Deep learning can accelerate grasp-optimized motion planning

Jeffrey Ichnowski*, Yahav Avigal, Vishal Satish, Ken Goldberg

Robots for picking in e-commerce warehouses require rapid computing of efficient and smooth robot arm motions between varying configurations. Recent results integrate grasp analysis with arm motion planning to compute optimal smooth arm motions; however, computation times on the order of tens of seconds dominate motion times. Recent advances in deep learning allow neural networks to quickly compute these motions; however, they lack the precision required to produce kinematically and dynamically feasible motions. While infeasible, the network-computed motions approximate the optimized results. The proposed method warms starts the optimization process by using the approximate motions as a starting point from which the optimizing motion planner refines to an optimized and feasible motion with few iterations. In experiments, the proposed deep learning–based warm-started optimizing motion planner reduces compute and motion time when compared to a sampling-based asymptotically optimal motion planner and an optimizing motion planner. When applied to grasp-optimized motion planning, the results suggest that deep learning can reduce the computation time by two orders of magnitude (300×), from 29 s to 80 ms, making it practical for e-commerce warehouse picking.

INTRODUCTION

The Coronavirus Disease 2019 pandemic greatly increased demand for e-commerce and reduced the ability of warehouse workers to fill orders in close proximity, driving interest in robots for order fulfillment. However, despite recent advances in grasp planning (e.g., Mahler et al. (1)), the planning and executing of robot motion remain a bottleneck. To address this, in prior work, we introduced a Grasp-Optimized Motion Planner (GOMP) (2) that computes a time-optimized motion plan (see Fig. 1) subject to joint velocity and acceleration limits and allows for degrees of freedom in the pick-and-place frames (see Fig. 2). The motions that GOMP produces are fast and smooth; however, by not taking into account the motion’s jerk (change in acceleration), the robot arm will often rapidly accelerate at the beginning of each motion and rapidly decelerate at the end. In the context of continuous pick-and-place operations in a warehouse, these high-jerk motions could result in wear on the robot’s motors and reduce the overall service life of a robot. In this paper, we introduce jerk limits and find that the resulting sequential quadratic program (SQP) and its underlying quadratic program (QP) require computation on the order of tens of seconds, which is not practical for speeding up the overall pick-and-place pipeline. We then present DJ (Deep-learning Jerk-limited)–GOMP, which uses a deep neural network to learn trajectories that warm start computation, yielding a reduction in computation times from 29 s to 80 ms, making it practical for industrial use.

For a given workcell environment, DJ-GOMP speeds up motion planning for a robot and a repeated task through a process of tasks. The first phase randomly samples tasks from the distribution of tasks the robot is likely to encounter and generates a time- and jerk-minimized motion plan using an SQP. The second phase trains a deep neural network using the data from the first phase to compute time-optimized motion plans for a given task specification (Fig. 3). The third phase, used in pick-and-place, uses the deep network from the second phase to generate a motion plan to warm start the SQP from the first phase. By warm starting the SQP from the deep network’s output, DJ-GOMP ensures that the motion plan meets the constraints of the robot (something the network cannot guarantee) and greatly accelerates the convergence rate of the SQP (something the SQP cannot do without a good initial approximation).

This paper describes algorithms and training process of DJ-GOMP. In Results, we perform experiments on a physical Universal Robotics UR5 manipulator arm, verifying that the trajectories GOMP generates are executable on a physical robot and result in fast and smooth motion. This paper provides the following contributions: (i) J-GOMP, an extension of GOMP that computes time-optimized jerk-limited motions for pick-and-place operations; (ii) DJ-GOMP, an extension of J-GOMP that uses deep learning of time-optimized motion plans that empirically speeds up the computation time of the J-GOMP optimization by two orders of magnitude (300×); (iii) comparison to optimally time-parameterized Probabilistic Road Maps “Star” (PRM*) and TrajOpt motion planners in compute and motion time suggesting that DJ-GOMP computes fast motions quickly; and (iv) experiments in simulation and on a physical UR5 robot suggesting that DJ-GOMP can be practical for reducing jerk to acceptable limits.

RESULTS

Time-optimized motion planning

We consider the problem of automating grasping and placing motions of a manipulator arm while avoiding obstacles and minimizing jerk and time. Minimizing motion time requires incorporating the robot’s velocity and acceleration limits. We cast this as an optimization problem with nonconvex constraints and compute an approximation using an SQP.

To plan a robot’s motion, we compute a trajectory $\tau$ as a sequence of configurations $(q_0, q_1, \ldots, q_n)$, in which each configuration $q_i$ is the complete specification of the robot’s degrees of freedom. Of the set of all configurations $C$, the robot is in collision for a portion $C_{\text{obs}} \subset C$. The remainder $C_{\text{free}} = C \setminus C_{\text{obs}}$ is the free space. For the motion to be valid, each configuration must be in the free space $q \in C_{\text{free}}$ and be within the joint limits $[q_{\text{min}}, q_{\text{max}}]$.

Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, USA.

*Corresponding author. Email: jeff@berkeley.edu
The motion starts with the robot’s end effector at a grasp frame \( g_0 \in SE(3) \) and ends at a place frame \( g_H \in SE(3) \). Grasps for parallel-jaw grippers have an implied degree of freedom about the axis defined by the grasp contact points. Similarly, suction-cup grippers have one about the contact normal. The implied degrees of freedom means that the start of the motion is constrained to a set \( G_0 = \{ g_i | g_i = R_c(\theta) g_0 + t, \theta \in [\theta_{\text{min}}, \theta_{\text{max}}], t \in [t_{\text{min}}, t_{\text{max}}] \} \) where \( R_c(\cdot) \) is a rotation about the free axis \( c \), \( \theta_{\text{min}} \) and \( \theta_{\text{max}} \) bound the angle of rotation, and \( t_{\text{min}} \in \mathbb{R}^3 \) and \( t_{\text{max}} \in \mathbb{R}^3 \) bound the translation. The place frame may have similarly formulated, but different degrees of freedom based on packing requirements.

To be dynamically feasible, trajectories must also remain within the velocity, acceleration, and jerk limits (\( v_{\text{max}}, a_{\text{max}}, \text{ and } j_{\text{max}} \)) of the robot.

Treating \( \tau: \mathbb{R} \to C \) as a function of time and defining a function \( h:T \to \mathbb{R} \) as the duration of the trajectory, where \( T \) is the set of all trajectories, the objective of DJ-GOMP is to compute

\[
\text{argmin}_{\tau} h(\tau) \\
\text{s.t. (such that)} \quad \tau(t) \in [q_{\text{min}}, q_{\text{max}}] \cup C_{\text{free}} \quad \forall t \in [0, h(\tau)]
\]

\[
\frac{dx}{dt} \in [-v_{\text{max}}, v_{\text{max}}] \quad \forall t \in [0, h(\tau)]
\]

\[
\frac{d^2 x}{dt^2} \in [-a_{\text{max}}, a_{\text{max}}] \quad \forall t \in [0, h(\tau)]
\]

\[
\frac{d^3 x}{dt^3} \in [-j_{\text{max}}, j_{\text{max}}] \quad \forall t \in [0, h(\tau)]
\]

\[
p(\tau(0)) \in G_0
\]

\[
p(\tau(h(\tau))) \in G_H
\]

where \( p:C \to SE(3) \) is the robot’s forward kinematic function to gripper frame. In addition, should multiple trajectories satisfy the above minimization, DJ-GOMP computes a trajectory that minimizes sum-of-squared jerks over time.
Each of the time steps \( t_0 \) and place \( (t_r, \theta) \) uses Operator Splitting solver for Quadratic Program (OSQP) \((6)\) to generate a discretized trajectory. The J-GOMP optimizer is written in C++ and relies on accurately predicting the optimal time horizon and the warm-start trajectory results in a median with deep learning of 80 ms; when compared to J-GOMP, this shows two orders of magnitude improvement, an approximate 300× speedup.

To evaluate the ability of the deep-learning approach of DJ-GOMP to speed up motion planning, we computed 1000 random motion plans both without and with deep learning–based warm start and plotted the results in Fig. 5. The median compute time without deep learning is 29.0 s. Using a network to estimate the optimal time horizon, but not the trajectory, can speed up computation significantly but at a cost of increased failure rate. Using the network to both predict the time horizon and the warm-start trajectory results in a median with deep learning of 80 ms; when compared to J-GOMP, this shows two orders of magnitude improvement, an approximate 300× speedup.

To evaluate the effect on the optimality of the computed trajectories, we compared the sum-of-squared jerks between trajectories generated with the full SQP versus those generated with a warm-started prediction with the optimal horizon. We observe that more than 99% of the test trajectories are within \( 10^{-3} \) of each other, which is an error value that is within the tolerance bounds we set for the SQP optimizer. For a small fraction (less than 1%), we observe that the warm-started optimization and the full optimization find different local minima, without clear benefit to either optimization.

Because the optimality of the trajectory and the failure rate is dependent on accurately predicting the optimal time horizon of a trajectory, we separately evaluated this prediction. We observe that shorter values of the horizon lead inevitably to SQP failures, whereas longer values lead to suboptimal trajectories. Because failures are likely to be more problematic than slightly slower trajectories, we...
went down with additional training data and longer network training due to the tight constraints. In experiments, the failure rate because the optimizer cannot move the trajectory into a feasible region. The upper limit on the x axis is shown in red to highlight the difference in scale—plots (B) and (C) are magnified by two orders of magnitude.

We propose a simple heuristic to predict longer horizons. When the network predicts a horizon longer than the optimal, we observe that the optimization of trajectories with suboptimal horizon can be faster than that of the optimal horizon (shown in Fig. 5B). This is likely due to the suboptimal trajectory being less constrained and thus faster to converge. In practice, we propose that using a readily available multicore CPU to simultaneously compute multiple SQPs for different horizons around the estimated horizon would be a practical way to address the failures and suboptimal trajectories. However, if constrained to a single-core computation, using a longer horizon may also be practical because the compute time saved may be more than time saved by using the optimal horizon.

To evaluate the effect on failure rate, we recorded the number of failures with both cold-started and warm-started optimization with the optimal horizon (observing that predicting short horizon is the other source of failures). Cold-started optimizations fail 10.7%, whereas warm-started optimizations fail 5.7%. These failures occur because the optimizer cannot move the trajectory into a feasible region due to the tight constraints. In experiments, the failure rate went down with additional training data and longer network training, suggesting that further improvement is possible.

We compare compute time and motion time performance to PRM* (9, 10) and TrajOpt (3). For PRM*, we precompute graphs of 10,000, 100,000, and 1,000,000 vertices over the workspace in front of the robot. Because PRM* is an asymptotically optimal motion planner, graphs with more vertices should produce shorter paths, at the expense of longer graph search time. For TrajOpt, we configure the optimization parameters to match that of DJ-GOMP, observing that this improves success rate over the default. Straight-line initialization in TrajOpt fails in this environment due to the bin wall between the start and end configurations; whereas DJ-GOMP’s specialized obstacle model moves the trajectory out of collision, TrajOpt’s obstacle model result in linearizations that do not push the trajectory out of collision. We thus initialize TrajOpt with a trajectory above the obstacles in the workspace. Because both PRM* and TrajOpt do not directly produce time-parameterized trajectories, we use Kunz et al.’s method (11) to compute time-optimal parameterization. This time parameterization method first “rounds corners” by adding smooth rounded segments to connect the piecewise linear motion plan from PRM* before computing the optimal timing for each waypoint. Without the rounded corners, the robot would have to stop between each linear segment of the motion plan to avoid an instantaneous infinite acceleration. The radius of the corner rounding is tunable; however, rounding corners too much can result in a motion plan that collides with obstacles. This time parameterization also does not minimize or limit jerk and thus produces high jerk trajectories with peaks in the range $5 \times 10^5$ to $8 \times 10^5 \text{rad/s}^3$ (Fig. 6A), meaning that they should have an advantage in motion time over jerk-limited motions (Fig. 7). As a final step, because 180° rotated parallel jaw grasps are equivalent, we compute trajectories for each pick and place combination and select the fastest motion. The results for 1000 pick-place pairs are shown in Fig. 6. We observe that PRM* has consistent fast compute times but produces the slowest trajectories. TrajOpt is slower to compute but produces faster trajectories than PRM*. DJ-GOMP, because it directly optimizes for a time-optimal path, produces the fastest motions, whereas the deep-learning horizon prediction and warm start allow it to compute quickly despite complex constraints and result in the overall fastest combined compute and motion time.

To evaluate whether motion plans that DJ-GOMP generates work on a physical robot, we have a UR5 follow trajectories that DJ-GOMP generates. An example motion is shown in Fig. 4. The UR5 controller does not allow the robot to exceed joint limits and issues an automated emergency stop when it does. The trajectories that DJ-GOMP generates are constrained to the documented limits and thus do not cause the stop. However, we have observed that, without jerk limits, a high-jerk trajectory can cause the UR5 to overshoot its target and bounce back. With DJ-GOMP’s jerk-limited trajectories, the UR5 empirically does not overshoot.

**DISCUSSION**

Experiments suggest that warm starting the J-GOMP optimizing motion planner with an approximation from deep learning can speed up motion planning with J-GOMP by two orders of magnitude, over 300×, and compute time-optimized jerk-limited trajectories with an 80-ms median compute time. The time optimization has potential to reduce picks per hour, an important metric in warehouse packing operations, whereas the jerk limits can reduce wear on robots, leading to longer lifespans and reduced downtime.

---

The optimization smoothly varies around obstacles by performing continuous collision detection based on the spline between waypoints. To encourage continuity, we took the following steps: (i) the cold-started optimizations starts from a deterministic and smoothly varying interpolation, and (iii) using the optimal trajectories with suboptimal horizons in the training dataset. We also observe that for a given start-goal pair, there can be multiple minimum time trajectories due to discretization of time. By minimizing jerk as well, J-GOMP provides a consistent mechanism for selecting a trajectory to learn.

**Continuous learning**

In continuous operation, a system will produce trajectories that can be used to train the deep network. When running the experiments,
we found that more training data improved the predictions of the network. We hypothesize that we did not reach the limit of improvement, and continuous operation would provide a method by which additional training data can be generated. An additional benefit may come from such a feedback system. The initial training dataset that we propose is from a uniform random distribution over two volumes—the pick bin and place bin (Fig. 4). In practice, the distribution of trajectories is likely to be nonuniform, e.g., based on how objects form piles in each bin. Hence, the initial training distribution will likely be out of distribution with the system during operation, and other precomputation strategies (12) may produce a better initial results. By leveraging the data from repeated operation, the system should continue to gain data from which it can learn and thus produce better trajectories that will speed up the SQP computation.

Application to other robots and environments
We propose a system for speeding up motion planning and execution time and experimented on a UR5 robot in a pick-and-place operation. The kinematic design of this robot has favorable properties in this application and motion planning algorithm. The robot has two joints that can lift the end effector up from any configuration—with the depth map as the obstacle, this means that there will always be an obstacle-free trajectory, provided that there are a sufficient number of waypoints allocated to the trajectory to make the traversal. In addition, because of its 6-DOF (degrees of freedom) design, for any end-effector location, there exist eight analytic inverse kinematic solutions (13), allowing for rapid computation of multiple initial and final poses to seed the optimization process.

Application to robots with additional degrees of freedom would not only result in more inverse kinematic solutions but also allow the robot to have more options (in the form of different configurations) to avoid obstacles. In these cases, changes in the initial trajectory seeded to the optimization can result in the robot converging on a different homotopic path. For example, with a different obstacle environment, one seed might lead to an arm going above an obstacle, whereas a different seed would lead an arm going to the side of an obstacle. We hypothesize that this could be addressed in the proposed system by having a consistent solution to seeding a trajectory—one that results in a smooth function for the deep network to approximate.

Applications to other environments would require an additional data generation and training step specific to the new condition. In the experiments, we generated training and test datasets from the same distribution. If the test dataset were to come from a different (or held out) distribution, then the resulting covariate shift would decrease performance. In practice, however, we would generate training data from the new distribution.

Speeding up other optimized motion planners
The deep learning-based warm start of the optimization used by DJ-GOMP may also help speed up other optimizing motion planners such as TrajOpt (3), Covariant Hamiltonian Optimization for Motion Planning (CHOMP) (14), Stochastic Trajectory Optimization for Motion Planning (STOMP) (15), and Incremental Trajectory Optimization for Motion Planning (ITOMP) (16), ones based on interior-point optimization (17) and gradients (18). Many of these planners already compute solutions quickly, although with increased constraints, more complex obstacle environments, or additional waypoints in the discretization, they may slow down to the point where they become impractical to use without something like the deep learning-based warm start proposed in DJ-GOMP.

Integrated grasp and motion planning
In this paper, we explore speeding up the computation of jerk-limited motions for the pick-and-place task from GOMP in which both pick and place frames have an additional degree of freedom. The degree of freedom comes from the four degree-of-freedom representation commonly used by grasp analysis approaches such as Dexterity Network (Dex-Net) (1, 19–21), Grasp Quality Convolutional Neural Network (GG-CNN) (22), Grasp Pose Detection (GPD) (23), or Fully Convolutional GQ-CNN (FC-GQ-CNN) (24). These data-driven methods often represent grasps using a center axis (1) or rectangle formulation (25) in the image plane (e.g., from a depth camera), which results in 4 DOF (a three-dimensional translation, plus a rotation about the camera z axis). Although we use FC-GQ-CNN (24) in experiments, we propose that many grasp analysis algorithms could be incorporated into the computation and learning process. However, on the basis of the grasp analysis software and gripper, modifications to the network design may be necessary. For example, recent work exploring additional degrees of freedom for grasps (26–29) and showing that top-down grasps leave out many high quality grasps on many objects (30) may require an alternate formulation of the input to the network used for predicting the warm-start trajectory.

In future work, DJ-GOMP could be integrated with a grasp planner to optimize among multiple grasp configurations. Whether the grasp analysis method is from the first wave of grasping research based on analytic algorithms with physical models of contact dynamics and known geometry (31–34), the second wave of research based on
data-driven learning and neural networks (25, 35–41), or the recent wave of research combining the two (J), many grasp analysis methods often have the ability to generate multiple ranked candidate grasps. With multiple forward passes using the DJ-GOMP network, grasp candidates from these methods could be rapidly weighted on the basis of their potential execution speed. This would allow the combination of grasp analysis software and DJ-GOMP to rapidly determine which grasp of multiple candidates leads to the fastest motion or to perform a cost-benefit analysis—for example, trading off some reliability of the grasp for speed of motion.

**Use in other applications**
We propose and experiment with an optimizing motion planning method in the context of a repeated pick-and-place scenario, in which the optimization is slowed down because of constraints on dynamics, obstacle avoidance, and degrees of freedom at pick and place points. We envision that other scenarios may also benefit from the proposed approach, including applications in manufacturing, assembly, painting, welding, inspection, robot-assisted surgery, construction, farming, and recycling. In each of these scenarios, the constraints in the optimization would need to adapt to the task, and the inputs to the system would also vary accordingly.

**Opportunities for future research**
In future work, we will explore expanding DJ-GOMP to additional robots performing more varied tasks that would include increased variation of start and goal configurations and in more complex environments. We will also explore additional deep-learning approaches to find better approximations of the optimization process and thus allow for faster warm starting of the final optimization step of DJ-GOMP. For systems without access to a GPU or other neural network accelerator, it may be fruitful to explore other routes to compute a warm-start trajectory, e.g., different/smaller network design, or a nearest trajectory from the training dataset (42). There may be potential for using a deep learning–based warm start to speed up constrained optimizations in other fields of robotics, e.g., grasp contact models (43), task planning (44, 45), and model predictive control (46, 47)—potentially allowing such algorithms to run at interactive rates and enabling new applications.

**MATERIALS AND METHODS**
This section describes the methods in DJ-GOMP. Underlying DJ-GOMP is a jerk- and time-optimizing constrained motion planner based on an SQP. Because of the complexity of solving this SQP, computation time can far exceed the trajectory’s execution time. DJ-GOMP uses this SQP on a random set of pick-and-place inputs to generate training data (trajectories) to train a neural network. During pick-and-place operation, DJ-GOMP uses the neural network to compute an approximate trajectory for the given pick and place frames, which it then uses to warm start the SQP.

**Jerk- and time-optimized trajectory generation**
To generate a jerk- and time-optimized trajectory, DJ-GOMP extends the SQP formulated in GOMP (2). The solver for this SQP, following the method in TrajOpt (3) and summarized in Algorithm 1, starts with a discretized estimate of the trajectory as a sequence of \( H \) waypoints after the starting configuration, in which each waypoint represents the robot’s configuration \( q \), velocity \( v \), acceleration a, and jerk \( j \) at a moment in time. The waypoints are sequentially separated by \( t_{\text{step}} \) seconds. This discretization is collected into \( x^{(0)} \), where the superscript represents a refinement iteration. Thus

\[
\begin{bmatrix}
q^{(0)}_1 \\
v^{(0)}_1 \\
a^{(0)}_1 \\
j^{(0)}_1
\end{bmatrix}
= \begin{bmatrix}
q_k \\
v_k \\
a_k \\
j_k
\end{bmatrix},
\]

The choice of \( H \) and \( t_{\text{step}} \) is application specific, although in physical experiments, we set \( t_{\text{step}} \) to match (an integer multiple of) the control frequency of the robot, and we set \( H \) such that \( H \cdot t_{\text{step}} \) is an estimate of the upper bound of the minimum trajectory time for the workspace and task input distribution.

The initial value of \( x^{(0)} \) seeds (or warms up) the SQP computation. Without the approximation generated by the neural network (e.g., for training data set generation), this trajectory can be initialized to all zeros. In practice, the SQP can converge faster by first computing a trajectory between inverse kinematic solutions to \( g_0 \) and \( g_{HI} \) with only the convex kinematic and dynamic constraints (described below).

In each iteration \( k = (0,1,2, \ldots, m) \) of the SQP, DJ-GOMP linearizes the nonconvex constraints of obstacles and pick-and-place locations and solves a QP of the following form

\[
\begin{align*}
\min_{x} & \quad \frac{1}{2} x^T P x + p^T x \\
\text{s.t.} & \quad A x \leq b
\end{align*}
\]

where \( A \) defines constraints enforcing the trust region, joint limits, and dynamics, and \( P \) is defined such that \( x^T P x \) is a sum-of-squared jerks. To enforce the linearized nonconvex constraints, DJ-GOMP adds constrained nonnegative slack variables penalized using appropriate coefficients in \( p \). As DJ-GOMP iterates over the SQP, it increases the penalty term exponentially, terminating on the iteration \( m \) at which \( x^{(m)} \) meets the nonconvex constraints.

**Algorithm 1: Jerk-limited Motion Plan**

**Require:** \( x^{(0)} \) is an initial guess of the trajectory, \( h + 1 \) is the number of waypoints in \( x^{(0)} \), \( t_{\text{step}} \) is the time between each waypoint, \( g_0 \) and \( g_{HI} \) are the pick and place frames, \( \beta_{\text{shrink}} \in (0,1) \), \( \beta_{\text{grow}} > 1 \), and \( \gamma > 1 \)

1: \( \mu \leftarrow \text{initial penalty multiple} \)
2: \( \epsilon_{\text{trust}} \leftarrow \text{initial trust region size} \)
3: \( k \leftarrow 0 \)
4: \( P, p, A, b \leftarrow \text{linearize } x^{(0)} \text{ as a QP} \)
5: while \( \mu < \mu_{\text{max}} \) do
6: \( x^{(k+1)} \leftarrow \arg \min_{x} \frac{1}{2} x^T P x + p^T x \text{s.t. } A x \leq b \)
7: \( /\ast \) warm start with \( x^{(k)} \)*
8: if sufficient decrease in trajectory cost then
9: \( k \leftarrow k + 1 \)
10: /\ast \) accept trajectory */
11: \( \epsilon_{\text{trust}} \leftarrow \epsilon_{\text{trust}} \beta_{\text{grow}} \)
12: /\ast \) grow trust region */
13: \( A, b \leftarrow \text{update linearization using } x^{(k)} \)
14: if \( \epsilon_{\text{trust}} < \epsilon_{\text{min}, \text{trust}} \) then
15: \( \mu \leftarrow \gamma \mu \)
16: /\ast \) increase penalty */
17: \( \epsilon_{\text{trust}} \leftarrow \text{initial trust region size} \)
18: \( P \leftarrow \text{update penalty in QP to match } \mu \)
19: return \( x^{(k)} \)
To enforce joint limits and dynamic constraints, Algorithm 1 creates a matrix $A$ and a vector $b$ that enforce the following linear inequalities on joint limits

$$
\begin{align*}
q_{\min} & \leq q_t \leq q_{\max} \\
-v_{\max} & \leq v_t \leq v_{\max} \\
-a_{\max} & \leq a_t \leq a_{\max} \\
-j_{\max} & \leq j_t \leq j_{\max}
\end{align*}
$$

and the following equalities that enforce dynamic constraints between variables

$$
\begin{align*}
q_{t+1} = q_t + v_{\text{step}} v_t + \frac{1}{2} s_{\text{step}}^2 a_t + \frac{1}{6} s_{\text{step}}^3 j_t \\
v_{t+1} = v_t + v_{\text{step}} a_t + \frac{1}{2} t_{\text{step}}^2 j_t \\
a_{t+1} = a_t + t_{\text{step}} j_t
\end{align*}
$$

In addition, Algorithm 1 linearizes nonconvex constraints by adding slack variables to implement $L_1$ penalties. Thus, for a nonconvex constraint $g(x) \leq c$, the algorithm adds the linearization $g_0(x)$ as a constraint in the form

$$
g(x) - \mu y_j + \mu y_j^\ast \leq c
$$

where $\mu$ is the penalty, and the slack variables are constrained such that $y_j \geq 0$ and $y_j^\ast \geq 0$.

In the QP, obstacle avoidance constraints are linearized on the basis of the waypoints of the current iteration’s trajectory (Algorithm 2). To compute these constraints, the algorithm evaluates the spline

$$
q_{\text{spline}}(s; t) = q_t + s v_t + \frac{1}{2} s^2 a_t + \frac{1}{6} s^3 j_t
$$

between each pair of waypoints $(x_t, x_{t+1})$ against a depth map of obstacles to find the time $s \in [0, t_{\text{step}}]$ and corresponding configuration $q_t$ that minimizes signed distance separation from any obstacle. In this evaluation, a negative signed distance indicates that the configuration is in collision. The algorithm then uses this $q_t$ to compute a separating hyperplane in the form $n^T q + d = 0$. The hyperplane is either the top plane of the obstacle or the plane that separates $q_t$ from the nearest obstacle (see Fig. 8). By selecting the top plane of the penetrated obstacle, this pushes the trajectory above the obstacle, which is a specialization of TrajOpt’s more general obstacle avoidance approach that is useful in bin picking. By selecting the hyperplane of the nearest obstacle, the algorithm keeps the trajectory from entering the obstacle. The linearize constraint for this point is

$$
n^T J_{\text{s}}(s) x_t \leq -d - n^T p(q_t) + n^T J_{\text{s}}(s) x_t
$$

where $J_{\text{s}}$ is the Jacobian of the robot’s position at $q_t$. Because $n_t$ and $J_{\text{s}}$ are at an interpolated state between optimization variables at $x_t$ and $x_{t+1}$, linearizing this constraint requires computing the chain rule for the Jacobian

$$
J_{\text{s}} = J_{\text{s}}(q_t) J_{\text{s}}(s)
$$

where $J_{\text{s}}(q_t)$ is the Jacobian of the position at configuration $q_t$, and $J_{\text{s}}(s)$ is the Jacobian of the configuration on the spline at $s$.

We observe that linearization at each waypoint will safely avoid obstacles with a sufficient buffer around obstacles (e.g., via a Minkowski difference with obstacles); however, slight variations in pick or place frames can shift the alignment of waypoints to obstacles. This shift of alignment (see Fig. 8C) can lead to solutions that vary disproportionately to small changes in input. Although this may be acceptable in operation, it can lead to data that can be difficult for a neural network to learn.

**Algorithm 2: Linearize Obstacle-Avoidance Constraint**

1: for $t \in [0, H]$ do
2: \hspace{1em} $(n_{\text{min}}, d_{\text{min}}) \leftarrow$ linearize obstacle nearest to $p(q_t)$
3: \hspace{1em} $q_{\text{min}} \leftarrow q_t$
4: \hspace{1em} for all $s \in [0, t_{\text{step}}]$ do /* line search $s$ to desired resolution */
5: \hspace{2em} $q_s \leftarrow q_t + s v_t + \frac{1}{2} s^2 a_t + \frac{1}{6} s^3 j_t$
6: \hspace{2em} $(n_s, d_s) \leftarrow$ linearize obstacle nearest to $p(q_s)$
7: \hspace{2em} if $n_s^T p(q_s) + d_s < n_{\text{min}}^T p(q_{\text{min}}) + d_{\text{min}}$ then /* compare signed distance */
8: \hspace{3em} $(n_{\text{min}}, d_{\text{min}}, q_{\text{min}}) \leftarrow (n_s, d_s, q_s)$
9: \hspace{2em} $J_{\text{s}} \leftarrow$ Jacobian of $q_s$
10: \hspace{2em} $p_{\text{s}} \leftarrow$ Jacobian of position at $q_{\text{min}}$
11: \hspace{1em} $J_{\text{s}} \leftarrow J_{\text{s}} - J_{\text{s}} J_{\text{s}}^T$
12: \hspace{2em} $b_t \leftarrow -d_{\text{min}} - n_{\text{min}}^T p(q_{\text{min}}) + n_{\text{min}}^T J_{\text{s}} x_t - \mu y_t^\ast$ /* lower bound with slack $y_t^\ast$ */
13: \hspace{2em} Add $(n_{\text{min}}^T J_{\text{s}} x_t \geq b_t)$ and $(y_t^\ast \geq 0)$ to set of linear constraints in QP

As with GOMP, DJ-GOMP allows degrees of freedom in rotation and translation to be added to start and goal grasp frames. Adding this degree of freedom allows DJ-GOMP to take a potentially shorter path when an exact pose of the end effector is unnecessary. For example, a pick point with a parallel-jaw gripper can rotate about the axis defined by antipodal contact points (see Fig. 2), and the pick point with a suction gripper can rotate about the normal of its contact plane. Similarly, a task may allow for a place point anywhere within a bounded box. The degrees of freedom about the pick point can be optionally added as constraints that are linearized as

$$
w_{\text{min}} \leq J_{\text{s}}^T q_{\text{min}} - (g_0 - p(q_{\text{min}})) + J_0^T q_{\text{min}} \leq w_{\text{min}}
$$

where $q_{\text{min}}$ and $J_{\text{s}}$ are the configuration and Jacobian of the first waypoint in the accepted trajectory, $q_{\text{min}}$ is one of variables the QP is minimizing, and $w_{\text{min}} \leq w_{\text{max}}$ defines the twist allowed about the pick point. Applying a similar set of constraints to $g_0$ allows degrees of freedom in the place frame as well.

The SQP establishes trust regions to constrain the optimized trajectory to be within a box with extents defined by a shrinking trust region size. Because convex constraints on dynamics enforce the
relationship between configuration, velocity, and acceleration of each waypoint, we observe that trust regions only need to be defined as box bounds around one of the three for each waypoint. In experiments, we established trust regions on configurations.

Algorithm 3: Time-optimal Motion Plan

Require: $g_0$ and $g_T$ are the start and end frames,

$γ > 1$ is the search bisection ratio

1: $H_{upper} ←$ fixed or estimated upper limit of maximum time
2: $H_{lower} ← 3$
3: $v_{upper} ← ∞$ /* constraint violation */
4: while $v_{upper} >$ tolerance do /* find upper limit */
5: $\langle x_{upper}, v_{upper} \rangle ←$ call Alg. 1 with cold-start trajectory for $H_{upper}$
6: $H_{upper} ← \max(H_{upper} + 1, \lceil γ H_{upper} \rceil)$
7: while $H_{lower} < H_{upper}$ do /* search for shortest $H*$ /
8: $H_{min} ← H_{lower} + (H_{upper} - H_{lower}) / γ$
9: $\langle x_{mid}, v_{mid} \rangle ←$ call Alg. 1 with warm-start trajectory $x_{upper}$ interpolated to $H_{mid}$
10: if $v_{mid} ≤$ tolerance then
11: $(H_{upper}, x_{upper}, v_{upper}) ← (H_{mid}, x_{mid}, v_{mid})$
12: else
13: $H_{lower} ← H_{mid} + 1$
14: return $x_{upper}$

To find the minimum time-time trajectory, J-GOMP searches for the shortest jerk-minimized trajectory that solves all constraints. This search, shown in Algorithm 3, starts with a guess of $H$ and then performs an exponential search for the upper bound, followed by a binary search for the shortest $H$ by repeatedly performing the SQP of Algorithm 1. The binary search warm starts each SQP with an interpolation of the trajectory of the current upper bound of $H$. The search ends when the upper and lower bounds of $H$ are the same.

Deep learning of trajectories

To speed up motion planning, we add a deep neural network to the pipeline. This neural network treats the trajectory optimization process as a function $f_t$ to approximate

$$ f_t: SE(3) × SE(3) → \mathbb{R}^{H_t × n × 4} $$

where the output to the function are the pick and place frames, and the output is a discretized trajectory of variable length $H^*_t$ waypoints, each of which has a configuration, velocity, acceleration, and jerk for all $n$ joints of the robot. We assume that the neural network $f_t$ can only approximate $f_t$, thus $f_t(γ) = f_t(γ) + E(γ)$ for some unknown error distribution $E(γ)$. Hence, the output of $f_t$ may not be dynamically or kinematically feasible. To address this potential issue, we use the network’s output to warm start a final pass through the SQP. This process can be thought of as polishing the output of the neural network approximation to overcome any errors in the network’s output.

In this section, we describe a proposed neural network architecture, its loss function, training and testing dataset generation, and the training process. Although we posited that a more general approximation could include the amount of pick and place degrees of freedom as inputs, for brevity, we assume that $f_t$ and its neural network approximation are parameterized by a preset amount of pick and place degrees of freedom. In practice, it may also be appropriate to train multiple neural networks for different parameterizations of $f_t$.

Architecture

The deep neural network architecture we propose is depicted in Fig. 3. It consists of an input layer connected through four fully connected blocks to multiple output blocks. The input layer takes in the concatenated grasp frames $[g_0^T, g_T^T]$. Because the optimal trajectory length $H^*$ can vary, the network has multiple output heads for each of the possible values for $H^*$. To select which of the outputs to use, we use a separate classification network with two fully connected layers with one-hot encoding trained using a cross-entropy loss.

We refer to the horizon classification and multiple-output network as HYDRA (Horizon Yielding Distillation through Retained Activations) network. The network yields both an optimal horizon length and the trajectory for that horizon. To train this network (detailed below), the trajectory output layers’ activation values for horizons not in the training sample are retained using a zero gradient so as to weight the contribution of the layers during backprop to the input layers.

In experiments, a neural network with a single output head was unable to produce a consistent result for predicting varied length horizons. We conjecture that the discontinuity between trajectories of different horizon lengths made it difficult to learn. In contrast, we found that a network was able to accurately learn a function for a single horizon length but was computationally and space inefficient, because each network should be able to share information about the function between the horizons. This led to the proposed design in which a single network with multiple output heads shares weights through multiple shared input layers.

Dataset generation

We propose generating a training dataset by randomly sampling start and end pairs from the likely distribution of tasks. For example, in a warehouse pick-and-place operation, the pick frames will be constrained to a volume defined by the picking bin, and the place frames will be constrained to a volume defined by the placement or packing bin. For each random input, we generate optimized trajectories for all time horizons from $H_{max}$ to the optimal $H^*$. Although this process generates more trajectories than necessary, generating each trajectory is efficient because the optimization for a trajectory of size $H$ warm starts from the trajectory of size $H + 1$. Overall, this process is efficient and, with parallelization, can quickly generate a large training dataset.

This process can also help detect whether the analysis of the maximum trajectory duration was incorrect. If all trajectories are significantly shorter than $H_{max}$, then one may reduce the number of output heads. If any trajectory exceeds $H_{max}$, then the number of output heads can be increased.

We also note that in the case where the initial training data do not match the operational distribution of inputs, the result may be that the neural network produces suboptimal motions that are substantially, kinematically, and dynamically infeasible. In this case, the subsequent pass through the optimization (see “Fast pipeline for trajectory generation” section) will fix these errors, although with a longer computation time. We propose, in a manner similar to DAGger (48), that trajectories from operation can be continually added to the training dataset for subsequent training or refinement of the neural network.

Training

To train the network in a way that encourages matching the reference trajectory while keeping the output trajectory kinematically and dynamically feasible, we propose a multipart loss function $L$. This
loss function includes a weighted sum of MSE loss on the trajectory $\mathcal{L}_T$ a boundary loss $\mathcal{L}_B$, which enforces the correct start and end positions, and a dynamics loss $\mathcal{L}_{\text{dyn}}$ that enforces the dynamic feasibility of the trajectory. The MSE loss is the mean of the sum of squared differences of the two vector arguments: $\mathcal{L}_{\text{MSE}}(\hat{a}, a) = \frac{1}{2} \| \hat{a} - a \|^2$. The dynamics loss attempts to mimic the convex constraints of the SQP. Given the predicted trajectories $\hat{X} = (\hat{x}^{H_0}, \ldots, \hat{x}^{H_{\text{max}}})$, where $\hat{x}^h = (\hat{q}, \hat{v}, \hat{a})_{t=0}^t$ and the ground-truth trajectories from dataset generation $X = (x^H, \ldots, x^{H_{\text{max}}})$, the loss functions are

$$\mathcal{L}_T = \alpha_T \mathcal{L}_{\text{MSE}}(\hat{q}, q) + \alpha_v \mathcal{L}_{\text{MSE}}(\hat{v}, v) + \alpha_a \mathcal{L}_{\text{MSE}}(\hat{a}, a) + \alpha_j \mathcal{L}_{\text{MSE}}(\hat{j}, j)$$

$$\mathcal{L}_B = \mathcal{L}_{\text{MSE}}(\hat{q}_0, q_0) + \mathcal{L}_{\text{MSE}}(\hat{q}_H, q_H)$$

$$\mathcal{L}_{\text{dyn}} = \frac{1}{H} \sum_{h=0}^{H-1} \left[ \frac{1}{t_{\text{step}}} \left\| \dot{q}_t + \frac{1}{2} t_{\text{step}} \ddot{q}_t + \frac{1}{6} t_{\text{step}}^3 \dddot{q}_t - \dddot{q}_{t+1} \right\|^2 + \frac{1}{t_{\text{step}}} \left\| \dot{v}_t + \frac{1}{2} t_{\text{step}} \ddot{v}_t - \ddot{v}_{t+1} \right\|^2 + \frac{1}{t_{\text{step}}} \left\| \dot{a}_t + \frac{1}{2} t_{\text{step}} \ddot{a}_t - \ddot{a}_{t+1} \right\|^2 + \frac{1}{t_{\text{step}}} \left\| \frac{1}{t_{\text{step}}} (\dddot{j}_t + \dddot{j}_{t+1}) \right\|^2 \right]$$

$$\mathcal{L}_h = \alpha_T \mathcal{L}_T^h + \alpha_B \mathcal{L}_B^h + \alpha_{\text{dyn}} \mathcal{L}_{\text{dyn}}^h$$

where values of $\alpha_T = 10, \alpha_v = 1, \alpha_a = 1, \alpha_j = 1, \alpha_B = 4 \times 10^3$, and $\alpha_{\text{dyn}} = 1$ were chosen empirically. This loss is combined into a single loss for the entire network by summing the losses of all horizons while multiplying by an indicator function for the horizons that are valid

$$\mathcal{L} = \frac{1}{H_{\text{max}}} \sum_{h=H_{\text{min}}}^{H_{\text{max}}} \mathcal{L}_h |_{[H_{\text{min}}, H_{\text{max}}]}(h)$$

Because the ground-truth trajectories for horizons shorter than $H^*$ are not defined, we must ensure that some finite data are present so that the multiplication of an indicator value of 0 results in 0 (and not NaN). Multiplying by indicator function in this way results in a zero gradient for the part of the network that is not included in the trajectory data.

During training, we observed that the network would often exhibit behavior of coadaptation in which it would learn either $\mathcal{L}_T$ or $\mathcal{L}_{\text{dyn}}$ but not both. This showed up as a test loss for one going to small values, whereas the other remained high. To address this problem, we introduced dropout layers (49) with dropout probability $p_{\text{drop}} = 0.5$ between each fully connected layer, shown in Fig. 3. We annealed (50) $p_{\text{drop}}$ to 0 over the course of the training to reduce the loss further.

**Fast pipeline for trajectory generation**

The end goal of this proposed motion planning pipeline is to generate feasible, time-optimized trajectories quickly. The SQP computes feasible, time-optimized trajectories but is slow when starting from scratch. The HYDRA neural network computes trajectories quickly (e.g., the forward pass on the network in the results section requires ~1 to compute) but without guarantees on correctness. In this section, we propose combining the properties of the SQP and HYDRA into a pipeline (see Fig. 9) to get fast computation of correct trajectories by using a forward pass on the neural network to warm start the SQP.

The first step in the pipeline is to compute $H^*$, an estimate of the optimal time horizon. This requires a single forward pass through the one-hot classification network. Because predicting horizons shorter than $H^*$ result in an infeasible SQP, it can be beneficial to either compute multiple SQPs around the predicted horizon or increase the horizon if the difference in the one-hot values for $H^*$ and $H^* + 1$ is within a threshold.

The second step in the pipeline is to compute $\hat{x}^{(0)}$ to warm start the SQP. In this step, because the warm-start trajectory is close to the final trajectory and generating a smooth training dataset is not a requirement, we can speed up the SQP process by relaxing the termination conditions to the tolerances of the robot and task, e.g., terminating when the pick point (and other constraints) is within $10^{-3}$ m of the target frame, instead of the $10^{-6}$ m used in dataset generation.

We observe that symmetry in grippers, such as found in parallel and multifinger grippers, means that multiple top-down grasps can result in the same contact points [e.g., see Fig. 2 (A and D)]. In this setting, we can use $\hat{f}_h(\cdot)$ to estimate optimal horizons for all the grasp configurations and quickly select the one with the lowest horizon. Experimentally, we find that breaking ties for optimal horizons using the associated one-hot values leads to faster trajectory optimization compute times.

**Experimental hardware setup**

The experimental workspace consists of two bins position in front of a UR5 robotic arm fitted with a Robotiq 2F-85 parallel-jaw gripper. DJ-GOMP’s network is trained on inputs in which the gripper picks from the bin in front of it and places in the bin to its right while avoiding the bin wall between the pick and place points. The pick frame allows a degree of freedom in rotation about the grasp axis on the pick point and a degree in left-right and forward-back translation (but not up-down).

**Experimental procedure**

We generate uniform random pick and place points from box volumes bounded by their respective bins and with random top-down rotation of $0^\circ$ to $180^\circ$. For each pick-place pair, we compute a J-GOMP trajectory to all four combinations of their symmetric grasp points. The resulting dataset consists of 100,000 (input, trajectory) pairs × 4 for the symmetric grasps. With this dataset, we train the neural network. In experiments, we use a different set of 1000 random inputs from the same distribution to compare the time to compute an optimized trajectory with and without warm start. We run a subset of these results on the physical robot to spot check that the generated trajectories will run on the UR5.

**SUPPLEMENTARY MATERIALS**

[Link to supplementary materials]

**REFERENCES AND NOTES**


Acknowledgements: This research was performed at the AUTOLAB at UC Berkeley in collaboration with the Berkeley AI Research (BAIR) Lab, Berkeley Deep Drive (BDD), the Real-Time Intelligent Secure Execution (RISE) Lab, and the CITRIS “People and Robots” (CPAR) Initiative. We thank our colleagues who provided helpful feedback and suggestions, particularly A. Balakrishna and B. Thananjeyan. Funding: We were also supported by the Scalable Collaborative Human-Robot Learning (SCHLO) Project, a NSF National Robotics Initiative Award 1734633, and in part by donations from Google and Toyota Research Institute. Author contributions: JJ. devised the method and neural network design, designed and ran the experiments, and wrote the manuscript. YA. designed and experimented with neural network data generation and training and edited the manuscript. V.S. designed and implemented the neural network training and edited the manuscript. K.G. supervised the project, advised the design and experiments, and
edited the manuscript. **Competing interests:** J.I., Y.A., V.S., and K.G. are co-inventors on a patent application related to this work. Ambidextrous Robotics, a startup company commercializing algorithms for robot grasping, has no financial interest and played no role in the work presented in this paper; V.S. has worked there as a summer intern, and K.G. is part-time Chief Scientist there. **Data and materials availability:** All data needed to evaluate the conclusions in this paper are present in the paper. This article solely reflects the opinions and conclusions of its authors and does not reflect the views of the sponsors or their associated entities.

Deep learning can accelerate grasp-optimized motion planning

Jeffrey Ichnowski, Yahav Avigal, Vishal Satish, and Ken Goldberg

DOI: 10.1126/scirobotics.abd7710

View the article online
https://www.science.org/doi/10.1126/scirobotics.abd7710
Permissions
https://www.science.org/help/reprints-and-permissions