Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Automated Custom Code Generation for Embedded, Real-time Second Order Cone Programming

Daniel Dueri, * **Jing Zhang,** * **and Behçet Açikmeşe** *

* *The University of Texas at Austin, Austin, TX 78712 USA (e-mail: daniel.dueri@utexas.edu, serenaj.zhang@gmail.com, behcet@austin.utexas.edu).*

Abstract: In this paper, we discuss the development of an Interior Point Method (IPM) solver for Second Order Cone Programming optimization problems that is capable of producing customized ANSI-C code for embedded, real-time applications. The customized code is generated for a given problem structure and makes use of no dynamic memory allocation, minimizes branching, wastes no mathematical or logical computations, and has minimal dependencies to standard libraries. The resulting software is designed to be easy to implement on embedded hardware with limited computing capabilities, while still providing accurate results rapidly enough for real-time use. The core IPM algorithm is a fairly standard primal-dual IPM, which makes use of Mehrotra predictor-corrector method with Nesterov-Todd scalings and Newton search directions. We make use of the Approximate Minimum Degree heuristic to maximize the sparsity of the Cholesky factorizations that are ultimately used to solve for the search directions. We conclude the paper by presenting the computational performance results from two example problems: a Mars landing optimal control problem and a reaction wheel allocation problem. The code generated for the Mars landing problem was successfully validated in three flights onboard a NASA test rocket, and was used in real-time to generate the optimal landing trajectories that guided the rocket. To the best of our knowledge, this was the first time that a real-time embedded convex optimization algorithm was used to control such a large vehicle, where mission success and safety critically relied on the real-time optimization algorithm.

1. INTRODUCTION

Historically, optimization based algorithms have been relegated to ground-based design and Monte-Carlo simulations on desktop computers or even clusters. The main reason behind this choice was that numerical optimization was not believed to be reliable enough for real-time, autonomous computations. This belief was somewhat justified since many traditional nonlinear programming techniques (Bersekas, 1999) have no guarantees of finding optimal (or, even feasible) solutions to constrained optimization problems. On the other hand, Convex Optimization (CO) contains a large class of optimization problems that can be reliably solved in polynomial time to global optimality, without any expert tweaking of solution parameters (Boyd and Vandenberghe, 2004). This motivated researchers, including us, to formulate many challenging control problems in a convex optimization framework. Furthermore, recent research has shown that Linear and Quadratic Programming (LP and QP) problems can be solved by *custom* algorithms in real-time (Mattingley and Boyd, 2010b). There have also been other approaches to real-time convex optimization, such as multi-parametric programming (Jones et al., 2007) that systematically generates tables, and later interpolates from them in real-time to obtain the optimal solutions. Motivated by these developments, our goal in this paper is to introduce a new software capability (one of the first of its kind) to customize a generic Interior Point Method (IPM) algorithm for real-time solutions of Second Order Cone Programming (SOCP) problems onboard autonomous systems.

Recently, we generated a real-time IPM for the Mars landing path-planning problem that was successfully validated in three flights onboard a NASA test rocket, and was used in real-time to generate the optimal landing trajectories that guided the rocket (JPL et al., 2013; Scharf et al., 2014; Açıkmeşe et al., 2013; JPL and Systems, 2012; Aung et al., 2013). To the best of our knowledge, this was the first time that a real-time embedded IPM algorithm was used to control such a large vehicle, where mission success and safety critically relied on the real-time optimization algorithm. The custom IPM algorithm solved a trajectory optimization problem in real-time onboard the rocket in order to obtain a fuel-optimal landing trajectory. This capability enabled the rocket to fly laterally more than 2.5 times more than was previously possible. Consequently, real-time convex optimization proved to be an enabling technology for a dramatic improvement in the flight envelope of the rocket. It is demonstrated, for the first time, that real-time convex optimization can be used reliably in time-critical applications for high performance missions.

Convex Optimization lends itself well to the solution of constrained control problems, provided that the cost function and constraints are convex. The latter requirement may seem restrictive, however, recent discoveries in the field of optimal control theory have proven that a very

general class of control problems with non-convex control constraints can be posed as convex optimization problems without loss of generality via a procedure known as "lossless convexification" (Harris and Açikmeşe, 2013; Açıkmeşe and Blackmore, 2011; Blackmore et al., 2012; Açıkmeşe and Ploen, 2007; Açikmeşe et al., 2013; Blackmore et al., 2010). It has been shown that a convex relaxation of many non-convex control constraints can be used to obtain a globally optimum solution that is the same as the globally optimal solution of the original non-convex problem. As a consequence, the CO solver presented in this paper can be used to solve a variety of control problems with mixed convex and non-convex control constraints.

In this paper, we discuss the implementation of an IPM algorithm that has the capability of producing customized solvers for use on real-time computational hardware. The ultimate aim of our work is to solve optimal control problems in real-time onboard vehicles by making use of IPM algorithms. For the purposes of this paper, we define "customized" to mean "tailored to a particular problem structure;" that is, when a solver is produced for a particular problem structure, it can only solve instances of this problem class, but will do so with much increased computational efficiency. This tailoring not only reduces the number of operations required to solve the problem, but also reduces the number of `if` statements that are used - a property that makes it very appealing for real-time code, which should have as few branches (or `if` statements) as possible. Another property of the customized C solvers that is advantageous for real-time code is that they use absolutely *no dynamic memory allocation* and use only the memory necessary for the problem at hand.

Several recent implementations of LP and QP solvers can generate solvers for specific problem structures, but cannot handle SOCP problems (a generalization of QPs) (Wang and Boyd, 2010; Mattingley and Boyd, 2010a; Ferreau et al., 2008). Other implementations can handle second order cones, but use first order methods instead of IPM algorithms at the core of the solver (Ullmann, 2011). The ECOS (Domahidi et al., 2013) SOCP solver is a recently produced, compact, and fast generic SOCP solver that can tackle SOCP problems with IPMs, but does not generate customized code for a particular problem structure.

We have developed a single threaded C++ based solver that also has the ability to generate ANSI-C "customized" solvers. We note that the customized solvers have no library dependencies (other than standard libraries, such as `math` and `stdlib`), which makes the software portable. The specific implementation of the solver shares many similarities with (Domahidi et al., 2013) in that it employs a standard Mehrotra predictor-corrector IPM algorithm with self-dual embedding and makes use of the Approximate Minimum Degree heuristic to increase the efficiency of sparse Cholesky factorizations (Amestoy et al., 2004). Furthermore, Nesterov-Todd scalings (Nesterov and Todd, 1997) are used to condition matrices.

2. SOCP OPTIMIZATION

In this section, we give a brief overview of the solver implementation and the techniques that we use to make the computations efficient. The formulation introduced follows the formulation introduced by (Peng et al., 2001; Wang, 2003) closely.

We give three key definitions in this section and refer readers to (Boyd and Vandenberghe, 2004) for a comprehensive study of convexity. From set theory, we define a linear cone, $\mathcal{K}_L$, as (Boyd and Vandenberghe, 2004):

$$\mathcal{K}_L = \{x : x \geq 0\} = \mathbb{R}_+ \tag{1}$$

where we see that all non-negative real numbers (and vectors of non-negative real numbers) are elements of the linear cone. Furthermore, we can define a second order cone as (Boyd and Vandenberghe, 2004):

$$\mathcal{K}_S = \{\boldsymbol{x} = [x_1, \boldsymbol{x}_{2:n}]^{\mathrm{T}} \in \mathbb{R}^n : x_1^2 \geq \|\boldsymbol{x}_{2:n}\|^2\} \tag{2}$$

for any $n \geq 2$. One can use second order cones to express quadratic inequality constraints and costs. Finally, the dual, $\mathcal{K}^*$, of a set $\mathcal{K}$ is defined as (Boyd and Vandenberghe, 2004):

$$\mathcal{K}^* = \{\boldsymbol{s} : \boldsymbol{x}^{\mathrm{T}}\boldsymbol{s} \geq 0 \quad \forall \boldsymbol{x} \in \mathcal{K}\} \tag{3}$$

It can easily be shown that the linear and second order cones are self-dual (Boyd and Vandenberghe, 2004; Wang, 2003); that is, the dual cone of the linear cone is the linear cone, etc. This property will be useful in the following section.

2.1 SOCP Formulation

The goal of an SOCP solver is to find an optimal solution of the Primal (P) problem, given (in canonical form) by:

$$\begin{aligned} \text{minimize} &: \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \\ \text{subject to} &: \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ &\boldsymbol{x} \in \mathcal{K} \end{aligned} \tag{4}$$

where $\boldsymbol{x}$ is the solution variable, $\boldsymbol{c}$ maps the solution variable to a cost, $\boldsymbol{A}$ relates solution variables to constraint equations with $\boldsymbol{b}$ on the right hand side, and $\mathcal{K}$ is given by:

$$\mathcal{K} = \mathcal{K}_L \times \mathcal{K}_{S_1} \times \ldots \times \mathcal{K}_{S_m} \tag{5}$$

where m second order cones are used to define the domain of the solution variable, which is comprised of $k = m + l$ total cones (l linear cones in addition to m SOCs). We now introduce the Dual (D) problem to bring more information to the formulation. The dual problem is given by:

$$\begin{aligned} \text{maximize} &: \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} \\ \text{subject to} &: \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}, \\ &\boldsymbol{s} \in \mathcal{K}^* \quad \text{where} \quad \mathcal{K}^* = \mathcal{K} \end{aligned} \tag{6}$$

where the goal of the dual problem is to find a $\boldsymbol{y}$ and $\boldsymbol{s}$ that maximize $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y}$ and the last constraint on $\boldsymbol{s}$ can be rewritten as $\boldsymbol{s} \in \mathcal{K}$ since $\mathcal{K}$ is comprised of purely self-dual cones. It is also useful to note the duality gap $\boldsymbol{x}^T\boldsymbol{s} = \boldsymbol{c}^T\boldsymbol{x} - b^T\boldsymbol{y} \geq 0$ for any feasible primal-dual pair $\boldsymbol{x}$, $(\boldsymbol{y}, \boldsymbol{s})$. Also, when a strictly feasible primal-dual pair exists (i.e., a pair that satisfies equality constraints and lie strictly inside the cone $\mathcal{K}$), the duality gap is zero for any optimal primal-dual pair, i.e., $\boldsymbol{x}_*^T\boldsymbol{s}_* = \boldsymbol{c}^T\boldsymbol{x}_* - \boldsymbol{b}^T\boldsymbol{y}_* = 0$ (Peng et al., 2001). Therefore, the duality gap can be used to determine the closeness to the optimal solution.

2.2 Interior Point Methods

In order to solve the Primal and Dual problems together, we use a primal-dual path-following IPM. For detailed

descriptions of IPM algorithms, we refer the reader to (Peng et al., 2001; Wang, 2003; Vandenberghe, 2010). In the following, we give a brief overview of the IPM algorithm we implemented.

Homogenous Self-Embedding There are two major issues with the primal/dual formulation we describe in the previous section. The first is that there is no readily apparent way of determining whether a given optimization problem is feasible. The second issue arises from the fact that to use of a path-following IPM requires an initial guess that is both feasible and in the strict interior of $\mathcal{K}$. For this reason, we lift the problem into the following equivalent self-dual formulation (Peng et al., 2001; Nesterov and Nemirovski, 1994). Given ($\boldsymbol{x}_0$, $\boldsymbol{s}_0$, $\boldsymbol{y}_0$, τ_0, κ_0), solve:

$$\begin{aligned} \text{minimize} : \ & \beta\nu \\ \text{subject to} : \ & \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\tau - \boldsymbol{r}_p\nu = 0, \\ & -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{c}\tau - \boldsymbol{s} - \boldsymbol{r}_d\nu = 0, \\ & \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} - \kappa - r_g\nu = 0, \\ & \boldsymbol{r}_p^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{r}_d^{\mathrm{T}}\boldsymbol{x} + r_g\tau = -\beta, \\ & \boldsymbol{x}, \boldsymbol{s} \in \mathcal{K}, \ \ \tau, \kappa \geq 0 \end{aligned} \tag{7}$$

where $\boldsymbol{y}$ and ν are free, and $\boldsymbol{r}_p$, $\boldsymbol{r}_d$, r_g, β are residuals (Peng et al., 2001; Wang, 2003) defined by:

$$\begin{aligned} \boldsymbol{r}_p &\triangleq \frac{\boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}\tau_0}{\nu_0}, & \boldsymbol{r}_d &\triangleq \frac{-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}_0 + \boldsymbol{c}\tau_0 - \boldsymbol{s}_0}{\nu_0}, \\ r_g &\triangleq \frac{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y}_0 - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}_0 - \kappa_0}{\nu_0}, & \beta &\triangleq -(\boldsymbol{r}_p^{\mathrm{T}}\boldsymbol{y}_0 + \boldsymbol{r}_d^{\mathrm{T}}\boldsymbol{x}_0 + r_g\tau_0) \end{aligned} \tag{8}$$

Using this formulation, a path-following algorithm can be initialized trivially (Peng et al., 2001; Wang, 2003). Any $\boldsymbol{x}_0$ and $\boldsymbol{s}_0$ that are in the interior of $\mathcal{K}$ serve as a strictly feasible (but not necessarily optimal) solution to the problem defined by Equation 7. Moreover, since $\boldsymbol{y}$ is free, $\boldsymbol{y}_0$ can be taken to be a zero vector of appropriate dimensions; finally, τ_0 and κ_0 must be non-negative, and are typically chosen to be 1.

Now that our embedded problem (Equation 7) is initialized with a strictly feasible solution, we look at the feasibility of the Primal and Dual problems. Once the path-following scheme converges, we can look at the values of τ and κ and make claims about feasibility. The following list contains all possible values that τ and κ can converge to, along with their significance (Peng et al., 2001; Wang, 2003; Domahidi et al., 2013; Nesterov and Nemirovski, 1994):

(1) $\tau > 0$, $\kappa = 0$: An optimal solution of the self-embedded problem has been found. $\boldsymbol{x}/\tau$ is the optimal solution of the primal problem.
(2) $\tau = 0$, $\kappa > 0$, and $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} < 0$: Primal is infeasible.
(3) $\tau = 0$, $\kappa > 0$, and $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} < 0$: Dual is infeasible.
(4) $\tau = 0$, $\kappa = 0$: Problem is numerically ill-posed.

Central Path Finding the optimal solution to the homogenous, self-embedded formulation (Equation 7) is equivalent to solving the following system of non-linear, more specifically bilinear, equations for $\mu = 0$ (Peng et al., 2001; Nesterov and Nemirovski, 1994; Wang, 2003):

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\tau - \boldsymbol{r}_p\nu &= 0, \\ -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{c}\tau - \boldsymbol{r}_d\nu &= 0, \\ \boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} - \kappa - r_g\nu &= 0, \\ \boldsymbol{x}^{\mathrm{T}}s &= 0 + \mu, \\ \kappa\tau &= 0 + \mu, \\ \mu &= \mu_0\nu \geq 0. \end{aligned} \tag{9}$$

where μ is a perturbation on the system of equations formed by enforcing the constraints associated with the homogenous, self-embedded formulation in the previous section and μ_0 is given by:

$$\mu_0 = \frac{\boldsymbol{x}_0^{\mathrm{T}}\boldsymbol{s}_0 + \tau_0\kappa_0}{k+1}. \tag{10}$$

The solution to the system of equations in (9) is unique for non-zero values of μ (Peng et al., 2001); therefore for each positive μ, there exists a unique solution to the system that can be found by using numerical techniques (such as Newton's method). Moreover, as $\mu \to 0$ (or equivalently, $\nu \to 0$), the solution to the perturbed problem converges to the solution of the original problem (Equation 7). The solutions, ($\boldsymbol{x}_\mu$, τ_μ, $\boldsymbol{y}_\mu$, $\boldsymbol{s}_\mu$, κ_μ, ν_μ), for any non-negative μ are referred to as the *central path*. The difference between one IPM and another lies primarily in how one tracks the central path towards the unperturbed system solution.

Newton Search Directions To proceed with the solution of the nonlinear equations (9), we need to introduce some terminology. Given a vector, $\boldsymbol{v} = [v_1, v_2, \ldots, v_n]^{\mathrm{T}} \in \mathbb{R}^n$, we define the arrowhead matrix to be:

$$\text{arrow}(\boldsymbol{v}) \triangleq \begin{pmatrix} v_1 & v_{2:n}^{\mathrm{T}} \\ v_{2:n} & \boldsymbol{I}v_1 \end{pmatrix} \tag{11}$$

Using this definition, we can replace the fourth equation in (9) with:

$$\boldsymbol{X}\boldsymbol{S}\boldsymbol{e} = \boldsymbol{0} \tag{12}$$

since it can be shown that $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{s} = 0 \iff \boldsymbol{X}\boldsymbol{S}\boldsymbol{e} = 0$, where $\boldsymbol{e} = [1, 0, \ldots, 0]^{\mathrm{T}}$,

$$\boldsymbol{X} \triangleq \text{blkdiag}(\text{arrow}(\boldsymbol{x}^{(1)}), \ldots, \text{arrow}(\boldsymbol{x}^{(k)})) \tag{13}$$

$$\boldsymbol{S} \triangleq \text{blkdiag}(\text{arrow}(\boldsymbol{s}^{(1)}), \ldots, \text{arrow}(\boldsymbol{s}^{(k)})) \tag{14}$$

and $\boldsymbol{x}^{(i)}$ represents the i^{th} cone in the solution variable. That is, we create a block diagonal matrix with an arrowhead matrix corresponding to each cone in the solution variable. Without loss of generality, one can arrange the solution variables such that the linear cones are all in the beginning, followed by the second order cones. Since the linear cones correspond to positive scalars, the first l elements of the block diagonal matrices reduce to just the diagonal entries.

Given our current estimate of the solution, ($\boldsymbol{x}$, τ, $\boldsymbol{y}$, $\boldsymbol{s}$, κ), we seek to find the solution for the next iteration. In order to do so, we can express the next solution as ($\boldsymbol{x} + \Delta\boldsymbol{x}$, $\tau + \Delta\tau$, $\boldsymbol{y} + \Delta\boldsymbol{y}$, $\boldsymbol{s} + \Delta\boldsymbol{s}$, $\kappa + \Delta\kappa$). Plugging our expression into the system in (9) and substituting Equation 12 into the fourth equality, yields:

$$\begin{aligned} \boldsymbol{A}\Delta\boldsymbol{x} - \boldsymbol{b}\Delta\tau &= \boldsymbol{r}_p\nu - (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\tau), \\ -\boldsymbol{A}^{\mathrm{T}}\Delta\boldsymbol{y} + \boldsymbol{c}\Delta\tau - \Delta\boldsymbol{s} &= \boldsymbol{r}_d\nu - (-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{c}\tau - \boldsymbol{s}), \\ \boldsymbol{b}^{\mathrm{T}}\Delta\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\Delta\boldsymbol{x} - \Delta\kappa &= r_g\nu - (\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} - \kappa), \\ \boldsymbol{X}\Delta\boldsymbol{S}\boldsymbol{e} + \boldsymbol{S}\Delta\boldsymbol{X}\boldsymbol{e} &= \nu\mu_0\boldsymbol{e} - \boldsymbol{E}_{xs}, \\ \kappa\Delta\tau + \tau\Delta\kappa &= \nu\mu_0 - \kappa\tau - E_{\kappa\tau} \end{aligned} \tag{15}$$

where $\boldsymbol{E}_{xs}$ and $E_{\kappa\tau}$ are approximations (that will be given later) to the second order terms, $\Delta\boldsymbol{X}\Delta\boldsymbol{S}\boldsymbol{e}$ and $\Delta\kappa\Delta\tau$, respectively. Also the $\Delta\boldsymbol{X}$ and $\Delta\boldsymbol{S}$ matrices are formed in the same fashion as $\boldsymbol{X}$ and $\boldsymbol{S}$ in Equations 13 and 14. In its current form, the problem cannot be reliably and efficiently solved (Wang, 2003), so we introduce cone scalings that allow for an efficient method of solving the system of equations.

Nesterov-Todd (NT) Scalings Nesterov and Todd introduced a set of symmetric scalings that are inexpensive to compute and make the problem numerically well conditioned (Nesterov and Todd, 1997). An important property of scaling matrices is that they do not change the cone or the central path. Now, consider the i^{th} linear or second order cone, we can define a scaling matrix, $\boldsymbol{G} = \boldsymbol{G}^{\mathrm{T}}$, and positive number, θ, as follows (Nesterov and Todd, 1997):

$$\theta^2 = \sqrt{\frac{\boldsymbol{s}^{(i)\mathrm{T}}\boldsymbol{Q}\boldsymbol{s}^{(i)}}{\boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{Q}\boldsymbol{x}^{(i)}}} \tag{16}$$

where $\boldsymbol{Q} = 1$ for linear cones and

$$\boldsymbol{Q} = \mathrm{diag}(1, -1, \ldots, -1) \tag{17}$$

for second order cones. For linear cones, $\boldsymbol{G} = 1$, and for second order cones,

$$\boldsymbol{G} = -\boldsymbol{Q} + \frac{(\boldsymbol{e}+\boldsymbol{g})(\boldsymbol{e}+\boldsymbol{g})^{\mathrm{T}}}{1+\boldsymbol{e}^{\mathrm{T}}\boldsymbol{g}} \tag{18}$$

where $\boldsymbol{g}$ is defined as:

$$\boldsymbol{g} = \frac{\boldsymbol{s}^{(i)}/\theta + \theta\boldsymbol{Q}\boldsymbol{x}^{(i)}}{\sqrt{2\left(\boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{s}^{(i)} + \sqrt{\boldsymbol{x}^{(i)\mathrm{T}}\boldsymbol{Q}\boldsymbol{x}^{(i)}\boldsymbol{s}^{(i)\mathrm{T}}\boldsymbol{Q}\boldsymbol{s}^{(i)}}\right)}} \tag{19}$$

We can now define the scaled cones, as follows:

$$\bar{\boldsymbol{x}}^{(i)} = \theta\boldsymbol{G}\boldsymbol{x}^{(i)}, \quad \bar{\boldsymbol{s}}^{(i)} = (\theta\boldsymbol{G})^{-1}\boldsymbol{s}^{(i)}. \tag{20}$$

This operation can be repeated for every cone in the solution variable (finding a unique $\boldsymbol{G}$ and θ for each second order cone) and the scaled solution variable can be reconstructed by concatenating all of the cones to form $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{s}}$. Equivalently, a pair of block diagonal matrices can be formed such that:

$$\bar{\boldsymbol{x}} = \boldsymbol{\Theta}\tilde{\boldsymbol{G}}\boldsymbol{x}, \quad \bar{\boldsymbol{s}} = (\boldsymbol{\Theta}\tilde{\boldsymbol{G}})^{-1}\boldsymbol{s}, \tag{21}$$

where

$$\begin{aligned}\boldsymbol{\Theta} &= \mathrm{diag}(\theta^{(1)}, \ldots, \theta^{(k)}),\\ \tilde{\boldsymbol{G}} &= \mathrm{blkdiag}(\boldsymbol{G}^{(1)}, \ldots, \boldsymbol{G}^{(k)}).\end{aligned} \tag{22}$$

We can see that by inverting the relations in Equation 21, the following expressions hold:

$$\boldsymbol{x} = (\boldsymbol{\Theta}\tilde{\boldsymbol{G}})^{-1}\bar{\boldsymbol{x}}, \quad \boldsymbol{s} = \boldsymbol{\Theta}\tilde{\boldsymbol{G}}\bar{\boldsymbol{s}} \tag{23}$$

Plugging Equation 23 into the fourth equality of the system in (15), we get:

$$\begin{aligned}\boldsymbol{A}\Delta\boldsymbol{x} - \boldsymbol{b}\Delta\tau &= \boldsymbol{r}_1,\\ -\boldsymbol{A}^{\mathrm{T}}\Delta\boldsymbol{y} + \boldsymbol{c}\Delta\tau - \boldsymbol{s} &= \boldsymbol{r}_2,\\ \boldsymbol{b}^{\mathrm{T}}\Delta\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\Delta\boldsymbol{x} - \Delta\kappa &= r_3,\\ \bar{\boldsymbol{X}}(\boldsymbol{\Theta}\tilde{\boldsymbol{G}})^{-1}\Delta\bar{\boldsymbol{s}} + \bar{\boldsymbol{S}}\boldsymbol{\Theta}\tilde{\boldsymbol{G}}\Delta\bar{\boldsymbol{x}} &= \boldsymbol{r}_4,\\ \kappa\Delta\tau + \tau\Delta\kappa &= r_5,\end{aligned} \tag{24}$$

where the following convenience variables are defined (as in (Wang, 2003)):

$$\begin{aligned}&\boldsymbol{r}_1 = \boldsymbol{r}_p\nu - (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\tau), \ \boldsymbol{r}_2 = \boldsymbol{r}_d\nu - (-\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} + \boldsymbol{c}\tau - \boldsymbol{s}),\\ &r_3 = r_g\nu - (\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} - \kappa), \ \boldsymbol{r}_4 = \nu\mu_0\boldsymbol{e} - \boldsymbol{E}_{xs},\\ &r_5 = \nu\mu_0 - \kappa\tau - E_{\kappa\tau}.\end{aligned} \tag{25}$$

Given the current solution ($\boldsymbol{x}$, τ, $\boldsymbol{y}$, $\boldsymbol{s}$, κ), $\boldsymbol{E}_{xs}$, and $E_{\kappa\tau}$, one can calculate $\boldsymbol{r}_1$ through r_5. We can begin to solve for individual terms in the system of equations. Looking at the last equality in (24) is easy to see that:

$$\Delta\kappa = \frac{r_5 - \kappa\Delta\tau}{\tau} \tag{26}$$

Further, we can use the fourth equality to obtain:

$$\Delta\boldsymbol{s} = \boldsymbol{\Theta}\tilde{\boldsymbol{G}}(\bar{\boldsymbol{X}})^{-1}\boldsymbol{r}_4 - (\boldsymbol{\Theta}\tilde{\boldsymbol{G}})^2\Delta\boldsymbol{x} \tag{27}$$

Let $\boldsymbol{D} = (\boldsymbol{\Theta}\tilde{\boldsymbol{G}})^{-1}$, then it can be shown that the following relations also hold (Wang, 2003):

$$\Delta\boldsymbol{x} = \boldsymbol{D}^2\left(\boldsymbol{r}_2' + \boldsymbol{A}^{\mathrm{T}}\Delta\boldsymbol{y} - \boldsymbol{c}\Delta\tau\right), \tag{28}$$

$$\Delta\tau = \frac{r_3' + \boldsymbol{a}_1^{\mathrm{T}}\Delta\boldsymbol{y}}{a_2}, \tag{29}$$

$$\boldsymbol{r}_2' = \boldsymbol{r}_2 + \boldsymbol{\Theta}\tilde{\boldsymbol{G}}(\bar{\boldsymbol{X}})^{-1}\boldsymbol{r}_4, \tag{30}$$

$$\begin{aligned}&r_3' = r_3 + \frac{r_5}{\tau} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{D}^2\boldsymbol{r}_2', \ \boldsymbol{a}_1 = -\boldsymbol{b} + \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{c},\\ &a_2 = \frac{\kappa}{\tau} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{D}^2\boldsymbol{c}.\end{aligned}$$

Finally, the system of equations boils down to solving the following linear equation:

$$\left(\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^{\mathrm{T}} + \bar{\boldsymbol{a}}\boldsymbol{a}_1^{\mathrm{T}}\right)\Delta\boldsymbol{y} = \boldsymbol{\xi} \tag{31}$$

where

$$\bar{\boldsymbol{a}} = \frac{-\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{c} - \boldsymbol{b}}{a_2}, \ \boldsymbol{\xi} = \boldsymbol{r}_1' + \frac{r_3'}{a_2}(\boldsymbol{A}\boldsymbol{D}^2\boldsymbol{c} + \boldsymbol{b}), \ \boldsymbol{r}_1' = \boldsymbol{r}_1 - \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{r}_2'.$$

The system in Equation 31 can be solved efficiently by making use of the Sherman-Morrison formula (Hager, 1989; Wang, 2003). Let $\hat{\boldsymbol{P}} = \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^{\mathrm{T}}$, then it can be shown that the solution to (31) can be found by solving the following 2 linear systems for $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$:

$$\hat{\boldsymbol{P}}\boldsymbol{v}_0 = \boldsymbol{\xi}, \tag{32}$$

$$\hat{\boldsymbol{P}}\boldsymbol{v}_1 = \bar{\boldsymbol{a}} \tag{33}$$

and plugging the results into:

$$\Delta\boldsymbol{y} = \boldsymbol{v}_0 - \frac{\boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{v}_0}{1 + \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{v}_1}\boldsymbol{v}_1 \tag{34}$$

We note that $\hat{\boldsymbol{P}}$ is a symmetric, positive definite matrix due to the Nesterov-Todd scalings that we make use of. Furthermore, since both Equation 32 and 33 have the same coefficient matrix, one Cholesky factorization of $\hat{\boldsymbol{P}}$ is sufficient to solve both linear systems. This offers a significant advantage since the Cholesky factorization is the most computationally expensive part of solving a linear system. In practice, $\hat{\boldsymbol{P}}$ is often a sparse matrix, so sparse techniques can be employed to further reduce solver runtimes. Once $\Delta\boldsymbol{y}$ has been found, it can be plugged into the expressions for the other variables to produce Newton directions. However, we still have not discussed a method for obtaining $\boldsymbol{E}_{xs}$ and $E_{\kappa\tau}$.

Mehrotra Predictor-Corrector The purpose of using a predictor-corrector scheme is to deal with the second order terms that arise from our non-linear system of equations and update the centering term, ν. The Mehrotra predictor-corrector has been shown to have excellent convergence

properties in practice while still being inexpensive to compute. The procedure can be broken down into 2 steps: prediction, and then correction. During the prediction step, $\boldsymbol{E}_{xs}$ and $E_{\kappa\tau}$ are taken to be zero; these values are then used to solve the system of equations in (24). At this point, a scaling term $0 < \alpha \leq 1$ is computed such that the next solution does not deviate far from the central path. There are many strategies for selecting this scaling term, but we employ the maximum Newton step size method given in (Wang, 2003). After the predicted search direction is obtained, we can compute the predicted complementarity gap as:

$$g_p = (\boldsymbol{x}+\alpha\Delta\boldsymbol{x}_p)^{\mathrm{T}}(\boldsymbol{s}+\alpha\Delta\boldsymbol{s}_p)+(\kappa+\Delta\kappa_p)(\tau+\Delta\tau_p), \quad (35)$$

and use it to find the new centering parameter, ν:

$$\nu = \left(\frac{g_p}{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{s}+\kappa\tau}\right)^2 \frac{g_p}{k+1} \quad (36)$$

Moreover, we can use the predicted Newton directions to approximate the second order terms:

$$\boldsymbol{E}_{xs} = \Delta\tilde{\boldsymbol{X}}_p\Delta\tilde{\boldsymbol{S}}_p\boldsymbol{e}, \quad (37)$$

$$E_{\kappa\tau} = \Delta\kappa_p\Delta\tau_p, \quad (38)$$

where

$$\Delta\tilde{\boldsymbol{X}}_p = \mathrm{blkdiag}(\mathrm{arrow}(\Delta\tilde{\boldsymbol{x}}_p^{(1)}), \ldots, \mathrm{arrow}(\Delta\tilde{\boldsymbol{x}}_p^{(k)})),$$

$$\Delta\tilde{\boldsymbol{S}}_p = \mathrm{blkdiag}(\mathrm{arrow}(\Delta\tilde{\boldsymbol{s}}_p^{(1)}), \ldots, \mathrm{arrow}(\Delta\tilde{\boldsymbol{s}}_p^{(k)})),$$

$$\Delta\tilde{\boldsymbol{x}}_p = \boldsymbol{\Theta}\tilde{\boldsymbol{G}}\Delta\boldsymbol{x}_p, \quad \Delta\tilde{\boldsymbol{s}}_p = (\boldsymbol{\Theta}\tilde{\boldsymbol{G}})^{-1}\Delta\boldsymbol{s}_p.$$

The second order approximations and centering parameter can be used to solve the system in (24) again. However, since the coefficient matrix is the same for both the predictor and corrector, one can reuse the factorized matrix from the prediction step.

2.3 Algorithm Overview

We now provide a summary of our IPM algorithm in order to bring together the core solver techniques that we have implemented.

Data: a tolerance, ϵ, an initial point $(\boldsymbol{x}_0, \boldsymbol{s}_0, \boldsymbol{y}_0, \tau_0, \kappa_0)$ as described above, and the problem structure given as: $\boldsymbol{A}$, $\boldsymbol{b}$, and $\boldsymbol{c}$.
Result: an optimal solution, $(\boldsymbol{x}^*, \boldsymbol{s}^*, \boldsymbol{y}^*, \tau^*, \kappa^*)$
begin
  $\boldsymbol{x} = \boldsymbol{x}_0$, $\tau = \tau_0$, $\boldsymbol{y} = \boldsymbol{y}_0$, $\boldsymbol{s} = \boldsymbol{s}_0$, $\kappa = \kappa_0$;
  while $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{s} + \kappa\tau > \epsilon$ **do**
    predict: solve (24) with $\nu = 0$, $\boldsymbol{E}_{xs} = \boldsymbol{0}$, $E_{\kappa\tau} = 0$;
    calculate Newton step size, α;
    update ν, Eq. (36);
    correct: solve (24) with $\boldsymbol{E}_{xs}$, $E_{\kappa\tau}$ in Eqs. (37, 38);
    calculate Newton step size, α;
    $\boldsymbol{x} = \boldsymbol{x} + \alpha\Delta\boldsymbol{x}$, $\tau = \tau + \alpha\Delta\tau$;
    $\boldsymbol{y} = \boldsymbol{y} + \alpha\Delta\boldsymbol{y}$, $\boldsymbol{s} = \boldsymbol{s} + \alpha\Delta\boldsymbol{s}$;
    $\kappa = \kappa + \alpha\Delta\kappa$;
  if $\tau > 0$ **then**
    $\boldsymbol{x}^* = \frac{\boldsymbol{x}}{\tau}$;
    $\boldsymbol{s}^* = \frac{\boldsymbol{s}}{\tau}$;
    $\boldsymbol{y}^* = \frac{\boldsymbol{y}}{\tau}$;
    $\tau^* = \tau$;
    $\kappa^* = \kappa$;
  else if $\kappa = 0$ **then** Problem is ill posed;
  else if $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} < 0$ **then** Dual is infeasible;
  else if $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{y} > 0$ **then** Primal is infeasible;

The above algorithm represents an overview of the optimization techniques we used to solve general SOCP problems, but does not delve into the customization aspect of our implementation.

3. CODE CUSTOMIZATION

This section presents the methods we developed to generate efficient, customized C code for specific SOCP problem classes and discusses the benefits of customized code, which make up the bulk of our contribution. Embedded systems typically operate in environments with stringent real-time requirements and limited memory. For this reason, it is important that the solver not allocate more memory than it needs and that it not perform any operations unnecessarily. Moreover, some autonomous systems rely on the results of the optimization in order to safely perform their task, so it is important that logical branches are avoided when possible, and are well-formulated otherwise. We make use of "explicit coding;" where we automatically generate code that performs certain specific, otherwise computationally expensive tasks (mostly, sparse linear algebra). To this end, we have implemented a generic IPM algorithm for SOCPs in C++ that also has the capability to generate a specialized ANSI-C solver tailored to the problem it was given. In the following, we provide details regarding our implementation of both the generic and customized IPM solvers.

3.1 Sparsity

Many problems that naturally occur in engineering tend to translate into sparse optimization problems, particularly discrete controls problems (Açıkmeşe and Blackmore, 2011). Therefore, it is advantageous to make use of this sparsity by avoiding computations that add or multiply zeros. As we discussed in the previous section, the optimization problem boils down to solving a pair of linear systems. Since $\hat{\boldsymbol{P}}$ is a symmetric, positive definite matrix, a Cholesky factorization of the coefficient matrix in (32) is performed to obtain:

$$\hat{\boldsymbol{P}} = \hat{\boldsymbol{L}}\hat{\boldsymbol{L}}^{\mathrm{T}} \quad (39)$$

where $\hat{\boldsymbol{L}}$ is a lower triangular matrix. The number of non-zeros present in $\hat{\boldsymbol{L}}$ depends on the sparsity pattern of the $\hat{\boldsymbol{P}}$ matrix (the locations of the non-zero entries). Therefore, a permutation matrix, $\boldsymbol{R}$, can be found that seeks to minimize the number of non-zero entries in $\hat{\boldsymbol{L}}$ by transforming $\hat{\boldsymbol{P}}$:

$$\boldsymbol{R}\hat{\boldsymbol{P}}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{L}\boldsymbol{L}^{\mathrm{T}} \quad (40)$$

where $\boldsymbol{L}$ is a lower triangular matrix with a minimal number of non-zero entries. We use the Approximate Minimum Degree method (Amestoy et al., 2004) to obtain an $\boldsymbol{R}$ that is both inexpensive to compute and very good at reducing the fill-in ratio of the Cholesky factorization. Substituting Equations 39 and 40 into Equation 32 gives:

$$\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{v}_0 = \boldsymbol{R}\boldsymbol{\xi}. \quad (41)$$

where $\boldsymbol{R}\boldsymbol{R}^{\mathrm{T}} = \boldsymbol{I}$, and the Cholesky factorization and corresponding forward/back substitutions have the least

number of non-zero entries to process. A similar transformation can be done to the linear equation found in (33) to obtain its final form:

$$\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{v}_1 = \boldsymbol{R}\bar{\boldsymbol{a}}. \tag{42}$$

where $\boldsymbol{L}$ and $\boldsymbol{R}$ are the same as in Equation 41.

With a methodology for solving sparse linear systems established, we turn our attention to other sparse operations. In general, sparse linear algebra operations (like sparse matrix-matrix multiplication) reduce the total number of elementary operations, but increase the cost of each operation; in other words, we reduce operations by avoiding the addition and multiplication of zeros, but must now determine how the remaining non-zero terms interact with each other. In the next section, we discuss our explicit coding scheme and how it can mitigate these issues.

3.2 Memory Allocation and Explicit Coding

Much of the computational effort associated with sparse linear algebra stems from not knowing the problem structure beforehand, and therefore having to dynamically determine which non-zero elements interact every time an operation is carried out. However, we observe that a given embedded system typically only solves a single class of problems, and that the problems' structures themselves remain uniform; that is, the locations of the non-zero elements in $\boldsymbol{A}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ do not change. In order to take advantage of this, we define a problem class; given a problem defined by $\boldsymbol{A}_0$, $\boldsymbol{b}_0$, $\boldsymbol{c}_0$, and $\mathcal{K}_0$, we define a problem class, $\mathcal{P}$, as:

$$\begin{aligned}\mathcal{P} = \{\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, \mathcal{K} : \mathrm{str}(\boldsymbol{A}) \leq \mathrm{str}(\boldsymbol{A}_0), \mathrm{str}(\boldsymbol{b}) \leq \mathrm{str}(\boldsymbol{b}_0),\\ \mathrm{str}(\boldsymbol{c}) \leq \mathrm{str}(\boldsymbol{c}_0), \mathcal{K} = \mathcal{K}_0\}\end{aligned} \tag{43}$$

where the $\leq$ operator denotes element-wise inequality, and str maps any non-zero element to a 1 and leaves 0 elements undisturbed (thereby forming the sparsity structure of its input). Clearly, $\boldsymbol{A}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ must have the same dimensions as $\boldsymbol{A}_0$, $\boldsymbol{b}_0$, and $\boldsymbol{c}_0$ respectively. In this framework, any $\boldsymbol{A}$ that is more sparse than $\boldsymbol{A}_0$ can still be in $\mathcal{P}$, so long as any zero element in $\boldsymbol{A}_0$ is also a zero element in $\boldsymbol{A}$ (similarly for $\boldsymbol{b}$ and $\boldsymbol{c}$). Now, assume that a given embedded system solves a sequence of problems $\mathcal{P}_i \in \mathcal{P}$, $i = 1, \ldots, N$ (which we have observed is not a bad assumption). Then, the problem class, and therefore an upper bound on the sparsity structure, is known and implementing a generic solver on an embedded device is wasteful. We have developed software that takes advantage of this knowledge by keeping track of non-zero element interactions and using this to generate code that is free of logical operations (which are introduced by using sparse algorithms). That is, once the impact of the interaction between two non-zero elements has been determined, one line of C code can be generated that correctly handles the interaction without any logic. As a consequence, an added benefit of our explicit code generation is that it reduces branching. Moreover, since the problem size is known, we can statically allocate exactly the amount of memory that is necessary to solve a specific problem class.

We note that the permutation matrix, $\boldsymbol{R}$, found above depends only on the sparsity structure of $\hat{\boldsymbol{P}}$. We recall that $\hat{\boldsymbol{P}} = \boldsymbol{A}\boldsymbol{D}^2\boldsymbol{A}^{\mathrm{T}}$, and observe that the sparsity structure of $\boldsymbol{A}$ can be bounded from above by some $\boldsymbol{A}_0$ assuming that we are solving a problem that is in $\mathcal{P}$. We further observe that $\boldsymbol{D}$ is constructed by forming a block diagonal matrix out of a series of arrowhead matrices corresponding to the solution variable; therefore, the sparsity structure of $\boldsymbol{D}$ relies solely on the sizes of the cones that are used in the solution variable. Since all problems in a problem class have the same cone definition ($\mathcal{K}$) this is invariant. Therefore, the sparsity structure of $\hat{\boldsymbol{P}}$ is constant for a given problem class, and the permutation matrix can be found when the custom solver is generated and hard coded into the solver. The permutation can be further optimized by avoiding the matrix multiplications ($\boldsymbol{R}\hat{\boldsymbol{P}}\boldsymbol{R}^{\mathrm{T}}$) and treating it as an element-wise mapping, where the mapping is determined when the custom IPM solver is auto-generated.

As problem size increases, the amount of C code that is generated grows rapidly. Therefore, the sheer amount of machine instructions eventually outweighs the algorithmic advantages of avoiding sparse logic because of instruction cache misses. For this reason, customization is best suited for small to medium problem sizes, as we see from the results in the next section.

4. RESULTS

In this section, we briefly discuss two optimization problems and give computational timing results for each problem using different solvers. Every problem class is run 500 times, and the average runtime on a workstation with an Intel Core i7 (3.4 GHz) processor and 15 GB of RAM is presented. The first problem is the planetary soft landing problem (Açikmeşe et al., 2013), where we compute a fuel-and-time-optimal landing trajectory. The second is a reaction wheel allocation problem, where we use optimization to obtain the spin rates that best match a desired torque while minimizing the power usage.

4.1 Planetary Soft Landing

Quickly finding an optimal trajectory onboard a lander vehicle is crucial. In this case, a lander vehicle uses thrusters to slow its descent and ultimately carry out a soft landing at a prescribed target (Açikmeşe et al., 2013). At the start of the maneuver, the vehicle obtains its relative position and velocity vectors with respect to the target, and uses these states to compute a fuel-and-time-optimal trajectory to the target. The longer it takes to compute the landing trajectory, the less accurate the initial conditions become - making this an ideal, time-critical application for our customized solver. A methodology for posing the planetary soft landing problem as a convex optimization problem, specifically as SOCPs, with velocity upper bounds, time-varying mass, minimum and maximum thrust magnitude constraints, glide slope angle bounds, and pointing constraints is given in (Açikmeşe and Ploen, 2005; Açıkmeşe and Ploen, 2007; Açikmeşe et al., 2013), and adapted herein.

We use SDPT3, (Tutuncu et al., 2002)), SeDuMi ((Sturm, 1999)), ECOS ((Domahidi et al., 2013), our generic solver (named Bsocp), and customized solvers over a range of problem sizes to find optimal solutions of the planetary soft landing problem described above. We increased the

problem size by increasing the number of points in the discretization of the dynamics (Açıkmeşe and Ploen, 2007). For small problems, our customized solver was able to solve problems about two times faster than the general Bsocp solver (Figure 1). Problems with about 2,000 solution variables gain almost no advantage by customization, as expected. ECOS and Bsocp performed similarly for small to medium problems, with each taking turns outperforming the other over different sub-regions.

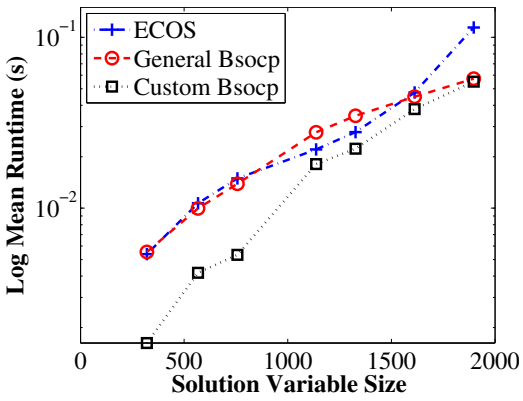

Fig. 1. Planetary Landing Benchmark

For problems with more solution variables than that of the simulations shown in Figure 1, instruction cache misses dominate any gains from customization. For this reason, we show benchmarking results for larger problems using the other solvers. As one can see, ECOS scales better than the general Bsocp solver for problems with more than about 10,000 solution variables. The linear algebra libraries that were written for Bsocp were not intended to be used on larger problems, therefore this result did not surprise us. Nonetheless, Bsocp outperforms SDPT3 and SeDuMi for all problem sizes in the benchmark.

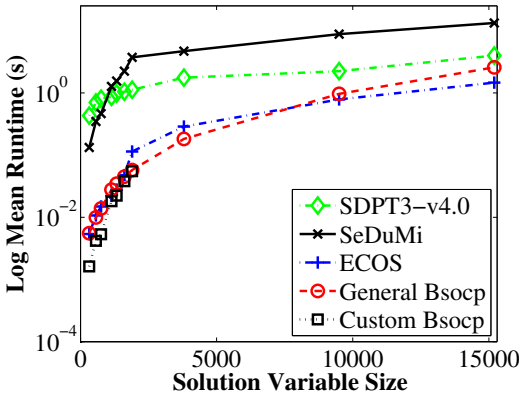

Fig. 2. Planetary Landing Benchmark 2

In Table 1, we present details regarding the types of cones that were used in the simulations for Figures 1 and 2. As we can see, a large amount of second order cones are solved for the bigger problem sizes.

n	# $\mathcal{K}_L$	# $\mathcal{K}_s$
320	153	49
567	270	88
757	360	118
1,137	540	178
1,327	630	208
1,612	765	253
1,897	900	298
3,797	1,800	598
9,497	4,500	1,498
15,197	7,200	2,398

Table 1: Benchmark Details

where n represents the solution variable size, the middle column corresponds to the number of linear cones in the solution variable, and the last column represents the number of second order cones in the solution variable.

4.2 Reaction Wheel Allocation

Spacecraft that make use of reaction wheels to control their attitude dynamics must have an algorithm that maps a given desired control torque to their corresponding flywheel rates. Typically, some variation of the Moore-Penrose pseudo-inverse is used to obtain a minimum torque error mapping; however, these algorithms do not take into consideration the physical constraints of the flywheels, such as upper bounds on flywheel rates and accelerations (Dueri et al., 2014). A methodology for posing this mapping problem as a constrained convex optimization problem that seeks to minimize power usage is given in (Dueri et al., 2014). Here we will explore how quickly these problems can be solved by using custom IPM solution algorithms.

Since this problem formulation leads to LPs and QPs, we were able to also compare our customized code with CVXGEN (Mattingley and Boyd, 2010a). We note that the timings presented here correspond to a system with 4 reaction wheels, making it minimally redundant.

	ECOS	CVXGEN	Cust. Bsocp
CPU-time (msec)	0.5	0.2	0.2

Table 2: CPU Times Using Different IPM Solvers

We can see from Table 2 that the customized Bsocp solver performs as well as CVXGEN for linear optimization problems, even though it is implementing a more general solver. We also see that for tiny problems like this one (with only 40 solution variables), the customization process yields a roughly 2.5 times improvement in runtime efficiency.

5. CONCLUSION

In this paper, we have presented the algorithms and techniques that were used to implement an SOCP solver capable of generating custom code for given problem structures. The core solver algorithm makes use of a fairly standard Mehrotra predictor-corrector IPM with NT cone scalings and scaled Newton search directions. The customized code is generated by creating explicit code for the otherwise cumbersome sparse linear algebra operations that are needed to compute solutions. Timings from both the customized IPM and the general IPM were compared to ECOS, SDPT3, SeDuMi, and CVXGEN (when possible). We found that for small to medium problems, solvers customized for a problem class out-

performed the competition; however, for problems with a large amount of solution variables (2,000 or more), instruction cache misses and prohibitively time consuming compilation times plagued the customized solvers due to source code length.

Future work will involve finding ways to make the linear algebra libraries that are used by Bsocp more scalable and to continue to find ways of implementing more efficient IPM algorithms. We are also currently exploring ways of reducing the compilation time of customized code.

REFERENCES

Açıkmeşe, B., Aung, M., Casoliva, J., Mohan, S., Johnson, A., Scharf, D., Masten, D., Scotkin, J., Wolf, A., and Regehr, M.W. (2013). Flight testing of trajectories computed by G-FOLD: Fuel optimal large divert guidance algorithm for planetary landing. In *AAS/AIAA Spaceflight Mechanics Meeting*.

Açıkmeşe, B. and Blackmore, L. (2011). Lossless convexification for a class of optimal control problems with nonconvex control constraints. *Automatica*, 47(2), 341–347.

Açıkmeşe, B. and Ploen, S.R. (2007). Convex programming approach to powered descent guidance for mars landing. *AIAA Journal of Guidance, Control and Dynamics*, 30(5), 1353–1366.

Açikmeşe, B., Carson, J.M., and Blackmore, L. (2013). Lossless convexification of non-convex control bound and pointing constraints of the soft landing optimal control problem. *IEEE Transactions on Control Systems Technology*.

Açikmeşe, B. and Ploen, S. (2005). A powered descent guidance algorithm for Mars pinpoint landing. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. San Francisco, USA.

Amestoy, P.R., Davis, T.A., and Duff, I.S. (2004). Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3), 381–388. doi:10.1145/1024074.1024081. URL `http://doi.acm.org/10.1145/1024074.1024081`.

Aung, M., Açıkmeşe, B., Johnson, A., Regehr, M., Casoliva, J., Mohan, S., Wolf, A., Masten, D., and Scotkin, J. (2013). ADAPT a closed-loop testbed for next-generation EDL GN&C systems. In *23rd AAS/AIAA 2013 GN&C Conference*. Breckenridge, USA.

Bersekas, D. (1999). *Nonlinear Programming*. Athena Scientific, 2 edition.

Blackmore, L., Açıkmeşe, B., and Carson, J.M. (2012). Lossless convexication of control constraints for a class of nonlinear optimal control problems. *Systems and Control Letters*, 61, 863–871.

Blackmore, L., Açikmeşe, B., and Scharf, D.P. (2010). Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization. *Journal of Guidance, Control, and Dynamics*, 33(4), 1161–1171. doi:10.2514/1.47202.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Domahidi, A., Chu, E., and Boyd, S. (2013). ECOS: An SOCP solver for Embedded Systems. In *Proceedings European Control Conference*.

Dueri, D., Leve, F., and Açıkmeşe, B. (2014). Reaction wheel dissipative power reduction control allocation via lexicographic optimization. In *In review, American Astronautical Society*.

Ferreau, H., Bock, H., and Diehl, M. (2008). An online active set strategy to overcome the limitations of explicit MPC. *Int. Journal of Robust and Nonlinear Control*, 18.

Hager, W.W. (1989). Updating the inverse of a matrix. *SIAM*, 31(2), 221–239.

Harris, M.W. and Açikmeşe, B. (2013). Lossless convexification of non-convex optimal control problems for state constrained linear systems. *Under review, Automatica*.

Jones, C., Bari, M., and Morari, M. (2007). Multiparametric linear programming with applications to control. *European Journal of Control*, 13(2-3), 152 – 170.

JPL and Systems, M.S. (2012). 650 meter divert Xombie test flight for G-FOLD, Guidance for Fuel Optimal Large Divert, validation. `http://www.youtube.com/watch?v=WU4TZlA3jsg`.

JPL, Systems, M.S., and of Texas, U. (2013). First flight testing of real-time G-FOLD, Guidance for Fuel Optimal Large Divert, validation. `http://www.jpl.nasa.gov/news/news.php?release=2013-247`.

Mattingley, J. and Boyd, S. (2010a). *Automatic Code Generation for Real-Time Convex Optimization. Convex Optimization in Signal Processing and Communications, Y. Eldar and D. Palomar, Eds.* Cambridge University Press.

Mattingley, J. and Boyd, S. (2010b). Real-time convex optimization in signal processing. *IEEE Signal Processing Magazine*, 27(3), 50–61.

Nesterov, Y. and Nemirovski, A. (1994). *Interior Point Polynomial Algorithms in Convex Programming*. SIAM.

Nesterov, Y.E. and Todd, M.J. (1997). Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1), 1–42.

Peng, J., Roos, C., and Terlaky, T. (2001). *Self-Regularity: A New Paradigm for Primal-Dual Interior-Point Algorithms*. Princeton Series in Applied Mathematics.

Scharf, D.P., Regehr, M.W., Dueri, D., Açıkmeşe, B., Vaughan, G.M., Benito, J., Ansari, H., M. Aung, A.J., Masten, D., Nietfeld, S., Casoliva, J., and Mohan, S. (2014). ADAPT demonstrations of onboard large-divert guidance with a reusable launch vehicle. *Submitted to IEEE Aerospace Conference*.

Sturm, J. (1999). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12, 625–653. Version 1.05 available from `http://fewcal.kub.nl/sturm`.

Tutuncu, R.H., Toh, K.C., and Todd, M.J. (2002). Solving semidefinite-quadratic-linear problems using SPDT3. *Mathematical Programming*.

Ullmann, F. (2011). *FiOrdOs: A Matlab Toolbox for C-Code Generation for First Order Methods*. Master's thesis, ETH Zurich.

Vandenberghe, L. (2010). The CVXOPT linear and quadratic cone program solvers. URL `http://www.seas.ucla.edu/~vandenbe`.

Wang, B. (2003). *Implementation of Interior Point Methods for Second order Conic Optimization*. Master's thesis, McMaster University.

Wang, Y. and Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, 18(2), 267–278.