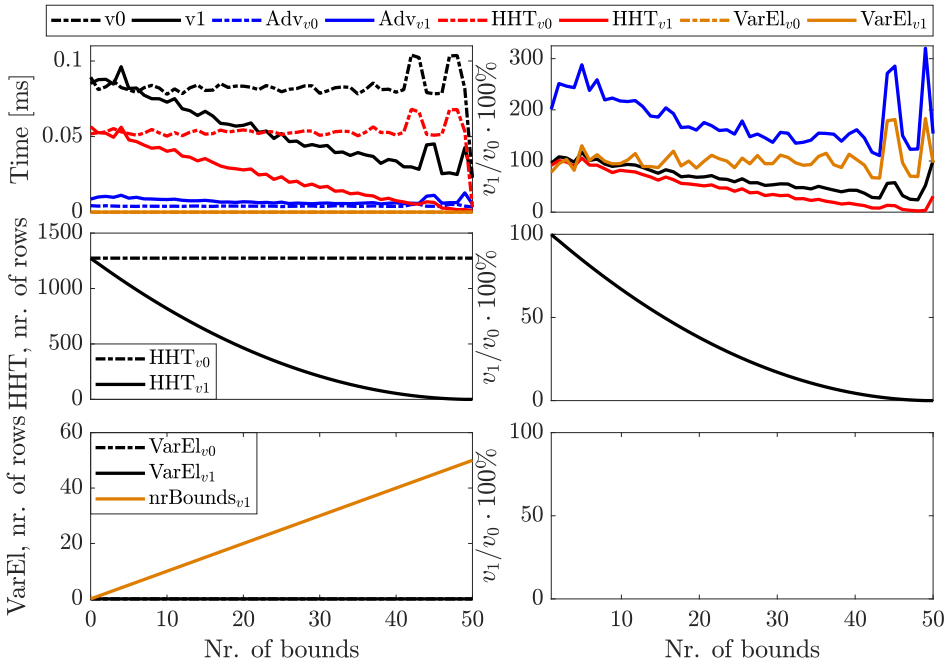


***Fig. 6.7. Test 3**, evaluation data for the decomposition of a square matrix with increasing number of bound constraints.*

eventually $\boldsymbol{A}$ only consists of bound constraints $\boldsymbol{A} = \boldsymbol{I}$. The general structure of the problem is given in fig. 6.6.

Timings are given in fig. 6.7. The computation time of v1 decreases linearly with the addition of bound constraints. Eventually, the computation times of v1 fall below 50% of the ones seen for v0. Foremost, the rank revealing QR decomposition is only applied on $\boldsymbol{M}$ (and not on $\begin{bmatrix}\boldsymbol{M}^T & \boldsymbol{0}^T\end{bmatrix}^T$) which is reflected in the decreasing number of HHT rows (HHT$_{v1}$) in the left graph of the second row of fig. 6.7. This is due to the knowledge of the column lengths which enables to make use of the sparsity of the bound constraints. Once v1 proceeds to the QR decomposition of $\boldsymbol{I}$ the simplified HHT is applied, leading to an additional time saving. This is reflected in the increasing

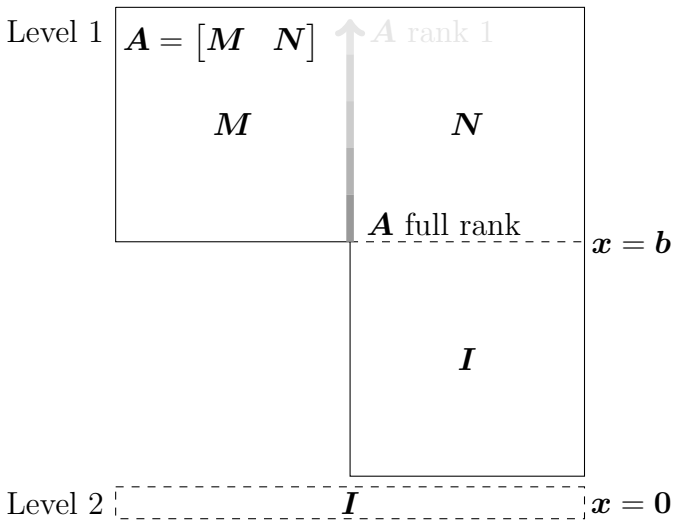


Fig. 6.8. ***Test 4****, schematic of the problem development. The rank of the matrix* $\boldsymbol{A}$ *is decreased successively.*

number of pure bound constraints $nrBounds_{v1}$.

For $\boldsymbol{A} = \boldsymbol{I} \in \mathbb{R}^{50,50}$ a sharp decrease in computation time is seen for v0. This is due to the fact that the HHT from the Eigen library Guennebaud et al. [2010] does not apply the trivial (since $\boldsymbol{H}^{22} = \boldsymbol{I}$) multiplication (6.3).

The timings for the case of 49 bound constraints $\boldsymbol{I} \in \mathbb{R}^{49,49}$ are as following: v0 takes 0.082 ms for the decomposition, with the HHT adding 0.053 ms and 'Advance' adding 0.004 ms. v1 on the other hand takes 0.043 ms for the decomposition, with HHT contributing with 0.002 ms and 'Advance' adding 0.013 ms to the overall computation time.

Since there is only one level no variable elimination is conducted.

6.5.4 Test 4: Taking advantage of bound constraints during the variable elimination

Test 4 shows the computational speed up achieved by leveraging bound constraints during the variable elimination. A two level problem is defined. Level 1 is a square matrix $\in \mathbb{R}^{100,100}$ composed of a rectangular matrix $\boldsymbol{A} \in \mathbb{R}^{50,100}$ and a bound constraint matrix $\boldsymbol{I} \in \mathbb{R}^{50,50}$ constraining the last 50 variables. The rank of $\boldsymbol{A}$ is incrementally decreased from 50 to 1. A fully determined problem is obtained with the minimal norm task $\boldsymbol{I} \in \mathbb{R}^{100,100}$ which is added on the second level, bounding all variables. The schematic of the test is given in fig. 6.8.

In this example significant speed-up is achieved (for $rank(\boldsymbol{A}) = 50$, v1 only takes 61% of the time needed for v0, and for $rank(\boldsymbol{A}) = 1$, v1 only takes 40% of the time needed for v0) since the problem is dominated by bound constraints (see fig. 6.9).

First, a rank revealing QR decomposition of $\boldsymbol{M}$ is conducted. Once its rank (or $\boldsymbol{A}$'s

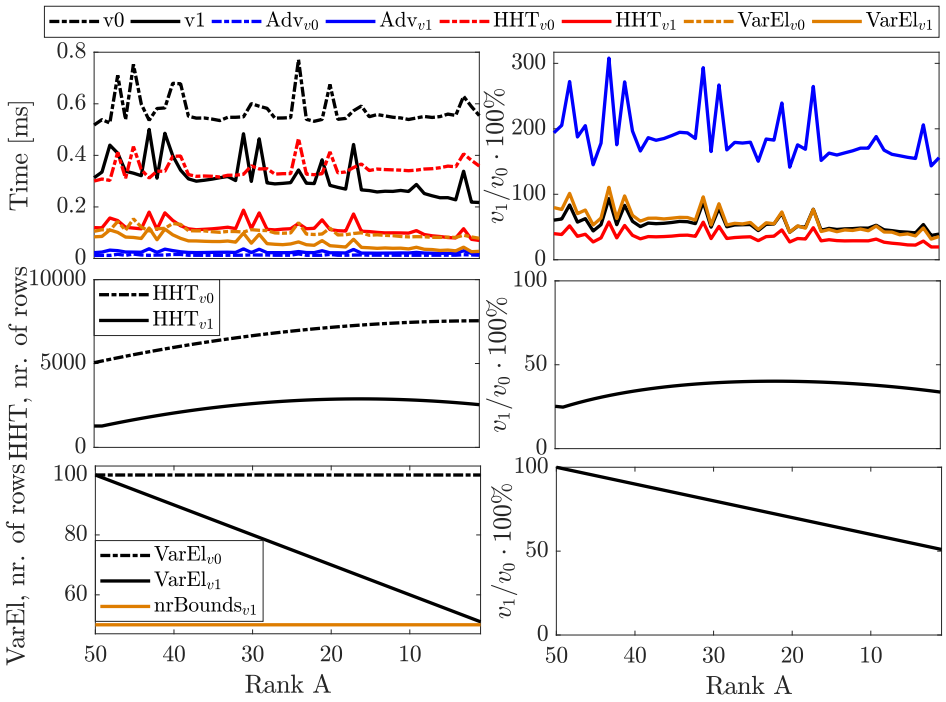


***Fig. 6.9. Test 4**, evaluation data with decreasing rank of **A** on the first level, leaving more and more variables for the decomposition of the bound constraints on the second level.*

rank, respectively) is revealed, the algorithm proceeds to the simplified decomposition of the identity matrices on the first and second level. v1 only Householder-transforms less than 50% of the rows transformed by v0. The number of rows involved in the HHT for v1 increases until $rank(A) \approx 15$ and then starts to decrease again. The increase comes from the fact that increasingly more rows on the second level need to be Householder transformed during the QR decomposition (in the beginning no variables are left for the QR decomposition of the second level). The saturation on the other hand is due to the fact that in the beginning a high number of bound constraints on the second level are permuted to the 'left side', creating a high count of general constraints during the variable elimination. This is also the reason why the number of pure bound rows 'nrBounds$_{v1}$' does not increase linearly with decreasing rank of the matrix $\boldsymbol{A}$ but stays constant at 'nrBounds$_{v1}$'= 50. This corresponds to the number of bound constraints on level 1 which can be handled as pure bound constraints, enabling a shortened HHT and a matrix block inversion during the variable elimination of the second level.

For $rank(\boldsymbol{A}) = 1$, v0 takes 0.55 ms for the decomposition, with the HHT adding 0.36 ms, 'Advance' adding 0.013 ms and 'VarEl' adding 0.078 ms to the overall computation time. v1 on the other hand takes 0.22 ms for the decomposition, with HHT adding 0.070 ms, 'Advance' adding 0.020 ms and 'VarEl' contributing with 0.028 ms to the overall computation time.

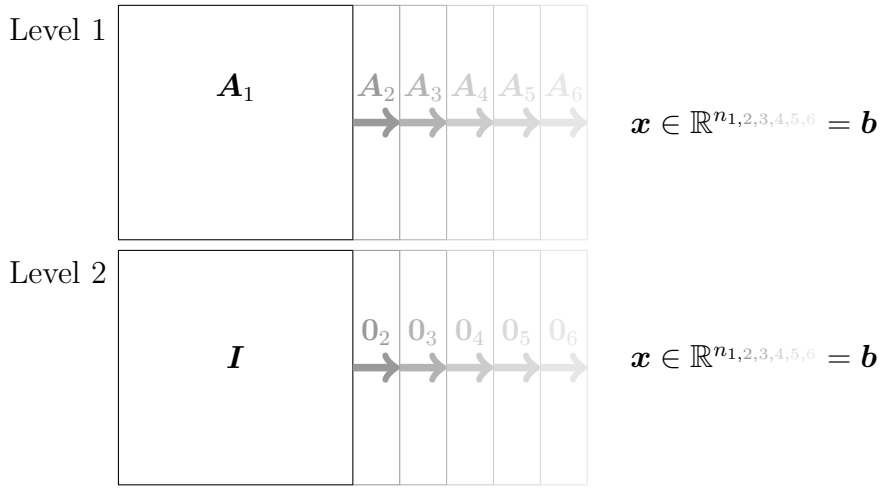


Fig. 6.10. Test 5*, schematic of the problem development. The number of variables n is increased successively.*

6.5.5 Test 5: Increasing the number of variables

Test 5 shows a problem on which the solver v1 can hardly achieve any computational accelerations despite the presence of bound constraints. The problem starts with full rank matrices $\boldsymbol{A} \in \mathbb{R}^{50,50}$ on the first and $\boldsymbol{I} \in \mathbb{R}^{50,50}$ on the second level. The number of variables n is increased from 50 to 100 ($\boldsymbol{x} \in \mathbb{R}^{50 \to 100}$) with $\boldsymbol{A} \in \mathbb{R}^{50,50 \to 100}$ on the first and $\begin{bmatrix} \boldsymbol{I} \in \mathbb{R}^{50,50} & \boldsymbol{0} \in \mathbb{R}^{50,0 \to 50} \end{bmatrix}$ on the second level. Note that the problem is always fully determined since $n \leq m_1 + m_2$. The schematic of the test is given in fig. 6.10.

The achieved speed-up is relatively small, if noticeable at all (see fig. 6.11). The reason is that the rank revealing QR decomposition of $\boldsymbol{A}$ chooses pivot columns indistinguishably of possible bound constraints on the second level. These bound constraints then cannot be used for a simplified pivot column search and HHT on the second level.

Nonetheless, for $n = 100$, v1 takes about 90% of the computation time required for v0. v1 applies the HHT only on about 66% of the rows of v0 but this only leads to a 15% speed-up in HHT's computation time. In fig. 6.12 it can be seen that the overall relative HHT effort (corresponds to the size '$nrRows \times nrCols$' of the matrices the HHT is applied on) is a bit higher (81%) than just the HHT row count (66%).

The number of rows involved in the HHT for v1 increases until the problem has grown to about 80 variables and then starts to decrease again. This is the same phenomenon as seen in test 4: bound constraints on the second level are swapped to the 'left side' during the QR decomposition of the first level. These constraints are then converted to general constraints which shapes the level 2 constraint matrix into the form (6.35). This increases the number of rows that need to be Householder transformed and prohibits the simplified HHT in general.

The computational effort for the variable elimination decreases to 56% of the one of v0. Its overall time is small compared to the one of the HHT and therefore only leads to a small decrease of overall computation time of v1. The decrease is due to the fact

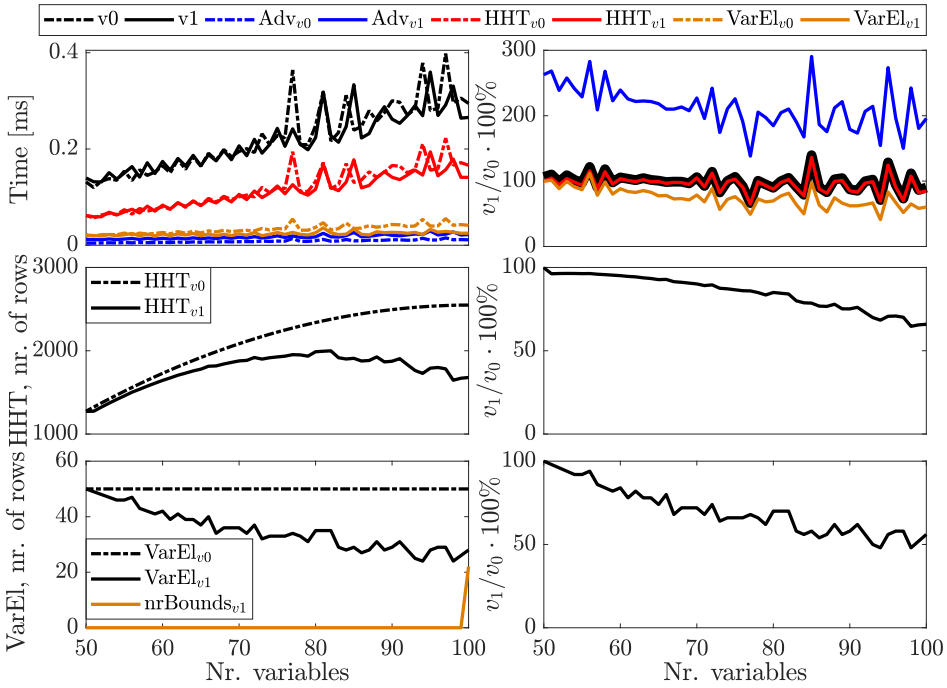


***Fig. 6.11. Test 5**, evaluation data with an increasing number of variables, leaving more and more variables for the decomposition of the bound constraints on the second level.*

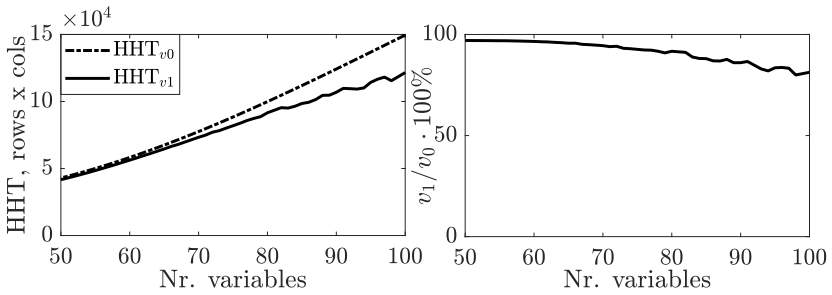


***Fig. 6.12. Test 5**, number of overall matrix operations during the HHT.*

that in the beginning all bound constraints are swapped to the 'left side' during the QR decomposition of the first level. However, with an increasing number of variables chances decrease to pivot a column which corresponds to a bounded variable on the second level. This leads to an approximately linear decrease of the rows involved in the variable elimination of the second level.

For $n = 100$, v0 takes 0.295 ms for the decomposition, with the HHT adding 0.166 ms, 'Advance' adding 0.011 ms and 'VarEl' adding 0.041 ms to the overall computation time. v1 on the other hand takes 0.266 ms for the decomposition, with HHT contributing with 0.141 ms, 'Advance' adding 0.022 ms and 'VarEl' contributing with 0.025 ms to the overall computation time.

6.6 Conclusion

In this chapter we adapted the given solver LexLSI by a few modifications in order to gain computational speed-up. Our implemented methods of restarting factorizations, the bound and column length handling and the matrix block inversion method indeed can lead to a computational improvement. However, the five tests showed that the achievable speed-up in computation time depends heavily on the structure of the problem.

For the factorization restart method it is relevant on which level the active-set change takes place and of what dimensions the decomposition of higher priority levels is. With active-set changes on high priority levels (as it is the case for typical robot applications, e.g. joint, joint torque, contact wrench and trust region limits) the savings in computation times might be negligible since the factorization of only a small number of levels can be kept.

The same holds for the bound handling method. While keeping track of the bound constraints only adds a negligible overhead, time savings can only be achieved if there is a certain amount of bound constraints present. In this case a simplified column pivot search, a simplified HHT, or a HHT on a reduced number of rows can be applied. Of advantage is that in typical robotic applications bound constraints are on high priority levels (e.g. joint, joint torque, contact wrench and trust region limits) which promotes cheap variable elimination on a high number of lower priority levels using block matrix inversion techniques.

For typical robot applications however, the introduced improvements are not sufficient to compensate for the high number of active-set iterations observed in our experiments, see sec. 5.4.2. It is therefore necessary to dedicate further research on how to counter these numerical issues and on how to incorporate factorization updates into LexLSI.

Summary and outlook

Summary

With this thesis we developed an efficient method for the resolution of kinematic and algorithmic singularities in multi-level constrained robotic control problems. Thereby, we achieved high convergence rates and overcame the necessity for damping tuning necessary for classical Levenberg-Marquardt based approaches. The methods were evaluated in simulation for the kinematic control case and on real robot experiments with the HRP-2Kai robot for the dynamic control case. Besides the advantages given by the formulation of hierarchical least-squares problems with constraint relaxation itself (and which are detailed in chp. 3), we achieved following goals with our singularity resolution methods LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS:

- Kinematic and algorithmic singularity resolution for multi-level constrained least-squares control problems while ensuring
  - Numerical (joint) stability
  - Joint smoothness
  - Strict prioritization
  - Better convergence than damping based approaches
  - Computational affordability.
- No need for damping tuning. Our Newton's method can be rather interpreted as 'optimal damping' as it is closely related to the LM algorithm.
- Least squares formulation enabling the usage of state-of-the-art and fast hierarchical least-squares solvers.
- Demonstration of numerical stability and good convergence when kinematic and algorithmic singularities are approached and reached in a test bench with 20 test cases and three real robot experiments.

On our way to these results, we first conducted an experimental validation of a controller based on a weighted two level hierarchical least-squares solver without constraint

relaxation. The necessary accuracy for the high precision industrial task of nut fastening is provided. However, a clear distinction between safety and physical stability relevant tasks is not possible. Additionally, the feasibility of constraints needs to be ensured at all times. This gave strong motivation for introducing multi-level constrained least-squares hierarchies with constraint relaxation.

Solving linearized control problems with such hierarchically constrained least-squares solvers requires the resolution of kinematic and algorithmic singularities. The approach we presented in this thesis is based on principles originating from unconstrained optimization, namely the GN algorithm and Newton's method. Both can be derived from the second order Taylor approximation of the quadratic error norm function and can be expressed as a least-squares problem. The GN algorithm thereby neglects second order information which leads to numerical instabilities close to singularities due to insufficient model representation. In these cases we switch to Newton's method with the help of a reliable switching method based on the feasibility of the least-squares problem.

We adapted both the GN algorithm and Newton's method for the case of multi-level constrained optimization by finding a formulation for the corresponding Lagrangian gradient and Hessian. The former allowed the design of a BFGS based approximation method for the Lagrangian Hessian (LexLSAug2BFGS). It is computationally cheap and yields positive definite updates. This is in contrast to the analytic calculation (LexLSAug2AH) or the approximation by the SR1 method (LexLSAug2SR1) which both require an expensive SVD based regularization approach (Higham regularization) for the possibly indefinite Lagrangian Hessian. However, LexLSAug2AH is superior in terms of robot behaviour.

We formulated both the GN algorithm and Newton's method as trust region methods. A trust region adaptation method for multi-level constrained optimization was devised which observes directly the robot's joint velocity state. This is in contrast to other approaches which are designed for unconstrained optimization or optimization with a single constraint level and are not easily transferable to constrained optimization. The adaptation method effectively suppresses occurring numerical instabilities after a few control iterations. A further method observing the robot's joint acceleration state introduces joint-wise damping to the direct BFGS Hessian approximation (LexLSAug2BFGS). This further improves numerical stability and joint smoothness.

The GN algorithm and Newton's method all rooted in optimization were then adapted to the robot control case. We achieved this by incorporating a control time step $\Delta t \ll 1$ s into our methods LexLSAug2AH, LexLSAug2SR1 and LexLSAug2BFGS. This required weighting the Lagrange multipliers with the time step for correctly calculating the Lagrangian gradient and Hessian for Newton's method.

As the last step we adapted our methods to the inclusion of dynamics constraints. The equation of motion and the corresponding motion controllers are of second order. However, Newton's method is not well defined in the acceleration domain. We therefore designed motion controllers which emulate acceleration-based control in the velocity domain. The equation of motion was accordingly adapted by forward integration which

was validated with the simulation of a freely swinging pendulum. We compromised between the consistency with the acceleration-based control and the correct calculation of the Lagrange multipliers. Our real robot experiments showed the validity of this compromise as it yields dynamic values in accordance with the reference values from acceleration-based controllers and wrench sensor data.

Outlook

With our proposed methods we are able to dynamically control kinematic structures in a multi-level hierarchy around kinematic and algorithmic singularities. Following issues however require more research effort:

- Problems occurred with the active search in dynamically challenging situations. Here, a high number of active-set iterations leads to violation of the real-time computation constraint. While a few computational adjustments for the hierarchical least squares solver LexLSI have been introduced they are not sufficient to overcome this problem. Further investigations first must address the reason why these numerical issues occur when solving dynamic robot problems. Secondly, the solver itself needs to be improved by implementing factorization updates or a more effective active-set search and warm-start strategy.
- Further developments of the trust region adaptation methods are necessary to allow very fast motions. A sufficiently small radius is necessary in the vicinity of singularities in order to ensure numerical stability by good model accuracy of the second order Taylor approximation. However, this limits the maximum speed in regular configurations and requires a trust region relaxation if faster motions are required.
- It would be desirable for LexLSAug2BFGS to directly update the Cholesky decomposition of the Lagrangian Hessian similarly to Fletcher [2006] in order to save computation time.
- For the indefinite Lagrangian Hessian of LexLSAug2AH and LexLSAug2SR1 cheaper regularization methods than the Higham regularization based for example on the Bunch-Kaufman decomposition require our attention. Also, occurrences of negative definite curvature during the BFGS updates (LexLSAug2BFGS) are ignored but rather the last positive definite update is kept and used for Newton's method. This slows down convergence and also might lead to numerical instability and needs to be investigated.
- We do not elaborate on the discontinuity that comes with a switch from the GN algorithm to Newton's method. A perturbation analysis for least-squares problems in the sense of Björck [1996] is relevant for full rank problems and gives an upper bound on the discontinuity. However, with our full rank augmentation the rank of the least-squares problem usually changes since the number of constraints is smaller than the number of variables. In this case only a lower bound

can be found [Wei 1992; Wedin 1973]. Putting these results into context of our hierarchical least-squares formulation needs to be addressed in future work.

- In the first set of evaluation tests we counted consecutive sign changes of the calculated joint velocity vector in order to obtain a quantitative value for numerical instabilities. While this measure gave a rough idea about the numerical stability performance of the methods it is desirable to formulate a more universal definition of numerical instability. For example, measures based on the singular values of the Jacobian or the Lagrangian Hessian could be considered.
- Finally, with our work we introduced the concept of dynamically feasible kinematic control by incorporating the equation of motion and corresponding dynamic constraints on a high priority level. This shapes the kinematic control output from motion controllers of lower priority levels in such a way that the motion is dynamically feasible and obeys the dynamics constraints. However, force control (for example on some low priority level) and the concept of dynamical consistency require more attention and it needs to be investigated how they can be incorporated into our singularity resolution schemes.

APPENDIX A

Trust region adaptation methods from constrained optimization

A common trust region adaptation method for unconstrained optimization Nocedal et Wright [2006] decides upon the scalar value

$$\rho = \frac{red_{\text{act}}}{red_{\text{pred}}} = \frac{\Phi(\boldsymbol{q}^{(k)}) - \Phi(\boldsymbol{q}^{(k)}) + \boldsymbol{\Delta q}^{(k)})}{\Phi(\boldsymbol{q}^{(k)}) - \epsilon^{(k)}} \tag{A.1}$$

$$= \Phi(\boldsymbol{q}^{(k)}) - \Phi(\boldsymbol{q}^{(k)} + \boldsymbol{\Delta q}^{(k)}) \tag{A.2}$$

$$\tag{A.3}$$

$$= \frac{\Phi(\boldsymbol{q}^{(k)}) - \Phi(\boldsymbol{q}^{(k)}) + \boldsymbol{\Delta q}^{(k)})}{\boldsymbol{\Delta q}^{(k),T}\boldsymbol{J}^{(k),T}\boldsymbol{e}^{(k)} - \frac{1}{2}\boldsymbol{\Delta q}^{(k),T}(\boldsymbol{J}^{(k),T}\boldsymbol{J}^{(k)} + \boldsymbol{H}^{(k-1)})\boldsymbol{\Delta q}^{(k-1)}} \tag{A.4}$$

whether a step is accepted or rejected. Φ and ϵ are the quadratic norms of the task error (4.7) and the slack (4.76), respectively. red_{act} and red_{pred} are the actual and predicted reduction between two consecutive control iterations. From a computational point of view, note that the calculation of $\Phi(\boldsymbol{q}^{(k)} + \Delta\boldsymbol{q}^{(k)})$ requires a full update of the robot forward kinematics. Additionally, ρ is not defined for a converged robot with $\boldsymbol{\Delta q} = \boldsymbol{0}$ or $\Phi^{(k)} = 0$ and $\epsilon^{(k)} = 0$.

In unconstrained optimization and with a well chosen trust region radius, the model yields a positive reduction $red_{\text{pred}} \geq 0$ at all times. If this is not the case, the step is rejected and a new step is calculated with an adapted, smaller trust region. Choosing a trust region such that the original function is well represented by the model is not trivial. Additionally, we allow the desired value $\boldsymbol{f}_d(t)$ to change over time which could mean that a target is moving away from the current value $\boldsymbol{f}(\boldsymbol{q}(t))$, leading to $red_{\text{pred}} < 0$ (this can be countered by calculating $\Phi(\boldsymbol{q}^{(k)} + \Delta\boldsymbol{q}^{(k)})$ of red_{act} with $\boldsymbol{f}_d(t^{(k)})$ instead of $\boldsymbol{f}_d(t^{(k+1)})$).

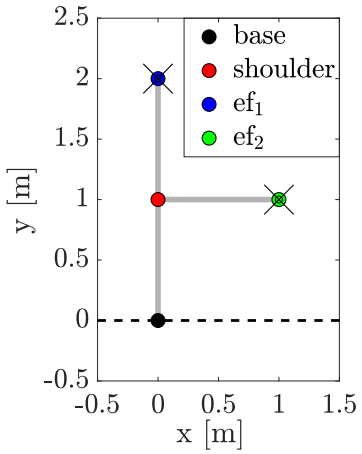


Fig. A.1. *Converged robot for test 4 with two just in reach static targets for each end-effector.*

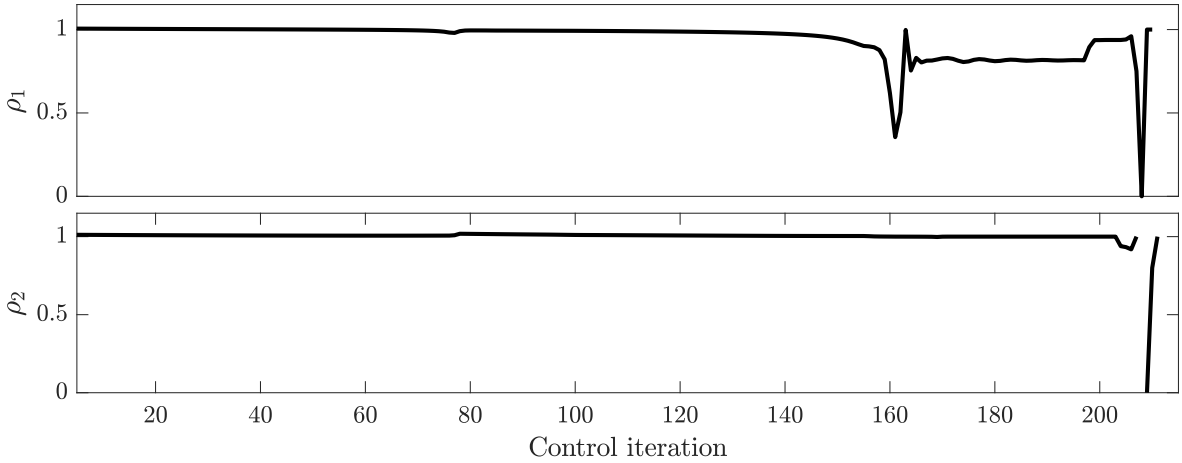


Fig. A.2. *Test 4 with two static targets for each end-effector. The upper graph shows ρ_1 of level 1 and the lower graph shows ρ_2 of level 2.*

First, we look at the behaviour of ρ during test 4 (T4) of our validation section 4.5.1. In the converged configuration, both targets are exactly reachable (see fig. A.1) with the level 1 end-effector in kinematic singularity and the level 2 task exactly on the limit of being in algorithmic singularity with the level 1 task. The values of ρ for the level 1 and level 2 end-effectors are given in the upper and lower graph of A.2 respectively. The actual and predicted reduction match well in the beginning with $\rho_1 \approx 1$ whereas at around 160 iterations the value drops which indicates a model misrepresentation with the predicted reduction being larger than the actual one. Shortly before convergence at iteration 206, the end-effector comes to a halt with zero actual reduction $red_{act,1} = 0$, $\rho_1 = 0$. In the converged and numerically stable (due to switching to Newton's method) state, for both levels we have $\Phi = 0$ and $\epsilon = 0$ for which ρ is not defined (from iteration 208). For level 2, ρ_2 in the bottom graph also shows a good model state despite its conflict with the level 1 task during the approach of the level 2 target.

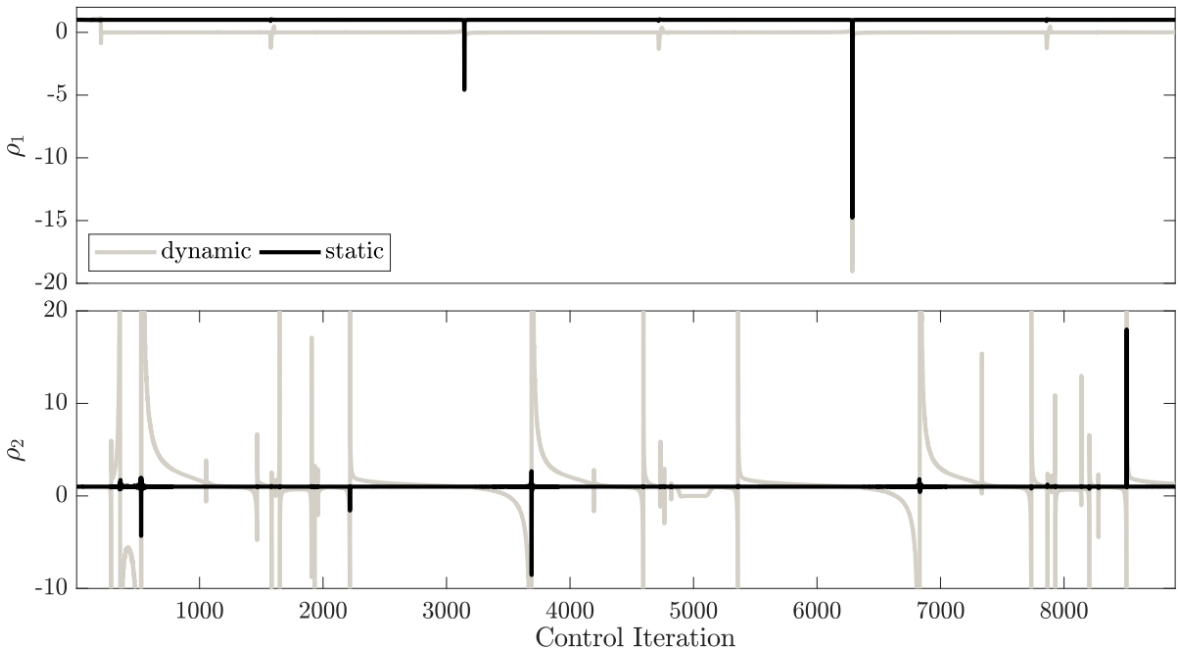


Fig. A.3. *Test 1 with diagonally oscillating targets for each end-effector. The upper graph shows ρ_1 of level 1 and the lower graph shows ρ_2 of level 2. The static value thereby accounts for the changing value of $\boldsymbol{f}_d(t)$ by using $red_{act}(\boldsymbol{f}_d(t^{(k)}))$ instead of $red_{act}(\boldsymbol{f}_d(t^{(k+1)}))$.*

Next, the behaviour of ρ for test 1 (T1, refer to the validation section 4.5.1) is analysed. In contrast to test 4, the targets for the two end-effectors are not static but change over time. The values of ρ both calculated with $red_{\text{act}}(\boldsymbol{e}(\boldsymbol{f}_d^{(k+1)}))$ ('dynamic') and $red_{\text{act}}(\boldsymbol{e}(\boldsymbol{f}_d^{(k)}))$ ('static') for the level 1 and level 2 end-effectors are given in fig. A.3. For level 1, the dynamic ρ value is almost constantly zero since $red_{\text{act}} = 0$ due to the constantly changing desired value $\boldsymbol{f}_d(t)$. However, the static values nicely compensate for this on both levels, showing good matching between the actual and predicted reduction with $\rho_{\text{static}} \approx 1$.

A possible trust region adaptation method could for example reduce the trust region radius for joints that are on the kinematic chain of a faulty task. Such a faulty task could be judged upon its deviation of ρ from the optimal value 1.

Another option could be based on the filter method of Fletcher et Leyffer [2002]. Here, $p-1$ filters are introduced for every of the p constrained QP problems except the first one. Infeasible constraints are accepted. However, it is difficult to adapt the concept of step domination and constraint minimization. Especially on lower priority levels, dominated steps might occur all the time due to conflict with higher priority tasks. This leads to a constant reduction of the trust region until all the joints on the kinematic chain of these tasks are set to zero.

APPENDIX B

Illustration of the effect of the second order augmentation

In the following we illustrate the influence of second order augmentation on the GN algorithm. For this, we take a fixed based robot in kinematic control mode (velocity-based, proportional controller $\dot{\boldsymbol{e}}^{\text{ctrl}}$ (2.4)) with eight revolute joints. Six of these joints (q_1, q_2, q_3, q_4, q_5 and q_6) are on the kinematic chain of the level 1 task. The end-effector tip is asked to move to a target along the y-axis which is out of reach. Another end-effector is branching off its kinematic chain and is composed of the joints q_1, q_2, q_3, q_7 and q_8. Its target is in reach (see fig. B.1).

In the beginning, no augmentation is necessary since joint velocities for the level 1 end-effector can be created by the robot which minimize the error between current and desired position. The problem is solved by the GN algorithm. Close to kinematic singularity however, this ability gets lost due to imminent rank loss of the task Jacobian $\boldsymbol{J}_1$ in the y-direction. A switch to Newton's method is necessary to ensure numerical stability on joint level of the level 1 kinematic chain. We augment the Jacobian $\boldsymbol{J}_1$ with the weighted dotted identity matrix $\boldsymbol{I}_1^*(q_1, q_2, q_3, q_4, q_5, q_6)$ as an approximation of the true Hessian.

At this point all the joints q_1 to q_6 must be fully occupied with the fulfilment of the level 1 task in order to minimize the error to the target *at best*. This can be easily seen from fig. B.1 with the level 1 kinematic chain being in kinematic singularity. The original non-linear problem ensures this naturally. However, it could not be achieved solely by the GN algorithm which can fully 'dedicate' only two joints to the level 1 task due to the Jacobian being of maximum rank $rank(\boldsymbol{J}_1) = 2$. Instead, we rely on a full rank (full rank with respect to the level 1 kinematic chain, $rank(\boldsymbol{I}_1^*) = 6$) augmentation of the GN algorithm to the Newton's method. The augmentation 'locks' the joints such that they are fully dedicated to the movement (or no movement thereof

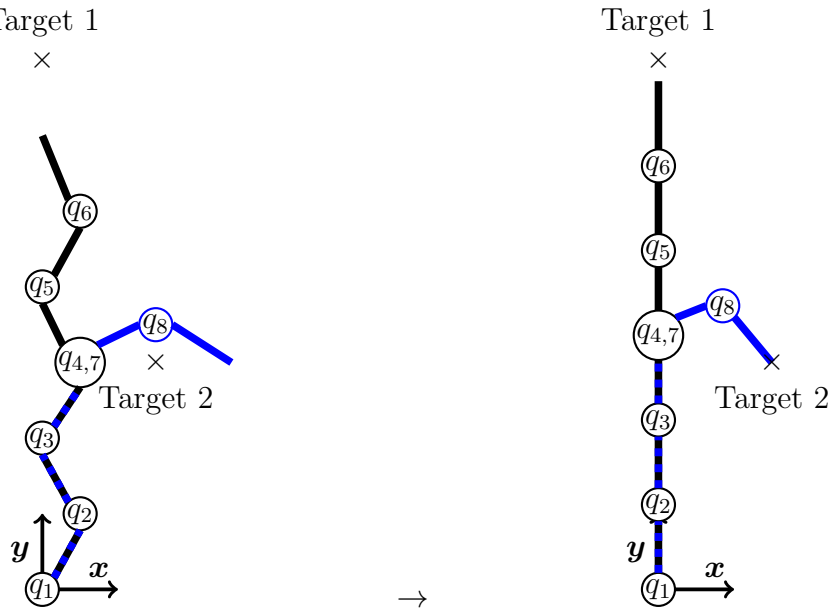


Fig. B.1. *Left: the robot in initial configuration. $q_{4,7}$ consists of two distinct revolute joints q_4 and q_7 such that the rotation of q_7 has no influence on the level 1 end-effector and the rotation of q_4 has no influence on the motion of the level 2 end-effector. The level 1 target is out of reach (black kinematic chain), the level 2 target is in reach (blue and dashed blue kinematic chain). Right: the robot in converged configuration. The black kinematic chain is fully dedicated to the level 1 task. The full rank augmentation $\boldsymbol{I}_1^*(\boldsymbol{q} \setminus q_7, q_8)$ does not influence the joints q_7 and q_8 which is the only one left for controlling the level 2 end-effector.*

by preventing numerical instabilities and influence from lower priority levels) of the corresponding singular task.

The joints q_1 to q_3, which are shared by the kinematic chains of both the level 1 and level 2 tasks, are now fully dedicated to the achievement of the level 1 task. By virtue of the dotted identity matrix $\boldsymbol{I}_1^*(\boldsymbol{q} \setminus q_7, q_8)$, the two joints q_7, q_8 are still used for the fulfilment of the level 2 end-effector task.

Figure B.2 shows the corresponding joint positions and velocities for the level 1 end-effector converging to target 1 in kinematic control for LexLSAug2AH. The joint positions and velocities of the level 2 end-effector are not displayed in order to avoid a cluttered graph. The time step is chosen as $\Delta t = 1$ s so $\boldsymbol{\Delta q} = \dot{\boldsymbol{q}}$ holds. Also, the proportional gain k_p for the end-effector tasks is chosen as $k_p = 0.01$ in order to achieve a slow robot behaviour. The algorithm switches to Newton's method at control iteration 129 for a trust region radius of $\Delta = 0.01$ rad/s. Note that the switch would occur earlier for a smaller trust region radius and later or not at all if the trust region is chosen larger or too large in the worst case.

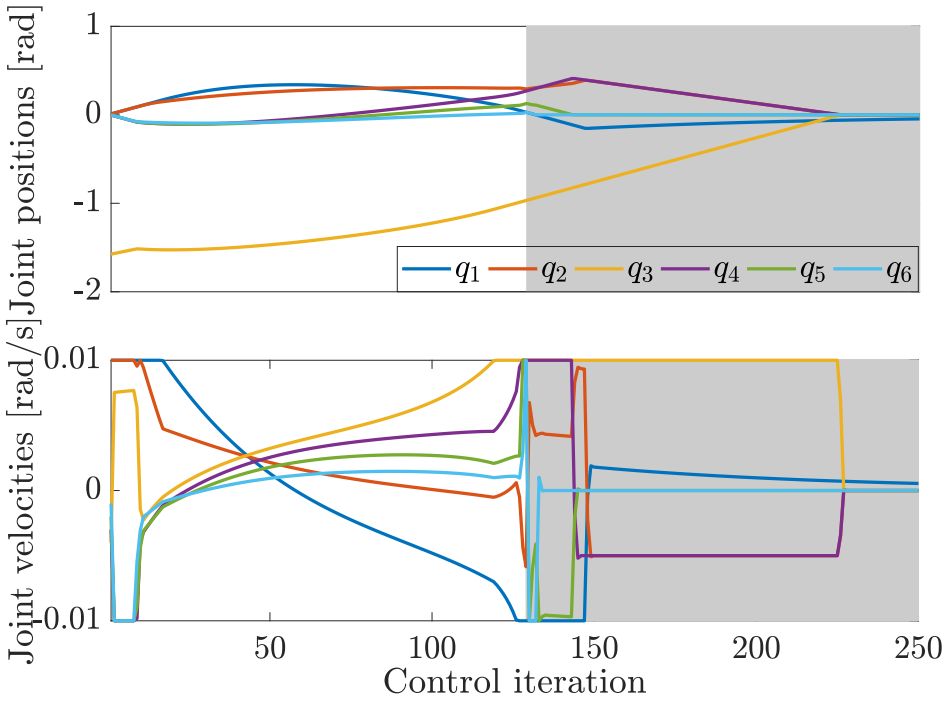


Fig. B.2. *Joint positions and velocities for LexLSAug2AH of the joints q_1, q_2, q_3, q_4, q_5 and q_6 of the level 1 end-effector converging to target 1. The switch to Newton's method takes place at control iteration 129 and is depicted with a gray background.*

APPENDIX C

Pseudo-algorithms

Algorithm 2 gives the structure of the main robot control loop. For simplicity, we assume only one task $-\dot{\boldsymbol{e}}_l^{\text{ctrl}} - \boldsymbol{J}_l\dot{\boldsymbol{q}}^{(k+1)} \leqq \boldsymbol{w}_l$ per level $l = 1, \ldots, p$. Note that $\dot{\boldsymbol{e}}_l^{\text{ctrl}}$ also includes $\dot{\boldsymbol{e}}_{\text{PD},l}^{\text{ctrl}}$ for dynamically feasible kinematic control. $LexLSI()$ solves problem (2.34) and implements an active-set search in the presence of inequality constraints. The found optimal active-set can then be accessed by $ActiveSet()$. $c_{\mathcal{A}}$ is an indicator for the highest among possible several priority levels where a change of the active-set occurred. $c_{\mathcal{A}}$ is set to -1 if no change of the active-set occurred. $LagrangeMultipliers()$ returns the Lagrange multipliers. $UpdateRobot()$ updates the tasks' Jacobians $\boldsymbol{J}$ and motion controllers $\dot{\boldsymbol{e}}^{\text{ctrl}}$ with the new joint positions and velocities $\boldsymbol{q}^{(k+1)}$ and $\dot{\boldsymbol{q}}^{(k+1)}$. The vector $\boldsymbol{H}$ stores the Hessian matrices of all geometric functions $\boldsymbol{f}_i \in \mathbb{R}^{m_i}$ of all levels p. They are calculated externally in the function $Hessian()$, for example as described in Erleben et Andrews [2017].

The augmentation $\boldsymbol{R}$ is calculated in LexLSAug2AH, LexLSAug2SR1 or LexLSAug2BFGS depending on the chosen method. Their pseudo-algorithms are given in the following. Bound constraints (i.e. constraints on the variable $\boldsymbol{x}$ of the form $\boldsymbol{I}\boldsymbol{x} \leqq \boldsymbol{c}$) do not need to be considered since $\boldsymbol{H} = \nabla_{\boldsymbol{q}}\boldsymbol{J} = \boldsymbol{0}$. This also holds for LexLSAug2SR1 and LexLSAug2BFGS since $\boldsymbol{J}^{(k)} - \boldsymbol{J}^{(k-1)} = \boldsymbol{0}$.

The augmentation in the lines 13, 14 and 15 is only necessary if the corresponding level contains a task which is formulated as Newton's method. Note that the augmentation is generally omitted for the equation of motion and dynamic and trust region constraints.

C.1 LexLSAug2AH

Algorithm 3 describes the calculation of the Lagrangian Hessian $\hat{\boldsymbol{H}}$ and its regularized decomposition $\boldsymbol{R}$ by LexLSAug2AH. $Higham(\boldsymbol{B})$ gives back the Cholesky factor $\boldsymbol{R}$ of

Algorithm 2 – Main control loop

$k = 0$
$\left[\underline{\boldsymbol{J}}^{(k)} \quad \underline{\dot{\boldsymbol{e}}}^{\text{ctrl},(k)}\right] = UpdateRobot(\boldsymbol{q}^{(k)}, \dot{\boldsymbol{q}}^{(k)} = \boldsymbol{0})$
while control = true **do**
  if LexLSAug2AH **then**
    $\boldsymbol{H} = Hessian()$
    $\underline{\boldsymbol{R}} = LexLSAug2AH(\boldsymbol{H}, \boldsymbol{\Lambda}^{\text{optim}})$
  else if LexLSAug2SR1 **then**
    $\underline{\boldsymbol{R}} = LexLSAug2SR1(\underline{\boldsymbol{J}}^{(k)}, \underline{\boldsymbol{J}}^{(k-1)}, \underline{\dot{\boldsymbol{e}}}^{\text{ctrl},(k)}, \boldsymbol{\Lambda}^{\text{optim},(k)}, \underline{\mathcal{A}}^{(k)}, c_{\mathcal{A}}, \boldsymbol{\Delta q}^{(k)}, \Delta t)$
  else if LexLSAug2BFGS **then**
    $\underline{\boldsymbol{R}} = LexLSAug2BFGS(\underline{\boldsymbol{J}}^{(k)}, \underline{\boldsymbol{J}}^{(k-1)}, \underline{\dot{\boldsymbol{e}}}^{\text{ctrl},(k)}, \boldsymbol{\Lambda}^{\text{optim},(k)}, \underline{\mathcal{A}}^{(k)}, c_{\mathcal{A}}, \boldsymbol{\Delta q}^{(k)}, \Delta t)]$
  end if
  for $l = 1 : p$ **do**
    $\boldsymbol{A}_l = \left[-\boldsymbol{J}_l^T \quad -\boldsymbol{R}_l^T\right]^T$
    $\boldsymbol{l}_l = \left[\dot{\boldsymbol{e}}_l^{\text{ctrl, lower bound},T} \quad \boldsymbol{0}^T\right]^T$
    $\boldsymbol{u}_l = \left[\dot{\boldsymbol{e}}_l^{\text{ctrl, upper bound},T} \quad \boldsymbol{0}^T\right]^T$
  end for
  $k \mathrel{+}= 1$
  $\left[\dot{\boldsymbol{q}}^{(k),T} \quad \boldsymbol{\tau}^{(k),T}/\Delta t \quad \boldsymbol{\gamma}^{(k),T}/\Delta t\right]^T = LexLSI.solve(\underline{\boldsymbol{A}}, \underline{\boldsymbol{l}}, \underline{\boldsymbol{u}})$
  $\left[c_{\mathcal{A}} \quad \underline{\mathcal{A}}^{(k)}\right] = LexLSI.ActiveSet()$
  $\boldsymbol{\Lambda}^{\text{ctrl},(k)} = LexLSI.LagrangeMultipliers()$
  $\boldsymbol{\Lambda}^{\text{optim},(k)} = \Delta t \boldsymbol{\Lambda}^{\text{ctrl},(k)}$
  $\boldsymbol{\Delta q}^{(k)} = \Delta t \dot{\boldsymbol{q}}^{(k)}$
  $\left[\underline{\boldsymbol{J}}^{(k)} \quad \underline{\dot{\boldsymbol{e}}}^{\text{ctrl},(k)}\right] = UpdateRobot(\boldsymbol{q}^{(k)}, \dot{\boldsymbol{q}}^{(k)}, \boldsymbol{\tau}^{(k)}, \boldsymbol{\gamma}^{(k)})$
  Wait until Δt is reached
end while

the input matrix $\hat{\boldsymbol{H}}$ regularized by the Higham method.

C.2 LexLSAug2SR1

Algorithm 4 describes the calculation of the Lagrangian Hessian $\hat{\boldsymbol{H}}$ and its regularized decomposition $\boldsymbol{R}$ by LexLSAug2SR1. Any underlined vector or matrix contains the values from level $l = 1, \ldots, p$. $\underline{\boldsymbol{B}}_i$ contains the SR1 approximations $\boldsymbol{B}$ from level $1, \ldots, i$.

Algorithm 3 – Lexicographic second order augmentation LexLSAug2AH

Input: $\boldsymbol{H}$, $\boldsymbol{\Lambda}^{(k)}$, Δt
for $l = 1 : p$ **do**
  for $i = l : p$ **do**
    $\hat{\boldsymbol{H}}_i \mathrel{+}= \sum_{d=1}^{m_i} \lambda_{i,l}^{d} \boldsymbol{H}_{d,l}$ ▷ Build Lagrangian Hessian
  end for
  if $\frac{1}{2}\|\boldsymbol{w}_l^J\|^2 > \Delta t^2 \nu$ **then** ▷ Newton's method
    $\boldsymbol{R}_l = Higham(\hat{\boldsymbol{H}}_l)$
  else ▷ GN algorithm
    $\boldsymbol{R}_l = \{\}$
  end if
end for
return $\underline{\boldsymbol{R}}$

C.3 LexLSAug2BFGS

Algorithm 5 describes the calculation of the Lagrangian Hessian $\hat{\boldsymbol{H}}$ and its decomposition $\boldsymbol{R}$ by LexLSAug2BFGS. $Cholesky(\boldsymbol{B})$ gives back the Cholesky factor $\boldsymbol{R}$ of the input matrix $\boldsymbol{B} + \iota \boldsymbol{I}^*$. The weighted identity matrix is added in order to make $\boldsymbol{B}$ strictly positive definite.

Algorithm 4 – Lexicographic second order augmentation LexLSAug2SR1

Input: $\underline{\boldsymbol{J}}^{(k)}, \underline{\boldsymbol{J}}^{(k-1)}, \underline{\dot{\boldsymbol{e}}}^{\mathrm{ctrl},(k)}, \boldsymbol{\Lambda}^{(k)}, \underline{\mathcal{A}}^{(k)}, c_{\mathcal{A}}, \boldsymbol{\Delta q}^{(k)}, k, \Delta t$
Internal variables: $\underline{\boldsymbol{B}}$

$\underline{\boldsymbol{y}} = \underline{\boldsymbol{0}}$
for $l = 1 : p$ **do**
  if $k = 0$ **or** $c_{\mathcal{A}} > -1$ **and** $l \geq c_{\mathcal{A}}$ **then**
    $\boldsymbol{B}_{1:m_l,l} = \sum_{i=1}^{l} \sum_{j \in \mathcal{A}_i^{(k)}} \max(\Delta t^2 \zeta, \frac{1}{2} \|\dot{\boldsymbol{e}}_{j,i}^{\mathrm{ctrl},(k)}\|_2^2) \boldsymbol{I}_{j,i}^{*}$ ▷ SR1 reinitialization
  end if
  if $k > 0$ **then**
    $\boldsymbol{y}_{1:m_l,l} = (\boldsymbol{J}_l^{(k)} - \boldsymbol{J}_l^{(k-1)})^T$ ▷ Gradient difference of $\boldsymbol{f}_l$
    if $(\boldsymbol{y}_{1:m_l,l} - \boldsymbol{B}_{1:m_l,l} \boldsymbol{\Delta q}^{(k)})^T \boldsymbol{\Delta q}^{(k)} > \Delta t^2 \xi$ **then**
      $\boldsymbol{B}_{1:m_l,l}$ += $\frac{(\boldsymbol{y}_{1:m_l,l} - \boldsymbol{B}_{1:m_l,l} \boldsymbol{\Delta q}^{(k)})(\boldsymbol{y}_{1:m_l,l} - \boldsymbol{B}_{1:m_l,l} \boldsymbol{\Delta q}^{(k)})^T}{(\boldsymbol{y}_{1:m_l,l} - \boldsymbol{B}_{1:m_l,l} \boldsymbol{\Delta q}^{(k)})^T \boldsymbol{\Delta q}^{(k)}}$ ▷ SR1 update
    else
      ▷ use last updated $\boldsymbol{B}_{i,l}$
    end if
  else ▷ No update since $\boldsymbol{J}_l^{(-1)}$ is not available
  end if
  for $i = l : p$ **do**
    $\hat{\boldsymbol{H}}_i$ += $\sum_{d=1}^{m_i} \lambda_{i,l}^{d} \boldsymbol{B}_{d,l}^{(k+1)}$
  end for
  if $\frac{1}{2} \|\boldsymbol{w}_l^J\|^2 >, \Delta t^2 \nu$ **then** ▷ Newton's method
    $\boldsymbol{R}_l = Higham(\hat{\boldsymbol{H}}_l^{(k+1)})$
  else ▷ GN algorithm
    $\boldsymbol{R}_l = \{\}$
  end if
end for
return $\underline{\boldsymbol{R}}$

Algorithm 5 – Lexicographic second order augmentation LexLSAug2BFGS

Input: $\underline{\boldsymbol{J}}^{(k)}, \underline{\boldsymbol{J}}^{(k-1)}, \underline{\dot{\boldsymbol{e}}}^{\mathrm{ctrl},(k)}, \boldsymbol{\Lambda}^{(k)}, \underline{\mathcal{A}}^{(k)}, c_{\mathcal{A}}, \boldsymbol{\Delta q}^{(k)}, k, \Delta t$
Internal variables: $\underline{\boldsymbol{B}}$

$\underline{\boldsymbol{y}} = \underline{\boldsymbol{0}}$
for $l = 1 : p$ **do**
  if $k = 0$ **or** $c_{\mathcal{A}} > -1$ **and** $l \geq c_{\mathcal{A}}$ **then**
    $\boldsymbol{B}_l = \sum_{i=1}^{l} \sum_{j \in \mathcal{A}_i^{(k)}} \max(\Delta t^2 \zeta, \frac{1}{2} \|\dot{\boldsymbol{e}}_{j,i}^{\mathrm{ctrl},(k)}\|_2^2) \boldsymbol{I}_{j,i}^{*}$ ▷ BFGS reinitialization
  end if
  if $k = 0$ **then**
    $\boldsymbol{R}_l = Cholesky(\boldsymbol{B}_l)$ ▷ No update since $\boldsymbol{J}_l^{(-1)}$ not available
  else
    $\boldsymbol{\Delta J}_l = (\boldsymbol{J}_l^{(k)} - \boldsymbol{J}_l^{(k-1)})^T$
    for all $j \geq l$ **and** $j \in \mathcal{A}$ **do**
      $\boldsymbol{y}_j$ += $\boldsymbol{\Delta J}_l \boldsymbol{\lambda}_{j,l}^{(k)}$ ▷ Build difference of Lagrangian gradient
    end for
    if $\frac{1}{2} \|\boldsymbol{w}_l^J\|^2 > \Delta t^2 \nu$ **then** ▷ Newton's method
      if $\boldsymbol{y}_l^T \boldsymbol{\Delta q}^{(k)} > \Delta t^2 \xi$ **then**
        $\boldsymbol{B}_l$ += $\frac{\boldsymbol{y}_l \boldsymbol{y}_l^T}{\boldsymbol{y}_l^T \boldsymbol{\Delta q}^{(k)}} - \frac{\boldsymbol{B}_l \boldsymbol{\Delta q} \boldsymbol{\Delta q}^{(k),T} \boldsymbol{B}_l}{\boldsymbol{\Delta q}^{(k),T} \boldsymbol{B}_l \boldsymbol{\Delta q}^{(k)}}$ ▷ BFGS update
        $\boldsymbol{R}_l = Cholesky(\boldsymbol{B}_l + \iota \boldsymbol{I}^{*})$
      else
        ▷ use last updated $\boldsymbol{R}_l$
      end if
    else ▷ GN algorithm
      $\boldsymbol{R}_l = \{\}$
    end if
  end if
end for
return $\underline{\boldsymbol{R}}$

APPENDIX D

Comparison between LexLSI and HQP

The solver HQP presented in Escande et al. [2014] provides the bases for the null-space and the range-space by complete orthogonal decompositions. While these bases are somewhat expensive to compute (in the case of linear dependency it requires a second non rank revealing QR decomposition) the factorizations are updatable in a straightforward manner.

If the active-set iteration count is close to one iteration, as it is usually the case for slowly developing robot problems, LexLSI outperforms HQP. However, in the two chapters 4 and 5 we observed control iterations which came with a large change of the active-set. Determining those requires a large number of active-set iterations. LexLSI does not incorporate an update strategy but rather recomputes the factorizations from scratch with each addition or deletion of a constraint from the active-set. The sum of all the factorizations is computationally expensive. Obviously this is not suitable for real-time robot control and incorporating an update strategy seems unavoidable.

A comparison between the computation times of LexLSI and HQP for a stretch demo is given in fig. D.1. The stretch demo is based on the same hierarchy given in fig. 5.18 of chapter 5 with the same contact configuration of the two feet standing on the ground and the left hand grabbing to a pole. The right hand then stretches to an out-of-reach target in its upper right direction. The right end-effector task therefore ends up in an algorithmic singularity with the geometric contact constraints which is resolved by a simple augmentation with a weighted identity matrix corresponding to the LM-method. For a better comparability, both HQP and LexLSI solve exactly the same problem. This is achieved by only using the solution of LexLSI after every control iteration for the robot update. The updated robot problem is then again solved by both HQP and LexLSI. Additionally, the problem is always started with an active-set only

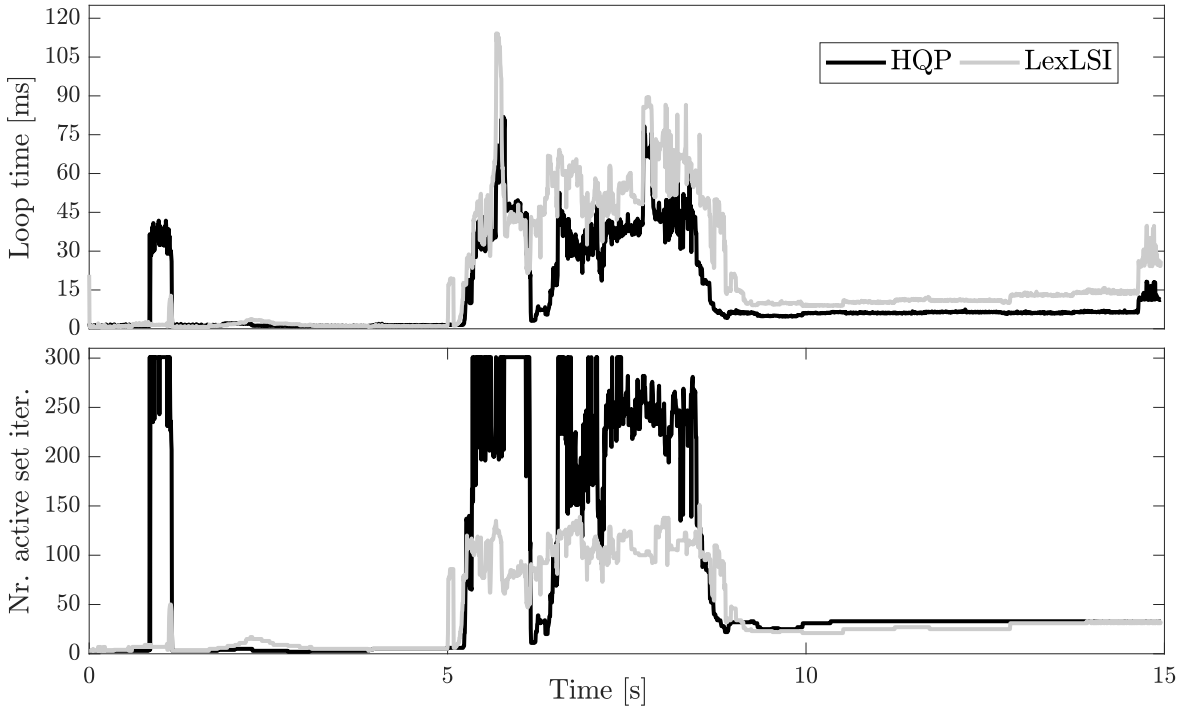


Fig. D.1. *Comparison of computation times and active-set iteration count between HQP and LexLSI for a simple stretch demo with HRP-2KAI. It can be observed that HQP, which implements a factorization update strategy, takes about 50% of the computation time of LexLSI once a certain number of active-set iterations is surpassed.*

containing the equalities (this is a so-called 'cold start' in contrary to a 'warm start' where the active-set search is commenced from the set found in the previous control iteration). Additionally, the heuristics to exit the active-set search from sec. 5.3 is disabled.

The active-set iteration count for HQP is higher than the one seen for LexLSI. Even though HQP implements a slightly different active-set search where the slack variables are always given implicitly (and therefore always being optimal) this result has to be taken with care since HQP does not implement a proper strategy for stalling and cycling (iterations where repeatedly the same constraints are activated and deactivated with a null step $\alpha = 0$, see 2.4). Despite this circumstance, HQP shows approximately the same computation times as LexLSI between control time 5 s and 8 s even though it requires over twice as much active-set iterations (the active-set search is exited when 300 iterations are reached). This is due to cheaply updating factorizations after constraint additions and removals. From ≈ 8 s to ≈ 15 s both HQP and LexLSI require about 30 active-set iterations. However, HQP is clearly superior to LexLSI in terms of computation times, requiring only about half of the computation time seen for LexLSI. However, even in this case HQP excesses the desired loop time of 5 ms with ≈ 6 ms. This stresses the need to further investigate the reason why these high active-set iteration counts occur.

Overall, it can be observed that HQP takes about 50% of the computation time of LexLSI once a certain number of active-set iterations is surpassed. This emphasizes the need for factorization updates in LexLSI in order to be computationally competitive in the case of high numbers of active-set iterations.

Bibliography

[Abe et al. 2007] Abe, Yeuhi ; Da Silva, Marco ; Popović, Jovan: Multiobjective control with frictional contacts. In: *ACM SIGGRAPH Symposium on Computer Animation* 1 (2007), S. 249–258. – ISBN 978-1-59593-624-4

[Ahmad et Lee 1990] Ahmad, Shaheen ; Lee, Chuan N.: Shape recovery from robot contour-tracking with force feedback. In: *Advanced Robotics* 5 (1990), Nr. 3, S. 257–273

[Audren et al. 2014] Audren, H. ; Vaillant, J. ; Kheddar, A. ; Escande, A. ; Kaneko, K. ; Yoshida, E.: Model preview control in multi-contact motion-application to a humanoid robot. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2014, S. 4030–4035. – ISSN 2153-0858

[Baerlocher et Boulic 2004] Baerlocher, Paolo ; Boulic, Ronan: An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. In: *Visual Computer* 20 (2004), Nr. 6, S. 402–417. – ISSN 01782789

[Bertsekas 1982] Bertsekas, Dimitri P.: Chapter 2 - The Method of Multipliers for Equality Constrained Problems. In: Bertsekas, Dimitri P. (Hrsg.): *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982, S. 95 – 157. – URL http://www.sciencedirect.com/science/article/pii/B9780120934805500064. – ISBN 978-0-12-093480-5

[Bien et Xu 1998] Bien, Zeungnam ; Xu, Jian-Xin: *Iterative Learning Control: Analysis, Design, Integration and Applications*. Norwell, MA, USA : Kluwer Academic Publishers, 1998. – ISBN 0-7923-8213-7

[Björck 1996] Björck, Å.: *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996. – URL https://epubs.siam.org/doi/abs/10.1137/1.9781611971484

[Blanes et al. 2009] Blanes, S. ; Casas, F. ; Oteo, J.A. ; Ros, J.: The Magnus expansion and some of its applications. In: *Physics Reports* 470 (2009), Nr. 5,

S. 151 – 238. – URL http://www.sciencedirect.com/science/article/pii/S0370157308004092. – ISSN 0370-1573

[Boggs et W. Tolle 1995] Boggs, Paul ; W. Tolle, Jon: Sequential Quadratic Programming. In: *Acta Numerica* 4 (1995), 01, S. 1–51

[Bonnet et al. 2018] Bonnet, Vincent ; Pfeiffer, Kai ; Fraisse, Philippe ; Crosnier, André ; Venture, Gentiane: Self-Generation of Optimal Exciting Motions for Identification of a Humanoid Robot. In: *International Journal of Humanoid Robotics* 15 (2018), Dezember, Nr. 6, S. 1850024. – URL https://hal.archives-ouvertes.fr/hal-02048085

[Bouyarmane et Kheddar 2011] Bouyarmane, Karim ; Kheddar, Abderrahmane: Using a multi-objective controller to synthesize simulated humanoid robot motion with changing contact configurations. In: *IEEE International Conference on Intelligent Robots and Systems* (2011), S. 4414–4419. – ISBN 9781612844541

[Bretl et Lall 2008] Bretl, Timothy ; Lall, Sanjay: Testing static equilibrium for legged robots. In: *IEEE Transactions on Robotics* 24 (2008), Nr. 4, S. 794–807. – ISBN 1552-3098

[Broyden 1970] Broyden, C G.: The Convergence of a Class of Double-rank Minization Algorithms. In: *Journal of the Mathematics and its Applications* 6 (1970), S. 76–90. – ISBN 0020-2932

[Bunch et al. 1976] Bunch, James R. ; Kaufman, Linda ; Parlett, Beresford N.: Decomposition of a symmetric matrix. In: *Numerische Mathematik* 27 (1976), Nr. 1, S. 95–109. – ISSN 0029599X

[Buss et Kim 2005] Buss, Samuel R. ; Kim, Jin-Su: Selectively Damped Least Squares for Inverse Kinematics. In: *Journal of Graphics Tools* 10 (2005), Nr. 3, S. 37–49. – URL https://doi.org/10.1080/2151237X.2005.10129202

[Caccavale et al. 1997] Caccavale, F. ; Chiaverini, S. ; Siciliano, B.: Second-order kinematic control of robot manipulators with Jacobian damped least-squares inverse: theory and experiments. In: *IEEE/ASME Transactions on Mechatronics* 2 (1997), Sep., Nr. 3, S. 188–194. – ISSN 1083-4435

[Carlson 1986] Carlson, David: What are Schur complements, anyway? In: *Linear Algebra and its Applications* 74 (1986), S. 257 – 275. – URL http://www.sciencedirect.com/science/article/pii/0024379586901278. – ISSN 0024-3795

[Caron et Kheddar 2016] Caron, S. ; Kheddar, A.: Multi-contact walking pattern generation based on model preview control of 3D COM accelerations. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, Nov 2016, S. 550–557. – ISSN 2164-0580

[Chiaverini et al. 1991] Chiaverini, S. ; Egeland, O. ; Kanestrom, R. K.: Achieving user-defined accuracy with damped least-squares inverse kinematics. In: *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, June 1991, S. 672–677 vol.1

[Chiaverini et al. 1994] Chiaverini, S. ; Siciliano, B. ; Egeland, O.: Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. In: *IEEE Transactions on Control Systems Technology* 2 (1994), June, Nr. 2, S. 123–134. – ISSN 1063-6536

[Chiaverini 1997] Chiaverini, Stefano: Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. In: *IEEE Transactions on Robotics and Automation* 13 (1997), Nr. 3, S. 398–410. – ISBN 1042-296X

[Collette et al. 2007] Collette, Cyrille ; Micaelli, Alain ; Andriot, Claude ; Lemerle, Pierre: Dynamic balance control of humanoids for multiple grasps and non coplanar frictional contacts. In: *IEEE/RAS International Conference on Humanoid Robots*. Pittsburgh, PA, November 29 - December 1 2007, S. 81–88

[Conn et al. 1991] Conn, A. R. ; Gould, N. I M. ; Toint, Ph L.: Convergence of quasi-Newton matrices generated by the symmetric rank one update. In: *Mathematical Programming* 50 (1991), Nr. 1-3, S. 177–195. – ISBN 0025-5610

[Conn et al. 1988] Conn, Andrew ; Gould, N.I.M. ; L Toint, Ph: Testing a class of methods for solving minimization problems with simple bounds on the variables. In: *Mathematics of Computation* 50 (1988), 04, S. 399–430

[Dantzig 1991] Dantzig, George B.: *Linear Programming and Extensions*. Princeton University Press, 1991. – URL http://www.jstor.org/stable/j.ctt1cx3tvg

[Dennis et al. 1981] Dennis, John E. ; Gay, David M. ; Walsh, Roy E.: An Adaptive Nonlinear Least-Squares Algorithm. In: *ACM Trans. Math. Softw.* 7 (1981), September, Nr. 3, S. 348–368. – URL http://doi.acm.org/10.1145/355958.355965. – ISSN 0098-3500

[Deo et Walker 1993] Deo, A. S. ; Walker, I. D.: Adaptive non-linear least squares for inverse kinematics. In: *IEEE International Conference on Robotics and Automation* Bd. 1, May 1993, S. 186–193

[Dietrich et al. 2012] Dietrich, A. ; Albu-Schffer, A. ; Hirzinger, G.: On continuous null space projections for torque-based, hierarchical, multi-objective manipulation. In: *2012 IEEE International Conference on Robotics and Automation*, May 2012, S. 2978–2985. – ISSN 1050-4729

[Dietrich et al. 2015] Dietrich, Alexander ; Ott, Christian ; Albu-Schffer, Alin: An overview of null space projections for redundant, torque-controlled robots. In:

The International Journal of Robotics Research 34 (2015), Nr. 11, S. 1385–1400. – URL https://doi.org/10.1177/0278364914566516

[Dimitrov et al. 2015] Dimitrov, Dimitar ; Sherikov, Alexander ; Wieber, Pierre-Brice: *Efficient resolution of potentially conflicting linear constraints in robotics*. August 2015. – URL https://hal.inria.fr/hal-01183003. – Submitted to IEEE TRO (05/August/2015)

[Erleben et Andrews 2017] Erleben, Kenny ; Andrews, Sheldon: Inverse Kinematics Problems with Exact Hessian Matrices. In: *Proceedings of the Tenth International Conference on Motion in Games*. New York, NY, USA : ACM, 2017 (MIG '17), S. 14:1–14:6. – URL http://doi.acm.org/10.1145/3136457.3136464. – ISBN 978-1-4503-5541-4

[Escande et al. 2013] Escande, Adrien ; Mansard, Nicolas ; Wieber, Pierre-Brice: Hierarchical Quadratic Programming: Companion report. URL https://hal.archives-ouvertes.fr/hal-00970816, November 2013. – Forschungsbericht. 10p.

[Escande et al. 2014] Escande, Adrien ; Mansard, Nicolas ; Wieber, Pierre-Brice: Hierarchical quadratic programming: Fast online humanoid-robot motion generation. In: *The International Journal of Robotics Research* 33 (2014), Nr. 7, S. 1006–1028. – ISBN 0278-3649

[Faverjon et Tournassoud 1987] Faverjon, Bernard ; Tournassoud, P.: A local based approach for path planning of manipulators with a high number of degrees of freedom / INRIA. URL https://hal.inria.fr/inria-00075933, Februar 1987 (RR-0621). – Forschungsbericht

[Feng et al. 2013] Feng, Siyuan ; Xinjilefu, X ; Huang, Weiwei ; Atkeson, Christopher G.: 3D Walking Based on Online Optimization. In: *IEEE-RAS International Conference on Humanoid Robots*, 2013

[Flacco et De Luca 2014] Flacco, F. ; De Luca, A.: Discrete-time velocity control of redundant robots with acceleration/torque optimization properties. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, S. 5139–5144. – ISSN 1050-4729

[Flash et Hogan 1985] Flash, T ; Hogan, N: The coordination of arm movements: an experimentally confirmed mathematical model. In: *Journal of Neuroscience* 5 (1985), Nr. 7, S. 1688–1703. – URL https://www.jneurosci.org/content/5/7/1688. – ISSN 0270-6474

[Fletcher 1970] Fletcher, R.: A new approach to variable metric algorithms. In: *The Computer Journal* 13 (1970), 01, Nr. 3, S. 317–322. – URL https://doi.org/10.1093/comjnl/13.3.317. – ISSN 0010-4620

[Fletcher 2006] Fletcher, Roger: A new low rank quasi-Newton update scheme for nonlinear programming. In: *IFIP TC7 Conference* (2006), Nr. August, S. 275–293

[Fletcher et Leyffer 2002] Fletcher, Roger ; Leyffer, Sven: *Nonlinear programming without a penalty function*. 2002. – 239–269 S. – ISBN 1010701002

[Frank et Wolfe 1956] Frank, Marguerite ; Wolfe, Philip: An algorithm for quadratic programming. In: *Naval Research Logistics Quarterly* 3 (1956), Nr. 12, S. 95–110. – URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030109

[Gill et al. 1986] Gill, P. E.Philip E. ; Hammarling, Sven J. ; Murray, Walter ; Saunders, Michael A. ; Wright, Margaret H.: User's guide for LSSOL (version 1.0): a fortran package for constrained linear least-squares and convex quadratic programming / Standford University. Standord, California 94305, January 1986 (86-1). – Forschungsbericht

[Goldfarb 1970] Goldfarb, Donald: A Family of Variable-Metric Methods Derived by Variational Means. In: *Mathematics of Computation* 24 (1970), Nr. 109, S. 23–26. – URL http://www.jstor.org/stable/2004873. – ISSN 00255718, 10886842

[Golub et Van Loan 1996] Golub, Gene H. ; Van Loan, Charles F.: *Matrix Computations (3rd Ed.)*. Baltimore, MD, USA : Johns Hopkins University Press, 1996. – ISBN 0-8018-5414-8

[Goswami 1999] Goswami, Ambarish: Postural Stability of Biped Robots and the Foot-Rotation Indicator (FRI) Point. In: *The International Journal of Robotics Research* 18 (1999), Nr. 6, S. 523–533. – URL https://doi.org/10.1177/02783649922066376

[Guennebaud et al. 2010] Guennebaud, Gaël ; Jacob, Benoît et al.: *Eigen v3*. http://eigen.tuxfamily.org. 2010

[Hansen 1987] Hansen, Per C.: The truncatedSVD as a method for regularization. In: *BIT Numerical Mathematics* 27 (1987), Dec, Nr. 4, S. 534–553. – URL https://doi.org/10.1007/BF01937276. – ISSN 1572-9125

[Harish et al. 2016] Harish, Pawan ; Mahmudi, Mentar ; Callennec, Benoît Le ; Boulic, Ronan: Parallel Inverse Kinematics for Multithreaded Architectures. In: *ACM Trans. Graph.* 35 (2016), Februar, Nr. 2, S. 19:1–19:13. – URL http://doi.acm.org/10.1145/2887740. – ISSN 0730-0301

[Harris 1978] Harris, Frederic J.: On the use of Harmonic Analysis with the Discrete Fourier Transform. In: *Proceedings of the IEEE* 66 (1978), January, Nr. 1, S. 51 – 83

[Herzog et al. 2016] Herzog, Alexander ; Rotella, Nicholas ; Mason, Sean ; Grimminger, Felix ; Schaal, Stefan ; Righetti, Ludovic: Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid. In: *Autonomous Robots* 40 (2016), Mar, Nr. 3, S. 473–491. – URL https://doi.org/10.1007/s10514-015-9476-6. – ISSN 1573-7527

[Higham 1986] Higham, N.: Computing the Polar Decompositionwith Applications. In: *SIAM Journal on Scientific and Statistical Computing* 7 (1986), Nr. 4, S. 1160–1174. – URL https://doi.org/10.1137/0907079

[Higham 1988] Higham, Nicholas J.: Computing a nearest symmetric positive semidefinite matrix. In: *Linear Algebra and its Applications* 103 (1988), S. 103 – 118. – URL http://www.sciencedirect.com/science/article/pii/0024379588902236. – ISSN 0024-3795

[Kajita et al. 2003] Kajita, S. ; Kanehiro, F. ; Kaneko, K. ; Fujiwara, K. ; Harada, K. ; Yokoi, K. ; Hirukawa, H.: Biped walking pattern generation by using preview control of zero-moment point. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)* Bd. 2, Sep. 2003, S. 1620–1626 vol.2. – ISSN 1050-4729

[Kanoun et al. 2011] Kanoun, O ; Lamiraux, F ; Wieber, P-B.: Kinematic control of redundant manipulators: generalizing the task priority framework to inequality tasks. In: *IEEE Trans. on Robotics* 27 (2011), Nr. 4, S. 785–792

[Karush 1939] Karush, William: *Minima of Functions of Several Variables with Inequalities as Side Conditions*. Chicago, IL, USA, Department of Mathematics, University of Chicago, Diplomarbeit, 1939

[Khalfan et al. 1993] Khalfan, H. ; Byrd, R. ; Schnabel, R.: A Theoretical and Experimental Study of the Symmetric Rank-One Update. In: *SIAM Journal on Optimization* 3 (1993), Nr. 1, S. 1–24. – URL https://doi.org/10.1137/0803001

[Khalil et Dombre 2002] Khalil, W ; Dombre, E: Chapter 5 - Direct kinematic model of serial robots. In: Khalil, W (Hrsg.) ; Dombre, E (Hrsg.): *Modeling, Identification and Control of Robots*. Oxford : Butterworth-Heinemann, 2002, S. 85 – 115. – URL http://www.sciencedirect.com/science/article/pii/B9781903996669500051. – ISBN 978-1-903996-66-9

[Khatib 1987] Khatib, O.: A unified approach for motion and force control of robot manipulators: The operational space formulation. In: *IEEE Journal on Robotics and Automation* 3 (1987), February, Nr. 1, S. 43–53. – ISSN 0882-4967

[Khatib 1995] Khatib, Oussama: Inertial Properties in Robotic Manipulation: An Object-Level Framework. In: *The International Journal of Robotics Research* 14 (1995), Nr. 1, S. 19–36. – URL https://doi.org/10.1177/027836499501400103

[Kim et Oh 2007] Kim, Park ; Oh: Walking Control Algorithm of Biped Humanoid Robot on Uneven and Inclined Floor. In: *Journal of Intelligent and Robotic Systems* Bd. 48, 2007, S. 457–484

[Kuhn et Tucker 1951] Kuhn, H. W. ; Tucker, A. W.: Nonlinear Programming. In: *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley, Calif. : University of California Press, 1951, S. 481–492. – URL https://projecteuclid.org/euclid.bsmsp/1200500249

[Kuindersma et al. 2014] Kuindersma, Scott ; Permenter, Frank ; Tedrake, Russ: An efficiently solvable quadratic program for stabilizing dynamic locomotion. In: *IEEE International Conference on Robotics and Automation*. Hong Kong, China, 2014

[Kyong-Sok Chang et Khatib 1995] Kyong-Sok Chang ; Khatib, O.: Manipulator control at kinematic singularities: a dynamically consistent strategy. In: *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots* Bd. 3, Aug 1995, S. 84–88 vol.3

[L. Toint 1987] L. Toint, Ph: On Large Scale Nonlinear Least Squares Calculations. In: *Siam Journal on Scientific and Statistical Computing* 8 (1987), 05

[Maciejewski et Klein 1988] Maciejewski, Anthony A. ; Klein, Charles A.: Numerical filtering for the operation of robotic manipulators through kinematically singular configurations. In: *Journal of Robotic Systems* 5 (1988), Nr. 6, S. 527–552. – ISSN 10974563

[Mansard 2012] Mansard, N.: A dedicated solver for fast operational-space inverse dynamics. In: *2012 IEEE International Conference on Robotics and Automation*, May 2012, S. 4943–4949

[Mi et Jia 2004] Mi, Liangchuan ; Jia, Yan-Bin: High Precision Contour Tracking with a Joystick Sensor. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004

[Murray et al. 1994] Murray, Richard M. ; Sastry, S. S. ; Zexiang, Li: *A Mathematical Introduction to Robotic Manipulation*. 1st. Boca Raton, FL, USA : CRC Press, Inc., 1994. – ISBN 0849379814

[Nakamura et Hanafusa 1986] Nakamura, Yoshihiko ; Hanafusa, Hideo: Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control. In: *J. Dyn. Sys., Meas., Control* 108 (1986), Nr. 3, S. 163–171

[Nenchev et al. 2000] Nenchev, Dragomir N. ; Tsumaki, Yuichi ; Uchiyama, Masaru: Singularity-Consistent Parameterization of Robot Motion and Control. In: *The International Journal of Robotics Research* 19 (2000), Nr. 2, S. 159–182. – URL https://doi.org/10.1177/02783640022066806

[Nitsche et al. 2015] Nitsche, Matias ; Krajník, Tomáš ; Čížek, Petr ; Mejail, Marta ; Duckett, Tom: WhyCon: an efficient, marker-based localization system. In: *IEEE/RAS IROS Workshop on Open Source Aerial Robotics*, 2015

[Nocedal et Wright 2006] Nocedal, Jorge ; Wright, Stephen J.: *Numerical Optimization*. second. New York, NY, USA : Springer, 2006

[Paolillo et al. 2014] Paolillo, A. ; Cherubini, A. ; Keith, F. ; Kheddar, A. ; Venditelli, M.: Toward autonomous car driving by a humanoid robot: A sensor-based framework. In: *2014 IEEE-RAS International Conference on Humanoid Robots*, Nov 2014, S. 451–456. – ISSN 2164-0572

[Pfeiffer et al. 2017] Pfeiffer, K. ; Escande, A. ; Kheddar, A.: Nut fastening with a humanoid robot. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, S. 6142–6148. – ISSN 2153-0866

[Pratt et al. 2006] Pratt, Jerry ; Carff, John ; Drakunov, Sergey: Capture Point : A Step toward Humanoid Push Recovery. In: *International Conference on Humanoid Robots* (2006), S. 200–207. ISBN 142440200X

[Righetti et al. 2011] Righetti, L ; Mistry, M ; Buchli, j. ; Schaal, S: Inverse Dynamics Control of Floating-Base Robots With External Contraints: an Unified View. In: *2011 Ieee International Conference on Robotics and Automation (Icra)* (2011), S. 1085–1090. – URL http://www-clmc.usc.edu/publications/R/righetti-ICRA2011.pdf. ISBN 9781612843858

[Saab et al. 2013] Saab, Layale ; Ramos, Oscar E. ; Keith, Francois ; Mansard, Nicolas ; Soueres, Philippe ; Fourquet, Jean Y.: Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. In: *IEEE Transactions on Robotics* 29 (2013), Nr. 2, S. 346–362. – ISBN 1552-3098

[Sardain et Bessonnet 2004] Sardain, P. ; Bessonnet, G.: Forces acting on a biped robot. Center of pressure-zero moment point. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 34 (2004), Sep., Nr. 5, S. 630–637. – ISSN 1083-4427

[Sentis et Khatib 2004] Sentis, L. ; Khatib, O.: Prioritized multi-objective dynamics and control of robots in human environments. In: *4th IEEE/RAS International Conference on Humanoid Robots, 2004.* Bd. 2, Nov 2004, S. 764–780 Vol. 2

[Sentis et Khatib 2005] Sentis, L. ; Khatib, O.: Control of Free-Floating Humanoid Robots Through Task Prioritization. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, April 2005, S. 1718–1723. – ISSN 1050-4729

[Shamir 1990] Shamir, T.: The Singularities of Redundant Robot Arms. In: *The International Journal of Robotics Research* 9 (1990), Nr. 1, S. 113–121. – URL https://doi.org/10.1177/027836499000900105

[Shanno 1970] Shanno, D. F.: Conditioning of Quasi-Newton Methods for Function Minimization. In: *Mathematics of Computation* 24 (1970), Nr. 111, S. 647–656. – URL http://www.jstor.org/stable/2004840. – ISSN 00255718, 10886842

[Sherikov et al. 2015] Sherikov, Alexander ; Dimitrov, Dimitar ; Wieber, Pierre B.: Balancing a humanoid robot with a prioritized contact force distribution, 2015, S. 223–228. – ISBN 9781479968855

[Siciliano et Slotine 1991] Siciliano, Bruno ; Slotine, Jean-Jacques E.: The General Framework for Managing Multiple Tasks in High Redundant Robotic Systems. In: *International Conference on Advanced Robotics*, 1991, S. 1211 – 1216 vol.2. – ISBN 0-7803-0078-5

[Stewart 2000] Stewart, David E.: Rigid-Body Dynamics with Friction and Impact. In: *SIAM Rev.* 42 (2000), März, Nr. 1, S. 3–39. – URL http://dx.doi.org/10.1137/S0036144599360110. – ISSN 0036-1445

[Sugihara 2011] Sugihara, Tomomichi: Solvability-Unconcerned Inverse Kinematics by the LevenbergMarquardt Method. In: *IEEE Transactions on Robotics* 27 (2011), October, Nr. 5, S. 984–991

[Tourassis et Marcelo H. Ang 1992] Tourassis, Vassilios D. ; Marcelo H. Ang, JR: Identification and Analysis of Robot Manipulator Singularities. In: *The International Journal of Robotics Research* 11 (1992), Nr. 3, S. 248–259. – URL https://doi.org/10.1177/027836499201100307

[Udwadia et E. Kalaba 1992] Udwadia, Firdaus ; E. Kalaba, R: A New Perspective on Constrained Motion. In: *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences* 439 (1992), 11, S. 407–410

[Udwadia et Schutte 2010] Udwadia, Firdaus ; Schutte, Aaron: Equations of motion for general constrained systems in Lagrangian mechanics. In: *Acta Mech* 213 (2010), 08

[Vaillant et al. 2016] Vaillant, Joris ; Kheddar, Abderrahmane ; Audren, Hervé ; Keith, François ; Brossette, Stanislas ; Escande, Adrien ; Bouyarmane, Karim ; Kaneko, Kenji ; Morisawa, Mitsuharu ; Gergondet, Pierre ; Yoshida, Eiichi ; Kajita, Suuji ; Kanehiro, Fumio: Multi-contact vertical ladder climbing with an HRP-2 humanoid. In: *Autonomous Robots* 40 (2016), Nr. 3, S. 561–580. – ISSN 15737527

[Vukobratovi et Stepanenko 1972] Vukobratovi, M. ; Stepanenko, J.: On the stability of anthropomorphic systems. In: *Mathematical Biosciences* 15 (1972), Nr. 1, S. 1 – 37. – URL http://www.sciencedirect.com/science/article/pii/0025556472900612. – ISSN 0025-5564

[Walker et Orin 1982] Walker, M. W. ; Orin, D. E.: Efficient Dynamic Computer Simulation of Robotic Mechanisms. In: *Journal of Dynamic Systems, Measurement, and Control* 104 (1982), 09, Nr. 3, S. 205–211. – URL https://doi.org/10.1115/1.3139699. – ISSN 0022-0434

[Wampler et Leifer 1988] Wampler, II ; Leifer, L. J.: Applications of Damped Least-Squares Methods to Resolved-Rate and Resolved-Acceleration Control of Manipulators. In: *Journal of Dynamic Systems, Measurement, and Control* 110 (1988), 03, Nr. 1, S. 31–38. – URL https://doi.org/10.1115/1.3152644. – ISSN 0022-0434

[Wang et al. 2010] Wang, Jingguo ; Li, Yangmin ; Zhao, Xinhua: Inverse Kinematics and Control of a 7-DOF Redundant Manipulator Based on the Closed-Loop Algorithm. In: *International Journal of Advanced Robotic Systems* 7 (2010), Nr. 4, S. 37. – URL https://doi.org/10.5772/10495

[Wedin 1973] Wedin, Per-Åke: Perturbation theory for pseudo-inverses. In: *BIT Numerical Mathematics* 13 (1973), Jun, Nr. 2, S. 217–232. – URL https://doi.org/10.1007/BF01933494. – ISSN 1572-9125

[Wei 1992] Wei, M.: Perturbation Theory for the Rank-Deficient Equality Constrained Least Squares Problem. In: *SIAM Journal on Numerical Analysis* 29 (1992), Nr. 5, S. 1462–1481. – URL https://doi.org/10.1137/0729084

[Wieber 2006] Wieber, P.: Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations. In: *2006 6th IEEE-RAS International Conference on Humanoid Robots*, Dec 2006, S. 137–142. – ISSN 2164-0572

[Wieber 2002] Wieber, Pierre-Brice: On the stability of walking systems. In: *Proceedings of the International Workshop on Humanoid and Human Friendly Robotics*. Tsukuba, Japan, 2002. – URL https://hal.inria.fr/inria-00390866

[Wieber et al. 2017] Wieber, Pierre-Brice ; Escande, Adrien ; Dimitrov, Dimitar ; Sherikov, Alexander: Geometric and numerical aspects of redundancy. In: *Geometric and Numerical Foundations of Movements*. URL https://hal.inria.fr/hal-01418462, 2017

[Wolovich et Elliott 1984] Wolovich, W. ; Elliott, H.: A computational technique for inverse kinematics. In: *IEEE Conference on Decision and Control* 23 (1984), Nr. December, S. 1359–1363. – ISSN 01912216

[Zhao et Badler 1994] Zhao, Jianmin ; Badler, Norman I.: Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures. In: *ACM Trans. Graph.* 13 (1994), Oktober, Nr. 4, S. 313–336. – URL http://doi.acm.org/10.1145/195826.195827. – ISSN 0730-0301

List of Figures

List of Tables