A General Algorithm for Time-Optimal Trajectory Generation Subject to Minimum and Maximum Constraints

Stephen D. Butler, Mark Moll, and Lydia E. Kavraki

Rice University, Department of Computer Science, Houston, TX 77005, USA

Abstract. This paper presents a new algorithm which generates time-optimal trajectories given a path as input. The algorithm improves on previous approaches by generically handling a broader class of constraints on the dynamics. It eliminates the need for heuristics to select trajectory segments that are part of the optimal trajectory through an exhaustive, but efficient search. We also present an algorithm for computing all achievable velocities at the end of a path given an initial range of velocities. This algorithm effectively computes bundles of feasible trajectories for a given path and is a first step toward a new generation of more efficient kinodynamic motion planning algorithms. We present results for both algorithms using a simulated WAM arm with a Barrett hand subject to dynamics constraints on joint torque, joint velocity, momentum, and end effector velocity. The new algorithms are compared with a state-of-the-art alternative approach.

1 Introduction

A path is a continuous curve describing the joint positions desired for a robot to move through. It is often represented by a sequence of waypoints with positions between them obtained through interpolation. A trajectory is a re-parameterization of a path as a function of time. The trajectory generation process is subject to the dynamics constraints of the underlying system. Finding time-optimal trajectories for robots with many degrees of freedom (DOFs) subject to complex dynamics and geometric constraints is a challenging problem that arises naturally in many robotics applications ranging from welding and painting to humanoid robots [1] and even spacecraft [2]. In practice, conservative safety constraints (e.g., by assuming quasistatic dynamics) are often used to simplify the trajectory generation problem, but this leads to either sub-optimal performance or more powerful hardware requirements than strictly necessary. If a task cannot be accomplished quasistatically, the task may still be dynamically feasible, in which case there are necessarily *minimum* velocity constraints which have generally, previously been ignored.

Many in the control community, such as [3], have made progress in solving planning problems which are constrained by dynamics, e.g., through constraints on torque, velocity, and momentum. Likewise, those in the path planning community, typically using sampling-based planning [4,5], have made progress in cluttered environments where collision and geometric constraints arise, e.g., on pose, orientation, and end effector contact. Nevertheless, motion planning problems for high DOF systems subject to both complex geometric and dynamics constraints still pose a formidable challenge. The typical motion planning pipeline for high DOF systems often consists of first performing sampling-based planning to generate a path which is then fed into a trajectory generator or controller. In this case, any sense of optimality or feasibility for the dynamics is lost as

the path is generated without the dynamics. Kinodynamic motion planners that consider both the dynamics and geometric constraints do exist, e.g. [6], but the size of the state space of the problem still increases exponentially for high-DOF systems. Our work aspires to push the envelop on solving high-DOF problems with dynamics efficiently.

Admissible velocity propagation (AVP) [7] is a promising new method which combines sampling-based planning and time-optimal trajectory generation to make the kinodynamic planning problem *significantly* more tractable. AVP builds on tree-based motion planning algorithms (e.g., [8,9]), which iteratively construct a tree of configurations connected by path segments. In AVP, a tree-based planner performs an additional step for each new path segment: it computes a bundle of all dynamically feasible trajectories. In this manner, the planner finds the interval of all admissible velocities (AVI) that can be reached at each sampled configuration. By computing the path segment-AVI pairs the search space is significantly shrunk since while position is sampled as a point, the velocity component of state is sampled as a volume. By sampling entire volumes of the state space, kinodynamic motion planning times are significantly improved. Unfortunately, the classical algorithms [10,11] which compute the AVI needed by AVP planners have been criticized as difficult to implement, not very robust, and difficult to extend to other dynamics constraints such as observed in [12,13]. The classical methods only work for those specific constraints for which they were written. No proof of completeness exists for these algorithms despite their determinism.

This paper presents two algorithms. The first algorithm, which we will refer to as Traj_Alg hereafter, generates a time-optimal trajectory over a path given some starting and ending velocity. That is, given a path, a robot description, and a set of additional constraint functions, the trajectory returned is of the shortest time possible. The second algorithm, which we will refer to as AVI_Alg hereafter, computes the AVI over a path instead of a single trajectory like Traj_Alg.

Both algorithms we present are efficient, robust, and generalize to a large class of constraints. While previous algorithms would require considerable extension or indeed a complete rework to reason over constraints other than torque or velocity, our algorithms require the user only input a parameterization of their novel constraint function, which is the bear minimum to even define a problem. So long as the user's constraint adheres to the class of constraints which we detail in 6, our algorithm will either return the solution or report that no solution exists. This class includes all those constraints presented previously in the phase plane literature and a considerable number more for which we present a few examples.

In addition to a considerable expansion in generality, we show in the results section that our algorithms outperform the state-of-the-art methods as implemented in TOPP [14] in terms of compute time. Additionally, our algorithms never fail to find a solution despite the stressful test cases which cause TOPP to not find solutions. Our algorithms challenge the fundamental assumptions of the classical methods through an exhaustive, but efficient search process. We plan as future work to present a proof of completeness.

2 Related Work

Work on the path-constrained time-optimal problem began in the early 1980s by [10,11]. These works attempt to encode dynamical constraints into a phase plane delineated

by inadmissible regions and proved that time-optimal trajectories result from finding profiles in the phase plane which maximize the integral of phase velocity. [10,11] are both specific only to torque constraints, fail at undifferentiable points, and cannot guarantee that they will find a solution even if one exists.

Recently, convex optimization [13,15] is another approach which has been used to solve the path parameterization problem. Convex optimization is similarly efficient in terms of computation time to phase plane navigation and has the added benefit of being able to optimize for quantities other than time. However, introducing new constraints requires that convexity is maintained and no extension has been presented so far to efficiently solve the AVI problem.

Phase plane navigation, however, can solve the AVI problem as efficiently as it solves the path parameterization problem. Much like the move from the workspace to the configuration space for geometric robot motion planning, the phase plane takes the state of the robot and transforms it into a point and trajectories into curves in the phase plane. This gives us a new space in which to solve the optimal-trajectory generation problem. Unfortunately, the early algorithms of [10,11] are difficult to extend to new constraints beyond joint torque and joint velocity constraints [16].

Recent work in phase plane navigation by [12], which builds on [11,16], enforces a specific type of path to simplify the problem and works only over velocity and acceleration constraints. Specifically, [12] uses a type of "circular blend" at each way-point which prevents a general path which does not have this feature from being input to the algorithm. There is no clear way to extend this method to torque or other constraints.

In the case of torque constraints, which the classical method, e.g. [10,11], was designed for, zero inertia points along the path which cause the maximum and minimum phase acceleration to be undefined were overlooked. These points were identified by [17], but continued to be mishandled or ignored leading to poor, erratic looking solution trajectories. An approach for identifying and determining the slope at zero inertia switching states for rigid body robots was developed by [18].

The TOPP library [14] is the most recent implementation of the algorithms originated by [10,11] and incorporates the extensions and improvements by the authors mentioned previously, e.g., [16,17,18], and we use it as our point of comparison.

Through all of the mentioned related works, the focus has been on specific constraints, typically torque and velocity. This paper generalizes to a much broader class of constraints, as defined in 6, and eliminates the need for a case-by-case analysis, thereby making it trivial to provide novel constraints as input to our algorithms. Additionally, these previous works operate by seeking only specifically those switching states and corresponding profiles which form the solution. We abandon this tenet and perform an exhaustive search over all possible switching states.

3 Problem Definitions

Traj_Alg computes a time-optimal trajectory from the following inputs:

A path. A path is specified by C^2 curves for each DOF, $\mathbf{q}(s)$. Without C^2 continuity the acceleration is undefined where the second derivative is discontinuous which could lead to undefined behavior in the navigation policy at these points.

System-control dynamics. Specifically, the algorithm requires functions that compute the maximum acceleration $\ddot{s}_{max}(s,\dot{s})$ and minimum acceleration $\ddot{s}_{min}(s,\dot{s})$ in the phase plane. The phase plane itself is described in more detail in section 4.

Path start and end states. The initial position and velocity $(s_o,\dot{s}_o)$ and the final position and velocity $(s_f,\dot{s}_f)$ along the path.

Dynamics constraints. Inequalities of the form $\dot{s}_{\min}(s) \leq \dot{s}(s) \leq \dot{s}_{\max}(s)$ which determine the maximum or minimum phase velocity for a point s along the path. We will show in detail how one parameterizes constraints for the phase plane in section 6.

The **objective function** for minimization is time: $T = \int_{t_o}^{t_f} dt$. The **output** of our algorithm is then a trajectory, $\mathbf{q}(t)$ and $\dot{\mathbf{q}}(t)$ $(t_o \leq t \leq t_f)$, where $\mathbf{q}$ is a full configuration of the system.

AVI_Alg extends Traj_Alg and takes as input in place of discrete starting and ending states a *range* of starting velocities $[\dot{s}_{\min,o},\dot{s}_{\max,o}]$, which we will refer to as the input AVI. The output correspondingly is the AVI at s_f: $[\dot{s}_{\min,f},\dot{s}_{\max,f}]$. For any admissible output velocity we can efficiently extract the corresponding time-optimal trajectory.

4 Classical Phase Plane Navigation Overview

Before we present our algorithms, it is important to understand classical phase plane navigation. The phase plane is defined by the path velocity $\dot{s}$ and the path position s. The scalar s indicates the position along the path $\mathbf{q}(s)$. The velocity and acceleration for all joints in the configuration space can be derived from $\mathbf{q}(s(t))$ using the chain rule.

$$\dot{\mathbf{q}} = \frac{d\mathbf{q}}{ds}\dot{s} \qquad \ddot{\mathbf{q}} = \frac{d\mathbf{q}}{ds}\ddot{s} + \frac{d^2\mathbf{q}}{ds^2}\dot{s}^2 \qquad (1)$$

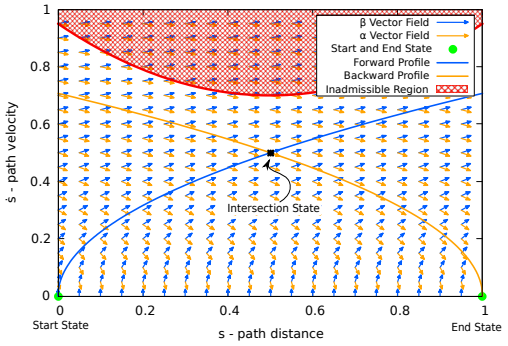

Fig. 1. Example phase plane.

The transformation of constraints to the phase plane is useful because it creates a lower-dimensional space in which the time-optimal trajectory can be computed more efficiently. A *profile*, such as the blue and gold curves in figure 1, is defined as a curve in the phase plane. A *solution trajectory* is any profile or concatenation of profiles from the starting to ending state which do no cross into the inadmissible region. If a solution trajectory maximizes the area under its curve, then it is the *time optimal-trajectory* [10]. Hence, maximizing $\dot{s}$ over a path is equivalent to minimizing the total time for its corresponding trajectory.

For the simple example in figure 1, the solution trajectory can be found by following the β and α vector fields *forward* and *backward* from the start and end states, respectively. The β vector field is defined by the maximum acceleration, $\beta = \ddot{s}_{max}(s,\dot{s})$, which the system is capable of at a given state and α is defined likewise as the minimum acceleration, $\alpha = \ddot{s}_{min}(s,\dot{s})$. We will refer to profiles which progress monotonically left

to right, and thus which try to follow the β vector field as forward profiles and having the forward directional type. Backward profiles are the opposite, they flow right to left following α.

Clearly, there are no valid profiles originating at the start or end state which can rise to values of $\dot{s}$ above those shown in figure 1 since they were created by applying extremal control inputs, i.e., following the β/α fields. Thus, the trajectory resulting from these profiles is time-optimal.

In figure 1, the intersection between the forward and backward profile is denoted by an intersection state. In prior work this intersection would not be computed explicitly, intersections were assumed when it was detected that one profile was above another at some value s. For our methods, intersections between profiles are discovered using curve intersection methods. These intersections are valuable to our algorithms since we need them to develop a graph of profiles in the phase plane.

To construct profiles we *navigate*. We refer to their construction as *navigation*, instead of merely integration, because a *navigation policy* is employed to step-by-step generate profiles. For example, to construct the profile originating from the starting state, the algorithm asks the navigation policy what phase acceleration $\ddot{s}$ it should use to integrate the equations of motion based on its current $[s, \dot{s}]$ state. The phase acceleration which is returned by the policy is then integrated to arrive at some new state. Usually, the acceleration returned by the policy will be equivalent to the β or α vector fields. This "bang-bang control" ensures that profiles are always maximizing $\dot{s}$ with each step.

However, the navigation policy may deviate to some value between β and α when it hits an *inadmissible region*. Inadmissible regions are induced by the dynamics constraints of a system and are delineated by maximum or minimum velocity curves (maxVC) (minVC). Since, the states in these regions violate constraints they cannot be entered by a valid profile. In the instance where a profile has intersected the maxVC, the navigation policy will attempt to guide the profile along the maxVC and so may return a value between β and α. Since no valid states exists above the maxVC, following the maxVC constitutes maximizing $\dot{s}$.

In general, the inadmissible regions and β/α vector field interactions are complex, requiring many profiles and a *phase plane navigation algorithm* is needed. At a high level, all phase plane navigation algorithms perform the following steps, which will be explained in more detail below:

1. Compute maximum and minimum velocity curves (maxVC/minVC), respectively.
2. Compute constraint switching states.
3. Construct maximum limiting profiles (MLP) from constraint switching states.
4. Connect start to end state (Traj_Alg) or connect start to end AVI (AVI_Alg)

Computing the minVC and maxVC is usually a straightforward substitution of the terms in equation 1 into the functions which define specific system constraints. We show how this is done for several constraints in section 6.

If a profile originating at the start or end state intersects the maxVC and has no viable direction of navigation, then it is necessary to identify new states in the space to connect to. These states are referred to as *switching states*. In terms of the control, they are points where the control input switches from following α to β or vice versa. A priori, it is unknown which switching states will be needed to traverse the plane. Any point along

the maxVC could potentially be a switching state. In [14], the identification of these necessary switching states follows from several heuristics designed for different types of constraints.

The MLP is formed from a set of profiles which originate from switching states along the maxVC. MLP construction follows by navigating profiles forward and backward from the constraint switching states, which we will refer to as *expansion*. In [14], profiles are expanded without means of accounting for connectivity between profiles or continuity to states such as the start or end states. This lack of information leads to many problems, both in determining if the resulting profiles form a solution trajectory and in determining if the resulting velocity intervals are in fact feasible. That is, the prior methods may report a feasible trajectory infeasible, i.e., a false negative, or report an infeasible trajectory as feasible, i.e., a false positive. Our algorithms do not have this problem thanks to a key contribution in the form a graph data structure which allow us to efficiently extract the time-optimal trajectory if it exists and report definitively that no solution trajectory exists if it does not.

For trajectory generation, the classical method attempts to connect the starting and ending states either directly together as we saw in figure 1 or through the MLP. For AVI, the algorithm presented in [14] attempts a bisection search. Profiles are iteratively expanded from the start range to find the maximum final $\dot{s}$. It then proceeds to perform the same search at the end trying to find the minimum final $\dot{s}$. This process is described in more detail later as we deviate from the classical method in AVI computation.

5 Traj_Alg and AVI_Alg, Algorithms for Phase Plane Navigation

This section elaborates the key contributions of our work in phase plane navigation. Our algorithms follow the same general flow as what is presented in the previous section. However, we replace the methods used in each step from section 4 with our own to resolve the issues we identified and improve the generality to new novel dynamics constraints. For example, rather than trying to only find the subset of switching states which forms the solution, we exhaustively find all possible switching states as outlined in 5.1. To manage the exhaustive set of profiles and intersection states, we introduce a data structure that can be searched for the time-optimal trajectory or used in service of finding the AVI as outlined in 5.2. The following subsections apply to both Traj_Alg and AVI_Alg until section 5.3 where they split.

5.1 Identification of Constraint Switching States

We refer to a state as *navigable* if there exists an infinitesimal expansion to the left and to the right in the phase plane which does not enter the inadmissible region. In figure 2, we can see that point marked 1 on the maxVC is navigable. If the slope of the maxVC is between the slopes formed by the β and α vectors, which we will refer to as the *navigation cone*, then the maxVC can be followed. The navigation cone for the point marked 2 is directed into the inadmissible region and thus this point is un-navigable.

We do not seek switching states along the minVC. If there exists a state above the minVC at some s, then this state is necessarily better than the one on the minVC, since it

has a higher $\dot{s}$. However, if the maxVC and minVC coincide, i.e., no state exists above the minVC, then the path is infeasible as there is no way to traverse the plane without entering the inadmissible region.

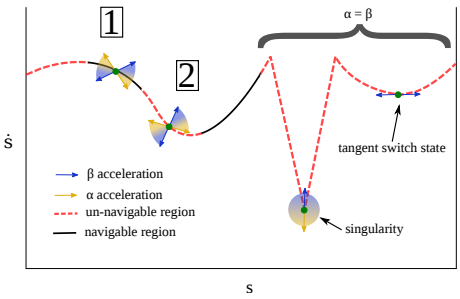

Fig. 2. Phase plane plot illustrating navigability along the maxVC. The α and β vectors are shown only the points marked in the figure.

General Switching States In general, any state along the maxVC which is navigable is a potential switching state. Also, all states within a navigable range are equivalent. That is, any state on the maxVC is equivalent to its navigable neighbors because expansion of said state will generate a profile which necessarily passes through its neighbors. The navigation policy always directs a profile in the direction which maximizes $\dot{s}$, which in this case is along the maxVC since no direction can result in a higher $\dot{s}$ in the next step. Thus, finding all navigable regions and picking at least one point in the range is equivalent to finding all possible constraint switching states. We define a function $navigable(s,\dot{s})$ which evaluates whether the slope of the maxVC bisects the navigation cone. This function is evaluated at finely discretized points along the path to find the valid regions and hence all general switching states. We assume $navigable(s,\dot{s})$ is Lipschitz continuous and that the integration step size is chosen such that the function is monotonic between any two consecutive discretized points. There are two degenerate cases where the general method fails: tangent switching states between discrete states and zero inertia switching states.

Tangent Switching States When the β vector is tangent to the slope of the maxVC we have a point of navigability. If the β vector is above the maxVC at a discrete state and below the maxVC at an adjacent discrete state, then we can use bisection to find the tangent switch state in the interval between them.

Zero Inertia Points As we will show in section 6, a change in sign of the inertial term causes a divide-by-zero scenario. The zero inertia points (ZIPs) corresponding to such an event are easy to identify. Similar to [14], we find these switching states via checking each discretized neighboring state along the maxVC for whether there is a change in sign of the denominator term in the equation which defines $\ddot{s}_{max}$ and $\ddot{s}_{min}$. The ZIP is then located through interpolation between these two states.

Once located, a simple method to determine the slope of best fit for a ZIP is to project segments to the left and right of the ZIP. For each segment a search can be performed to find the angle which minimizes the difference in slope between the segment and the β or α vector at the segments end point. Typically, the segment's s component should be held equal to the discretization length of the path to ensure that that the segment's projected endpoint is sufficiently far from the ZIP which is being avoided.

5.2 Constructing the Maximum Limiting Profiles

With all switching states found, the order of expansion does not matter and, indeed, they can be generated in parallel. Additionally, all switching states are expanded in the same

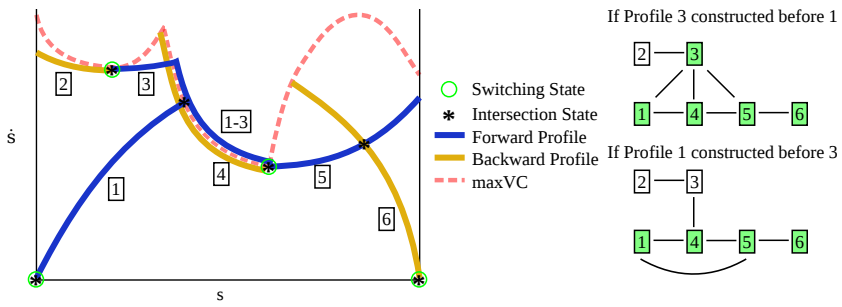

Fig. 3. Phase plane plot with numbered profiles and corresponding continuity graphs. Shaded nodes have continuity to the start and end states.

way except for those which occur at ZIPs which we discussed in section 5.1. This makes our algorithms very easy to parallelize.

In the event that any expansion would lead to a profile intersecting the minVC the algorithm will report that the path is infeasible and halt. Since the profile which intersected the minVC was following the β or α vector field, there exists no control which would avoid the inadmissible region. Also, the profile in question must have originated from the maxVC since it was a candidate MLP profile. Since, it originates from the maxVC, it is then clear that no profile can span the phase plane without also intersecting this profile and hence leading to the minVC. Thus, in this event, the path is known to be infeasible.

We will construct a graph, called the *continuity data structure* or *continuity graph*, with profiles and their intersections corresponding to vertices and edges, respectively. Our algorithms will search this graph to find the profiles and intersections leading from start to end state. If such a path from start to end exists, this path can be used to extract the time-optimal trajectory, or else the path is reported infeasible.

Since switching states can be expanded in any order, there are many possible graphs which can be constructed for the same problem. One might then be concerned that this could lead to multiple solution trajectories. We illustrate in figure 3 two possible graphs, which result depending on the order of construction of profiles 1 and 3.

However, the nodes and edges in our graph all have spatial meaning. Regardless of which graph is constructed, the spatial relationship between elements is the same. For example, paths [1-3-4-5-6] and [1-4-5-6] are equivalent as when we evaluate the paths monotonically from left to right, the exact same curve in the phase plane is constructed. Hence, the first path returned by any search method over the graph is the time-optimal solution. Based on the properties of profiles, the algorithms cannot generate multiple spatially-distinct paths between states in the phase plane. We do not provide a full proof here; however, we outline some of the intuition and properties of profiles.

In the first case, 1 is constructed before 3 resulting in the second graph. Navigation of 3 would halt when it intersects 1. Further expansion of a profile after it intersects a profile of the same directional type, i.e., forward or backward, is unnecessary. This is clear because profiles of the same type follow the same navigation policy and hence if two profiles share a state they will deterministically navigate the same way from the shared state regardless of how they got there. Halting a profile's navigation after like-type

intersection shrinks the amount of additional navigation and the number of intersections that would result. In the second case, 3 is constructed before 1, and so 1 would halt upon intersecting 3 generating the first graph.

Note, profiles of different type cannot intersect more than once as this would lead to a contradiction of α and β. Without loss of generality consider a forward profile which has just intersected a backward profile. The forward profile will have intersected from below and to the left of the backward profile. The profile cannot have intersected from the left and above because this would mean that $\alpha > \beta$ at this point and thus would be inadmissible. After intersection, the forward profile would necessarily have to intersect the backward profile from above which as we have just pointed out is inadmissible.

5.3 Solution Extraction

Traj_Alg With all of the hard work finished, i.e., constructing the MLP, all that Traj_Alg needs to do is expand the starting and ending states. The resulting continuity graph as described in the previous section can then be searched for the solution trajectory.

AVI_Alg Computing admissible velocity intervals requires a bit more work. There is not a single $\dot{s}$ value to start from and so there is not a single start state to expand. Different $\dot{s}_o$ values can result in different $\dot{s}_{f,max}$ and $\dot{s}_{f,min}$. We illustrate the various steps of AVI_Alg in figure 4 needed to find $\dot{s}_{f,max}$ and $\dot{s}_{f,min}$.

First, AVI_Alg will find the $\dot{s}_{f,max}$. In the ideal situation, plot 2 in figure 4 or step 1(b) in AVI_Alg 1, the MLP already intersects the starting AVI in which case we can proceed to step 1(c). However, a set of profiles, such as 1 and 2, expanded during MLP

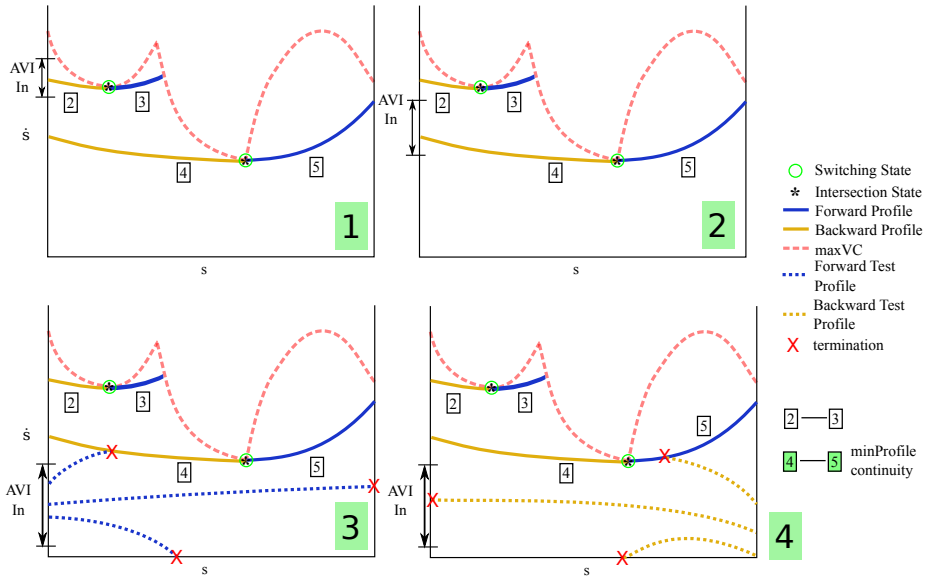

Fig. 4. Plot 1 illustrates condition 1a. Plot 2 illustrates condition 1b and 1c. Plot 3 illustrates condition 1d. Plot 4 illustrates both conditions 1(e) and 2(a) in algorithm 1.

Algorithm 1: AVI_Alg: algorithm to find the maximum and minimum reachable $\dot{s}$ at the end of the path.

1. Find the $\dot{s}_{max,f}$:
   (a) If the MLP (the set of profiles with continuity to the minProfile) reaches s_o and is below the starting AVI, return no solution exists, else go to step 1(b).
   (b) If the MLP intersects the starting AVI, go to step 1(c), else go to step 1(d).
   (c) If the MLP reaches s_f, then $\dot{s}_{max,f}$ is equivalent to this intersection and go to step 2, else go to step 1(e).
   (d) Bisection search, using forward *test profiles*, from start AVI to MLP or to highest $\dot{s}_f$. If MLP is intersected go to step 1(c), else if s_f reached then $\dot{s}_{max,f}$ has been found, else return no solution.
   (e) Bisection search, using backward *test profiles*, for highest $\dot{s}$ at $\dot{s}_f$ which intersects the MLP or reaches the start AVI. If MLP intersected or start AVI reached, $\dot{s}_{max,f}$ is found, else return no solution exists.
2. Find the $\dot{s}_{min,f}$:
   (a) Bisection search, using backward *test profiles*, to find $\dot{s}_{min,f}$ between the minVC(s_f) as a lower bound and $\dot{s}_{max,f}$ found in part 1 as the upper bound.

construction may be disconnected from the profiles which actually span the plane such as in plot 1. To deal with this situation we redefine the MLP.

We will refer to the profile, out of all profiles constructed for the MLP, with the lowest value of $\dot{s}$ along its entire length as the *minProfile*. Clearly, if the MLP is to span the plane, this profile must be part of the solution. This profile like all those generated during MLP construction originates on the maxVC. No other profile can span the s space of the minProfile without also passing through the origin switching state of the minProfile. AVI_Alg can then use the continuity data structure to find all those profiles with continuity to the minProfile which now constitute the new MLP.

If the start AVI is above the MLP or below the minVC, steps 1(a) and 1(b) in algorithm 1, then the path velocity is too fast or too slow and so AVI_Alg returns no solution exists.

If we reach step 1(d), i.e., the MLP either reaches s_o and is above the AVI input or does not reach s_o, then AVI_Alg will perform a bisection search. The bisection search is conducted over the starting AVI, i.e., at s_o, from $\dot{s}_{o,min}$ to $\dot{s}_{o,max}$ as illustrated in plot 3. Over this interval of starting states, forward *test profiles* are expanded. The nature of the termination condition for a test profile determines if the search moves up or down the $\dot{s}_o$ interval until the test $\dot{s}$ converges on some ε. If the test profile terminates on the minVC, then the test $\dot{s}$ increases as higher $\dot{s}$ are less likely to hit the minVC. Likewise, if the test profile terminates on the maxVC then $\dot{s}$ is decreased. If the test profile terminates at s_f, then the $\dot{s}$ is increased. If the MLP is intersected at any point, then AVI_Alg proceeds to step 1(c). These are all the features in the space which can cause termination and thus the search must converge with one of these final conditions. The resulting conditions and the branch they cause in the algorithm are enumerated in step 1(d).

If step 1(e) is reached, AVI_Alg has not found $s_{f,max}$ but has connected the start AVI to the MLP and the MLP does not reach s_f. Thus we perform a similar bisection search as 1(d) except at s_f. Clearly, backward test profiles are used in the place of forward test profiles. The $\dot{s}_f$ interval used for bisection spans from the $maxVC(s_f)$ to the $minVC(s_f)$.

With $\dot{s}_{f,max}$ found, the next step, 2(a), is to find $\dot{s}_{f,min}$. To do this, perform the same search as in 1(e), except instead of biasing search up after terminating at a profile which has continuity to the start, we bias our search down as the goal is to find the smallest $\dot{s}$ possible. The $\dot{s}_f$ interval used for bisection spans from the $\dot{s}_{f,max}$ to the $minVC(s_f)$.

6 Phase Plane Parameterizations

The constraints which Traj_Alg and AVI_Alg can handle is defined as those constraints which form maximum and/or minimum curves which do not lead to more than one homotopy class of solution profiles. Our algorithms currently do not handle those constraints which [11] described as "inadmissible islands," so called because all of the space around these regions is admissible isolating them from other regions of inadmissibility. All of the constraints presented here fall into the class which Traj_Alg and AVI_Alg can handle. Determining if a constraint can create "inadmissible islands" is usually straight forward as the minimum and maximum curves defined by such a constraint form an *or relationship*, i.e. a state is valid if it is above the minimum or below the maximum, instead of an *and relationship*, i.e. a state is valid if it is both above the minimum and below the maximum.

Torque Constraints For torque constraints, consider rigid body robots whose dynamics can be described as:

$$\boldsymbol{\tau}_{\mathbf{min}} \leq \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \dot{\mathbf{q}}^T \mathbf{C}(\mathbf{q})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \leq \boldsymbol{\tau}_{\mathbf{max}} \tag{2}$$

We substitue the terms from 1 into 2 and alias the terms as a, b, c as seen in [14]. We have also changed our notation so that evaluation results in a scalar value of torque on the axis of some joint i.

$$a_i(s) = \mathbf{M}_i(\mathbf{q}(s))\frac{d\mathbf{q}}{ds} \qquad b_i(s) = \mathbf{M}_i(\mathbf{q}(s))\frac{d^2\mathbf{q}}{ds^2} + \frac{d\mathbf{q}}{ds}^T \mathbf{C}_i(\mathbf{q}(s))\frac{d\mathbf{q}}{ds} \qquad c_i(s) = \mathbf{G}_i(s) \tag{3}$$

Substituting 3 back into 2 leaves us with the simplified phase plane parameterized equations of motion for a rigid body robot.

$$\tau_{i,min} \leq a_i\ddot{s} + b_i\dot{s}^2 + c_i \leq \tau_{i,max} \tag{4}$$

Note that the a_i term is the inertial component, b_i is the Coriolis component, and c_i incorporates the external force terms. Rearranging terms from 4 we get

$$\frac{\tau_{i,min} - b_i\dot{s}^2 - c_i}{a_i} \leq \ddot{s}_i \leq \frac{\tau_{i,max} - b_i\dot{s}^2 - c_i}{a_i} \tag{5}$$

taking care to note that the sign of the inertial term a will determine whether the resulting $\ddot{s}$ is that joints maximum, $\ddot{s}_{i,max}$, or minimum, $\ddot{s}_{i,min}$, since the sign of a can flip the direction of the inequalities. With the system parameterized for the phase plane, we see that it is very easy to solve for the maximum acceleration for a given state $[s, \dot{s}]$ from 5.

To compute the maxVC and minVC we must solve for when $\alpha(s,\dot{s}) \geq \beta(s,\dot{s})$. To do this we take some maximum constraint, i, and some minimum constraint, j, as defined in 5 to get:

$$\frac{-b_i\dot{s}^2 - c_i}{a_i} \leq \frac{-b_j\dot{s}^2 - c_j}{a_j} \qquad \Rightarrow \qquad \frac{-b_i\dot{s}^2}{a_i} + \frac{b_j\dot{s}^2}{a_j} \leq \frac{-c_j}{a_j} + \frac{c_i}{a_i} \tag{6}$$

Note that we have pulled the constant τ term into the constant c term. Isolating the $\dot{s}$ terms, we see that whether $\dot{s}$ is a solution to the maximum or minimum velocity curve at s is dependent upon the sign of the denominator in 7.

$$\dot{s}(s) \leq \sqrt{\left(\frac{-c_j}{a_j} + \frac{c_i}{a_i}\right) / \left(\frac{-b_i}{a_i} + \frac{b_j}{a_j}\right)} \tag{7}$$

Thus we can evaluate 7 with every combination of joints i, and joints j to find the minVC and maxVC.

Velocity Constraints Unlike torque constraints, which involve considering the full body dynamics of the robot, velocity constraints are much more straightforward. Here we repeat the constraint as seen in prior work [16]:

$$\dot{s}_{max}(s) = \min_i \dot{q}_{i,max} / \frac{dq}{ds}_i(s) \tag{8}$$

Momentum Constraints While the torque constraint parameterization appears intimidating, many constraints are very easy to parameterize. A common constraint in safety critical environments where robots are in close proximity to people or fragile hardware is the momentum constraint. Here we present a parameterization to prevent the robot from exceeding some maximum linear momentum for a given link in the system. The angular momentum is parameterized similarly. Linear momentum in a rigid body robot, for a link i, can be computed from the Jacobian and center of mass as follows

$$\mathbf{h}_i = \mathbf{v}_i m_i = \mathbf{J}_i(\mathbf{q})\dot{\mathbf{q}} m_i \tag{9}$$

Substituting 1 into 9 and solving for $\dot{s}$ gives us

$$\dot{s}_{i,max}(s) = \frac{\mathbf{h}_{i,max}}{m_i} \left(\mathbf{J}_i(\mathbf{q}(s)) \frac{d\mathbf{q}}{ds}\right)^{-1} \tag{10}$$

which we can compute for each link. As usual, we take the lowest $\dot{s}$ as the maxVC.

Workspace Proximity Constraints In the early work of [10,11], the phase plane constraints were parameterized in the context of the end effector for workspace planning and control instead of joint constraints. We extend this to a new, interesting constraint wherein a user might wish to slow the end effector or some other link when it is in proximity to some tool or object they wish it to interact with. When controlling an end effector to grasp an object, we might wish to add a threshold or some function which lowers the velocity limit as the end effector gets close to an object to improve accuracy. A hard threshold on end effector velocity when within a certain distance of an object can create jumps in the maxVC which our results show are handled without fail. We can find the maxVC for an arbitrary point on the robot by removing mass from equation 10:

$$\dot{s}_{max}(s) = \mathbf{v}_{max}(s) \left(\mathbf{J}(\mathbf{q}(s)) \frac{d\mathbf{q}}{ds}\right)^{-1} \tag{11}$$

7 Results

To evaluate our algorithms, we set up a series of simulation experiments for a 7-DOF WAM arm with the attached 4-DOF Barret hand using OpenRAVE [19] (see Fig. 5). To generate paths for the experiments, we randomly sample four configurations in

the C-space of the Barret-WAM arm. A C^2 curve is interpolated through these four configurations via piece-wise quintic spline interpolation. In this fashion, we generate a test set of 1,000 random paths. The s length of each path segment is set to one resulting in paths of length three and discretized by $\varepsilon = 0.001$. Every state along these curves is reachable quasi-statically by the Barret-WAM arm. The input AVI are also set for each curve to span from the minVC to the maxVC. Thus, if an algorithm reports the path infeasible, it is a failure of the algorithm, i.e., a false negative. To check for false positives we evaluate the solution profiles for continuity and constraint violation. Only those cases which pass both tests are counted in the success rate.

In our experiments we compare the TOPP library [14] with our algorithms. TOPP supports both joint torque and velocity constraints and so we compare our algorithms using both of these types of constraints. These limits are included in the OpenRAVE supplied model. Our algorithms generalize to a broad class of constraints. We show below results for momentum and end effector velocity constraints which are within this class. Prior methods (like TOPP) cannot handle these types of constraints, so no direct comparison is possible. For momentum, we impose linear momentum limits of $[0.75, 0.75, 0.75, 0.65, 0.65, 0.75, 0.75, 0.75]$ (kg·m/s) to links wam0 through wam7, respectively. For end effector velocity we impose a limit of 0.5 m/s on wam7's center of mass.

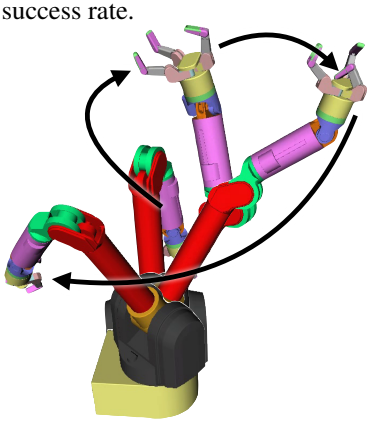

Fig. 5. A path for the Barrett-WAM arm between four random waypoints, used as input for TOPP and our algorithms.

Traj_Alg The objective of this work is to improve the success rate and generality of time-optimal trajectory generation; however, the algorithms presented also outperform the state-of-the-art method in computation time (see table 1). One would think that growing the number of potential switching states, profiles, and introducing intersection routines would decrease performance. However, the prior method, such as implemented in TOPP, expends considerable effort vetting switching states before expansion as including an unnecessary or missing a necessary switching state can result in failure. In table 1 we can see that this effect results in an over 65% improvement in the time it takes to generate the profiles which store the solution. Our graph structure also greatly speeds up the time it takes to extract the trajectory by several orders of magnitude.

Comparing the success rates, TOPP does fairly well. However, considering that these are deterministic algorithms we should expect 100% success rates. For Traj_Alg we see that the time-optimal trajectory was found for every test case.

AVI_Alg In this section we evaluate the performance of TOPP and AVI_Alg. We measure success rate, generation time, and the height of the velocity intervals found. The velocity intervals found should be fairly similar for both algorithms given the same constraints, but one would expect some loss in TOPP due to the heuristics employed.

For TOPP, we see a notable drop in the success rate for the harder AVI problem which is the focus of this work. This is mostly due to the fact that some of the heuristics

Table 1. Success rate and trajectory computation time for TOPP and Traj_Alg. The trajectory generation time can be broken down into generation time (T_g) and extraction time (T_e). Results are an average over 1000 random test cases.

Algorithm	**Constraints**	**Success (%)**	$\boldsymbol{T_g}$ **(s)**	$\boldsymbol{T_e}$ **(s)**	$\boldsymbol{T_g + T_e}$ **(s)**
TOPP	torque	98.0	0.248	0.174	0.422
TOPP	torque + velocity	99.6	0.278	0.324	0.601
Our algorithm	torque	100	0.184	0.004	0.188
Our algorithm	torque + velocity	100	0.184	0.003	0.187
Our algorithm	torque + momentum	100	0.191	0.003	0.194
Our algorithm	torque + momentum + end eff. vel.	100	0.192	0.002	0.194

Table 2. Success rate and AVI generation time for TOPP and AVI_Alg. Results are an average over 1000 random test cases. †: 3 successful cases are not enough to produce statistically significant times or ranges for comparison.

Algorithm	**Constraints**	**Success (%)**	**AVI Generation Time (s)**	**AVI Size ($\dot{s}$ units)**
TOPP	torque	90.8	0.226	0.757
TOPP	torque + velocity	0.3	†	†
Our algorithm	torque	100	0.182	0.761
Our algorithm	torque + velocity	100	0.181	0.399
Our algorithm	torque + momentum	100	0.188	0.316
Our algorithm	torque + momentum + end eff. vel.	100	0.190	0.272

used by TOPP for the point-to-point problem cannot be easily generalized to the AVI problem. Comparing tables 1 and 2, we see a drop from 98.0% to 90.8% for the cases with joint torque constraints and complete failure when joint velocity constraints are added. Ignoring the complete failure case, losing 10% of cases would be catastrophic for a kinodynamic motion planning algorithm relying on these algorithms for determining feasibility. This data suggests that the algorithm implemented in [14] is not complete.

Other arms The performance differences between TOPP and our algorithms vary based on the kinematics and dynamics of the robot, but they can be quite large. For example, for a 5-DOF arm with unit-length and unit-mass links the success rate from TOPP drops to 84.3% for trajectory generation and 52.7% for AVI computation under torque constraints of 20 units for each joint. In contrast, the success rate of our algorithms remains 100%.

Minimum Velocity Constraints: A Case Study In this section we illustrate a fully dynamic planning problem representative of the painting/welding application mentioned in the introduction. In addition to all the previous constraints, we now also impose a minimum velocity limit on the end effector of 0.05 m/s. The end effector velocity limits make sure that neither too little nor too much paint is deposited. The momentum limits ensure some measure of safety for human co-workers. The joint torque and velocity limits simply follow from the robot's dynamics constraints.

Finding a time-optimal trajectory for a given path subject to all these constraints is extremely challenging. Standard workspace time-optimal control methods that solve boundary value problems cannot be applied as we wish to follow a *given* path. Algorithms

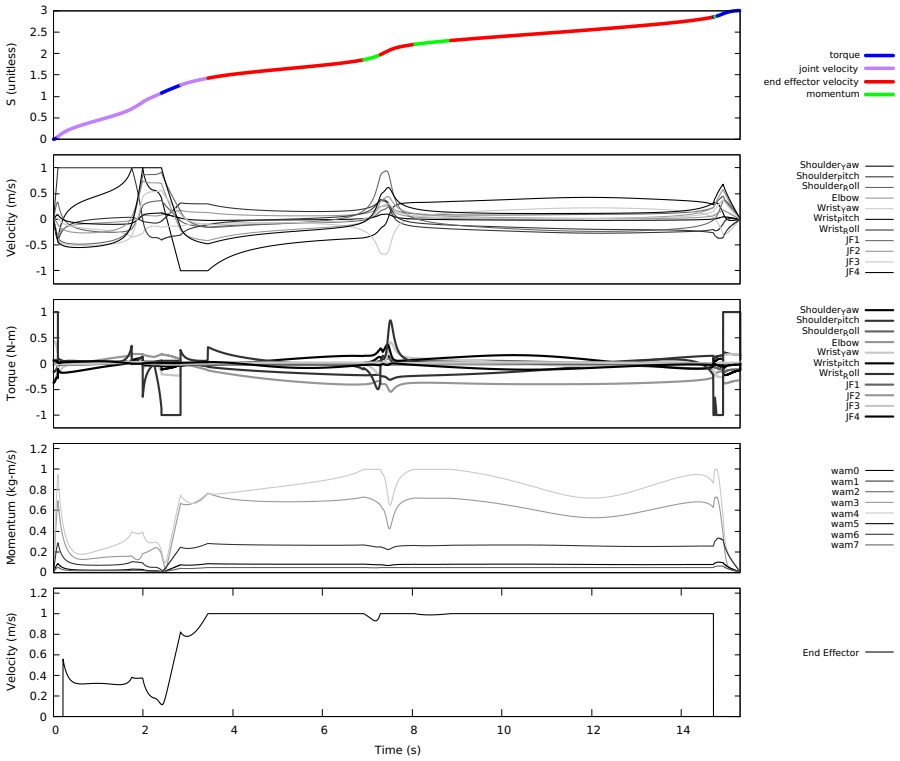

Fig. 6. Phase plane plot along with the resulting trajectories which correctly saturate the constraints. The colored time vs. s plot indicates which constraint is saturated.

like TOPP do not support minimum constraints, momentum constraints, or end effector velocity constraints. In addition, they tend to perform poorly on non-quasistatic paths. In contrast, by merely inputing the parameterizations for these constraints (discussed in Section 6) into our algorithm we are able to find time-optimal trajectories.

Figure 6 shows the trajectory generated for a given path. The plots below it show the torques and velocities for all joints, the momentum for each link, and the end effector velocity, respectively. Each curve has been normalized based on its constraint limit. A value above 1 or below -1 would indicate that a constraint is being violated. The first plot is color coded based on the active constraint at each point in time. For example, the first purple segment indicates that joint velocity is the active constraint. We can trace down to the velocity vs. time plot and see that indeed at least one joint's limit is saturated. This is indicated by the plateau at the normalized value of 1. Walking along the first plot we can see that at every time point some constraint is saturated.

8 Conclusion

We have presented novel algorithms for time-optimal trajectory generation and the computation of AVI. We have shown that they outperform classical methods in terms

of success rate and performance, and generalize to a broader class of constraints than prior work. A proof of completeness and further theoretical analysis of the algorithms are future work.

Acknowledgments Work on this paper has been supported in part by NSF 1139011, NSF 1317849, and NSF 1514372.

References

1. Pham, Q.C., Nakamura, Y.: Time-optimal path parameterization for critically dynamic motions of humanoid robots. In: IEEE-RAS Intl. Conf. on Humanoid Robots. (2012) 165–170
2. Nguyen, H., Pham, Q.: Time-optimal path parameterization of rigid-body motions: Applications to spacecraft reorientation. J. of Guidance Control and Dynamics **39**(7) (2016) 1667–1671
3. Ansari, A.R., Murphey, T.D.: Sequential action control: Closed-form optimal control for nonlinear systems. IEEE Trans. on Robotics **32**(5) (October 2016) 1196–1214
4. Berenson, D., Srinivasa, S.S., Ferguson, D., Kuffner, J.J.: Manipulation planning on constraint manifolds. In: IEEE Intl. Conf. on Robotics and Automation. (2009) 625–632
5. Jaillet, L., Porta, J.: Path planning under kinematic constraints by rapidly exploring manifolds. IEEE Trans. on Robotics **29**(1) (2013) 105–117
6. Elbanhawi, M., Simic, M.: Sampling-based robot motion planning: A review. IEEE Access **2** (2014) 56–77
7. Pham, Q.C., Caron, S., Nakamura, Y.: Kinodynamic planning in the configuration space via admissible velocity propagation. In: Robotics: Science and Systems, Berlin, Germany (2013)
8. Hsu, D., Latombe, J.C., Motwani, R.: Path planning in expansive configuration spaces. Intl. J. of Computational Geometry and Applications **9**(4-5) (1999) 495–512
9. Kuffner, J., LaValle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: Proc. 2000 IEEE Intl. Conf. on Robotics and Automation, San Francisco, CA (April 2000) 995–1001
10. Bobrow, J., Dubowsky, S., Gibson, J.: Time-optimal control of robotic manipulators along specified paths. The Intl. Journal of Robotics Research **4**(3) (1985) 3–17
11. Shin, K., McKay, N.: Minimum-time control of robotic manipulators with geometric path constraints. IEEE Trans. on Automatic Control **30**(6) (1985) 531–541
12. Kunz, T., Stilman, M.: Time-optimal trajectory generation for path following with bounded acceleration and velocity. In: Robotics: Science and Systems, Sydney, Australia (2012)
13. Hauser, K.: Fast interpolation and time-optimization on implicit contact submanifolds. In: Robotics: Science and Systems, Berlin, Germany (2013)
14. Pham, Q.C.: A general, fast, and robust implementation of the time-optimal path parameterization algorithm. IEEE Trans. on Robotics **30**(6) (2014) 1533–1540
15. Verscheure, D., Demeulenaere, B., Swevers, J., Schutter, J.D., Diehl, M.: Time-optimal path tracking for robots: A convex optimization approach. IEEE Trans. on Automatic Control **54**(10) (2009) 2318–2327
16. Žlajpah, L.: On time optimal path control of manipulators with bounded joint velocities and torques. In: IEEE Intl. Conf. on Robotics and Automation. (1996) 1572–1577
17. Shiller, Z.: On singular time-optimal control along specified paths. IEEE Trans. on Robotics and Automation **10**(4) (1994) 561–566
18. Pham, Q.C.: Characterizing and addressing dynamic singularities in the time-optimal path parameterization algorithm. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems. (2013) 2357–2363
19. Diankov, R.: Automated Construction of Robotic Manipulation Programs. PhD thesis, Carnegie Mellon University, Robotics Institute (August 2010)