Motion Trajectory Generation for Robotic Manipulators via Sequential Convex Optimization

James Schuback
75 James Street, Toronto, ON, M8W1L7
james.schuback@mail.utoronto.ca

Abstract

To interact in a dynamic environment alongside humans, it is essential that robots not just be able to move as commanded, but also constrain their motions with respect to other entities - to react. An example of such a situation is obstacle avoidance: given a target manipulator position and a sensed obstacle, find a new path for the end effector which reaches the desired target while avoiding the obstacle. Specifying the robot dynamics and kinematics in the 3D workspace leads to functions involving trigonometric polynomials, causing non-convex constraints. In this work we explore an iterative convex approximation method and analyze the computational resources required. An example application is presented on a 4 degree of freedom robotic arm of a commercial robot. The complete software is available at https://github.com/js117/CVX/tree/master/MotionTrajGenSCP

1 Introduction

Robotic manipulators consist of a series of *joints*, rotational components comprised of actuators such as DC motors, and *links*, rigid sections connecting joints. A human arm can be thought of as having 7 joints and 2 links: 3 rotational joints at the shoulder, the femur link (upper arm), another joint at the elbow, followed by the forearm link, then 3 joints at the wrist, and finally the hand, which can be thought of as the *end effector*. Typical industrial robots may have grippers or other custom tools as end effectors. The structure comprising of joints, links, and end effector is known as a *robotic manipulator*. A robotic arm in an assembly line is a typical example.

Specifying the position (as well as orientation) of the end effector of a robotic manipulator as a function of the joint angles is done via *forward kinematics*: given the joint angles, find the position of the end-effector [1]. Through 3D geometry, these equations are found to be polynomials of trigonometric functions of joint angles, for example in [1]. The inverse problem, *inverse kinematics*, takes as input a desired *workspace target* (position and orientation of the end effector) and outputs the joint angles that will realize it. Inverse kinematics will often be of use to specify an end target configuration of joint angles.

By *trajectory generation* we mean finding a sequence of joint angles, $\theta_t \in \mathbb{R}^n, t = 1, ..., T$ (where n is the number of joints of the manipulator and T is the number of discrete time steps defining the trajectory), that satisfy our objective given constraints. For example, $n = 4$ for a 4-DOF arm and $T = 50$ for a 1-second trajectory with 20ms samplings. Using the kinematics equations, we can confine the end effector to always be within a region - inside of which the manipulator does not contact any obstacles.

2 Problem Statement

In this work we consider the problem of constraining the end-effector to be within a sequence of rectangular prisms in 3D space while moving to a target configuration from an initial configuration. Let $f(\theta_{1:T})$ be a convex objective function of the joint angles over the trajectory. Typical functions to be minimized are vector norms which roughly correspond to different types of energy utilization or torque output. Let $K(\theta_t)_{x,y,z}$ denote the (non-convex) forward kinematics functions which map the joint angles to the end effector's x, y, or z position in the workspace.

Finally, we constrain the range of joint angles to be within some lower and upper limits between $[0, 2\pi)$. As well, we limit the maximum possible *transition magnitude* of joint angles between time steps. This means that in a time step t, $|\theta_t - \theta_{t-1}| \leq \alpha$ and physically this corresponds to the actuators having a maximum velocity.

The problem is stated as follows:

$$\begin{gathered}
\text{minimize} \\
\|\theta_{1:T}\| \\
\text{subject to} \\
\theta_0 = \theta_{init} \\
\theta_T = \theta_{final} \\
|\theta_t - \theta_{t-1}| \leq \alpha, t = 2, ..., T \\
x_{min} \leq K_x(\theta_t) \leq x_{max}, t = 1, ..., T \\
y_{min} \leq K_y(\theta_t) \leq y_{max}, t = 1, ..., T \\
z_{min} \leq K_z(\theta_t) \leq z_{max}, t = 1, ..., T
\end{gathered}$$

We assume that the given initial configuration θ_{init} and final configuration θ_{final} correspond to feasible workspace points, i.e. they satisfy the inequality constraints. As well, we set our initial guess of the trajectory $\theta_{1:T}$ as being a linearly spaced sequence between θ_{init} and θ_{final}.

3 Problem Modeling

To solve the optimization problem in the proposed framework, convex approximations of the kinematics functions are required. The sequential programming approach involves iteratively solving the modified problem on a restricted domain ('trust region') where the convex approximations are accurate enough to 'make progress' towards better solutions. While one cannot guarantee global convergence of this approach (and they need not even be descent methods), in practice they can yield reasonably effective solutions. A summary can be found in [8].

A natural approximation approach for smooth differentiable functions is to take the second order Taylor expansion about a nearby point, and constrain the trust region to be a bounding box about this point:

$$f(x) \approx f(x_0) + \nabla f(x_0)^T(x - x_0) + \frac{1}{2}(x - x_0)^T \nabla^2 f(x_0)(x - x_0)$$

Where $f(x) = K(\theta_t)_{x,y,z}$ and we will label the approximation $k(x)$. Convexity of $f(x)$ depends on the definiteness of the Hessian matrix $\nabla^2 f(x_0)$, which in general is indefinite. Note that for the constraints $f(x) \leq u$ we would need the Hessian to be positive semidefinite, while for $l \leq f(x)$ the Hessian needs to be negative semidefinite. The chosen approach was to decompose $k(x)$ into a convex and concave part via eigenvalue and singular value decomposition such that $k(x) = k_{cvx}(x) + k_{ccv}(x)$. Now the $6T$ inequality constraints are approximated as:

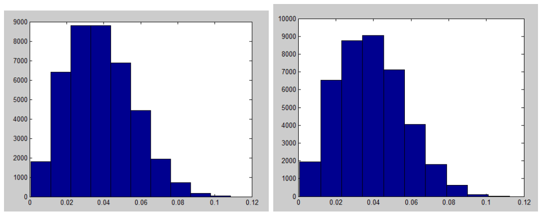
$$k^{p,t}_{cvx}(\theta_t) \leq u_{p,t}$$

$$l_{p,t} \leq k^{p,t}_{ccv}(\theta_t)$$

For all $p = x, y, z, t = 1, ..., T$.

To numerically test the accuracy of these approximations, we repeatedly form the approximation functions $k_0(\phi)$ about a uniform randomly generated θ_0 point in the joint angle domain, then compare the value of $K(\phi)$ to $k_0(\phi)$ for a random sample of points $\{\phi\colon \|\theta_0 + \phi\|_\infty \leq r\}$. Below we show results for 200 samplings of ϕ about 200 θ_0 samples in the joint angle domain, for a particular angle radius value r. We do this for both the convex and concave approximations, and average the error values over the x, y, and z functions.

Figure 1: Histogram of errors of k_{cvx} and k_{ccv} - mean errors 3.8cm and 3.88cm respectively, $r = 0.05$ rad (2.9 degrees) with a max error of 11cm. At $r = 0.2$ rad (11.5 degrees), the mean errors were close to 15cm, with max errors approaching 45cm.



4 Algorithms and Software

For this work we chose to use ECOS: an SOCP solver for embedded systems [7]. Since the approximate convex problems being solved iteratively have quadratic constraints, the problem can be phrased as a second order cone program. The SOCP framework also provides room to extend the accuracy of the function approximations.

ECOS uses a primal-dual Mehrotra predictor-corrector method with Nesterov-Todd scaling and self-dual embedding. According to [4], primal-dual methods are "often more efficient than barrier methods, especially when high accuracy is required." This type of algorithm is considered state-of-the-art, and numerical results in [7] show its competitiveness with commercial products. Having this power of solver available as open-source software, written in easily portable C code makes it ideal for robotics applications.

A key requirement of these sequential methods is the ability to start from infeasible points, as in general we might not know a feasible path around the obstacle a priori. The difficulty in finding a feasible point is on the same order of magnitude as finding an optimal solution. Self-dual embedded symmetric primal-dual algorithms can operate at infeasible points (e.g. initial sequence guess). Thus we do not require phase I/II methods to find initial feasible points ([5], [6]).

An additional benefit of these methods is that they have an interface to return certificates of infeasibility and unboundedness [7]. This is helpful in discerning both the efficacy of function approximations (with respect to the trust region), as well as infeasibility of the original problem to be solved. In a real-time robot controller, having access to quick certificates of infeasibility can be a major advantage in not taking a potentially dangerous action or movement.

Example Sequential Convex Programming (SCP) routines are described in [8] and [11], with the latter being applied to an analogous robotics trajectory problem.

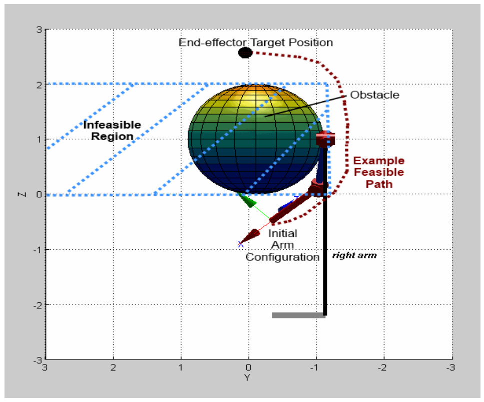
5 Sample Problem: The Nao Robot

To test the methodology presented, we form an instance of the main optimization problem pertaining to the 4 degree of freedom right arm of the Nao Humanoid Robot [9]. Using the MATLAB Robotics toolbox [3], we obtain an accurate kinematics model as well as a visual representation of the arm.

Simple manipulators such as these have analytic inverse kinematics solutions, nullifying the need for iterative methods that use the forward kinematics to navigate a feasible path. However the kinematics are still indicative of the type of nonconvex functions present in more advanced manipulators which may not have closed-form inverse solutions. This makes the Nao a suitable test robot, with the added advantage that one can use the explicit inverse functions to compare optimization routine solutions to other solutions.

The problem explored was to avoid an obstacle in the y-z plane represented by box constraints, in moving the end-effector from its initial configuration to its final target.

Figure 2: The Nao robot's right arm, from a view facing forward, with an obstacle represented by box constraints in the y and z dimensions.



The precise optimization problem solved in this example is as follows:

$$\text{minimize}$$
$$\delta 1^T s + \gamma \|\theta_{1:T} - \theta^{lin}_{1:T}\|_1$$
$$\text{subject to}$$
$$\theta_0 = \theta_{init}$$
$$\theta_T = \theta_{final}$$
$$\theta_{min} \leq \theta_t \leq \theta_{max},$$
$$-s_{t,y} + y_{min} \leq K_y(\theta_t) \leq y_{max} + s_{t,y}$$
$$-s_{t,z} + z_{min} \leq K_z(\theta_t) \leq z_{max} + s_{t,z}$$

This problem is defined for variables $s \in \mathbb{R}^{2T}$, $\theta \in \mathbb{R}^{nT}$ and over the time horizon $t = 1, ..., T$. For the Nao right arm considered, $n = 4$ joints, and $T = 15$ was the chosen time horizon. The chosen θ radius, θ_r, was 0.075 rad (4.3 degrees), defining our angle limits per iteration as $\theta_t \pm \theta_r$. Therefore in each iteration, the joints can change at most 4.3 degrees before the next iteration recomputes the approximation functions and resolves the problem.

The regularization constants δ and γ can be thought of as controlling the strength of two opposing forces: the first force aims to reduce infeasibility, while the second works to reduce energy usage. Their chosen values were 1.5 and 1.1 respectively.

The norm in the objective is with respect to the difference of the optimal path with the initial linearly interpolated path. This choice was made because it is reasonable to assume that the straight-line linear path is a prototype "efficient" path by requiring the least amount of effort or energy. So while the iterative optimization routine constraints aim to take the arm's path outside of the infeasible box, this objective acts as a force to pull it closer to the optimal path (the path chosen when there is no obstacle).

The L1-norm was chosen to show a different flavor of solution this methodology can produce. A different norm on the joints could also be used, and this is an area to be explored in future work. For example, the L2-norm of the velocity of joints ($\theta_t - \theta_{t-1}$) could yield a smoother path. Indeed, in the above formulation the trajectory is found to have more abrupt transitions between time steps, analogous to bang-bang control [10]. The desired norm may be application specific; this framework is amenable to various choices.

For completeness, the explicit format of the kinematics functions is given below for argument $\phi = \theta_t$. Symbols a through j represent real numbers specific to robot arm parameters.

$K_y(\phi) = (asin(\phi_2))/b + (ccos(\phi_4)sin(\phi_2))/e +$

$gcos(\phi_2)cos(\phi_3)sin(\phi_4)/i - j$

$K_z(\phi) = 1 - (dsin(\phi_4)(cos(\phi_1)sin(\phi_3) - cos(\phi_3)sin(\phi_1)sin(\phi_2)))/e - (fcos(\phi_2)cos(\phi_4)sin(\phi_1))/e - (acos(\phi_2)sin(\phi_1))/b$

6 Deviations from Standard SCP

The above formulation contains some notable departures from traditional SCP techniques as in [8]. Instead of modifying the trust region based on problem infeasibility, we keep a fixed trust region size based on a predefined level of accuracy in the functions that require approximation: $K_{y,z}(\theta_t)$. In addition, our problem is always feasible due to the slack variable s. Constraining the joint angles within min and max values properly ensures physical feasibility as well. Ideally, if the method works, the sum of infeasibilities should be decreasing as iteration count increases.

This is in contrast to related work such as [11] where if sufficient progress is not made per iteration, the trust region must be shrunk (to try again). As well, the standard algorithm attempts to increase the trust region if sufficient progess is being made in the true objective. However, running numerical tests on the constraint functions, we can decide a priori what a reasonable trust region might be. For example, along the path, are we willing to accept a mean error of 10cm and a max error of 30cm? If not, we shrink the trust region.

In this application in particular there ought to be a standard upper bound on error we should accept based on the accuracy of which we sense the object, and the bounding box we assign it. Clearly the larger the trust region, the bigger the local approximation step each iteration can take (and faster the algorithm runs due to less steps), so it is this upper limit that we seek. Note that we could also trade-off trust region size with approximate global optimality as follows: a bigger trust region requires a more conservative bounding box about the obstacle (due to position function errors), and a bigger bounding box necessarily means a larger deviation from the optimal straight-line path, and thus more energy expended.

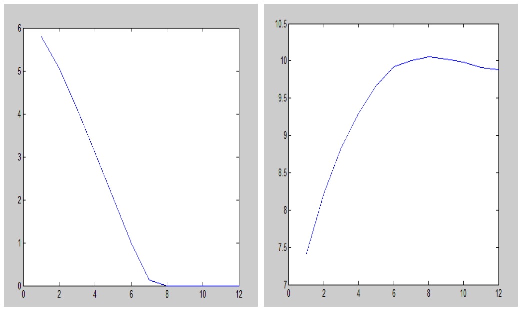
As shown in the next section, our simpler method tends to reduce the sum of infeasibilities monotonically towards zero (evaluated using the true kinematics functions). This is because on average our second-order convex approximations are very accurate within the chosen θ range. In practice our method may require less SCP iterations, and certainly less algorithm parameters.

Therefore by studying the quality of function approximations as in section (3), we can potentially have a faster algorithm. It should be noted, however, that we are only able to do this in practice due to the low dimensionality of the functions considered. For a general non-convex function in higher dimensions, a conservative trust-region approach is appropriate.

7 Experimental Results

For the example problem, we ran 12 iterative solves as described. By iteration 8, the sum of infeasibilities with respect to the true kinematics functions was found to be zero (and remained so for later iterations), indicating that we were outside of the obstacle box.

Figure 3: The true sum of infeasibilities (left) was found to decrease monotonically towards zero with each iteration. The true objective value is plotted on the right.

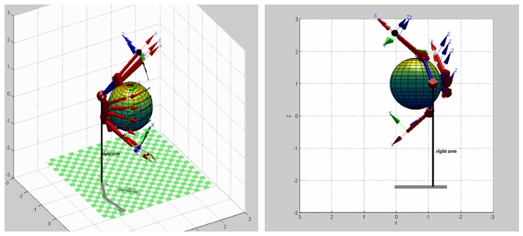


However, as in general with SCP methods, the true objective did not decrease monotonically, i.e. this is not a descent method. In fact, the true objective increased until the trajectory was outside of the bounding box (iteration 8). This is due to the fact that the sum of infeasibilities decreasing necessarily opposes energy usage, measured as deviation from the straight-line path. However, observe that after iteration 8, the objective value starts decreasing again until it appears to flatten.

The algorithm may be thought of as overcoming a potential well and then fine-tuning the path once outside of it. This is thus a balancing act between the regularization constants. Indeed it was found that for a ratio of $\gamma/\delta \geq 1.8$ the slacks stalled at a positive number, even for a large (30+) iteration count; the obstacle was not avoided.

Below we have the example trajectory plotted from various viewpoints.

Figure 4: Time-lapsed plot of sample arm motion trajectory.



In terms of computational resources, each iterative solve represents a Second Order Cone Program with 165 variables, 285 linear inequality constraints, and 120 second order cone constraints. These iterations were solved in roughly 0.4s (the solve) plus 1.66s (the re-approximation) , for a total time of 25 seconds on an AMD A8-3520M APU (1.6 GHz quad core processor) running in MATLAB software.

8 Conclusion

Overall, this iterative convex approximation approach was found to be an effective method for navigating non-convex constraints. The flexibility and generality of the approach opens avenues for future work to explore dynamics problems of similar systems.

As well, there is ample opportunity to create software that is better performance optimized than the MATLAB prototype. Example techniques would be: preallocating all needed data structures, and coming up with explicit update equations in the re-approximations (such as the eigen decomposition of small matrices). One can also trade off time horizon length, T, by creating more conservative obstacle bounding boxes to make up for the poorer continuous discretization. Real-time systems may need a solution within 33ms: a typical sampling rate for a camera system that can detect an obstacle.

The models of the functions can also be improved within the SOCP framework. SOC constraints yield quadratic forms that can have a single eigenvalue of the non-definite sign if the matrix can be expressed (or approximated) by a low-rank update [12]. By capturing a greater portion of the true Hessian, the approximations can be more accurate.

References

[1] M. Spong, S. Hutchinson, M. Vidyasagar *Robot Modeling and Control* New York: John Wiley & Sons, INC., 2005

[2] P. Corke, *Robotics, Vision & Control: Fundamental Algorithms in Matlab*, Springer, 2011.

[3] P. Corke, *A Robotics Toolbox for MATLAB*, IEEE Robotics and Automation Magazine, Volume 3(1), March 1996, pp. 24-32.

[4] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004

[5] L. Vandenberghe, *EE236C, Lectures 14-18*, UCLA, Spring 2011

[6] A. Ben-Tal, A. Nemirovski, *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, MPS-SIAM Series on Optimization

[7] A. Domahidi, E. Chu, S. Boyd, *ECOS: An SOCP Solver for Embedded Systems* Proceedings European Control Conference, pages 3071-3076, Zurich, July 2013.

[8] S. Boyd, *Sequential Convex Optimization, EE364B Lecture 11*, Stanford, 2008

[9] N. Kofinas, *Forward and Inverse Kinematics for the NAO Humanoid Robot*, Diploma Thesis, Department of Electronic and Computer Engineering, Technical University of Crete, July 2012.

[10] G. Vossen, H. Maurer, *On L1-minimization in optimal control and applications to robotics*, Optimal Control Applications and Methods, 2006; 27:301321

[11] J. Schulman, Y. Duan, et al, *Motion Planning with Sequential Convex Optimization and Convex Collision Checking*, International Journal of Robotics Research (IJRR), 2014.

[12] A. Mahajan, T. Munson, *Exploiting Second-Order Cone Structure for Global Optimization*, Argonne National Laboratory, 2010.