Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 311

A comparision of interior point and SQP methods with augmented Lagrangian methods in robotics applications

David Schwarz

Course of Study: Softwaretechnik

Examiner: Prof. Dr. M. Marc Toussaint

Supervisor: Prof. Dr. M. Marc Toussaint

Commenced: March 17, 2016

Completed: September 16, 2016

CR-Classification: I.2.9, G.1.6, G.4

Abstract

This Paper intends to interface the existing MLR Framework KOMO with two solvers for non linear optimization Problems.
These solvers are the Interior Point Method implemented in IPOPT and the sequential quadratic Programming Method SLSQP implemented in NLOPT.

In the second step I will compare these two Methods with KOMOs internal augmented Lagrangian Method in terms of precision, required number of evaluations and any-time (before convergence) quality of solutions.

Kurzfassung

Diese Arbeit beabsichtigt ein Interface zwischen dem existierenden MLR Framwork KOMO und zwei Solvern von nicht linearen Optimierungs-Problemen zu erschaffen. Diese sind die in IPOT implementierte Interior Point Methode und die sequential quadratic Programming Methode SLSQP die in NLOPT implementiert ist.

In einem weiteren Schritt werden diese zwei Methoden mit der augmented Lagrangian Methode die in KOMO implementiert verglichen. Dieser Vergleich findet statt unter den Aspekten von, Präzision, nötige Anzahl von Evaluationen, und der Qualität der Lösung bevor sie zu einem Endergebnis konvergiert ist.

Contents

List of Figures

1 Introduction

An important aspect of Robotics is planning an efficient trajectory based on existing constraints, like how exactly different parts of the robot can move, where these parts are in a specific starting state and what Obstacles exist.

All these aspects of a Robotic Problem can be modelled as a Mathematics optimization Problems.
There exist a number of methods to solve such optimization Problems.

In this Paper I intend to interface the existing MLR Framework KOMO[Tou14] with two solvers for non linear optimization Problems.
These solvers are the Interior Point Method implemented in IPOPT and the sequential quadratic Programming Method SLSQP implemented in NLOPT.

In the second step I will compare these two Methods with KOMOs internal augmented Lagrangian Method in terms of precision, required number of evaluations and any-time (before convergence) quality of solutions.

Structure

The Paper is structured in the following way:

Chapter 2 – AMPL: Here I will give a brief History and an explanation of what AMPL is, which is a key Part of most Methods of solving optimization Problems

Chapter 3 – Algorithms: Here I will describe the two Algorithms KOMO is interfaced with

Chapter 4 – Interfaces: A Description of the Interfaces to KOMO

Chapter 5 – Comparison: of the three mentioned Algorithms

Chapter 6 – Conclusion: based on the Comparisons Results.

2 AMPL

An important Part of solving non linear Optimization Problems is communicating a Problem Representation to a Solver.
The biggest Problem here is that the Problem representation a solver understands differs greatly from a Problem Representation a user could understand somewhat easily.
Fortunately today several modelling Languages exist to bridge this gap.

One of these modelling Languages is AMPL. Which allows the User to present their Problems similar to mathematical functions on paper. [FGK02]

2.1 A short History of AMPL

The Enthusiasm for the practical potential of linear programming systems was great in the early 1950s but started to fade until the late 1970s.
The reason for this was not the quality or speed of the optimization itself, but the process required to manage data, formulate and build the model and report on and analyse the results. Powerful techniques of computational optimization weren't as useful as expected because the communication between the User and the Optimizer ended up being the biggest bottleneck.

A response to this Problem was the development of increasingly advanced software for optimization modelling. This development started out with matrix generators and expanded to modelling languages later on.
Early on it was clear that even small problems contained to many coefficients, right-hand sides, and bounds to be managed by typing them in directly. Similar problems came up when examining and analyzing the results.
Therefore it was clear that computers would have to be used to create and manage problems.

Since the early development was focused on linear programming and the biggest effort in setting up a linear problem was spent in setting up the matrix coefficients such programs were called matrix generators. These matrix generators improved the application of linear programming vastly. But still they had their failings.
The biggest problems were that they weren't easy to use and debug, since their problem representation differed vastly from how a human would understand a problem regularly.

Modelling Languages were the logical alternative to trying to make debugging Matrix generators easier. They solved the problem by removing the need to write them manually, by taking over the job of translating the Problems to a Form the Solver could use. And allowed the human modeller to write his problem in a way that was close to a form he would normally use.
In general purpose modelling the most widely implemented and used approach is based on regular variable and equations known from algebra and calculus. Since generic optimization problems can be viewed as a minimization or maximization of some function of decision variables, subject to equations and inequalities.

Robert Fourer one of the creators of AMPL, first encountered modelling languages in the 1970s, while working for the National Bureau of Economic Research (NBER) in a Computer Research Center in Cambridge, Massachusetts.
Since the Research Center's mission was to design and develop innovative modelling software, ideas for new languages and tools were a regular field of discussion. And especially the younger members of the linear programming team started to consider these ideas in the context of Linear Programming software.
When he left for graduate school, in 1976 he, had begun writing the Report "A modern Approach to Computer Systems for Linear Programming" together with Michael J Harrison.
In this report algebraic modelling languages played a prominent part as a method to improve linear programming systems.
An attempt to submit this report to the M.I.T. Sloan Schools working paper series made clear, that many others didn't share Fourers view of modelling languages. Many reviewers at that point weren't convinced of the advantages of using modelling languages over traditional means.
This prompted him to write another report, focused on modelling languages, which investigated in detail the differences between modeller's and algorithm's forms, resulting difficulties in debugging matrix generators and many other related issues.
This report "Modelling Languages versus Matrix Generators" also faced some troubles in publication, but was finally released in a computer science journal devoted to mathematical software in 1983.

The development of AMPL started in 1984. Robert Fourer was invited by Dave Gay, who he knew from his work at NBER, to spend a sabbatical at the Computing Sciences Research Center at Bell Laboratories.
The researchers there were given a free hand in initiating new projects. Fourers arrival coincided with the completion of Brian Kerninghan's latest project.
The three of them felt a fresh attempt at designing a algebraic modelling language would be a great fit.
There aim was to create a declarative modelling language where the definition of variables, objectives and constraints told you everything you needed to know about them.
And thus began the development of AMPL.
Finding a name for a new software package was taken seriously by Bell labs, and the choice of "A Mathematical Programming Languag" short AMPL came well after the project had begun.
In the late 1980s the concept of modelling languages had become far more established and they had far less problems in publishing their paper "A modelling language for mathematical programming" (1990).

AMPL started out as a research project and therefore it only had a few interested users in its first seven years.
At that time Bell Labs had no clear how to disseminate the resulting software to the public. But they had a strong tradition in disseminating written work.
So they wrote an AMPL book which happened to contain a disk in the back.

Since then algebraic modelling languages have become an established approach.
Four general-purpose languages - AIMMS, AMPL, GAMS, MPL - and their associated software have been in active development for more than than two decades. They all are based on the same underlying concept but differ significantly in how this concept is presented to the user.
Additionally a variety of modelling languages dedicated to specific solvers have been developed by now.

Modelling Languages proved to be powerful encouragement to applied optimization. They especially reduced the barrier of entry. And even allow people who aren't practised programmers to use optimization methods. [Fou12]

3 Algorithms

3.1 IPOPT

IPOPT is an implementation of an Interior Point Method and stands for Interior Point Optimizer.
The code was written in C++ by Andreas Wächter and Carl Laird. And is released as open source code under the Eclipse Public License (EPL).
This code is available from the COIN-OR iniative.
It is designed to find optimal solutions for Problems with the following Form.

$$\begin{aligned} \min_{\forall x \in R^n} \quad & f(x) \\ \text{s.t.} \quad & g_L <= g(x) <= g_U \\ & x_L <= x <= x_U \end{aligned}$$

gL/gU are the lower and upper bounds of the constraints and xL/xU are respectively the lower and upper bounds of the x variables. Equality constraints can be set by setting gL=gU.
The functions f(x) and g(x) can be non-linear and non-convex, but should be twice continuously differentiable.

Interior Point Methods augment the objective function with a barrier term which causes the optimal unconstrained value to be in the feasible space. They converge by moving through the interior of the feasible space.[WB06]
IPOPT uses some external libraries:

BLAS Library: I am using the MKL (Math Kernel Library) by Intel.[Mkl]

A sparse symmetric indefinite linear solver: I am using MA57 from the HSL Mathematical Software Library.[Hsl]

3.2 NLOPT

NLOPT is a interface to many different Algorithms. In this the sequential quadratic Programming Method SLSQP is used. NLOPT was written and released by Steven G. Johnson.

It is designed to find optimal solutions for Problems with the following Form.

$$\begin{aligned} \min_{\forall x \in R^n} \quad & f(x) \\ \text{s.t.} \quad & g(x) <= 0 \\ & h(x) = 0 \\ & lb <= x <= ub \end{aligned}$$

h(x) are the non linear equality constraints, g(x) are the non-linear inequalit constraints and lb/ub are optional upper and lower bounds for x.

SLSQP treats the problem as a sequence of constrained least-squares problems. The algorithm optimizes successive second-order (quadratic/least-squares) approximations of the objective function (via BFGS updates), with first-order (affine) approximations of the constraints. [Joh][Kra88][Kra49]

4 Interfaces

4.1 IPOPT

The interface to IPOPT consists of the classes ipOPTinterface.cpp, ipOPTinterface.h and my implementation of the virtual class TNLP contained in IPOPT, called ipoptTNLP.cpp and ipoptTNLP.hpp.

4.1.1 ipOPTinterface

ipOPTinterface contains the public function callIPopt.
CallIPopt receives a ConstrainedProblem and the array of starting values for the x variables.
In callIPopt a instance of ipoptTNLP is initialized, some optimization options are specified and the optimization is started.

4.1.2 ipootTNLP

IpoptTNLP is initialized with the ConstrainedProblem and the x array which ipOPTinterface received. It contains the following functions realizing the communication with IPOPT.

set_x: this utility function calls ConstrainedProblem.fc to recalculate the value of f(x), the gradients of f(x), the constraints g(x) and the jacobian of the constraints at point x.

get_NLP_info: This is called once and supplies Ipopt with the size of the problem.

get_bounds_info: supplies Ipopt with the bounds of the constraints and the variables.

get_starting_point: sets the starting point for IPOPT.

eval_f: returns the value of the objective funcion at point x.

evalgrad_f: returns the values of the gradients of the function at point x.

eval_g: returns the values of the constraints at point x.

eval_jac_g: returns values contained in the jacobian at point x.

intermediate_callback: Is used to return some information about the running optimization

finalize_solution: is called in the end to return the result.

4.1.3 Hessian

IPOPT could also make use of a Hessian of the Lagrangian in the function eval_h. KOMO is able to provide the Hessian of the Lagrangian through UnconstrainedProblem.fs. But this can't be used by IPOPT since they add a custom obj_factor in front of the objective Term f(x) and lambdas to the second derivatives of the constraints g(x).
I can't let IPOPT calculate the Hessian of the Lagrangian either, because KOMO doesn't have the second derivatives of g(x) readily available.

Nevertheless IPOPT is able to optimize the Problem, since IPOPT has a option to calculate an approximate Hessian of the Lagrangian by a limited-memory quasi-Newton method.

4.2 NLOPT

The interface to NLOPT consists of the classes nlOPTinterface.cpp and nlOPTinterface.h. It contains the following functions to communicate with NLOPT.

callNLopt: CallNLopt receives a ConstrainedProblem and the array of starting values for the x variables.
This function initializes the optimization. Creates a myConverter object which holds the information about the ConstrainedProblem and the x array. The myConverter object allows myfunc and myconstraint to access the neccessary information. Here its also possible to set some options like the optimization terminationtolerance and the tolerance for the constraints. It is also possible to specify a different algorithm contained in NLOPT, in this case we are using SLSQP
Finally it returns the solution.

set_x: this utility function is contained by the myConverter structureand calls ConstrainedProblem.fc to recalculate the value of f(x), the gradients of f(x), the constraints and the jacobian of the constraints at point x.

myfunc: recieves a data object and calls textttset_x. it gives NLOPT the information about the current value of f(x) and the value of the gradients of f(x) at the current point x.

myconstraint: gives NLOPT the information about the current value of the constraints g(x) and their gradients contained in the jacobian at point x.

5 Comparison

5.1 Testcase description

For the comparison I will use two of the examples contained in KOMO/share/examples/KOMO/easy/. These examples are the examples testEasy and testEasyPR2.
I will compare both examples with Timesteps=1 , which makes them end-pose problems, and with Timesteps=10 and Timesteps=100 which are trajectory problems.

5.1.1 TestEasy

TestEasy consists of an arm with seven joints. The target is to reach a specific point in the 3-Dimensional room(green) and has an spherical obstacle (red). (see Figure 5.1)
TestEasy has the following amount of variables.

$$7 + 7 * Timesteps$$

And constraints equivalent to the number of Timesteps plus one.

5.1.2 TestEasyPR2

TestEasyPR2 is a model of the PR2 robot.
Its traget is to touch a spehrical object (red) in a shelf (blue) the shelf also acts as an obstacle. (Figure 5.2)
TestEasyPR2 has the following amount of variables.

$$25 + 25 * Timesteps$$

And constraints equivalent to the number of Timesteps plus one.

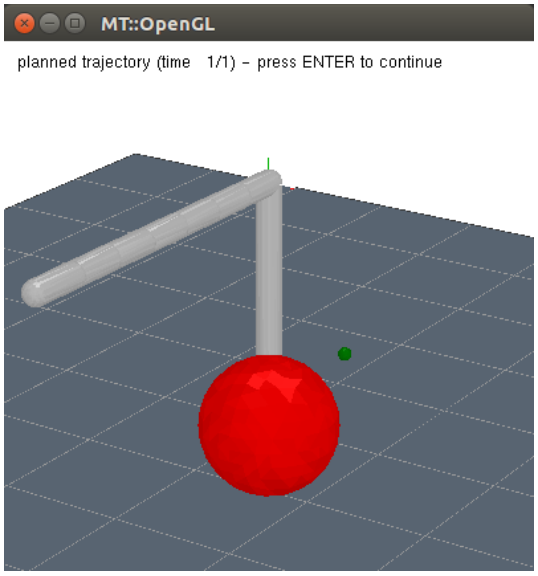

Figure 5.1: TestEasy

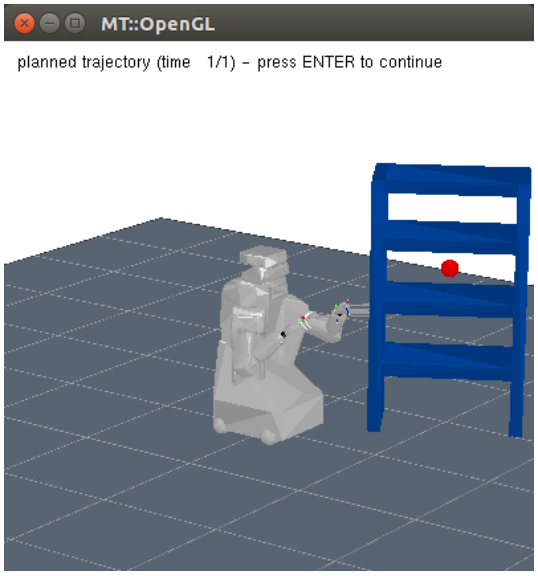

Figure 5.2: TestEasyPR2

5.2 TestEasy Timesteps=1

5.2.1 Results

In this case only KOMO gives a useful result.

KOMO converges to a f value of 0,322982 (Fig. 5.3/5.7).

IPOPT converges to a state of local infeasability (Fig. 5.6).

NLOPT converges to a local minimum with a f value of 228,952 (Fig. 5.5/5.8).

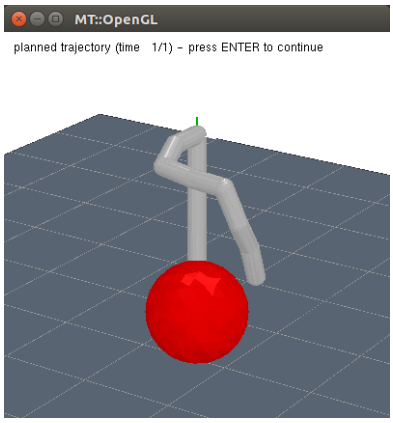

Figure 5.3: KOMO

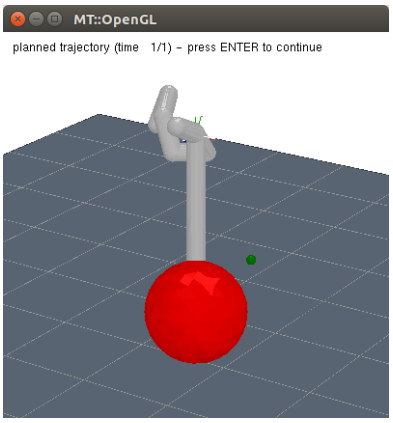

Figure 5.4: IPOPT

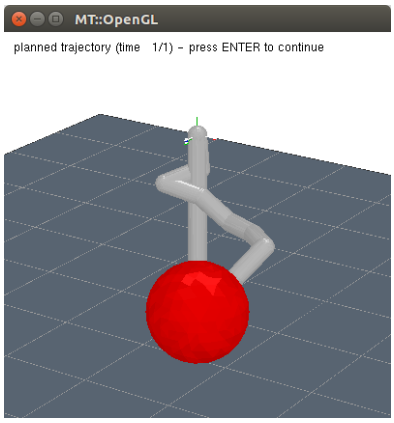

Figure 5.5: NLOPT

End states

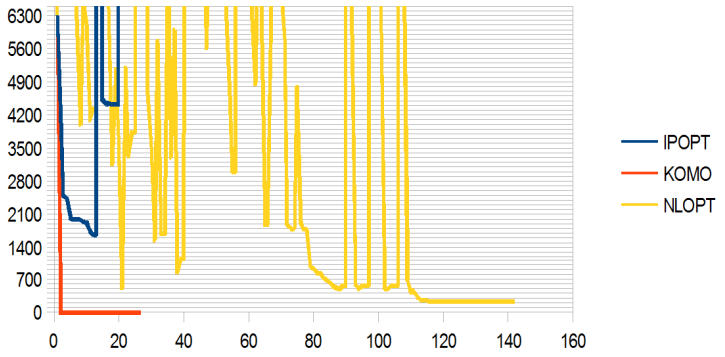

Figure 5.6: Global Convergence

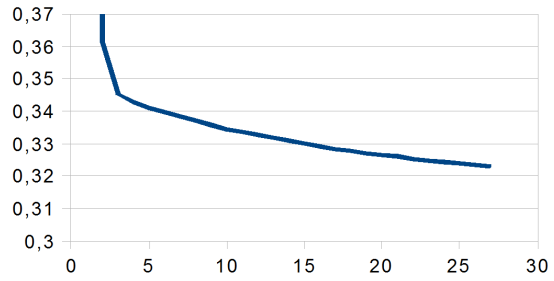

Figure 5.7: KOMO

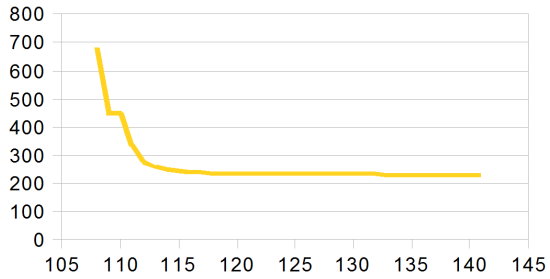

Figure 5.8: NLOPT Convergence

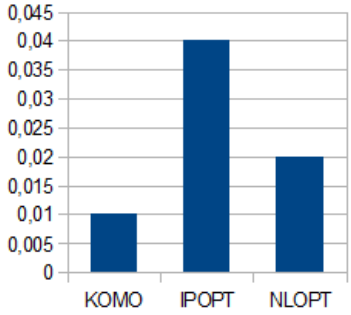

Figure 5.9: Time

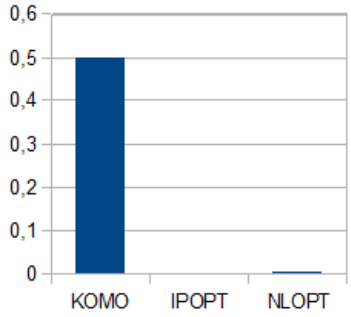

Figure 5.10: Constraint violation

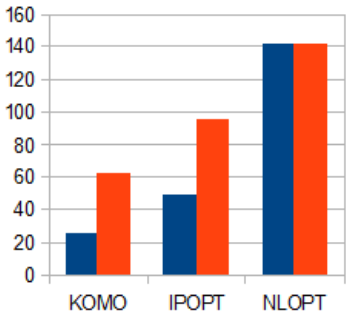

Figure 5.11: Iterations(Blue) Evaluations(Orange)

5.2.2 Time

In this case KOMO is the fastest, but NLOPT and IPOPT finish in a reasonable amount of Time (Fig 5.9)

5.2.3 Constraint violations

The result found by NLOPT violates the constraints significantly less than KOMO. (Fig 5.10)

KOMO = 0,5

NLOPT = 0,00533

5.2.4 Iterations and Evaluations

Komo requires the least iterations and evaluations but IPOPT and KOMO are still in a reasonable range. (Fig 5.11)

5.3 TestEasy Timesteps=10

5.3.1 Results

In this case KOMO and IPOPT find similar results. IPOPTS result is even slightly better.

KOMO converges to a f value of 1,97928 (Fig. 5.12/5.16).

IPOPT converges to a f value of 1,97076 (Fig. 5.13/5.17).

NLOPT isn't able to find a useful result in this case (Fig. 5.14/5.15).

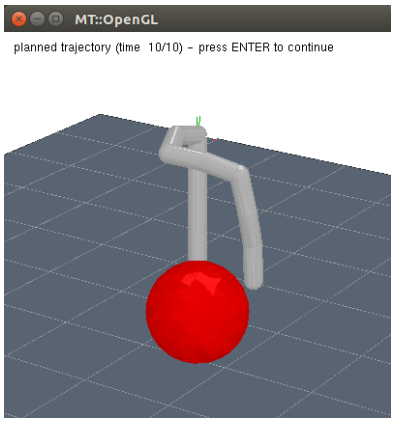

Figure 5.12: KOMO

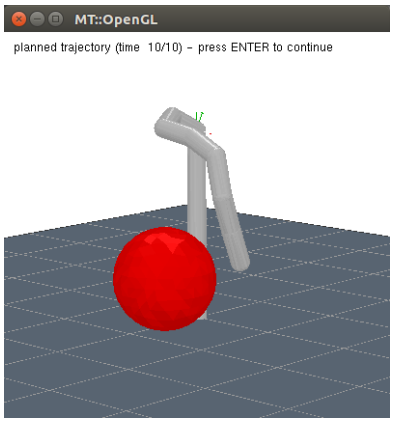

Figure 5.13: IPOPT

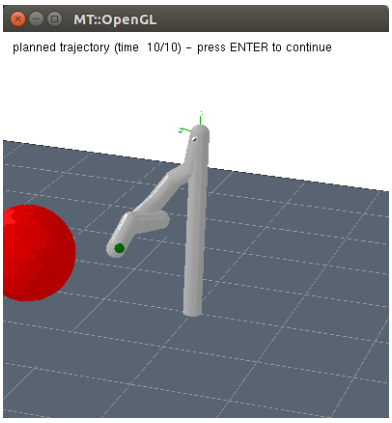

Figure 5.14: NLOPT

End states

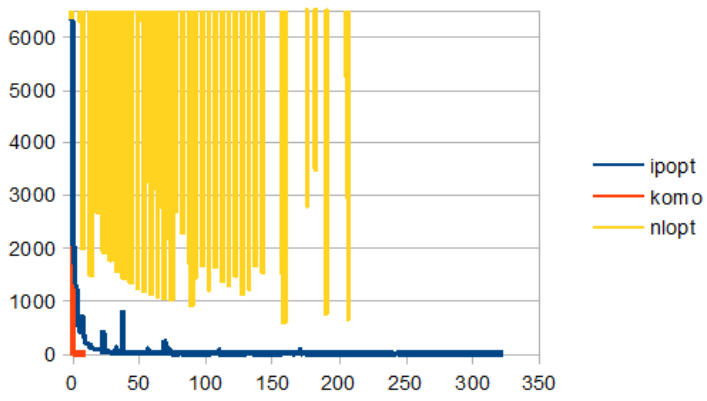

Figure 5.15: Global Convergence

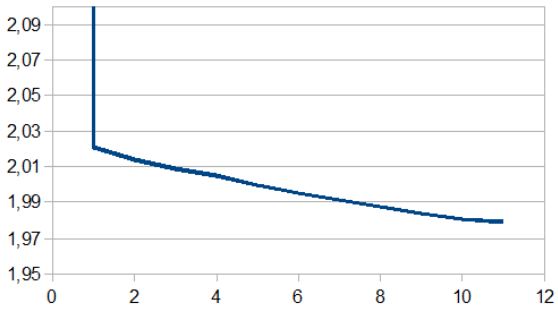

Figure 5.16: KOMO

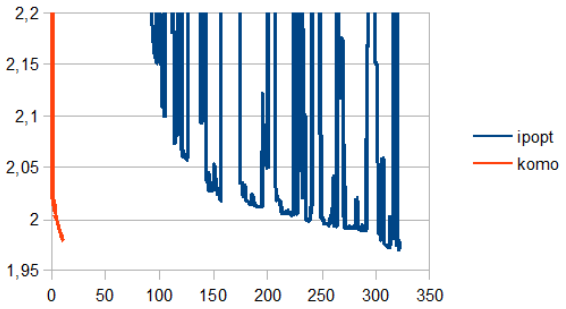

Figure 5.17: IPOPT/KOMO Convergence

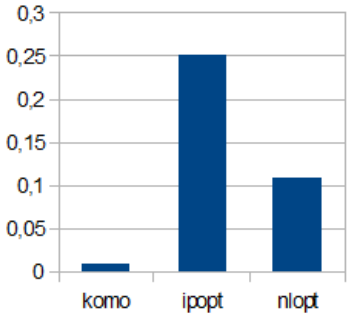

Figure 5.18: Time

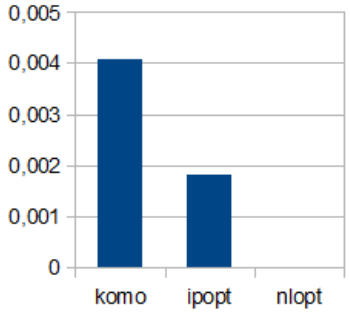

Figure 5.19: Constraint violation

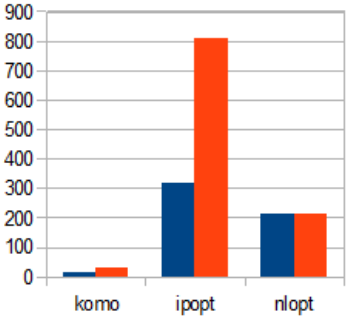

Figure 5.20: Iterations(Blue) Evaluations(Orange)

5.3.2 Time

In this case KOMO is the fastest, NLOPT and IPOPT already take a significantly longer amount of Time (Fig 5.18)

KOMO = 0,01s

IPOPT = 0,25s

NLOPT = 0,11s (doesn't find a useful result)

5.3.3 Constraint violations

The result found by IPOPT violates the constraints significantly less than KOMO. (Fig 5.19)

KOMO = 0,00408555

IPOPT = 0,001788

5.3.4 Iterations and Evaluations

KOMO requires the least iterations and evaluations, IPOPT and KOMO need a significantly higher amount. (Fig 5.18)

KOMO: Iterations=11 Evaluations=30

IPOPT: Iterations=322 Evaluations=812

NLOPT: Iterations=215 Evaluations=215

5.4 TestEasy Timesteps=100

5.4.1 Results

In this case KOMO and NLOPT find similar results. NLOPT result is even slightly better.

KOMO converges to a f value of 2,61789 (Fig. 5.21/5.25).

IPOPT converges to a state of local infeasibility (Fig. 5.22/5.24).

NLOPT converges to a f value of 2,56924 (Fig. 5.23/5.26).

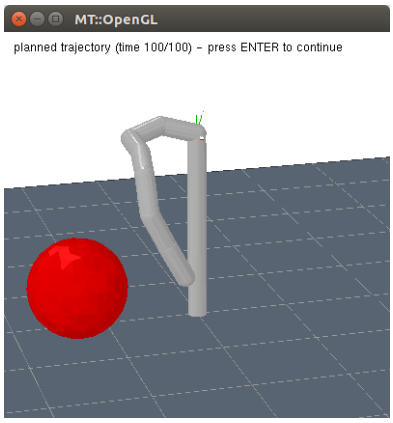

Figure 5.21: KOMO

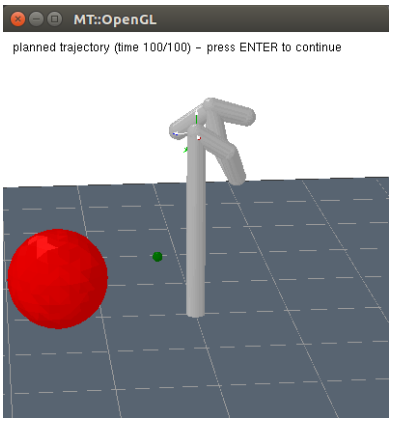

Figure 5.22: IPOPT

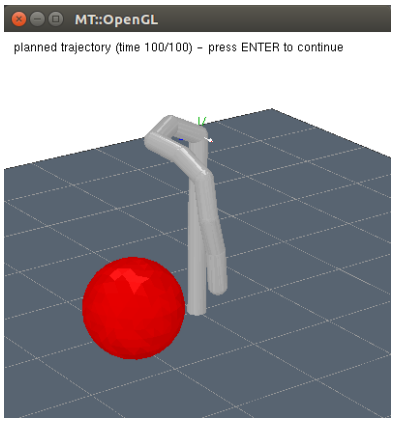

Figure 5.23: NLOPT

End states

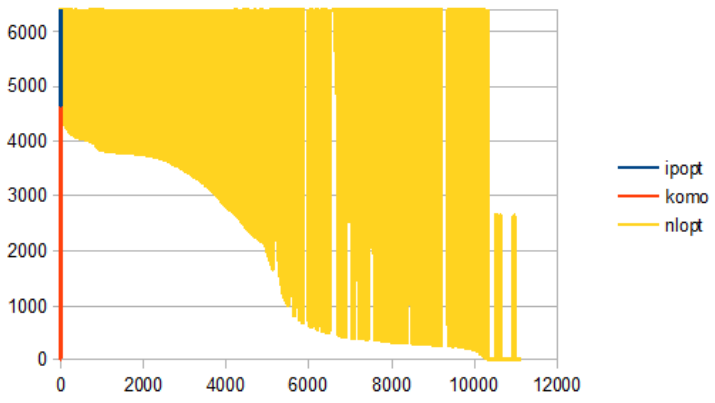

Figure 5.24: Global Convergence

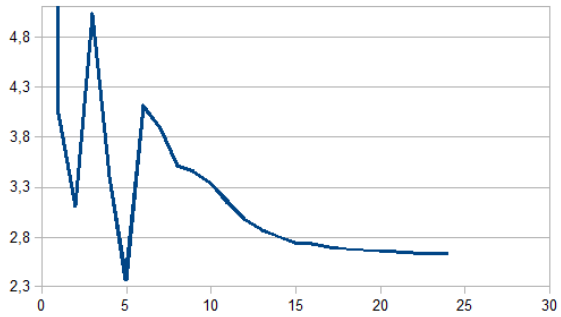

Figure 5.25: KOMO

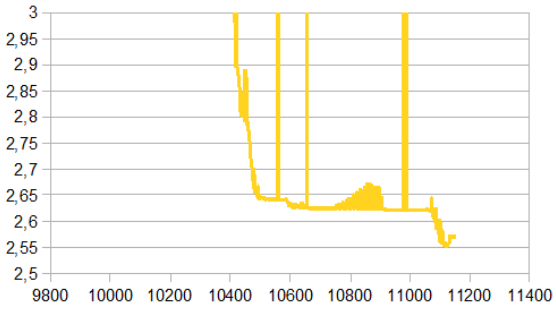

Figure 5.26: NLOPT Convergence

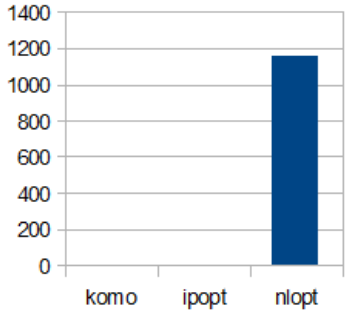

Figure 5.27: Time

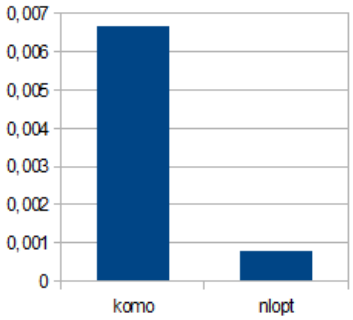

Figure 5.28: Constraint violation

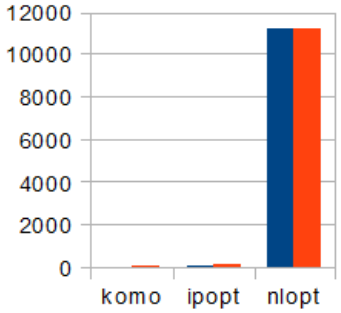

Figure 5.29: Iterations(Blue) Evaluations(Orange)

5.4.2 Time

In this case KOMO is the fastest, NLOPT takes extremely long to converge to a good result, while IPOPT finishes in somewhat reasonable time (Fig 5.27)

KOMO = 0,09s

IPOPT = 0,66s

NLOPT = 1156,13s

5.4.3 Constraint violations

The result found by NLOPT violates the constraints significantly less than KOMO. (Fig 5.28)

KOMO = 0,00664823

NLOPT = 0,000766757

5.4.4 Iterations and Evaluations

KOMO requires the least iterations and evaluations, IPOPT finishes in a reasonable amount of iterations, NLOPT takes a ridiculous amount of iterations to find a similar result like KOMO. (Fig 5.29)

KOMO: Iterations=24 Evaluations=60

IPOPT: Iterations=52 Evaluations=140

NLOPT: Iterations=11156 Evaluations=11156

5.5 TestEasyPR2 Timesteps=1

5.5.1 Results

In this case only KOMO finds a useful result.

KOMO converges to a f value of 0,333924 (Fig. 5.30/5.34).

IPOPT converges to a state of local infeasibility (Fig. 5.31/5.33).

NLOPT converges to a local minimum with a f value of 1.15933 (Fig. 5.32/5.35).

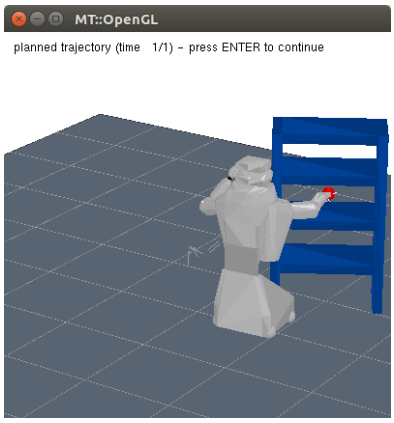

Figure 5.30: KOMO

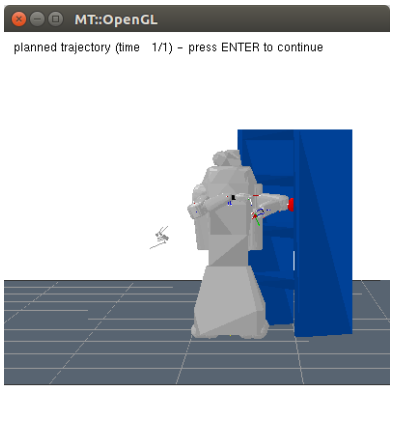

Figure 5.31: IPOPT

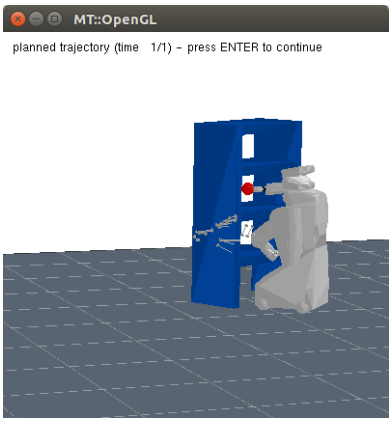

Figure 5.32: NLOPT

End states

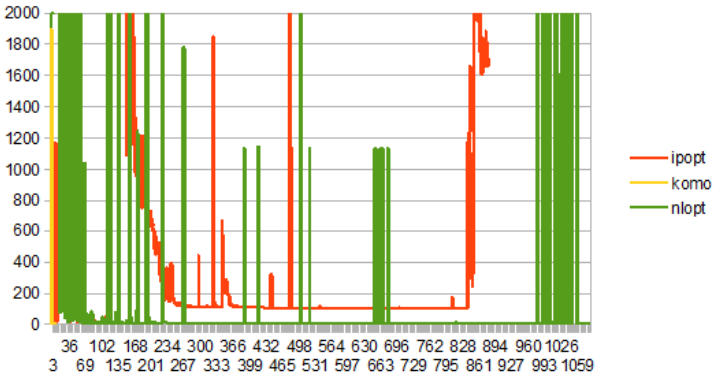

Figure 5.33: Global Convergence

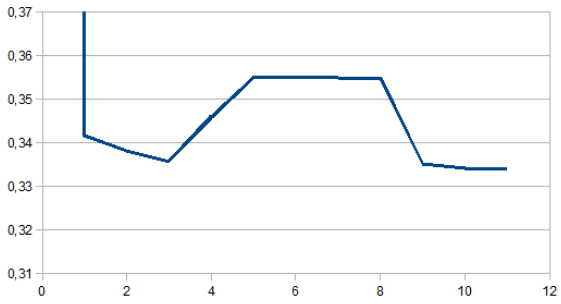

Figure 5.34: KOMO

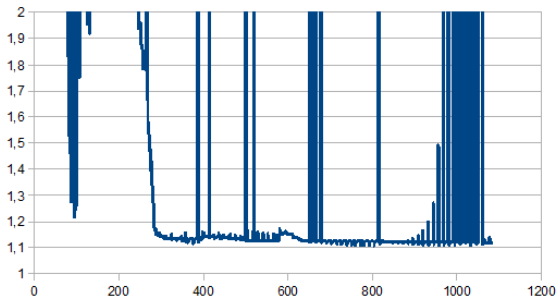

Figure 5.35: NLOPT Convergence

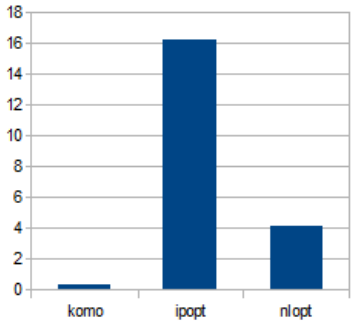

Figure 5.36: Time

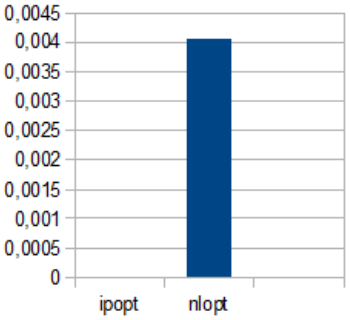

Figure 5.37: Constraint violation

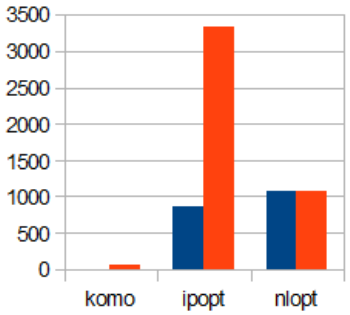

Figure 5.38: Iterations(Blue) Evaluations(Orange)

5.5.2 Time

In this case KOMO is the fastest, while NLOPT finishes in somewhat reasonable time (Fig 5.36)

KOMO = 0,23s

IPOPT = 16,13s

NLOPT = 4,04s

5.5.3 Constraint violations

Here KOMO finds a result which doesn't violate the constraints. NLOPT finds a result that violates the results slightly (Fig 5.37)

KOMO = 0

NLOPT = 0,00405637

5.5.4 Iterations and Evaluations

KOMO requires the least iterations and evaluations. Here IPOPT actually needs more evaluations than NLOPT but finds now viable result anyway. Nlopt takes significantly longer than komo (Fig 5.38)

KOMO: Iterations=11 Evaluations=63

IPOPT: Iterations=881 Evaluations=3332

NLOPT: Iterations=1087 Evaluations=1087

5.6 TestEasyPR2 Timesteps=10

5.6.1 Results

In this case KOMO and NLOPT find similar results. NLOPT result is even slightly better.

KOMO converges to a f value of 1,86955 (Fig. 5.39/5.44).

IPOPT converges to a state of local infeasibility (Fig. 5.39/5.44).

NLOPT converges to a a f value of 1,86213
(Fig. 5.41/5.43/5.45).

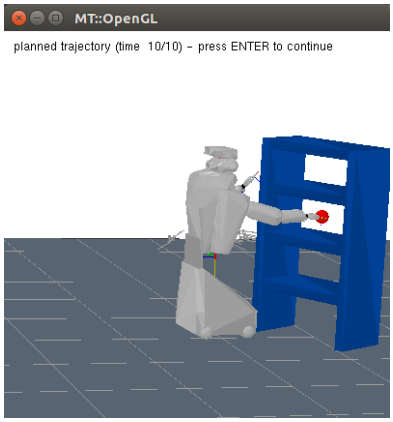

Figure 5.39: KOMO

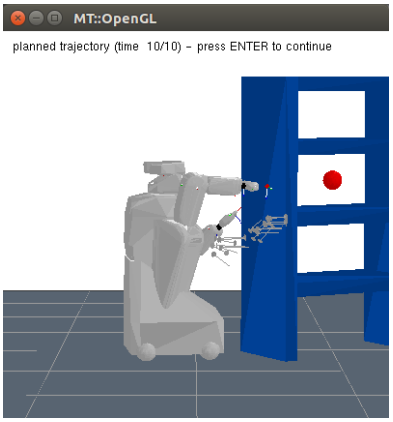

Figure 5.40: IPOPT

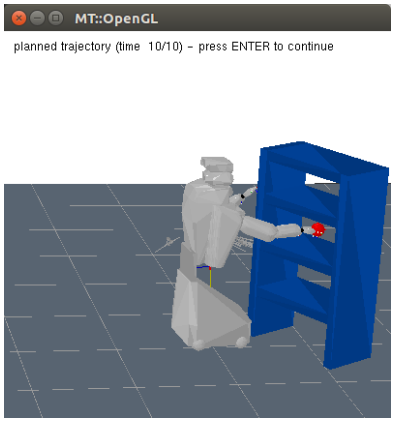

Figure 5.41: NLOPT

End states

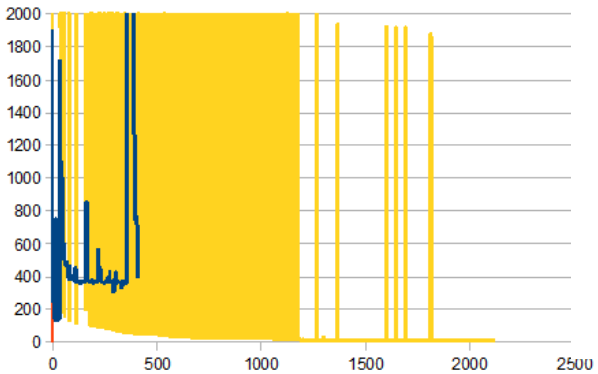

Figure 5.42: TestEasy

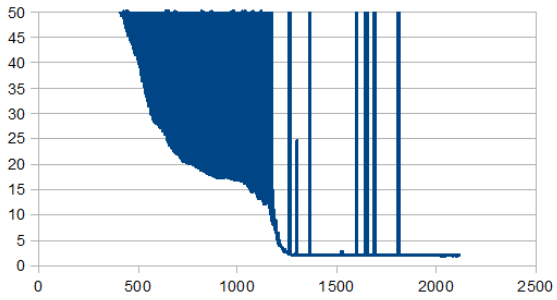

Figure 5.43: NLOPT

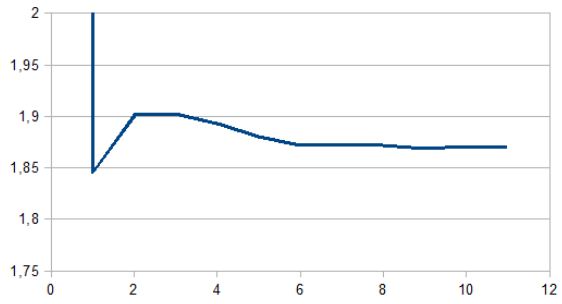

Figure 5.44: KOMO

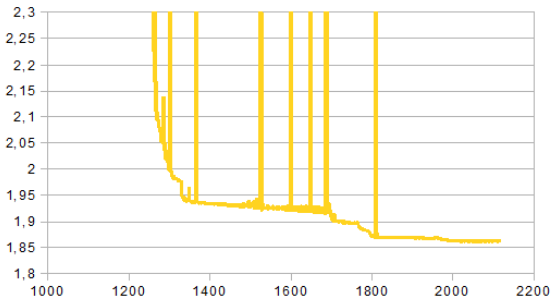

Figure 5.45: NLOPT Convergence

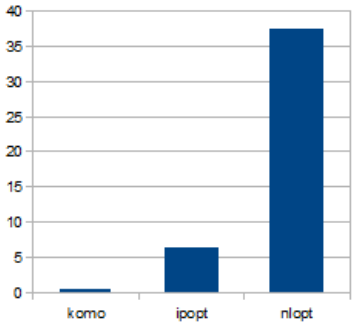

Figure 5.46: Time

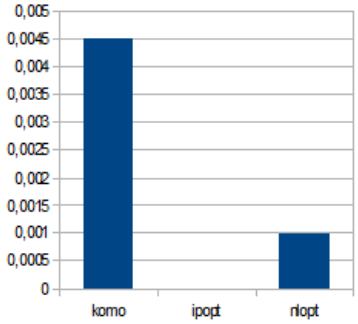

Figure 5.47: Constraint violation

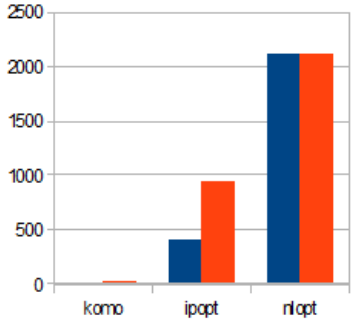

Figure 5.48: Iterations(Blue) Evaluations(Orange)

5.6.2 Time

In this case KOMO is the fastest, NLOPT takes a significant amount of time. (Fig 5.46)

KOMO = 0,26s

IPOPT = 6,3s

NLOPT = 37s

5.6.3 Constraint violations

Here KOMO finds a result which doesn't violate the constraints. NLOPT finds a result that violates the results slightly (Fig 5.47)

KOMO = 0,0044933

NLOPT = 0,000978789

5.6.4 Iterations and Evaluations

KOMO requires the least iterations and evaluations. IPOPT and NLOPT need signifiacntly more iterations and Evaluations than KOMO (Fig 5.48)

KOMO: Iterations=11 Evaluations=63

IPOPT: Iterations=407 Evaluations=942

NLOPT: Iterations=2121 Evaluations=2121

5.7 TestEasyPR2 Timesteps=100

5.7.1 Results

KOMO converges to a f value of 2.61283 (Fig. 5.53/5.49).

IPOPT gets stuck in a local minimum at 369,223 and was terminated after 20000 iterations. (Fig. 5.49/5.51/5.52).

NLOPT takes extremely long for each iteration and hasn't returned a result within two hours.

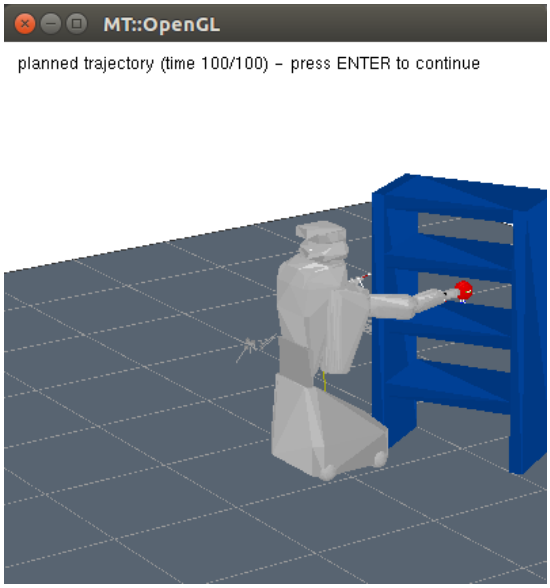

Figure 5.49: KOMO

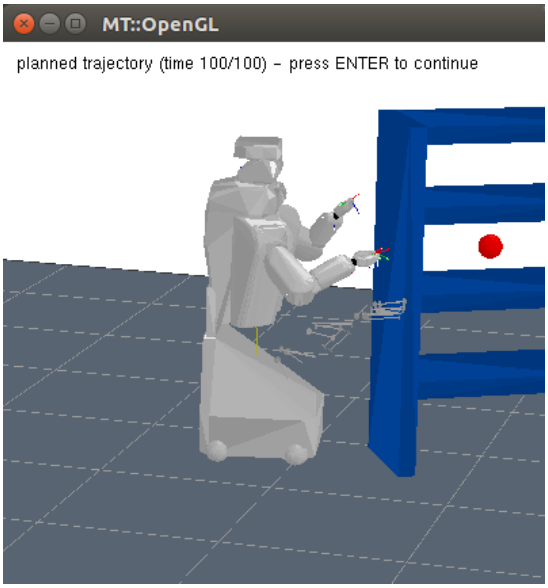

Figure 5.50: IPOPT

End states

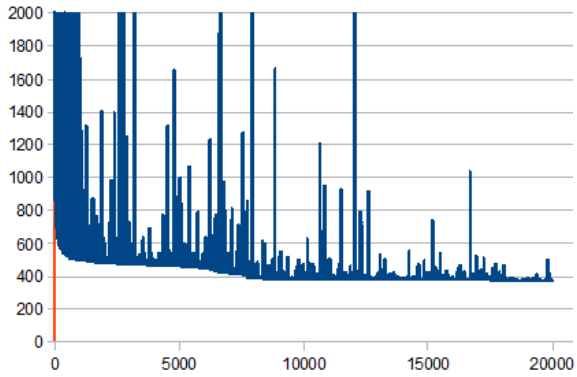

Figure 5.51: Global Convergence

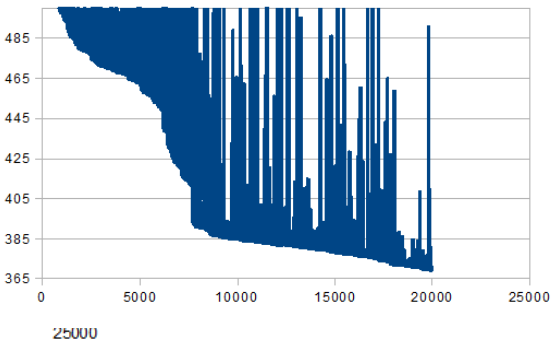

Figure 5.52: IPOPT

5.7.2 Time

In this case KOMO finds a result in a reasonable amount of time.
IPOPT isn't able to leave a local minimum in a most likely indefinet amount of time. (Fig 5.51/5.52/5.53)

KOMO = 0,75s

IPOPT = 2018s

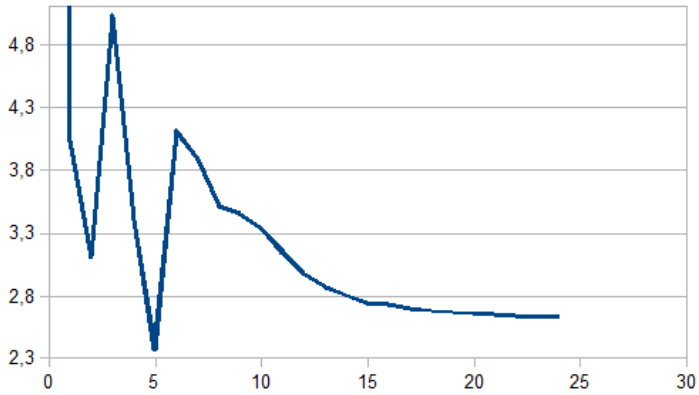

Figure 5.53: KOMO

5.7.3 Iterations and Evaluations

KOMO requires the least iterations and evaluations. After 20000 iterations IPOPT is still stuck in a local minimum (Fig 5.51/5.52/5.53)

KOMO: Iterations=11 Evaluations=26

IPOPT: Iterations=20000 Evaluations=52475

5.8 Analysis

Both IPOPT and KOMO are unreliable in achieving viable solutions. IPOPT tends to converge to a state of local infeasibility and NLOPT tends to get stuck in local minima(see Fig 5.8 or Fig 5.35).
IPOPT can also become stuck in a local minimum (Fig 5.51/5.52) And even if they find a solution they take significantly more time, Iterations, and evaluations.
They also seem to spend a significant amount of time in local minima (NLOPT example see Fig 5.24 and 5.26) and therefore tend to converge much slower, if they are even able to escape them.
KOMO on the other hand will achieve a relatively good solution on its first iteration and converge from there, in a short amount of time and a few further iterations reliably.

On the bright side in the cases IPOPT or NLOPT achieve solutions similar to KOMO they always have a slightly lower f value and the overall constraint violations tends to be less than KOMO this is the case in.

NLOPT: TestEasyPR2 Timesteps=10, TestEasy Timesteps=100.

IPOPT: TestEasy Timesteps=10

5.8.1 Time/Iterations/Evaluations

In all used examples KOMO requires far less time/iterations and evaluations.
Since IPOPT and NLOPT didn't converge to the same f value or even achieved a useful value in the same example. I can't compare there needed time/iterations and evaluations to each other.
But it is noticable that generaly speaking the time needed by IPOPT increase more slowly than the time needed by NLOPT with an increased number of variables and constraints. Also, for NLOPT, the needed Time per iteration seems to increase faster with a increased number of variables, than for IPOPT or KOMO.

6 Conclusion

Compared to KOMO, IPOPT and NLOPT aren't useful for the problems I tested.
They are both to unreliable in returning results comparable to KOMO in quality.
And in cases, where they give viable results, they take far to long compared to KOMO.

The biggest advantage of KOMO seems to be, the ability to get pretty close to the final result within the first iteration.
Therefore I suspect that IPOPT and/or NLOPT might become useful if they received a starting point closer to the desired minimum, which KOMO currently isn't able to provide.
This might be only true for IPOPT since NLOPT becomes extremely slow with a rising number of variables and constraints.

Bibliography

[FGK02] R. Fourer, D. M. Gay, B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Cengage Learning, 2002 (cit. on p. 11).

[Fou12] R. Fourer. "On the Evolution of Optimization Modeling Systems." In: *Documenta Math. Extra Volume: Optimization Stories* (2012), pp. 377–388 (cit. on p. 13).

[Hsl] *HSL(2013). A collection of Fortran codes for large scale scientific computation*. URL: http://www.hsl.rl.ac.uk (cit. on p. 15).

[Joh] S. G. Johnson. *The NLopt nonlinear-optimization package*. URL: http://ab-initio.mit.edu/nlopt (cit. on p. 16).

[Kra49] D. Kraft. "Algorithm 733: TOMP–Fortran modules for optimal control calculations." In: *ACM Transactions on Mathematical Software* 20.3 (1949), pp. 262–281 (cit. on p. 16).

[Kra88] D. Kraft. *A software package for sequential quadratic programming*. Tech. rep. DFVLR-FB 88-28. Institut für Dynamik der Flugsysteme, 1988 (cit. on p. 16).

[Mkl] *Intel(R) Math Kernel Library 11.1 Update 1 for Linux* (cit. on p. 15).

[Tou14] M. Toussaint. *KOMO Newton methods for k-order Markov Constrained Motion Problems*. 2014. URL: http://arxiv.org/abs/1407.0414 (cit. on p. 9).

[WB06] A. Wächter, L. T. Biegler. "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming." In: *Mathematical Programming* 106(1) (2006), pp. 25–57 (cit. on p. 15).

All links were last followed on September 16, 2016.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature