ARTIFICIAL INTELLIGENCE

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 warm 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 three-phase process. 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 τ as a sequence of configurations $(\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_n)$, in which each configuration $\mathbf{q}_i$ is the complete specification of the robot's degrees of freedom. Of the set of all configurations $\mathcal{C}$, the robot is in collision for a portion $\mathcal{C}_{\text{obs}} \subset \mathcal{C}$. The remainder $\mathcal{C}_{\text{free}} = \mathcal{C} \backslash \mathcal{C}_{\text{obs}}$ is the free space. For the motion to be valid, each configuration must be in the free space $\mathbf{q} \in \mathcal{C}_{\text{free}}$ and be within the joint limits $[\mathbf{q}_{\min}, \mathbf{q}_{\max}]$.

Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, USA.
*Corresponding author. Email: jeffi@berkeley.edu

Fig. 1. Grasp-optimized motion planning in action. The proposed motion planner computes a time- and jerk-optimized motion for pick-and-place operations, using a combination of deep learning and optimization. Time optimization makes the motions fast (sub-second). Jerk (change in acceleration) optimization avoids overshooting and reduces wear over long term repeated operation. For a given pair of start and end robot configurations, deep learning rapidly computes an approximation of the optimal motion that can violate motion constraints (e.g., collides with a bin, exceeds joint limits). The motion planner then feeds the approximation to an optimization process to minimize jerk and fix up the constraint violations. By using the deep-learning-based approximation, the computation time speeds up by 300×.

The motion starts with the robot's end effector at a grasp frame $\mathbf{g}_0 \in SE(3)$ and ends at a place frame $\mathbf{g}_H \in SE(3)$. 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

$$G_0 = \{\mathbf{g}_i \mid \mathbf{g}_i = R_\mathbf{c}(\theta)\,\mathbf{g}_0 + \mathbf{t}, \theta \in [\theta_{\min}, \theta_{\max}], \mathbf{t} \in [\mathbf{t}_{\min}, \mathbf{t}_{\max}]\}$$

where $R_\mathbf{c}(\cdot)$ is a rotation about the free axis $\mathbf{c}$, $\theta_{\min}$ and $\theta_{\max}$ bound the angle of rotation, and $\mathbf{t}_{\min} \in \mathbb{R}^3$ and $\mathbf{t}_{\max} \in \mathbb{R}^3$ 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 ($\mathbf{v}_{\max}$, $\mathbf{a}_{\max}$, and $\mathbf{j}_{\max}$) of the robot.

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

$$\begin{aligned}
&\underset{\tau}{\operatorname{argmin}}\; h(\tau) \\
&\text{s.t. (such that) } \tau(t) \in [\mathbf{q}_{\min}, \mathbf{q}_{\max}] \cup \mathcal{C}_{\text{free}}\ \forall t \in [0, h(\tau)] \\
&\frac{d\tau}{dt} \in [-\mathbf{V}_{\max}, \mathbf{V}_{\max}]\ \forall t \in [0, h(\tau)] \\
&\frac{d^2\tau}{dt}(t)\,[-\mathbf{a}_{\max}, \mathbf{a}_{\max}]\ \forall t \in [0, h(\tau)] \\
&\frac{d^3\tau}{dt}(t) \in [-\mathbf{j}_{\max}, \mathbf{j}_{\max}]\ \forall t \in [0, h(\tau)] \\
&p(\tau(0)) \in G_0 \\
&p(\tau(h(\tau))) \in G_H
\end{aligned}$$

where $p: \mathcal{C} \to \text{SE}(3)$ 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.

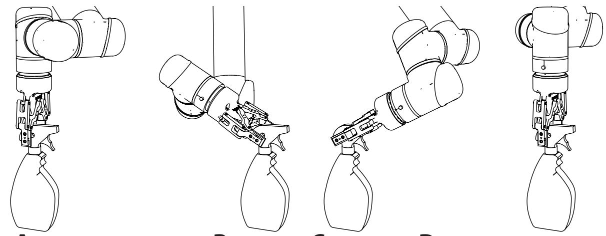

Fig. 2. Grasp-optimized motion planning degrees of freedom. The optimized motion planning allows for degrees of freedom to be added to the pick and or place frames. In (**A**), grasp analysis produces a top-down grasp. Because the analysis is based on two contact points, the motion planner allows for rotation about the grasp contact points shown as ±60° rotations in (**B**) and (**C**). Similarly, reversing the contact points, visible in (**D**) as a different arm pose, will still be valid according to grasp analysis. DJ-GOMP computes an optimal rotation for pick and place frames that minimizes time and jerk of the motion.

Computing motion plans

We propose a multistep process for computing motion plans quickly. The underlying motion planner is based on an SQP proposed in GOMP (*2*), which is a time-optimizing extension of TrajOpt (*3*) that incorporates a depth map for obstacle avoidance, degrees of freedom at pick and place points, and robot dynamic limits. In GOMP and its extensions in this work, trajectories are discretized into a sequence of $H + 1$ waypoints separated by a fixed time interval t_{step}, where t_{step} is a tunable parameter, and H is the computed (time) horizon of the motion (borrowing the term from receding horizon control methods). In this work, we extend the SQP in GOMP to include jerk limits and minimization to create J-GOMP, a jerk-limited motion planner. J-GOMP produces jerk-limited motion plans but at a substantially increased compute time.

To address the slow computation, we train a deep neural network to approximate J-GOMP. Because the network approximates J-GOMP, we use J-GOMP to generate a training dataset consisting of trajectories for random pick and place points likely to be encountered at runtime (e.g., from location in a picking bin to a location in

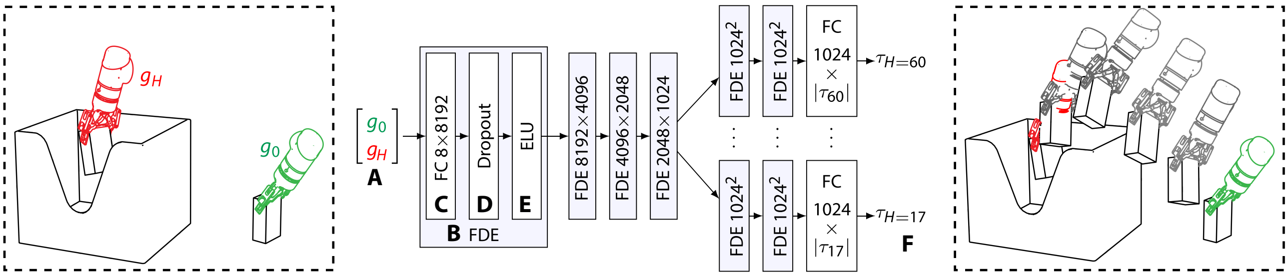

Fig. 3. A deep neural network architecture for grasp optimized motion planning. The input is the start and goal grasp frames (**A**). Each "FDE" block (**B**) sequences a fully connected (FC) layer (**C**), a dropout layer (**D**), and an exponential linear unit (ELU) layer (**E**). The output (**F**) is a trajectory τ_H from the start frame to the goal frame for each of the time steps H supported by the network. A separate network uses one-hot encoding to predict which of the output trajectories is the shortest valid trajectory.

a placement bin). With GPU (graphics processing unit)–based acceleration, the network can compute approximate trajectories in milliseconds. However, the network cannot guarantee that the trajectories it generates will be kinematically or dynamically feasible or avoid obstacles.

To fix the trajectories generated by the network, we propose using the network's trajectory to warm start the SQP from J-GOMP. The warm start allows the SQP to start from a trajectory much closer to the final solution and thus allows it to converge to an optimal solution quickly. Because the SQP enforces the pick, place, kinematic, dynamic, and obstacle constraints, the resulting trajectory is valid.

Physical experiments

We tested DJ-GOMP on a physical UR5 robot (*4*) fitted with a Robotiq 2F-85 (*5*) parallel gripper. In the experiment setup (see Fig. 4), the robot must move objects from one fixed bin location to another. We set DJ-GOMP to be constrained according to the specified joint configuration and velocity limits of the UR5. We derived an acceleration limit based on the UR5's documented torque and payload capacity, and we limited the jerk to a multiple of the computed acceleration limit. In practice, we surmise that an operator would define jerk limits by taking into account the desired service life of the robot.

To generate train/test data for the deep neural network, we use all 80 hardware threads of an NVIDIA DGX-1 to compute 100,000 optimized input and trajectory ($[\mathbf{g}_0^{\mathrm{T}} \ \ \mathbf{g}_H^{\mathrm{T}}]^{\mathrm{T}}, \mathbf{x}^*$) pairs, where $\mathbf{x}^*$ is the discretized trajectory. The J-GOMP optimizer is written in C++ and uses Operator Splitting solver for Quadratic Program (OSQP) (*6*) as the underlying QP solver. The inputs it generates consist of random pick ($\mathbf{t}_0$) and place ($\mathbf{t}_H$) translations drawn uniformly from the pick and place physical space. For each generated translation, we also generate a top-down rotation angle (θ_0 and θ_H) uniformly drawn from $(0, \pi)$. Because a parallel gripper's grasp has an equivalent, although kinematically different [see Fig. 2 (A and D)], grasp with a 180° rotation, for each translation + rotation grasp, we also add its rotation by 180°. Thus, for each random $[\mathbf{t}_0^{\mathrm{T}} \ \ \mathbf{t}_H^{\mathrm{T}}]^{\mathrm{T}}$ pair, we add four grasp frames with rotation (θ_0, θ_H), $(\theta_0 + \pi, \theta_H)$, $(\theta_0, \theta_H + \pi)$, $(\theta_0 + \pi, \theta_H + \pi)$ and their trajectories.

We trained the deep network with the Adadelta (*7*) optimizer for 50 epochs after initializing the weights using a He uniform initializer (*8*). The network architecture and optimization framework were written in Python using PyTorch. All training and deep network computations were accelerated by GPUs on NVIDIA DGX-1's Tesla V100 SXM2 GPU and Intel Xeon E5-2698 v4 central processing units (CPUs).

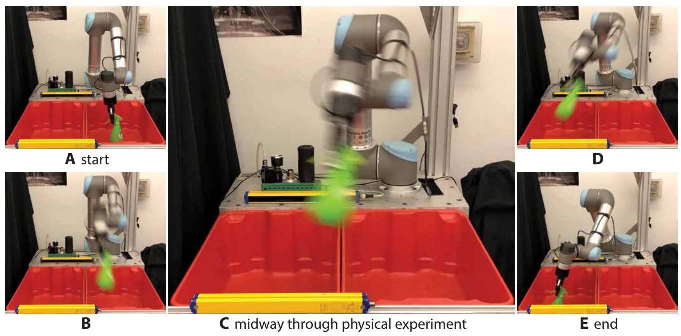

Fig. 4. Physical experiment executing jerk-limited motion computed by DJ-GOMP on a UR5. The motion starts by picking an object from the right bin (**A**), moves over the divider (**B** to **D**), and ends after placing the object in the left bin (**E**). Without the jerk limits, the motion takes 448 ms but results in a high jerk at the beginning and end of the motion, which, in this case, causes the UR5 robot to overshoot its end frame by a few millimeters. With jerk limits, the motion takes 544 ms, reduces wear, and does not overshoot the end frame.

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 plot 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 QP 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 slighty slower trajectories, we

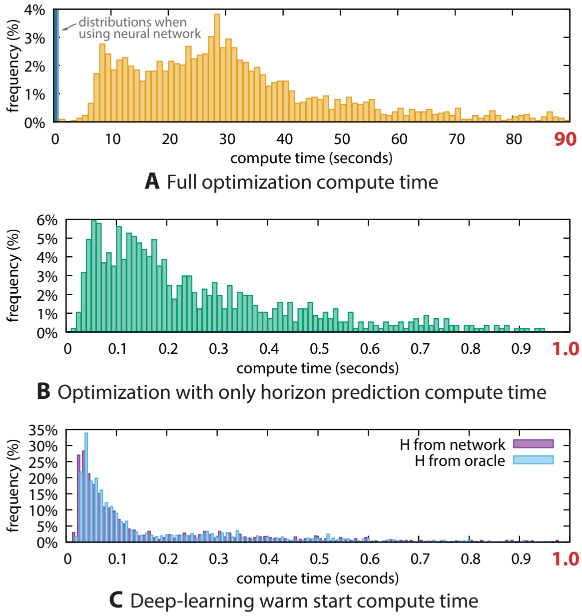

Fig. 5. Compute time distribution for 1000 random motion plans. In these plots, the *x* axis shows total compute time in seconds for a single optimized trajectory. Plot (**A**) extends to 90 s, whereas plots (**B**) and (**C**) extend to 1 s. The *y* axis shows the distribution compute time required. The full optimization process without the deep-learning prediction, shown in the histogram in (A), requires orders of magnitude longer to compute. Using a deep network to predict the optimal time horizon for a trajectory, but not warm-starting the trajectory (B), leads to improvements in compute time, although with increased failures. Using the deep network to compute a trajectory to warm start the optimization (C) further improves the compute time. In (C), the plots include results for both estimated trajectory horizon *H* and the exact *H* from the full optimization to show the effect of misprediction of trajectory length—inexact predictions can result in a faster compute time because the resulting trajectory is suboptimal, thus less tightly constrained. 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.

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 simultaneous 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 time 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 5×10^5 to 8×10^5 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 fast 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.

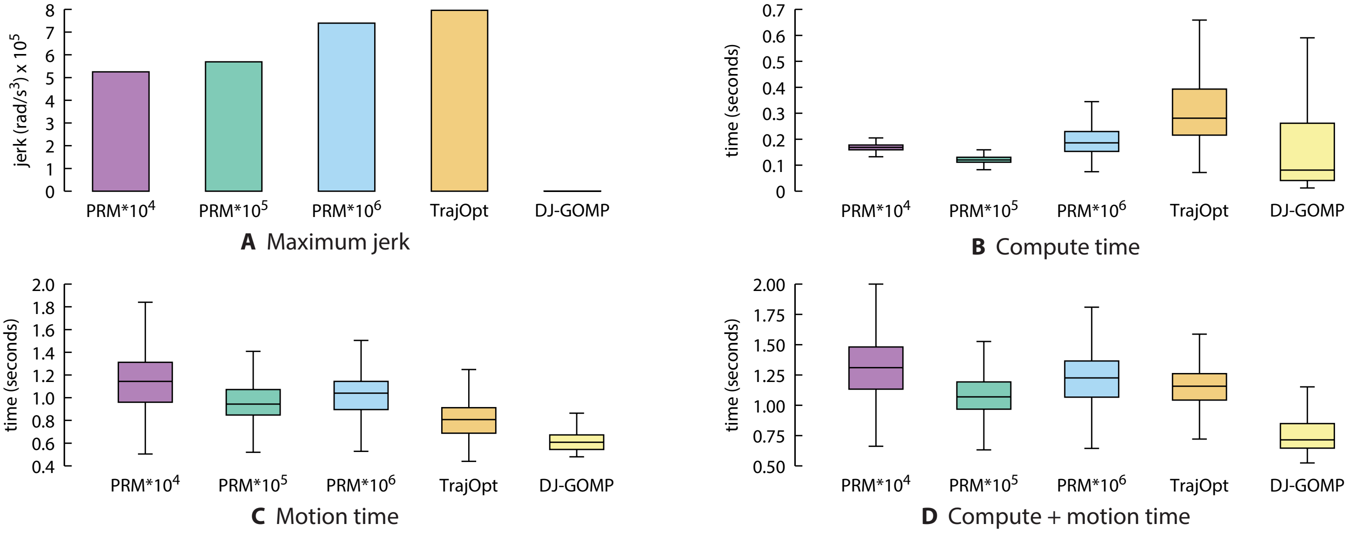

Fig. 6. Maximum jerk and timing comparisons for 1000 pick-place pairs computed with PRM*, TrajOpt, and DJ-GOMP. These graphs compare motion plan (**A**) jerk, (**B**) compute time, (**C**) motion time, and (**D**) combined compute + motion time. The filled boxes spans the first through third quartile with a horizontal line at the median. The whiskers extend from the minimum to maximum values. Paths computed by PRM* (*9*, *10*) and TrajOpt (*3*) are subsequently optimally time parameterized (*11*). The time parameterization does not limit jerk as DJ-GOMP does, which allows for faster but high jerk motions. Even so, because DJ-GOMP directly optimizes the path, unlike PRM* and TrajOpt, DJ-GOMP generates the fastest motions; whereas its deep learning–based warm start allows for fast compute and motion times.

Deep network design

The design and training of the deep network that approximates the trajectories of J-GOMP have an important impact on the performance of the overall motion planning system. Trajectories that are closer to the J-GOMP solution will take fewer optimizations iterations, whereas trajectories further from the solution will take more optimization iterations. In the method we propose, we use deep network to predict both the optimal time horizon of the trajectory and the full trajectory for any horizon. In ablation studies, we tried a policy-style network that predicts an action to take based on the current state and the goal state. By feeding each state back into the network, it computes a sequence of states that form a trajectory. This network produced less stable results and resulted in longer times to produce an optimization. Although an 80-ms median compute time may be sufficient for many applications, further improvement may be possible with different design.

The choice of loss function used in the training strongly influences the warm-start speed. Although a mean squared error (MSE) loss, because it measures the difference between training data and the network's output, should be sufficient if reduced to 0, we propose using a loss that is a weighted combination of MSE and a loss that encourages the network to produce dynamically feasible motions. Because the dynamics loss is consistent with the generated trajectories, using it did not significantly affect the reported MSE test loss but did result in trajectories that were smoother. The resulting smoothed trajectories were closer to a consistent solution and resulted in the optimizer requiring fewer SQP iterations to complete.

Training the network also benefits from a J-GOMP implementation and dataset that approaches a continuous function. Experimentally, we found that discontinuities made training the network difficult. To encourage continuity, we took the following steps: (i) The optimization smoothly varies around obstacles by performing a continuous collision detection based on the spline between waypoints, (ii) 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.

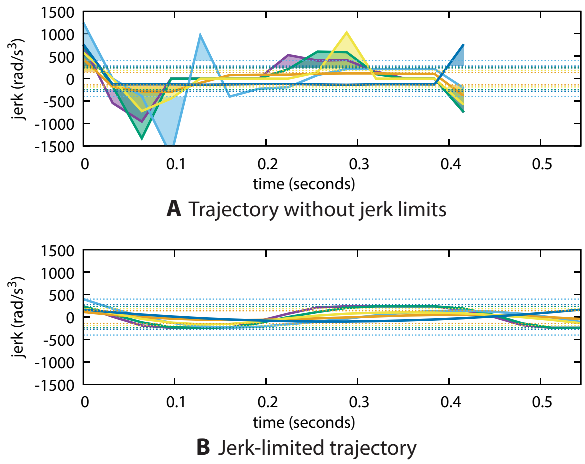

Fig. 7. Jerk limit's effect on computed and executed motion. We plot the jerk (*y* axis) of each joint in rad per cubic second over time in milliseconds (*x* axis) as computed (**A**) without jerk limits and (**B**) with jerk limits. Without jerk limits, the optimization computes trajectories with large jerks throughout the trajectory (shown in shaded regions). With jerk limits, each joint stays within the defined limits (the dotted lines) of the robot.

Continuous learning

In continuous operation, a system will produce trajectories that can be used to train the deep network. When running the experiments,

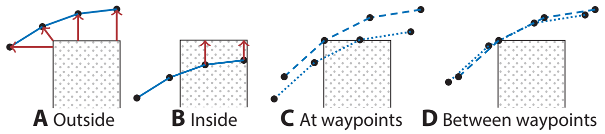

Fig. 8. Obstacle constraint linearization. The constraint linearization process keeps the trajectory away from obstacles by adding constraints based on the Jacobian of the configuration at each waypoint of the accepted trajectory $\mathbf{x}^{(k)}$. In this figure, the obstacle is shown from the side, the robot's path along part of $\mathbf{x}^{(k)}$ is shown in blue, and the constraints' normal projections into Euclidean space are shown in red. For waypoints that are outside the obstacle (**A**), constraints keep the waypoints from entering the obstacle. For waypoints that are inside the obstacle (**B**), constraints push the waypoints up out of the obstacle. If the algorithm adds constraints only at waypoints as in (**C**), the optimization can compute trajectories that collide with obstacles and produce discontinuities between trajectories with small changes to the pick or place frame. These effects are mitigated when obstacles are inflated to account for them, but the discontinuities can lead to poor results when training the neural network. The proposed algorithm adds linearized constraints to account for collision between obstacles, leading to more consistent results shown in (**D**).

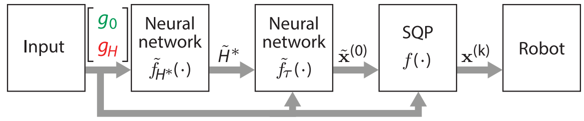

Fig. 9. The fast motion planning pipeline. The pipeline has three phases between input and robot execution. The first phase estimates the trajectory horizon H^* by computing a forward pass of the neural network. The second phase estimates the trajectories for H^* to create an initial trajectory for the SQP optimization process. The SQP then optimizes the trajectory, ensuring that it meets all joint kinematic and dynamic limits so that it can successfully execute on a robot.

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 exists 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 way points in the descretization, 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 (*1*), 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 τ as a sequence of H waypoints after the starting configuration, in which each waypoint represents the robot's configuration $\mathbf{q}$, velocity $\mathbf{v}$, acceleration $\mathbf{a}$, and jerk $\mathbf{j}$ at a moment in time. The waypoints are sequentially separated by t_{step} seconds. This discretization is collected into $\mathbf{x}^{(0)}$, where the superscript represents a refinement iteration. Thus

$$\mathbf{x}^{(0)} = (\mathbf{x}_0^{(0)}, \mathbf{x}_1^{(0)}, \ldots, \mathbf{x}_H^{(0)}), \text{ where } \mathbf{x}_t^{(k)} = \begin{bmatrix} \mathbf{q}_t^{(k)} \\ \mathbf{v}_t^{(k)} \\ \mathbf{a}_t^{(k)} \\ \mathbf{j}_t^{(k)} \end{bmatrix}$$

The choice of H and t_{step} is application specific, although in physical experiments, we set t_{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 $\mathbf{x}^{(0)}$ seeds (or warm starts) 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 $\mathbf{g}_0$ and $\mathbf{g}_H$ 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

$$\mathbf{x}^{(k+1)} = \underset{\mathbf{x}}{\operatorname{argmin}} \ \frac{1}{2}\mathbf{x}^{\top}\mathrm{P}\mathrm{x} + \mathbf{p}^{\top}\mathbf{x}$$
$$\text{s.t. } \mathrm{A}\mathrm{x} \le \mathrm{b}$$

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

Algorithm 1: Jerk-limited Motion Plan
Require: $\mathbf{x}^{(0)}$ is an initial guess of the trajectory, $h+1$ is the number of waypoints in $\mathbf{x}^{(0)}$, t_{step} is the time between each waypoint, $\mathbf{g}_0$ and $\mathbf{g}_H$ are the pick and place frames, $\beta_{\text{shrink}} \in (0,1)$, $\beta_{\text{grow}} > 1$, and $\gamma > 1$

```
μ ← initial penalty multiple
ϵ_trust ← initial trust region size
k ← 0
P, p, A, b ← linearize x^(0) as a QP
while μ < μ_max do
    x^(k+1) ← argmin_x ½ xᵀPx + pᵀx s.t. Ax ≤ b
                 /* warm start with x^(k) */
    if sufficient decrease in trajectory cost then
        k ← k + 1                  /*accept trajectory */
        ϵ_trust ← ϵ_trust β_grow    /* grow trust region */
        A, b ← update linearization using x^(k)
    else
        ϵ_trust ← ϵ_trust β_shrink  /* shrink trust region */
        b ← update trust region bounds only
    if ϵ_trust < ϵ_min_trust then
        μ ← γμ                     /* increase penalty */
        ϵ_trust ← initial trust region size
        p ← update penalty in QP to match μ
return x^(k)
```

To enforce joint limits and dynamic constraints, Algorithm 1 creates a matrix $\mathbf{A}$ and a vector $\mathbf{b}$ that enforce the following linear inequalities on joint limits

$$\begin{aligned} \mathbf{q}_{\min} &\leq \mathbf{q}_t \leq \mathbf{q}_{\max} \\ -\mathbf{v}_{\max} &\leq \mathbf{v}_t \leq \mathbf{v}_{\max} \\ -\mathbf{a}_{\max} &\leq \mathbf{a}_t \leq \mathbf{a}_{\max} \\ -\mathbf{j}_{\max} &\leq \mathbf{j}_t \leq \mathbf{j}_{\max} \end{aligned}$$

and the following equalities that enforce dynamic constraints between variables

$$\begin{aligned} \mathbf{q}_{t+1} &= \mathbf{q}_t + t_{\text{step}}\mathbf{v}_t + \frac{1}{2}t_{\text{step}}^2\mathbf{a}_t + \frac{1}{6}t_{\text{step}}^3\mathbf{j}_t \\ \mathbf{v}_{t+1} &= \mathbf{v}_t + t_{\text{step}}\mathbf{a}_t + \frac{1}{2}t_{\text{step}}^2\mathbf{j}_t \\ \mathbf{a}_{t+1} &= \mathbf{a}_t + t_{\text{step}}\mathbf{j}_t \end{aligned}$$

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

$$\bar{g}_j(\mathbf{x}) - \mu y_j^+ + \mu y_j^- \leq c$$

where μ is the penalty, and the slack variables are constrained such that $y_j^+ \geq 0$ and $y_j^- \geq 0$.

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

$$\mathbf{q}_{\text{spline}}(s; t) = \mathbf{q}_t + s\,\mathbf{v}_t + \frac{1}{2}s^2\mathbf{a}_t + \frac{1}{6}s^3\mathbf{j}_t$$

between each pair of waypoints $(\mathbf{x}_t, \mathbf{x}_{t+1})$ against a depth map of obstacles to find the time $s \in [0, t_{\text{step}})$ and corresponding configuration $\hat{\mathbf{q}}_t$ that minimizes signed distance separation from any obstacle. In this evaluation, a negative signed distance means that the configuration is in collision. The algorithm then uses this $\hat{\mathbf{q}}_t$ to computes a separating hyperplane in the form $\mathbf{n}^\mathrm{T}\mathbf{q} + d = 0$. The hyperplane is either the top plane of the obstacle it is penetrating or the plane that separates $\hat{\mathbf{q}}_t$ 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

$$\mathbf{n}^\mathrm{T}\hat{\mathbf{J}}_t^{(k)}\hat{\mathbf{x}}_t^{(k+1)} \geq -d - \mathbf{n}^\mathrm{T}p(\hat{\mathbf{x}}_t^{(k)}) + \mathbf{n}^\mathrm{T}\hat{\mathbf{J}}_t^{(k)}\hat{\mathbf{x}}_t^{(k)}$$

where $\hat{\mathbf{J}}_t$ is the Jacobian of the robot's position at $\hat{\mathbf{q}}_t$. Because $\hat{\mathbf{q}}_t$ and $\hat{\mathbf{J}}_t$ are at an interpolated state between optimization variables at $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$, linearizing this constraint requires computing the chain rule for the Jacobian

$$\hat{\mathbf{J}}_t = \mathbf{J}_p(\hat{\mathbf{q}}_t)\,\mathbf{J}_q(s)$$

where $\mathbf{J}_p(\hat{\mathbf{q}}_t)$ is the Jacobian of the position at configuration $\mathbf{q}_t$, and $\mathbf{J}_q(s)$ is the Jacobian of the configuration on the spline at s

$$\mathbf{J}_q(s) = \begin{bmatrix} \dfrac{\partial p}{\partial q_t} \\ \dfrac{\partial p}{\partial q_{t+1}} \\ \dfrac{\partial p}{\partial v_t} \\ \dfrac{\partial p}{\partial v_{t+1}} \end{bmatrix}^T = \begin{bmatrix} -3\dfrac{s^2}{t^2} + 2\dfrac{s^3}{t^3} + 1 \\ 3\dfrac{s^2}{t^2} - 2\dfrac{s^3}{t^3} \\ -2\dfrac{s^2}{t} + \dfrac{s^3}{t^3} + s \\ \dfrac{s^3}{t^2} - \dfrac{s^2}{t} \end{bmatrix}^\mathrm{T}$$

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

for $t \in [0, H)$ **do**
  $(\mathbf{n}_{\min}, d_{\min}) \leftarrow$ linearize obstacle nearest to $p(\mathbf{q}_t)$
  $\mathbf{q}_{\min} \leftarrow \mathbf{q}_t$
  for all $s \in [0, t_{\text{step}})$ **do** /* line search s to desired resolution */
    $\mathbf{q}_s \leftarrow \mathbf{q}_t + s\,\mathbf{v}_t + \frac{1}{2}s^2\mathbf{a}_t + \frac{1}{6}s^3\mathbf{j}_t$
    $(\mathbf{n}_s, d_s) \leftarrow$ linearize obstacle nearest to $p(\mathbf{q}_s)$
    if $\mathbf{n}_s^\mathrm{T}p(\mathbf{q}_s) + d_s < \mathbf{n}_{\min}^\mathrm{T}p(\mathbf{q}_{\min}) + d_{\min}$ **then** /* compare signed distance */
      $(\mathbf{n}_{\min}, d_{\min}, \mathbf{q}_{\min}) \leftarrow (\mathbf{n}_s, d_s, \mathbf{q}_s)$
      $J_q \leftarrow$ Jacobian of $\mathbf{q}_s$
  $J_p \leftarrow$ Jacobian of position at $\mathbf{q}_{\min}$
  $\hat{\mathbf{J}}_t \leftarrow \mathbf{J}_p\mathbf{J}_q$
  $b_t \leftarrow -d_{\min} - \mathbf{n}_{\min}^\mathrm{T}p(\mathbf{q}_{\min}) + \mathbf{n}_{\min}^\mathrm{T}\hat{\mathbf{J}}_t\mathbf{x}_t - \mu y_t^+$ /* lower bound with slack y_t^+ */
  Add $(\mathbf{n}_{\min}^\mathrm{T}\hat{\mathbf{J}}_t\mathbf{x}_t \geq b_t)$ and $(y_t^+ \geq 0)$ 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

$$\mathbf{w}_{\min} \leq \mathbf{J}_0^{(k)}\mathbf{q}_0^{(k+1)} - (\mathbf{g}_0 - p(\mathbf{q}_0^{(k)})) + \mathbf{J}_0^{(k)}\mathbf{q}_0^{(k)} \leq \mathbf{w}_{\min}$$

where $\mathbf{q}_0^{(k)}$ and $\mathbf{J}_0^{(k)}$ are the configuration and Jacobian of the first waypoint in the accepted trajectory, $\mathbf{q}_0^{(k+1)}$ is one of variables the QP is minimizing, and $\mathbf{w}_{\min} \leq \mathbf{w}_{\max}$ defines the twist allowed about the pick point. Applying a similar set of constraints to $\mathbf{g}_H$ 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: $\mathbf{g}_0$ and $\mathbf{g}_H$ are the start and end frames,
$\gamma > 1$ is the search bisection ratio
$H_{\text{upper}} \leftarrow$ fixed or estimated upper limit of maximum time
$H_{\text{lower}} \leftarrow 3$
$v_{\text{upper}} \leftarrow \infty$ /* constraint violation */
while $v_{\text{upper}} >$ tolerance **do** /* find upper limit */
  $(\mathbf{x}_{\text{upper}}, v_{\text{upper}}) \leftarrow$ call Alg. 1 with cold-start trajectory for H_{upper}
  $H_{\text{upper}} \leftarrow \max(H_{\text{upper}} + 1, \lceil \gamma H_{\text{upper}} \rceil)$
while $H_{\text{lower}} < H_{\text{upper}}$ **do** /* search for shortest H */
  $H_{\min} \leftarrow H_{\text{lower}} + \lfloor (H_{\text{upper}} - H_{\text{lower}})/\gamma \rfloor$
  $(\mathbf{x}_{\text{mid}}, v_{\text{mid}}) \leftarrow$ call Alg. 1 with warm-start trajectory $\mathbf{x}_{\text{upper}}$ interpolated to H_{mid}
  if $\mathbf{v}_{\text{mid}} \leq$ tolerance **then**
    $(H_{\text{upper}}, \mathbf{x}_{\text{upper}}, v_{\text{upper}}) \leftarrow (H_{\text{mid}}, \mathbf{x}_{\text{mid}}, v_{\text{mid}})$
  else
    $H_{\text{lower}} \leftarrow H_{\text{mid}} + 1$
return $\mathbf{x}_{\text{upper}}$

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_τ to approximate

$$f_\tau : \mathrm{SE}(3) \times \mathrm{SE}(3) \rightarrow \mathbb{R}^{H^* \times n \times 4}$$

where the arguments to the function are the pick and place frames, and the output is a discretized trajectory of variable length H^* waypoints, each of which has a configuration, velocity, acceleration, and jerk for all n joints of the robot. We assume that the neural network $\tilde{f}_\tau$ can only approximate f_τ, thus $\tilde{f}_\tau(\cdot) = f_\tau(\cdot) + E(\tau)$ for some unknown error distribution $E(\tau)$. Hence, the output of $\tilde{f}_\tau$ 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 posit that a more general approximation could include the amount of pick and place degrees of freedom as inputs, for brevity, we assume that f_τ 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_τ.

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 $[\mathbf{g}_0^{\mathrm{T}} \quad \mathbf{g}_H^{\mathrm{T}}]^{\mathrm{T}}$. 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 a 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 $\mathcal{L}$. This loss function includes a weighted sum of MSE loss on the trajectory $\mathcal{L}_{\mathcal{T}}$, a boundary loss $\mathcal{L}_B$, which enforces the correct start and end positions, and a dynamics loss $\mathcal{L}_{dyn}$ 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: $\mathcal{L}_{MSE}(\tilde{\mathbf{a}}, \underline{\mathbf{a}}) = \frac{1}{n}\sum_{i=1}^{n}(\tilde{\mathbf{a}}_i - \underline{\mathbf{a}}_i)^2$. The dynamics loss attempts to mimic the convex constraints of the SQP. Given the predicted trajectories $\widetilde{X} = (\tilde{\mathbf{x}}^{H_{min}}, \ldots, \tilde{\mathbf{x}}^{H_{max}})$, where $\tilde{\mathbf{x}}^h = (\tilde{\mathbf{q}}, \tilde{\mathbf{v}}, \tilde{\mathbf{a}}, \tilde{\mathbf{j}})_{t=0}^{h}$ and the ground-truth trajectories from dataset generation $\underline{X} = (\underline{\mathbf{x}}^{H^*}, \ldots, \underline{\mathbf{x}}^{H_{max}})$, the loss functions are

$$\mathcal{L}_{\mathcal{T}} = \alpha_q \mathcal{L}_{MSE}(\tilde{\mathbf{q}}, \underline{\mathbf{q}}) + \alpha_v \mathcal{L}_{MSE}(\tilde{\mathbf{v}}, \underline{\mathbf{v}}) + \alpha_a \mathcal{L}_{MSE}(\tilde{\boldsymbol{\alpha}}, \underline{\mathbf{a}}) + \alpha_j \mathcal{L}_{MSE}(\tilde{\mathbf{j}}, \underline{\mathbf{j}})$$

$$\mathcal{L}_B = \mathcal{L}_{MSE}(\tilde{\mathbf{q}}_0, \underline{\mathbf{q}}_0) + \mathcal{L}_{MSE}(\tilde{\mathbf{q}}_H, \underline{\mathbf{q}}_H)$$

$$\begin{aligned}\mathcal{L}_{dyn} = {} & \frac{1}{h}\sum_{t=0}^{h-1}\left\| \tilde{\mathbf{q}}_t + t_{step}\tilde{\mathbf{v}}_t + \frac{1}{2}t_{step}^2\tilde{\mathbf{a}}_t + \frac{1}{6}t_{step}^3\tilde{\mathbf{j}}_t - \tilde{\mathbf{q}}_{t+1}\right\|^2 \\ & + \frac{1}{h}\sum_{t=0}^{h-1}\left\| \tilde{\mathbf{v}}_t + t_{step}\tilde{\mathbf{a}}_t + \frac{1}{2}t_{step}^2\tilde{\mathbf{j}}_t - \tilde{\mathbf{v}}_{t+1}\right\|^2 \\ & + \frac{1}{h}\sum_{t=0}^{h-1}\|\tilde{\mathbf{a}}_t + t_{step}\tilde{\mathbf{j}}_t - \tilde{\mathbf{a}}_{t+1}\|^2 \\ & + \frac{1}{h}\sum_{t=0}^{h-1}\left\| \frac{1}{t_{step}}(\underline{\mathbf{j}}_{t+1} - \underline{\mathbf{j}}_t) - \frac{1}{t_{step}}(\tilde{\mathbf{j}}_{t+1} - \tilde{\mathbf{j}}_t)\right\|^2\end{aligned}$$

$$\mathcal{L}^h = \alpha_{\mathcal{T}}\mathcal{L}_{\mathcal{T}}^h + \alpha_B\mathcal{L}_B^h + \alpha_{dyn}\mathcal{L}_{dyn}^h$$

where values of $\alpha_q = 10$, $\alpha_v = 1$, $\alpha_a = 1$, $\alpha_j = 1$, $\alpha_B = 4 \times 10^3$, and $\alpha_{dyn} = 1$ 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

$$\mathcal{L} = \sum_{h=H_{min}}^{H_{max}} \mathcal{L}^h \mathbb{1}_{[\underline{H}^*, H_{max}]}(h)$$

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 $\mathcal{L}_{\mathcal{T}}$ or $\mathcal{L}_{dyn}$ 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 $p_{drop} = 0.5$ between each fully connected layer, shown in Fig. 3. We annealed (*50*) p_{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 $\widetilde{H}^*$, an estimate of the optimal time horizon H^*. 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 $\widetilde{H}^*$ and $\widetilde{H}^* + 1$ is within a threshold.

The second step in the pipeline is to compute $\tilde{\mathbf{x}}^{(0)}$, an estimate of the time-optimal trajectory for $\widetilde{H}^*$ using a forward pass through the HYDRA network.

The final step is to compute the trajectory using $\tilde{\mathbf{x}}^{(0)}$ 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 $\tilde{f}_H(\cdot)$ 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° to 180°. 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

robotics.sciencemag.org/cgi/content/full/5/48/eabd7710/DC1

Movie S1. Example of motions computed by grasp-optimized motion planning with a deep-learning warm start.

REFERENCES AND NOTES

1. J. Mahler, M. Matl, V. Satish, M. Danielczuk, B. DeRose, S. McKinley, K. Goldberg, Learning ambidextrous robot grasping policies. *Sci. Robot.* **4**, eaau4984 (2019).

2. J. Ichnowski, M. Danielczuk, J. Xu, V. Satish, K. Goldberg, "GOMP: Grasp-optimized motion planning for bin picking," in *2020 International Conference on Robotics and Automation (ICRA)* (IEEE, 2020).
3. J. Schulman, A. Lee, I. Awwal, H. Bradlow, P. Abbeel, Finding locally optimal, collision-free trajectories with sequential convex optimization. *Robot. Sci. Syst.* **9**, 1–10 (2013).
4. Universal Robotics, *UR5 Collaborative Robot Arm*, https://web.archive.org/web/20190815054522/https://www.universal-robots.com/products/ur5-robot/ [accessed 15 August 2019].
5. Robotiq, *2F-85 and 2F-140 Grippers*, https://web.archive.org/web/20190519030456/https://robotiq.com/products/2f85-140-adaptive-robot-gripper [accessed 19 May 2019].
6. B. Stellato, G. Banjac, P. Goulart, A. Bemporad, S. Boyd, OSQP: An operator splitting solver for quadratic programs. *Math. Prog. Comput.* **12**, 637–672 (2020).
7. M. D. Zeiler, ADADELTA: An adaptive learning rate method. *CoRR* 1212.5701 , (2012).
8. K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in *Proceedings of the IEEE International Conference on Computer Vision* (Santiago, Chile, 2015), pp. 1026–1034.
9. L. E. Kavraki, P. Svestka, J.-C. Latombe, M. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **12**, 566–580 (1996).
10. S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **30**, 846–894 (2011).
11. T. Kunz, M. Stilman, Time-optimal trajectory generation for path following with bounded acceleration and velocity, in *Proceedings of the 2012 Robotics: Science and Systems VIII (RSS)* (2012).
12. W. Merkt, V. Ivan, S. Vijayakumar, Leveraging precomputation with problem encoding for warm-starting trajectory optimization in complex environments, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2018), pp. 5877–5884.
13. K. P. Hawkins, Analytic inverse kinematics for the universal robots UR-5/UR-10 arms, Technical Report (Georgia Institute of Technology, 2013).
14. N. Ratliff, M. Zucker, J. A. Bagnell, S. Srinivasa, CHOMP: Gradient optimization techniques for efficient motion planning, in *2009 IEEE International Conference on Robotics and Automation* (IEEE, 2009), pp. 489–494.
15. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in *2011 IEEE International Conference on Robotics and Automation* (IEEE, 2011), pp. 4569–4574.
16. C. Park, J. Pan, D. Manocha, ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments, in *Twenty-Second International Conference on Automated Planning and Scheduling* (AAAI Press, 2012).
17. A. Kuntz, C. Bowen, R. Alterovitz, Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization, in *Robotics Research* (Springer, Cham, 2020), pp. 929–945.
18. M. Campana, F. Lamiraux, J.-P. Laumond, A gradient-based path optimization method for motion planning. *Adv. Robot.* **30**, 1126–1144 (2016).
19. J. Mahler,F. T. Pokorny, B. Hou, M. Roderick, M. Laskey, M. Aubry, K. Kohlhoff, T. Kröger, J. Kuffner, K. Goldberg, Dex-Net 1.0: A cloud-based network of 3D objects for robust grasp planning using a multi-armed bandit model with correlated rewards, in *Proceedings of IEEE International Conference in Robotics and Automation (ICRA)* (IEEE, Stockholm, Sweden, 2016), pp. 1957–1964.
20. J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, K. Goldberg, Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *Robot. Sci. Syst.* 1703.09312v3 , (2017).
21. J. Mahler, K. Goldberg, Learning deep policies for robot bin picking by simulating robust grasping sequences, in *Proceeding of the 1st Annual Conference on Robot Learning (CoRL)* (2017), pp. 515–524.
22. D. Morrison, P. Corke, J. Leitner, Learning robust, real-time, reactive robotic grasping. *Int. J. Robot. Res.* **39**, 183–201 (2019).
23. A. ten Pas, M. Gualtieri, K. Saenko, R. Platt, Grasp pose detection in point clouds. *Int. J. Robot. Res.* **36**, 1455–1473 (2017).
24. V. Satish, J. Mahler, K. Goldberg, On-policy dataset synthesis for learning robot grasping policies using fully convolutional deep networks. *IEEE Robot. Autom. Lett.* **4**, 1357–1364 (2019).
25. I. Lenz, H. Lee, A. Saxena, Deep learning for detecting robotic grasps. *Int. J. Robot. Res.* **34**, 705–724 (2015).
26. A. Mousavian, C. Eppner, D. Fox, 6-DOF GraspNet: Variational grasp generation for object manipulation, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 2901–2910.
27. A. Murali, A. Mousavian, C. Eppner, C. Paxton, D. Fox, 6-DOF grasping for target-driven object manipulation in clutter (2019); arXiv:1912.03628 [cs.RO] (8 December 2019).
28. X. Yan, J. Hsu, M. Khansari, Y. Bai, A. Pathak, A. Gupta, J. Davidson, H. Lee, Learning 6-DOF grasping interaction via deep geometry-aware 3D representations, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 1–9.
29. M. Liu, Z. Pan, K. Xu, K. Ganguly, D. Manocha, Generating grasp poses for a high-DOF gripper using neural networks, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2019).
30. C. Eppner, A. Mousavian, D. Fox, A billion ways to grasp: An evaluation of grasp sampling schemes on a dense, physics-based grasp data set, in *International Symposium of Robotics Research (ISRR)* (2019).
31. K. Goldberg, *MIT RoboSeminar - The New Wave in Robot Grasping* (2019); https://youtu.be/ATDrSWZXuwk.
32. R. M. Murray, Z. Li, S. S. Sastry, S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation* (CRC press, 1994).
33. D. Prattichizzo, J. C. Trinkle, *Springer Handbook of Robotics* (Springer, 2016), pp. 955–988.
34. E. Rimon, J. Burdick, *The Mechanics of Robot Grasping* (Cambridge University Press, 2019).
35. E. Johns, S. Leutenegger, A. J. Davison, Deep learning a grasp function for grasping under gripper pose uncertainty. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016), pp. 4461–4468.
36. U. Viereck, A. ten Pas, K. Saenko, R. Platt, Learning a visuomotor controller for real world robotic grasping using simulated depth images, in *Proceedings of the 1st Conference on Robot Learning (CoRL)* (2017).
37. D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, S. Levine. Scalable deep reinforcement learning for vision-based robotic manipulation, in *Proceedings fo the 2nd Conference on Robot Learning (CoRL)* (2018), pp. 651–673.
38. L. Pinto, A. Gupta, Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2016), pp. 3406–3413.
39. S. Levine, P. Pastor, A. Krizhevsky, D. Quillen, Learning hand-eye coordination for robotic grasping with large-scale data collection, in *International Symposium on Experimental Robotics (ISER)* (Springer, 2016), pp. 173–184.
40. K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, V. Vanhoucke, Using simulation and domain adaptation to improve efficiency of deep robotic grasping, in *Proceedings of the IEEE InternationalConference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 4243–4250.
41. D. Kappler, J. Bohg, S. Schaal, Leveraging big data for grasp planning, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2015), pp. 4304–4311.
42. T. S. Lembono, A. Paolillo, E. Pignat, S. Calinon, Memory of motion for warm-starting trajectory optimization. *IEEE Robot. Autom. Lett.* **5**, 2594–2601 (2020).
43. T. Watanabe, T. Yoshikawa, Grasping optimization using a required external force set. *IEEE Trans. Autom. Sci. Eng.* **4**, 52–68 (2007).
44. M. Toussaint, Logic-geometric programming: An optimization-based approach to combined task and motion planning, in *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI)* (2015), pp. 1930–1936.
45. D. Hadfield-Menell, C. Lin, R. Chitnis, S. Russell, P. Abbeel, Sequential quadratic programming for task plan optimization, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (IEEE, 2016), pp. 5040–5047.
46. Y. Wang, S. Boyd, Fast model predictive control using online optimization. *IFAC Proc. Vol.* **41**, 6974–6979 (2008).
47. N. Mansard, A. DelPrete, M. Geisert, S. Tonneau, O. Stasse, Using a memory of motion to efficiently warm-start a nonlinear predictive controller, in *2018 Proceedings of theIEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2018), pp. 2986–2993.
48. S. Ross, G. Gordon, D. Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (2011), pp. 627–635.
49. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014).
50. S. J. Rennie, V. Goel, S. Thomas, Annealed dropout training of deep networks, in *2014 IEEE Spoken Language Technology Workshop (SLT)* (IEEE, 2014), pp. 159–164.

Acknowledgements: This research was performed at the AUTOLAB at UC Berkeley in affiliation 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 (SCHooL) Project, a NSF National Robotics Initiative Award 1734633, and in part by donations from Google and Toyota Research Institute. **Author contributions:** J.I. devised the method and neural network design, designed and ran the experiments, and wrote the manuscript. Y.A. 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.

Submitted 12 July 2020
Accepted 15 October 2020
Published 18 November 2020
10.1126/scirobotics.abd7710

Citation: J. Ichnowski, Y. Avigal, V. Satish, K. Goldberg, Deep learning can accelerate grasp-optimized motion planning. *Sci. Robot.* **5**, eabd7710 (2020).

Science Robotics

Deep learning can accelerate grasp-optimized motion planning

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

Sci. Robot., **5** (48), eabd7710.
DOI: 10.1126/scirobotics.abd7710

Science Robotics (ISSN 2470-9476) is published by the American Association for the Advancement of Science. 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.