Task-Space Trajectories via Cubic Spline Optimization

J. Zico Kolter and Andrew Y. Ng
Computer Science Department, Stanford University, Stanford, CA 94305
{kolter,ang}@cs.stanford.edu

***Abstract*— We consider the task of planning smooth trajectories for robot motion. In this paper we make two contributions. First we present a method for cubic spline optimization; this technique lets us simultaneously plan optimal task-space trajectories and fit cubic splines to the trajectories, while obeying many of the same constraints imposed by a typical motion planning algorithm. The method uses convex optimization techniques, and is therefore very fast and suitable for real-time re-planning and control. Second, we apply this approach to the tasks of planning foot and body trajectory for a quadruped robot, the "LittleDog," and show that the proposed approach improves over previous work on this robot.**

I. Introduction

In this paper we consider the task of planning smooth trajectories for robot motion. This is one of the fundamental tasks of robotics, and has received a great deal of attention over the past several decades. One strategy that has proven particularly effective for this task is the use of smooth, parametrized splines to describe trajectories, either in joint-space or task-space. Cubic splines in particular are ubiquitous in robotic applications [1], as they provide a simple means of generating smooth (twice differentiable) trajectories for robot motion.

Cubic splines for robot trajectories are typically employed as follows. First, one uses a high-level planning algorithm to generate a series of kinematically feasibly *waypoints* that the robot should pass through on its way to the goal. Next, one fits the parameters of a cubic spline that passes through all these points; the smoothness of the resulting cubic spline leads to a smoother motion of the robot than would be obtained, for example, by a linear spline that just interpolated between the waypoints. We refer to this as the "two-phase" approach, since the planning and spline fitting are done in separate phases.

However, despite their advantages, cubic splines also suffer from a number of drawbacks. The chief problem is that in the typical two-phase application of cubic splines, the high-level waypoint planning is done separate from the cubic spline fitting procedure, which can lead to poor trajectories. To convey this intuition, consider the simple planning task shown in Figure 1 (a): the objective is to move a double pendulum, actuated at both joints, from the start to the goal, while avoiding the obstacle. Figure 1 (b) shows a possible output from a typical planner (for example, a randomized tree planner [2]) and the corresponding cubic spline fit to these waypoints. Due to the stochastic nature of the planner, the waypoints do not lead to a particularly nice final trajectory. Existing trajectory optimization techniques [3] can help mitigate this problem to some degree, but they usually involve a slow search process, and still typically do not take into account the final cubic spline form of the trajectory.

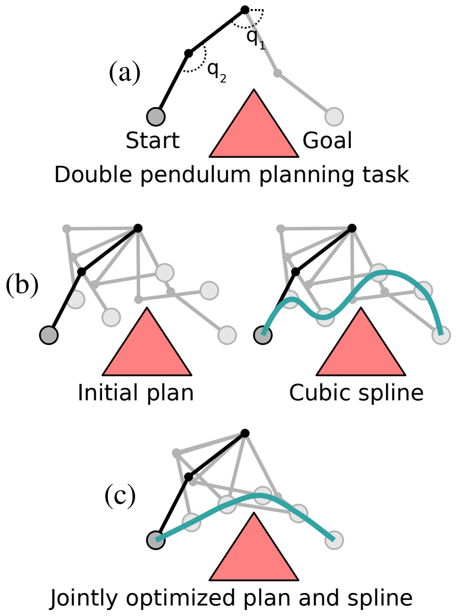
(a)
q_1
q_2
Start
Goal
Double pendulum planning task
(b)
Initial plan
Cubic spline
(c)
Jointly optimized plan and spline

Fig. 1. Simple planning task used to demonstrate the potential advantage of cubic spline optimization. See text for details.

The basic insight of the method we present in this paper is that if we initially parametrize the trajectory as a cubic spline, then in many cases we can accomplish both the planning and trajectory fitting simultaneously. That is, we can directly optimize the location of the cubic spline waypoints while obeying many of the same constraints (or approximations thereof) required by a typical planning algorithm. Specifically, in this paper we show how to plan smooth task-space trajectories — that is, trajectories where we care primarily about the position of the robot's end effector — while maintaining kinematic feasibility, avoiding collision, and limiting velocities or accelerations, all via a convex optimization problem. Convex optimization problems are beneficial in that they allow for efficiently finding global optimums [4] — this allows us to solve the planning tasks in a few milliseconds using off-the-shelf software, suitable for real-time re-planning and control. This approach is illustrated in Figure 1 (c). Because the waypoints and cubic spline and optimized simultaneously, the resulting trajectories are typically much smoother than those obtained by the two-phase approach.

We implement this proposed algorithm on a quadruped, the "Little Dog" robot, to demonstrate its usefulness; indeed, the task of planning foot and body trajectories for this robot originally motivated our approach. The cubic spline optimization approach that we present in this paper is a crucial element of our complete system on this robot, and as we demonstrate, the method substantially improves the quality of trajectories.

The remainder of this paper is organized as follows. In Section II we review the standard means of fitting cubic splines to a set of waypoints. Section III contains the chief algorithmic contribution of the paper: here we present our method for optimizing cubic spline trajectories using convex programming. In Section IV we discuss the application of this algorithm to a quadruped robot, and present empirical evaluations. Finally, in Section V we discuss related work, and conclude the paper in Section VI.

II. Cubic Splines

Here we review the standard methods for fitting cubic splines to a series of waypoints output by a planner. We suppose that we have access to a high-level planner that plans a feasible path from a start location to a goal location. This path is represented as a series of $T+1$ desired time-location pairs:

$$(t_0, x_0^\star), (t_1, x_1^\star), \ldots, (t_T, x_T^\star) \tag{1}$$

where $x_i^\star \in \mathbb{R}^n$ denotes the desired location of the robot at time $t_i \in \mathbb{R}$, specified in task space.

Given these waypoints, there is a unique piecewise-cubic trajectory that passes through the points and satisfies certain smoothness criteria. Specifically, we model the trajectory between times t_i and t_{i+1}, denoted $x_i(t) : \mathbb{R} \rightarrow \mathbb{R}^n$, as a cubic function

$$x_i(t) = a_i + b_i(t-t_i) + c_i(t-t_i)^2 + d_i(t-t_i)^3 \tag{2}$$

where $a_i, b_i, c_i, d_i \in \mathbb{R}^n$ are parameters of the cubic spline. The final trajectory $x(t) : \mathbb{R} \rightarrow \mathbb{R}^n$ is a piecewise-cubic function that is simply the concatenation of these different cubic trajectories

$$x(t) = \begin{cases} x_0(t) & \text{if } t_0 \leq t < t_1 \\ \vdots & \\ x_{T-1}(t) & \text{if } t_{T-1} \leq t \leq t_T \end{cases} \tag{3}$$

where we can assume that the trajectory is undefined for $t < t_0$ and $t > t_T$.

Given the desired waypoints, there exists a unique set of coefficients $\{a_i, b_i, c_i, d_i\}_{i=0,\ldots,T-1}$ such that the resulting trajectory passes through the waypoints and has continuous velocity and acceleration profiles at each waypoint.[1] To compute these coefficients, we first define the matrices $\mathbf{x}, \mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d} \in \mathbb{R}^{T+1 \times n}$

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} x_0^\star & x_1^\star & \cdots & x_T^\star \end{bmatrix}^T & (4) \\ \mathbf{a} &= \begin{bmatrix} a_0 & a_1 & \cdots & a_T \end{bmatrix}^T & (5) \end{aligned}$$

with $\mathbf{b}$, $\mathbf{c}$, and $\mathbf{d}$ defined similarly (we define $T+1$ sets of parameters in order to simplify the equations, though we will ultimately only use the $0, \ldots T-1$ parameters, as described above). Given the $\mathbf{x}$ matrix, we can find the parameters of the cubic splines using the following set of linear equations

$$\begin{aligned} \mathbf{a} &= \mathbf{x} & (6) \\ \mathbf{H}_1\mathbf{b} &= \mathbf{H}_2\mathbf{x} & (7) \\ \mathbf{c} &= \mathbf{H}_3\mathbf{x} + \mathbf{H}_4\mathbf{b} & (8) \\ \mathbf{d} &= \mathbf{H}_5\mathbf{x} + \mathbf{H}_6\mathbf{b} & (9) \end{aligned}$$

where the $\mathbf{H}_i \in \mathbb{R}^{(T+1)\times(T+1)}$ matrices depend only (non-linearly) on the times $t_0, \ldots, t_T$. We present the complete definition of these matrices, as well as a derivation of the equations, in Appendix A. However, the important point to gleam from this presentation is that the the parameters of the cubic splines are *linear* in the desired locations $\mathbf{x}$.

III. Cubic Spline Optimization

In this section we present our primarly algorithmic contribution: a method for optimizing task-space cubic spline trajectories using convex programming. As before, we assume that we are given an initial plan, now denoted

$$(t_0, \hat{x}_0), (t_1, \hat{x}_1), \ldots, (t_T, \hat{x}_T). \tag{10}$$

However, unlike the previous section, we will not require that our final cubic trajectory pass through these points. Indeed, most of the real planning is performed by the optimization problem itself, and the initial plan is required only for some of the approximate constraints that we will discuss shortly; an initial "plan" could simply be a straight line from the start location to the goal location.

The task of optimizing the location of the waypoints while obeying certain constraints can be written formally as

$$\begin{array}{ll} \displaystyle\min_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in \mathcal{C} \end{array}$$

where $\mathbf{x}$ is the optimization variable, representing the location of the waypoints, $f : \mathbb{R}^{(T+1)\times n} \rightarrow \mathbb{R}$ is the optimization objective, and $\mathcal{C}$ represent the set of constraints on the waypoints. In the subsequent sections, we discuss several possible constraints and objectives that we use in order to ensure that the resulting trajectories are both feasible and smooth. The following is not meant to be an exhaustive list, but conveys a general idea of what can be accomplished in this framework. We will present more concrete examples of such optimization problems when we discuss the quadruped robot in Section IV.

A. Additional Variables and Constraints

Spline derivatives at the waypoints. Often times we want objective and constraint terms that contain not only the position of the waypoints, but also the velocity, acceleration, and/or jerk (derivative of acceleration) of the resulting cubic spline. Using (6) – (9), these terms are *linear* functions of the desired positions, and can therefore be included in the optimization problem while maintaining convexity. For instance, since $\dot{x}_i(t_i) = b_i$, we can add $\dot{\mathbf{x}}$ variables that correspond to the velocity at each waypoint by introducing the constraint $\dot{\mathbf{x}} = \mathbf{b}$ — or rather, since to actually include

[1]Technically, in order to ensure uniqueness of the spline we also need to impose a constraint on the velocity or acceleration of the endpoints, but we ignore this for the time being.

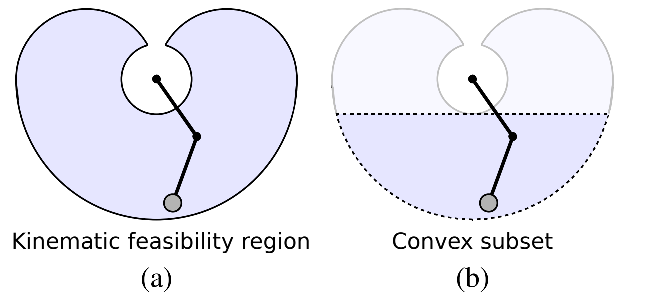

Fig. 2. Illustration of kinematic feasibility constraints. (a) Kinematically feasible region for $q_1 \in [-\pi/2, \pi/2]$, $q_2 \in [-2.6, 2.6]$. (b) Convex subset of the feasible region.

the $\mathbf{b}$ variables in the optimization problem, we can just directly include the constraint

$$\mathbf{H}_1 \dot{\mathbf{x}} = \mathbf{H_2 x}.$$

Adding variables in this manner allows us to include objectives and constraints that depend on the velocity terms, and the same procedure can be used to create variables representing the acceleration or jerk at each waypoint.

Spline position and derivatives at arbitrary times. Often times we want to constrain the position, velocity, etc, of the splines not only at the waypoints, but also at the intermediate times. Using the cubic spline formulation, such variables are also a linear function of the waypoint locations. For example, suppose we wanted to add a variable $x(t')$ representing the position of the trajectory at time $t_i < t' < t_{i+1}$. Using equations (2) and (3),

$$x(t') = a_i + b_i(t' - t_i) + c_i(t' - t_i)^2 + d_i(t' - t_i)^3.$$

But from (6)–(9), a_i, b_i, c_i and d_i are all linear in the desired positions $\mathbf{x}$, so the variable $x(t')$ is also linear in these variables. The same argument applies to adding additional variables that represent the velocity, acceleration, or jerk at any time.[2] Space constraints prohibit providing the complete definition of the following matrices, but it should be clear from the discussion above that if use use $\mathbf{x}' \in \mathbb{R}^{N \times n}$ to denote the spline positions at a variety of intermediate times, we can solve for these positions via a linear system

$$\mathbf{x}' = \mathbf{G}_1\mathbf{x} + \mathbf{G}_2\dot{\mathbf{x}}$$

and similarly for other derivatives.

Kinematic feasibility constraints. Since we are specifically focused on planning trajectories in task-space, a key requirement is that points on the spline must be kinematically feasible for the robot. While the kinematic feasibility region of an articulated body with joint stops is not typically a convex set, we can usually find a suitable *subset* of this kinematic region that *is* convex.

For example, consider the double pendulum shown in Figure 2. The kinematically feasible region, when joint one is

[2]In theory, if we wanted to ensure that the entire spline obeys a position or derivative constraint, we would have to add an infinite number of such variables. However, as we will show, in practice we can obtain good results by introducing a very small number of additional variables, greatly increasing the practicality of the approach. This same consideration applies to the kinematic feasibility and collision constraints.

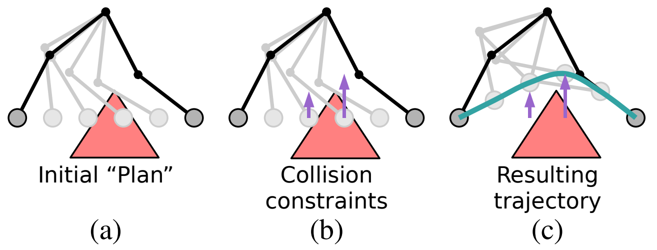

Fig. 3. Illustration of collision constraints. (a) Initial (infeasible) plan. (b) Height constraints imposed to avoid collision with obstacle. (c) Resulting optimized cubic spline trajectory.

restricted to the range $[-\pi/2, \pi/2]$ and joint two is restricted to the range $[-2.6, 2.6]$, is show in Figure 2 (a). Although this region is not convex, we can easily find a convex subset, such as the region shown in Figure 2 (b). This convex subset can be expressed as the intersection of a sphere and a half-plane, and so we can efficiently constrain the spline waypoints (or spline positions at intermediate times) to lie within this set, while maintaining convexity of the optimization problem.

Collision constraints. Although general collision constraints can be quite difficult to handle in our framework, in many simple cases we can approximate such constraints using a simple method shown in Figure 3. Here Figure 3 (a) shows the initial plan used by the cubic spline optimizer (recall from above such a plan need to be feasible). Two waypoints on this initial trajectory violate the collision constraint, so we simply add the constraint, at each of these times, that the resulting waypoint must lie above the the obstacle by some margin; Figure 3 (b) shows this constraint, and Figure 3 (c) shows the resulting trajectory. This technique is an approximation, because 1) it only constrains the end-effector position and still could lead to a collision with the articulated body, 2) it assumes that the x-position of the waypoints after optimization doesn't change, when in fact it can and 3) as mentioned in Footnote 2, these collision constraints are only imposed at a finite number of points, so we have to insure that the "resolution" of these points is smaller than any thin obstacles. Nonetheless, as we show in the subsequent section, this simple approximation works quite well in practice, and allows us to maintain convexity of the optimization problem.

Finally, we note that for some planning problems, adding enough constraints of any of the preceeding types can lead to an infeasible optimization problem. We mention this primarily to emphasize that this approach is *not* suited to all planning situations; if plans must traverse through non-convex, narrow "corridors" in the robot's configuration space, then slower, traditional motion planning algorithms may be the only possible approach. However, for situations where our method can be applied, such as the LittleDog planning tasks we describe below, our method can produce highly-optimized trajectories extremely quickly.

B. Optimization Objectives

Given the variables and constraints described above, we lastly need to define our final optimization objective. While we have experimented with several different possible optimization objectives, one that appears to work quite well is to penalize the squared velocities at the waypoints and at a few

Fig. 4. The LittleDog robot, designed and built by Boston Dynamics, Inc.

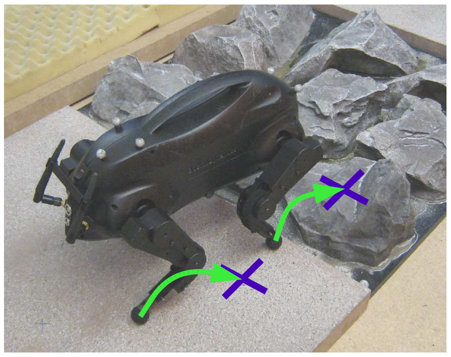

Fig. 5. Planning task, taking two steps over rough terrain.

intermediate points between each waypoint. More formally, we use the optimization objective

$$f(\dot{\mathbf{x}}, \dot{\mathbf{x}}') = \mathrm{tr}\, \dot{\mathbf{x}}^T \dot{\mathbf{x}} + \mathrm{tr}\, \dot{\mathbf{x}'}^T \dot{\mathbf{x}}'$$

where $\dot{\mathbf{x}}$ represents the velocity at each waypoint and $\dot{\mathbf{x}}'$ represents the velocities at the midpoint between each waypoint. This objective has the effect of discouraging very large velocities at any of the spline points, which leads to trajectories that travel minimal distances while keeping fairly smooth. However, while this objective works well for our settings, there are many other possible objective functions that might function better in other cases such as minimizing the maximum velocity, average or maximum acceleration, average distance between spline points, or (using approximations based on the Jacobians along the initial trajectory) average or maximum joint velocities or torques.

One objective that cannot be easily minimized is the total time of the trajectory. Recall that while the position, velocity, etc terms all linearly related by equations (6)–(9), these equations also involve non-linear, non-convex functions of the times. However, there has been previous work in approximately optimizing the times of cubic splines [5], [6], [7], and if the total time of the trajectory is ultimately the most important objective, these techniques can be applied.

IV. Application to a Quadruped Robot

In this Section we present the primary empirical result of this paper: the successful application of these methods to a quadruped, the "LittleDog" robot, shown in Figure 4. In particular, we use cubic spline optimization to plan foot and body trajectories for statically stable locomotion over rough terrain. The system we present here is used in our software

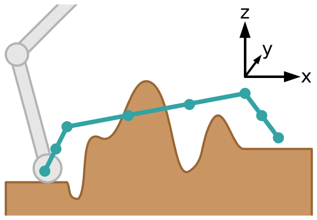

Fig. 6. Foot planning task, and initial trajectory.

architecture for the quadruped, submitted to DARPA for the Learning Locomotion project. Due to space constraints we only give a brief description of the quadruped task in general; we refer the reader to [8] for a more complete description of our system, excluding the cubic spline optimization.

The basic planning task that we consider in this paper is as follows: given a current position of the robot, and two upcoming footsteps plan trajectories for the feet and robot center of gravity (COG) that achieve these footsteps; Figure 5 illustrates this planning task. Successfully achieving the footsteps requires that the trajectories obey several constraints, such as ensuring that the COG and foot locations are jointly kinematically feasible and collision free, and ensuring that the robot is statically stable as it moves its feet. In the subsequent section, we show how to apply the techniques from the previous section to plan the foot and COG trajectories respectively.

A. Planning Foot Trajectories

The chief aim in planning foot trajectories is to achieve smooth motions that avoid collision with the terrain. An illustration of this task is shown in Figure 6, along with the initial plan we supply to the cubic spline optimization. The initial plan is a simple trapezoid, with three waypoints allocated for the upward and downward "ramps", and the remaining waypoints in a line, spaced in 2cm intervals. We use the following optimization problem to plan the foot trajectories:

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{x}',\dot{\mathbf{x}},\dot{\mathbf{x}}',\ddot{\mathbf{x}}} \quad & \mathrm{tr}\, \dot{\mathbf{x}}^T \dot{\mathbf{x}} + \mathrm{tr}\, \dot{\mathbf{x}'}^T \dot{\mathbf{x}}' && (11)\\
\text{subject to} \quad & \mathbf{H}_1 \dot{\mathbf{x}} = \mathbf{H}_2 \mathbf{x} && (12)\\
& \ddot{\mathbf{x}} = \frac{1}{2}(\mathbf{H}_3 \mathbf{x} + \mathbf{H}_4 \dot{\mathbf{x}}) && (13)\\
& \mathbf{x}' = \mathbf{G}_1 \mathbf{x} + \mathbf{G}_2 \dot{\mathbf{x}} && (14)\\
& \dot{\mathbf{x}}' = \mathbf{G}_3 \mathbf{x} + \mathbf{G}_4 \dot{\mathbf{x}} && (15)\\
& \mathbf{x}_{0,:} = \hat{x}_0,\ \mathbf{x}_{T,:} = \hat{x}_T,\ \dot{\mathbf{x}}_{0,:} = 0,\ \dot{\mathbf{x}}_{T,:} = 0 && (16)\\
& \ddot{\mathbf{x}}_{t,x} \geq 0,\ \ddot{\mathbf{x}}_{t,y} \geq 0,\quad t = 0, 1 && (17)\\
& \ddot{\mathbf{x}}_{t,x} \leq 0,\ \ddot{\mathbf{x}}_{t,y} \leq 0,\quad t = T-1, T && (18)\\
& \left.\begin{aligned}\ddot{\mathbf{x}}_{t,x} \geq \ddot{\mathbf{x}}_{t+1,x}\\ \ddot{\mathbf{x}}_{t,y} \geq \ddot{\mathbf{x}}_{t+1,y}\end{aligned}\right\} \quad t = 2, \ldots, T-3 && (19)\\
& \ddot{\mathbf{x}}_{t,z} < 0,\quad t = 2, \ldots, T-2 && (20)\\
& \begin{aligned}(\mathbf{x}_{t,x} - \mathbf{x}_{t+2,x})^2 + (\mathbf{x}_{t,y} - \mathbf{x}_{t+2,y})^2 \\ \leq (3\mathrm{cm})^2, \quad t = 0, T-2\end{aligned} && (21)\\
& {\mathbf{x}'}_{i,z} \geq \hat{x}_z(t'_i) + 2\mathrm{cm},\quad i = 1, \ldots, N && (22)\\
& \mathbf{x}_{t,z} \leq \max_{i=1,\ldots,N} \hat{x}_z(t'_i),\quad t = 1, \ldots T. && (23)
\end{aligned}$$

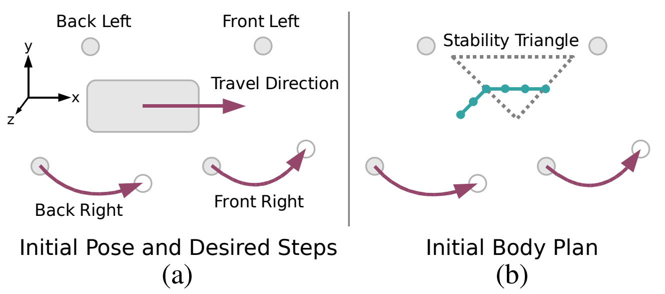

Fig. 7. (a) COG planning task. (b) Initial trajectory for the COG plan.

While there are many different terms in this optimization problem, the overall idea is straightforward. The optimization objective (11) is the squared velocity objective we discussed earlier; constraints (12)–(15) are the standard cubic spline equations for adding additional variables representing respectively the velocity at the waypoints, the acceleration at the waypoints, additional position terms, and additional velocity terms;[3] (16) insures that the spline begins and ends at the start and goal, with zero velocity; (17) and (18) ensure that the x, y accelerations are positive (negative) at the start (end) ramp, which in turn ensures that the trajectory will never overshoot the start and end locations;[4] (19) extends the previous constraint slightly to also ensure that the x, y accelerations during the main trajectory portion are monotonic; (20) forces the z accelerations during the main portion of the trajectory to be negative, which ensures that the spline moves over any obstacles in one single arch; (21) ensures that the last waypoints in the ramp don't deviate from the start and end positions by more than 3cm; finally, (22) and (23) ensure that the z position of the spline is 2cm above any obstacle, and that no waypoint is more than 2cm higher than the tallest obstacle.

Note that this precise optimization problem is not meant to be a general purpose algorithm. This particular objective and the constraints were developed specifically for the quadruped foot trajectory planning task, and many of the constraints were developed over time in response to specific situations that caused simpler optimization problems to produced suboptimal plans. As such, the main purpose of describing the optimization problem explicitly is as a demonstration of the techniques presented in the previous section, and to give a concrete example of the method.

B. Planning COG Trajectories

The aim of planning a trajectory for the COG is twofold: maintaining stability of the robot while allowing the feet to reach their targets. This planning task is illustrated in Figure 7 (a), and the initial trajectory supplied to the cubic spline optimizer is shown in Figure 7 (b). The triangle in Figure 7 (b) denotes the double supporting triangle, described more fully in [8]; as long as the COG is within this triangle, the robot can take either of the two steps while maintaining stability. Therefore, the initial plan proceeds in two phases, first moving into the supporting triangle, and then moving forward in the triangle while taking the two steps. Although not shown entirely in Figure 7 (b) for the sake of clarity, we always use nine waypoints in the COG trajectory splines: three to move the COG into the supporting triangle, and three for each foot movement. Since planning the COG trajectory requires knowledge of the foot locations, we first use the method above to plan trajectories for the moving feet, which we denote $x_{f_1}(t)$ and $x_{f_2}(t)$. We then use the following optimization problem to plan the COG trajectory:

$$\min_{\mathbf{x},\dot{\mathbf{x}},\dot{\mathbf{x}}'} \quad \begin{array}{l} \mathrm{tr}\, \dot{\mathbf{x}}^T \dot{\mathbf{x}} + \mathrm{tr}\, \dot{\mathbf{x}}'^T \dot{\mathbf{x}}' \\ +\lambda \sum_{i=3}^{5} \mathrm{feas}(\mathbf{x}_{i,:}, x_{f_1}(t_i), f_1) \\ +\lambda \sum_{i=5}^{7} \mathrm{feas}(\mathbf{x}_{i,:}, x_{f_2}(t_i), f_2) \end{array} \tag{24}$$

$$\text{subject to} \quad \mathbf{H}_1 \dot{\mathbf{x}} = \mathbf{H}_2 \mathbf{x} \tag{25}$$

$$\dot{\mathbf{x}}' = \mathbf{G}_3 \mathbf{x} + \mathbf{G}_4 \dot{\mathbf{x}} \tag{26}$$

$$\mathbf{x}_{0,:} = \hat{x}_0, \dot{x}_0, \ \mathbf{x}_{2,:} = \hat{x}_2, \ \mathbf{x}_{8,:} = \hat{x}_8 \tag{27}$$

$$\dot{\mathbf{x}}_{0,:} = \dot{x}_{\mathrm{init}} \tag{28}$$

$$\mathbf{x}_{i,:} \in \mathcal{S}, \quad i = 3, \ldots, 7 \tag{29}$$

where

$$\mathrm{feas}(x_{\mathrm{body}}, x_{\mathrm{foot}}, f)$$

denotes the squared distance from x_{foot} to the kinematically feasible region of foot f, given that the COG is positioned at point x_{body}, and where $\mathcal{S}$ denotes the support triangle. Intuitively, the optimization objective (24) is a weighted combination of the kinematic infeasibility of the moving feet plus the velocity terms we discussed earlier — in practice, we choose $\lambda = 100$ to try to make the system as close to kinematically feasible as possible, and only later try to minimize the velocities; constraints (25) and (26) are again the standard cubic spline equations for velocity terms — here we add eight additional velocity terms, one at the midpoint of each two waypoints; (27) ensures that the trajectory begins, enters the supporting triangle, and ends at the initially specified waypoints, while (28) ensures that the initial velocity of the spline is equal to the COG's current velocity; finally, (29) ensures that the COG waypoints will be inside the supporting triangle during the times that the feet are moving.

C. Experimental Results

We begin with a qualitative look at the trajectories generated by this method. Figure 8 shows a typical footstep trajectory generated by this method. As we can see, the trajectory moves the foot from its initial location to the desired location in one fluid motion, stepping high enough to avoid any obstacles. Likewise, Figure 9 shows a typically COG trajectory generated by the algorithm. Notice that the trajectory inside the supporting triangle is not just a straight line: the algorithm adjusts the trajectory in this manner to maximize the kinematic feasibility of the moving feet.

Of course, while examining the splines in this manner can help give an intuition about kind of trajectories generated

[3]In greater detail, we add four additional velocity terms, in the midpoints of the waypoints on the "ramp" portion of the initial trajectory. We add N additional position terms, one at each 1cm interval along the top portion of the initial trajectory.

[4]This constraint and next assume that the $\hat{x}_{0,x} \leq \hat{x}_{T,x}$ and $\hat{x}_{0,y} \leq \hat{x}_{T,y}$. In the case that these inequalities are reversed, the corresponding inequalities in the constraints are also reversed.

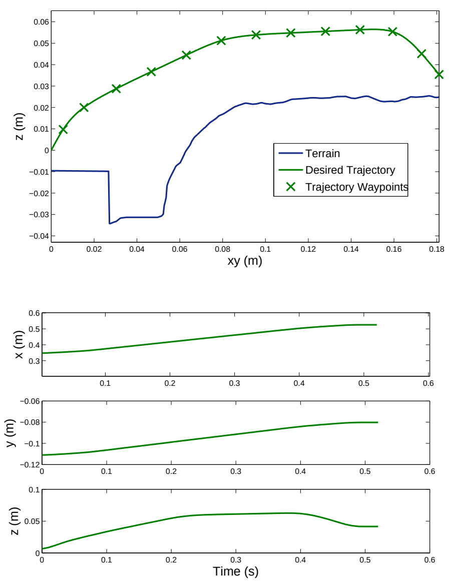

Fig. 8. Typical example of a foot trajectory generated by the algorithm. The top figure shows the resulting trajectory in 3D space, while the bottom shows each component as a function of time.

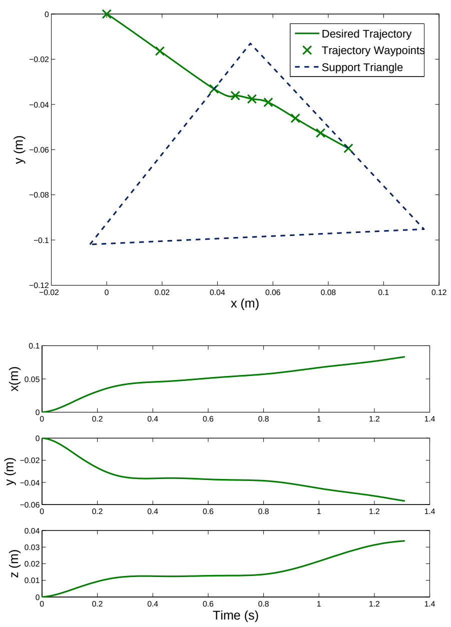

Fig. 9. Typical example of a COG trajectory generated by the algorithm.

by our algorithm, we are ultimately interested in whether or not the method actually improves performance on the LittleDog. To evaluate this, we tested the system on two terrains of varying difficulty, shown in Figure 10. We compare the cubic spline optimization approach to the trajectory planning method described in [8], which uses simple box-shapes to step over terrain, and linear splines for moving the COG. While, as mentioned previously, the entire system for planning and control of the quadruped is quite complex, all other elements of the system remain fixed in the experiments here.

Table I shows the performance of the quadruped both with and without the cubic spline optimization. We ran 10

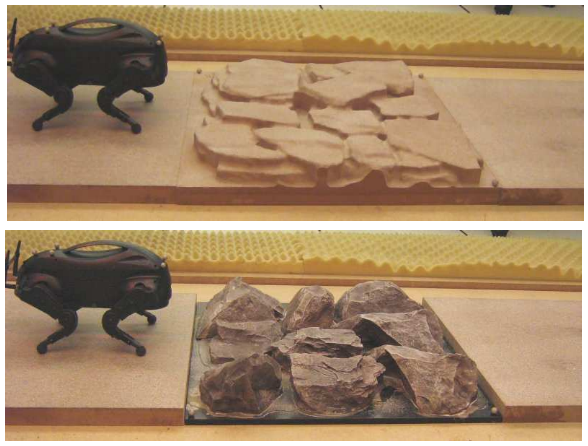

Fig. 10. Terrains used for evaluating the system. The top terrain is fairly trivial to navigate, while the bottom terrain is significantly more challenging.

trials on each of the terrains, and evaluated the systems using 1) fraction of successful runs, 2) speed over terrain, 3) average number of “recoveries”[5], and 4) average tracking error (i.e., distance between the planned and actual location) for the moving foot. Perhaps the most obvious benefit of the cubic spline optimization method is that the resulting speeds are faster; this is not particularly surprising, since the splines output by our planner will clearly be more efficient than a simple box pattern over obstacles. However, equally important is that the cubic spline optimization also leads to more *robust* behavior, especially on the challenging terrain: the previous method only succeeds in crossing the terrain 70% of the time, and even when it does succeed it typically needs to execute several recoveries, whereas the cubic spline optimization method crosses the terrain in all cases, and executes much fewer recoveries. We surmise that this is because the cubic spline method attempts to maintain kinematic feasibility of the moving foot at all times. This is seen in Table I from the fact that the cubic spline optimization approach has lower tracker error, implying more accurate placement of feet, and therefore greater robustness is challenging environments.

Finally, we want to briefly mention issues of running time. We solve the convex optimization problems using MOSEK, a general purpose convex solver, available at `http://www.mosek.com`. Using this software, solving for a foot spline takes an average of 4.4 milliseconds (with 1.1 millisecond standard deviation), while solving for the COG path takes an average of 5.4 ms (0.4 ms standard deviation). These quantities remain approximately the same regardless of the terrain, and are fast enough to allow for real-time re-planning and control on the quadruped.

V. Discussion and Related Work

As mentioned in the introduction, cubic splines are very common in robotic applications [1]. This paper, as well as [5], [6], [7] discuss methods for approximately optimizing the time of the trajectory, but do not consider optimizing the waypoints themselves. The work of Schlemmer et al.

[5]A recovery is required whenever the COG falls outside the supporting triangle. Again, see [8] for details.

	Easy Terrain		Challenging Terrain	
Metric	Spline Optimization	Previous Method	Spline Optimization	Previous Method
% Successful Trials	100%	100%	**100%**	70 %
Speed (cm/sec)	**7.02 ± 0.10**	5.99 ± 0.07	**6.30 ± 0.12**	5.59 ± 0.31
Avg. Tracking Error (cm)	**1.28 ± 0.03**	1.40 ± 0.06	**1.27 ± 0.05**	1.55 ± 0.09
Avg. # Recoveries	0.0	0.0	**0.5 ± 0.37**	2.22 ± 1.45

TABLE I

PERFORMANCE OF CUBIC SPLINE OPTIMIZATION ON THE TWO QUADRUPED TERRAINS. TERMS INCLUDE 95% CONFIDENCE INTERVALS.

[9] perhaps bears the most resemblance to our own, as they do consider the modification the cubic spline parameters themselves. However, the resulting techniques are quite different, since they break the trajectories into multiple segments with *constant* acceleration, not the continuous acceleration profiles that we consider here; furthermore, they focus on splines in joint space, whereas the most interesting constraints (namely the kinematic feasibility and collision constraints) in our setting are precisely those that arise due to the task-space planning.

Free-knot splines [10], [11], typically used in the statistics literature, also relate to our method, as they involve choosing both the waypoints and times of parametric splines, including cubic splines. However, these methods typically focus on fitting splines to data, not creating splines that relate to any kind of trajectory planning, so the actual optimization approaches are very different.

Our method also relates to the trajectory optimization literature in robotics [12], [13], [14], [15], though these methods typically focus on improving a pre-supplied (feasible) trajectory via gradient or higher-order methods. Although in this paper we focus on cases where the initial "plan" supplied to the cubic spline optimization problem is very simple and possibly infeasible, there is nothing that prevents the method from being used to optimize a feasible trajectory generated by a planner, as in the approaches above: in this setting the algorithm could also function as a trajectory optimization approach, specifically using a cubic spline parametrization of the trajectory, and using general convex optimization techniques rather than gradient methods alone.

Planning trajectories for the quadrupeds robot in particular has been considered in [8], [16], [17]. Although a full comparison between the different approaches is beyond the scope of this paper, we did compare to our previous controller, [8] and demonstrated that cubic spline optimization improves performance. The work by Mistry et al, [18] also considers a quadruped and specifically looks at the trade-off between balance and kinematic feasibility, but this work is generally orthogonal to our own, as it primarily considers control strategies for achieving a *given* trajectory.

VI. CONCLUSION

In this paper we presented a cubic spline optimization method, for planning smooth trajectories for robot motion. The algorithm uses convex optimization methods to efficiently plan task-space trajectories, while obeying several constraints, such as kinematic feasibility and avoiding contact. We apply this method to the task of planning trajectories for a quadruped robot, and demonstrate significant improvement over past work.

REFERENCES

[1] C.-S. Lin, P.-R. Chang, and J. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial robots," *IEEE Transactions on Automatic Control*, vol. 28, no. 12, pp. 1066–1074, 1983.

[2] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[3] J. Betts, "Survery of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

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

[5] B. Cao, G. Dodds, and G. Irwin, "Constrained time-efficient smooth cubic spline trajectory generation for industrial robots," in *Proceedings of the IEE Conference on Control Theory and Applications*, 1997.

[6] J.-H. Park, H.-S. Kim, and Y.-K. Choi, "Trajectory optimization and control for robot manipulator using evolution strategy and fuzzy logic," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 1997.

[7] A. I. F. Vaz and E. M. Fernandes, "Tools for robotic trajectory planning using cubic splines and semi-infinite programming," in *Recent Advances in Optimization*, A. Seeger, Ed. Springer, 2006, pp. 399–413.

[8] J. Z. Kolter, M. P. Rodgers, and A. Y. Ng, "A control architecture for quadruped locomotion over rough terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008.

[9] M. Schlemmer, R. Finsterwalder, and G. Grubel, "Dynamic trajectory optimization in real time for moving obstacles avoidance by a ten degrees of freedom manipulator," in *International Conference on Intelligent Robots and Systems*, 1995.

[10] C. de Boor, *A Practical Guide to Splines*. Springer-Verlag, 1978.

[11] D. Jupp, "Approximations to data by splines with free knots," *SIAM Journal of Numerical Analysis*, vol. 15, pp. 328–343, 1978.

[12] D. W. Miles and S. M. Rock, "Real-time dynamic trajectory optimization," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 1996.

[13] S. Fleury, P. Soueres, J. Laumound, and R. Chatila, "Primitives for smoother mobile robot trajectories," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 441–448, 1995.

[14] F. Lamiraux, D. Bonnafous, and O. Lefebvre, "Reactive path deformation for non-holonomic mobile robots," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 967–977, 2004.

[15] S. Lee, J. Park, F. M. Kim, and J. Bobrow, "Newtow-type algorithms for dynamics-based robot movement optimization," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 657–667, 2005.

[16] D. Pongas, M. Mistry, and S. Schaal, "A robust quadruped walking gait for traversing rough terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[17] J. R. Rebula, P. D. Neuhaus, B. V. Bonnlander, M. J. Johnson, and J. E. Pratt, "A controller for the littledog quadruped walking on rough terrain," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[18] M. Mistry, J. Nakanishi, and S. Schaal, "Take space control with prioritization for balance and locomotion," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[19] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*. Morgan Kaufmann, 1989.

APPENDIX

A. Derivation of the Cubic Spline Equations

In this section we derive the standard equations for cubic splines, presented in Section II. In particular, we show that given desired locations and final and initial velocities of the spline, we can solve for the cubic spline parameters using the linear equations (6)–(9). We reiterate that this is a standard derivation, but we include it for completeness and to solidify our notation. The derivation follows that in [19], but generalizes it to the case of arbitrary times for the spline points.

Recall that the trajectory from time t_i to t_{i+1} is given by the cubic function

$$x_i(t) = a_i + b_i(t - t_i) + c_i(t - t_i)^2 + d_i(t - t_i)^3 \tag{30}$$

In order to achieve a smoother, continuous trajectory, we impose four conditions on the spline: 1) at t_i it must have the value $x_i^\star$, 2) at t_{i+1} it must have the value $x_{i+1}^\star$, 3) its derivative (velocity) must be continuous at t_{i+1}, and 4) its second derivative (acceleration) must be continuous at t_{i+1}. These four conditions can be written formally as

$$\begin{aligned} x_i(t_i) &= x_i^\star \\ x_i(t_{i+1}) &= x_{i+1}^\star \\ \dot{x}_i(t_{i+1}) &= \dot{x}_{i+1}(t_{i+1}) \\ \ddot{x}_i(t_{i+1}) &= \ddot{x}_{i+1}(t_{i+1}). \end{aligned}$$

Using the cubic function definition (30), these conditions can be rewritten in terms of the cubic spline parameters:

$$\begin{aligned} a_i &= x_i^\star & (31)\\ a_i + b_i\Delta t_i + c_i\Delta t_i^2 + d_i\Delta t_i^3 &= x_{i+1}^\star & (32)\\ b_i + 2c_i\Delta t_i + 3d_i\Delta t_i^2 &= b_{i+1} & (33)\\ 2c_i + 6d_i\Delta t_i &= 2c_{i+1} & (34) \end{aligned}$$

where

$$\Delta t_i \equiv t_{i+1} - t_i.$$

Now we use simple variable elimination to get find explicit equations for the cubic spline parameters. Specifically, equation (31) immediately leads to the first linear equation (6)

$$\mathbf{a} = \mathbf{x}.$$

Next, we use equations (32) and (33), to solve for c_i and d_i in terms of the x_i's and b_i's, giving:

$$\begin{aligned} c_i &= \frac{3(x_{i+1}^\star - x_i^\star) - (2b_i + b_{i+1})\Delta t_i}{\Delta t_i^2} & (35)\\ d_i &= \frac{2(x_i^\star - x_{i+1}^\star) + (b_i + b_{i+1})\Delta t_i}{\Delta t_i^3}. & (36) \end{aligned}$$

This leads immediately to the third and fourth linear equations (8) and (9)

$$\begin{aligned} \mathbf{c} &= \mathbf{H}_3\mathbf{x} + \mathbf{H}_4\mathbf{b} \\ \mathbf{d} &= \mathbf{H}_5\mathbf{x} + \mathbf{H}_6\mathbf{b} \end{aligned}$$

where

$$\mathbf{H}_3 = \begin{bmatrix} \frac{-3}{\Delta t_0^2} & \frac{3}{\Delta t_0^2} & 0 & \cdots & 0 & 0 \\ 0 & \frac{-3}{\Delta t_1^2} & \frac{3}{\Delta t_1^2} & \cdots & 0 & 0 \\ 0 & 0 & \frac{-3}{\Delta t_2^2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{-3}{\Delta t_{T-1}^2} & \frac{3}{\Delta t_{T-1}^2} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{37}$$

$$\mathbf{H}_4 = \begin{bmatrix} \frac{-2}{\Delta t_0} & \frac{-1}{\Delta t_0} & 0 & \cdots & 0 & 0 \\ 0 & \frac{-2}{\Delta t_1} & \frac{-1}{\Delta t_1} & \cdots & 0 & 0 \\ 0 & 0 & \frac{-2}{\Delta t_2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{-2}{\Delta t_{T-1}} & \frac{-1}{\Delta t_{T-1}} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{38}$$

$$\mathbf{H}_5 = \begin{bmatrix} \frac{2}{\Delta t_0^3} & \frac{-2}{\Delta t_0^3} & 0 & \cdots & 0 & 0 \\ 0 & \frac{2}{\Delta t_1^3} & \frac{-2}{\Delta t_1^3} & \cdots & 0 & 0 \\ 0 & 0 & \frac{2}{\Delta t_2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{2}{\Delta t_{T-1}} & \frac{-2}{\Delta t_{T-1}} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{39}$$

$$\mathbf{H}_6 = \begin{bmatrix} \frac{1}{\Delta t_0^2} & \frac{1}{\Delta t_0^2} & 0 & \cdots & 0 & 0 \\ 0 & \frac{1}{\Delta t_1^2} & \frac{1}{\Delta t_1^2} & \cdots & 0 & 0 \\ 0 & 0 & \frac{1}{\Delta t_2^2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{\Delta t_{T-1}^2} & \frac{1}{\Delta t_{T-1}^2} \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}. \tag{40}$$

Finally, substituting (35) and (36) into (34) we obtain

$$\begin{aligned} &\frac{2b_i + 4b_{i+1}}{\Delta t_i} + \frac{4b_{i+1} + 2b_{i+2}}{\Delta t_{i+1}} = \\ &\qquad \frac{6(x_{i+1}^\star - x_i^\star)}{\Delta t_i^2} + \frac{6(x_{i+2}^\star - x_{i+1}^\star)}{\Delta t_{i+1}^2}. \end{aligned} \tag{41}$$

These equations, along with the additional constraints that can be expressed as the second set of linear equalities (7):

$$\mathbf{H}_1\mathbf{b} = \mathbf{H_2 x}$$

where

$$\mathbf{H}_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \alpha_0 & \beta_0 & \alpha_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \alpha_1 & \beta_1 & \alpha_2 & \cdots & 0 & 0 & 0 \\ 0 & 0 & \alpha_2 & \beta_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \beta_{T-3} & \alpha_{T-2} & 0 \\ 0 & 0 & 0 & 0 & \cdots & \alpha_{T-2} & \beta_{T-2} & \alpha_{T-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{42}$$

$$\alpha_i \equiv \frac{2}{\Delta t_i}, \quad \beta_i \equiv \frac{4}{\Delta t_i} + \frac{4}{\Delta t_{i+1}}$$

and

$$\mathbf{H}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\gamma_0 & \eta_0 & \gamma_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -\gamma_1 & \eta_1 & \gamma_2 & \cdots & 0 & 0 & 0 \\ 0 & 0 & -\gamma_2 & \eta_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \eta_{T-3} & \gamma_{T-2} & 0 \\ 0 & 0 & 0 & 0 & \cdots & -\gamma_{T-2} & \eta_{T-2} & \gamma_{T-1} \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{43}$$

$$\gamma_i \equiv \frac{6}{\Delta t_i^2}, \quad \eta_i \equiv \frac{6}{\Delta t_{i+1}^2} - \frac{6}{\Delta t_i^2}.$$