PokeRRT: Poking as a Skill and Failure Recovery Tactic for Planar Non-Prehensile Manipulation

Anuj Pasricha*, Yi-Shiuan Tung, Bradley Hayes, and Alessandro Roncone

***Abstract*— In this work, we introduce *PokeRRT*, a novel motion planning algorithm that demonstrates poking as an effective non-prehensile manipulation skill to enable fast manipulation of objects and increase the size of a robot's reachable workspace. We showcase poking as a failure recovery tactic used synergistically with pick-and-place for resiliency in cases where pick-and-place initially fails or is unachievable. Our experiments demonstrate the efficiency of the proposed framework in planning object trajectories using poking manipulation in uncluttered and cluttered environments. In addition to quantitatively and qualitatively demonstrating the adaptability of *PokeRRT* to different scenarios in both simulation and real-world settings, our results show the advantages of poking over pushing and grasping in terms of success rate and task time.**

I. Introduction

Humans engage naturally in multiple forms of dexterous manipulation that involve grasping, pushing, poking, rolling, and tossing objects [1], [2]. Consequently, the development of similar functionality in autonomous machines is an essential milestone for robotics and an area of active research with fundamental work needed ahead [3]. However, the human manipulation skill that has attracted the most attention from roboticists is prehensile manipulation, or *grasping*. Manipulation by grasping is attractive primarily because, once an object is grasped, it generally does not need to be tracked over time and uncertainty on its state is reduced. However, grasping is limited in capability by i) reachability of the robot arm, ii) mechanical design limitations of the end-effector, iii) physical properties of the object being manipulated, and iv) accuracy of the perception system.

Non-prehensile manipulation (i.e., any kind of manipulation not involving grasping, hereinafter referred to as NPM) offers a complementary solution to prehensile manipulation by significantly expanding the size (intended as the set of reachable configurations) and dimensionality (intended as the number of degrees of freedom) of the operational space of even the simplest robot manipulator [4]. In other words, NPM can be used to manipulate objects when conventional grasping-based manipulation is infeasible or unnecessary. Realistic robot applications might expect the robot to operate in dense clutter, in the presence of occlusions, or in ungraspable configurations—for example, the target object is in a pose that is not directly reachable by the end-effector or the target object is too large or too heavy. These applications may result in failure modes for robot operation through traditional grasping. Consequently, in such situations it is beneficial to complement the robot's skillset with NPM primitives. Indeed, NPM can be used both as a skill and a failure recovery mechanism, which points to the versatility of this paradigm.

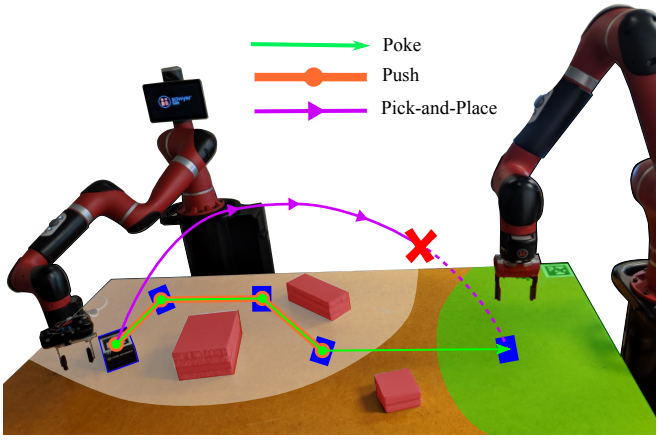

Fig. 1: This work demonstrates poking as a skill and a failure recovery tactic to increase the portfolio of capabilities at the robot's disposal. Here, an object (blue) is located in an obstacle-rich (red) workspace with non-overlapping reachable regions for each robot defined by beige and green shading. The first robot manipulates the object into the green region successfully via poking (path shown in green), but fails to do so via pushing or grasping (paths shown in orange and purple, respectively).

In this work, we demonstrate the utility of NPM through poking, a skill that allows fast object manipulation and expands the size of a manipulator's reachable workspace. The basic idea behind this work is detailed in Figure 1. *Poking* is a NPM primitive wherein a robot end-effector applies an instantaneous force to an object of interest to set the object in planar translational and rotational motion (*impact* phase). The object eventually slows down and comes to rest due to Coulomb friction (*free-sliding* phase). Poking has a multitude of desirable properties [5] that makes it complementary to grasping and serves as a generalized form of pushing where applied impulse forces are low. Advantages such as the ability of impulse forces to break static friction between the object and the surface it rests on can be desirable in manipulation scenarios where precision is important. Instantaneous impact is also an important property in the execution of certain tasks, such as striking a nail with a hammer. In this paper, we design and implement a sampling-based planner called *PokeRRT* which decouples skill modeling and path planning and specifically focuses on

[1]The authors are with the Department of Computer Science, University of Colorado Boulder, 1111 Engineering Drive, Boulder, CO USA firstname.lastname@colorado.edu

∗ Corresponding author.

leveraging the following advantages of poking over pushing and grasping: i) it does not require constant contact between the manipulator and the object, therefore greatly expanding the size of the manipulator's workspace, ii) it does not impose restrictions on the shape or size of objects that can be manipulated, and iii) it is inherently faster and therefore capable of covering large distances in short periods of time.

After discussing related work (Section II), we present *PokeRRT* which leverages a simulation model for poking to generate a planar, collision-free path between two points in the object configuration space (Section III). We conclude with an experimental validation (Section IV) and discussion (Section V) of *PokeRRT* in both simulated and real-world settings.

II. Background and Related Work

Non-prehensile manipulation planning and control: Research in non-prehensile manipulation dates back to the nineties [4], with the vast majority of prior work leveraging heuristics or analytical models based on a number of simplifying assumptions. Related work has focused on skills such as throwing [6], sliding [7], poking [8], [5], and pushing [9].

More recently, pushing manipulation has received increased attention due to availability of large-scale datasets [10] and the inherent controllability of the skill. Pushing operates under the quasistatic assumption to reduce modeling complexity, thereby limiting robot velocities and accelerations. On the contrary, poking must consider (and plan around) the non-negligible effects of inertial forces: the object continues moving after robot–object contact is broken, thus allowing for faster planar manipulation of objects. Past work in pushing manipulation incorporates simulation in a motion planning loop to get to the next feasible state [11]. Additional contributions in push modeling involve combining object state estimation with affordance prediction from image data to determine contact points for achieving the optimal push [12] and creating a deep recurrent neural network model to model push outcomes for a variety of objects [13]. However, both approaches use a greedy planner operating in obstacle-free environments. In this work, we propose a sampling-based framework that is capable of planning collision-free paths in the object configuration space.

The need for specialized impulse-delivery apparatus to achieve poking is explored in [5]. This makes planar manipulation of objects cumbersome due to the manual relocation of the apparatus required. In this work, we use a standard open-chain robot arm to deliver impulses using joint space velocity control. In order to generate impulse-based action paths that respect robot mechanics and represent feasible object motion, we use a simulation environment as the forward physics model in the planning loop.

Physics understanding for robotics: Past work on understanding physics to improve modeling and estimation can be divided into three categories: purely analytical models [14], data-driven learning approaches [15], [16], and hybrid methods [17] that combine the two. These methods are inextricably tied to the research on NPM planning detailed above, as a dynamical model is a pre-requisite of any NPM system.

Learning-based approaches for machine understanding of intuitive physics as precursors to more informed predictions are abound [18], [19]. Current approaches use data-driven methods which do not take advantage of existing empirical models of mechanics. More recent are works on combining analytical and learned models [20], [21], [6] and learning mechanical properties of objects through real-world interactions [22]. However, a majority of the techniques still rely on pushing, which can only operate within the robot's reachable workspace.

In all, evidence from prior work suggests that *poking*—sometimes referred to as "releasing" or "impulsive manipulation"—is a relatively unexplored primitive. Analytical models for poking are restricted to rotationally symmetric objects or situations where pusher design and dynamics can be geometrically modeled [5], [23]. In this work, we use PyBullet [24] as the forward physics model in our planner. While simulation may not accurately capture real-world dynamics [25], the closed-loop nature of our planning framework compensates for this approach as we explain later. This effective use of simulation captures essential characteristics of robot and object dynamics cheaply and safely.

III. Materials and Methods

In this section, we present an overview of the proposed approach for poking (Section III-A), its characterization in a simulated environment (Section III-B), and our motion planning algorithm, *PokeRRT*, that leverages the simulated environment to plan a collision-free path for the target object through its configuration space (Section III-C). We also introduce empirically-driven heuristics to *PokeRRT* that leverage the large and quick displacement property intrinsic to poking manipulation.

A. Formalization of the Poking Motion Primitive

Poking manipulation is modeled as a process composed of two phases: i) *impact*, where the robot end-effector makes instantaneous contact with the object; and ii) *free-sliding*, where the object slides on a planar surface and comes to a stop due to Coulomb friction. As detailed in Figure 2c, two parameters are required to describe the first phase of poking: the point of contact p_c (i.e., where on the contour of the object to strike), and the magnitude of the impact velocity $\|\vec{v}_{EE}\|$. Slippage at the contact point between the end-effector and the object may lead to non-linearities; therefore, we fix the direction of $\vec{v}_{EE}$ as being normal to the object's contour. Importantly, in order to apply an instantaneous force, the robot must come to a complete halt upon contact with the object. Therefore, collision between the end-effector and the object is treated as an elastic collision. The motor torques applied to stop the end-effector upon contact prevent the impulsive interaction from being truly elastic; however, this can be safely ignored by stopping the end-effector slightly past the contact point.

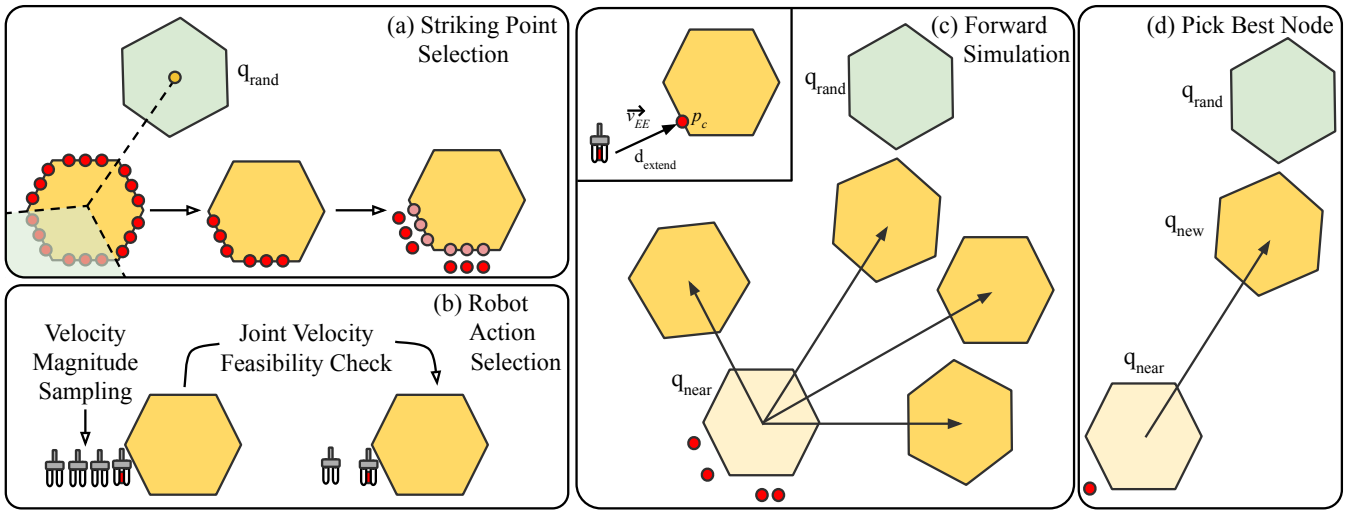

Fig. 2: Path planning for poking consists of 2 steps: action sampling (a, b) and graph expansion (c, d). (a) Points are sampled uniformly on the object contour (red) and filtered through a conical region originating from the target position (green). Striking points are generated by extending away from contour points in the normal direction. (b) End-effector velocity magnitudes are sampled for each striking point and filtered out if joint velocities are infeasible due to mechanical limitations of the robot. (c) Feasible actions are applied in simulation to get resultant poses. (d) The resultant pose closest to the target position is added to the planning graph.

Given a uniform-density object of mass m with a coefficient of friction μ whose center of mass starts in an initial planar pose $q_i = (x_i, y_i, \theta_i)$ and the control parameters (see Figure 2c) that determine the contour of the object to strike (p_c) and the velocity magnitude ($\|\vec{v}_{EE}\|$), we can solve the second phase of poking manipulation and determine the final pose of the object $q_f = (x_f, y_f, \theta_f)$ using a physics simulation engine.

B. Simulation Model for Poking

In order to understand the effects of impulsive forces on objects, we use the PyBullet physics simulation engine [24]. PyBullet models rigid body dynamics by performing numerical integration over time with equations of motion to solve for object position and velocity. Joint constraints, contact forces, and friction, in addition to external forces such as gravity, are taken into account in the forward dynamics solver of the engine and modeled as constraints in a Linear Complementarity Problem (LCP).

The main point of interest when modeling impulsive interactions is contact force. In the context of poking, contact is primarily dominated by frictional interactions between i) the robot end-effector and the object of interest, and ii) the object and the environment (i.e., the object's planar support surface and surrounding obstacles). Analytical models for contact make simplifying assumptions that do not fully represent the complexity of real-world dynamics, including the nonlinear nature of friction and actuator degradation and latency. Additionally, they do not generalize well to a diverse set of objects. Conversely, learning-based approaches, while capable of achieving generalization and modeling uncertainty and complexity, are data-inefficient. Collecting abundant and task-representative data in real-world robotics is expensive with regard to the time taken and the wear-and-tear caused on a real robot through repetitive, and potentially, high-acceleration trajectory executions such as those characteristic to poking manipulation. Executing such trajectories and modeling collisions along the action path with the object and the environment is therefore far safer in simulation.

While simulation does not perfectly capture the inherent complexity and stochasticity of real-world contact dynamics due to the simplistic nature of the underlying analytical models used, it nonetheless provide a good balance between pure learning and analytical models by ensuring interaction modeling is both cheap and safe while encapsulating the essential characteristics of robot–object and object–environment interactions. The closed-loop nature of our proposed motion planner also compensates for any inaccuracies in simulation modeling while executing poke plans in the real-world. That is, if a resultant pose violates a predefined object pose threshold, we compute a new poke plan from the current object pose to the goal region. This crucial feature allows our planner to plan feasible poke paths in simulation and execute them in the real-world.

C. Motion Planning for Poking

Using simulation as the forward physics model in our planning loop, we design and implement *PokeRRT* that takes advantage of the inherent speed and efficiency of poking manipulation. This global path planning approach leverages goal and obstacle information in object configuration space to introduce a bias into motion planning and to keep the sampling space low-dimensional to ensure fast planning.

1) Object Configuration Space: Our planning framework operates in the (x, y, θ) configuration space of the object. x and y are continuous spaces confined by the limits of the planar workspace, whereas θ is discretized in increments of $5°$ in the range $[-180°, 180°)$.

2) Action Sampling: The proposed planner decouples skill modeling and object path planning to allow for the evaluation of multiple skill models which directly improve planning outcomes. We achieve this through an action-oriented approach by employing a series of filtering steps to pick a set of valid actions $\boldsymbol{a_{valid}} = \langle \boldsymbol{p_c}, \|\vec{\boldsymbol{v}}_{\boldsymbol{EE}}\| \rangle$ to apply at a given object configuration q. A sampling approach that yields actions which can be simulated in a physics engine is desirable since inverse modeling of frictional contact and object motion may either be intractable or not guarantee feasible robot motion.

In order to achieve the correct direction of motion, candidates for object contour points that are ideal for impact must lie on the side nonadjacent to the target object position. Given an object configuration q, contour points $\boldsymbol{p_{contour}}$ are chosen as acceptable candidates if they lie within a cone originating from a target pose q_{rand} (see Figure 2a). The target pose is sampled during path planning as described in Section III-C.3. For each candidate contour point $p_c \in \boldsymbol{p_{contour}}$, a striking point p_s is computed at a fixed distance, d_{extend}, from p_c in the normal direction away from the object (see Figure 2c). The robot engages in joint space velocity control to apply a poke in its operational space by moving from p_s to p_c at velocity $\vec{v}_{EE}$. Collision-free robot joint configurations, $\boldsymbol{\theta}_s$ and $\boldsymbol{\theta}_c$, for both p_s and p_c are computed using *TRAC-IK* [26]. Since impactful contact is a core requirement for successful manipulation, we do not check for collisions between the robot end-effector and the object. A final filter is applied to see if $\dot{\boldsymbol{\theta}}_c = J^{-1}(\boldsymbol{\theta}_c)\vec{v}_{EE}$, i.e., the joint velocities at $\boldsymbol{\theta}_c$ given the given operational space velocity $\vec{v}_{EE}$, satisfies the joint velocity thresholds for the robot, $\langle \dot{\boldsymbol{\theta}}_{min}, \dot{\boldsymbol{\theta}}_{max} \rangle$ (see Figure 2b). Given this process for generating feasible actions that move the object from its current position towards the target position, we build a global path planner to generate an action path through object configuration space from the object's current pose to task goal region.

3) Path Planning: In this section, we introduce a novel motion planner named *PokeRRT*; it generates a graph of feasible robot actions in the object configuration space. With the aforementioned $\boldsymbol{a_{valid}}$ as a set of valid control parameters, exploration properties of the rapidly-exploring random tree (RRT) algorithm [27] are leveraged to generate a planning graph. The integration of poking-specific control inputs to RRT ensures that all nodes in the planning graph are achievable configurations, while the RRT algorithm ensures exploration bias to the largest Voronoi regions of the configuration space. This bias is especially central to poking manipulation due to its inherent ability to cover large distances in the operational space.

A new node q_{rand} is randomly sampled in the object configuration space with probability p_{bias}, otherwise q_{rand} is set as the goal pose. Actions from $\boldsymbol{a_{valid}}$ are applied from the nearest graph node q_{near} towards q_{rand} (see Figure 2c) and the resultant node q_{new} that minimizes the distance to q_{rand} is added to the graph (see Figure 2d). Resultant poses are added to the planning graph until the task goal region is reached, at which point the shortest path is calculated—the shortest path corresponds to one with the lowest overall number of pokes from q_{start} to q_{goal} to leverage poking's capacity to cover large distances quickly. The current implementation operates in a closed-loop paradigm—we replan on the fly if the resultant pose after a poking action is outside a specified threshold to compensate for inaccuracies of the simulation poking model. Using this planner, poking can be used both as a skill and as a failure recovery tactic and can be shown to operate faster and in a more diverse set of scenarios than pushing or grasping. Since poking is inherently capable of covering larger distances in short periods of time due to high impact interactions, we can leverage this insight and add extensions to *PokeRRT* to plan sparse paths in object configuration space.

4) Online Path Smoothing: In order to minimize the number of pokes and enable poke planning to cover large distances, we combine empirically-derived heuristics with insights from RRT* [28] to develop *PokeRRT**. RRT* allows for online path sparsification by introducing two additional steps to RRT—choosing the best parent node and rewiring—while preserving the probabilistic completeness guarantees of RRT and adding asymptotic optimality as a key feature. In RRT*, choosing the best parent node q_{best} replaces the edge from q_{near} to q_{new} with the edge from q_{best} to q_{new}. Therefore, planning time for *PokeRRT** can be reduced by using a data-driven heuristic to generate q_{new}—instead of applying actions from $\boldsymbol{a_{valid}}$ to get q_{new}, we sample Δq from a range of displacements from a dataset of random pokes. q_{new} is then computed as $q_{near} + \Delta q$ in the direction of q_{rand}.

Then q_{best} is chosen as the neighbor within radius r of q_{new} that minimizes the number of pokes from q_{start} to q_{new}. Radius r is selected empirically as the average displacement of the object in a pre-collected dataset of random pokes. In order to make sure each edge in the graph is a valid action, the best action from q_{best} to q_{new} is recomputed by sampling actions from $\boldsymbol{a_{valid}}$. Resultant pose q'_{new} is chosen as one that minimizes the distance to q_{new}. q'_{new} and the edge from q_{best} to q'_{new} are added to the graph. In the rewiring step, the neighborhood of q'_{new} is rewired to minimize the number of pokes along the path from q_{start} through q'_{new} to a neighboring node. As depicted in Figure 5, *PokeRRT** leads to fewer pokes in the planned path than *PokeRRT*, thereby taking advantage of poking's core competency of allowing larger displacements.

IV. Evaluation

In this section, we qualitatively and quantitatively evaluate *PokeRRT* and *PokeRRT** in simulation and the real-world under various environment setups. We measure success rates, task times (in seconds), and number of executed actions in the final planned path for both motion planners. Success rates are averaged across all scenarios for a given planner, whereas task times and number of executed actions are presented separately for each scenario. Task time is defined as the sum of planning, execution, and replanning times. Object start pose is kept fixed across various trials to improve

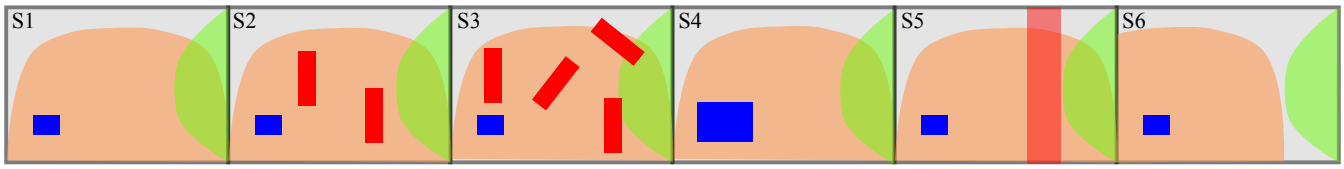

Fig. 3: The robot successfully pokes the object (blue) from its reachable workspace (orange) to the goal region (green) in all scenarios while avoiding obstacles (red). *PokeRRT*, *PokeRRT**, and baseline algorithms are evaluated in 6 scenarios—no obstacles (S1), 2 obstacles (S2), 4 obstacles (S3), wide object (S4), tunnel (S5), and non-overlapping shared workspace (S6). The robot is unable to i) push or pick-and-place in S6 due to limited robot reach, ii) push in S5 due to workspace obstruction in the action path, and iii) pick-and-place in S4 due to object being wider than gripper width. See Table I for more detail.

	Scenario Description	Poke	Push	Grasp
S1	No Obstacles	✓	✓	✓
S2	3 Obstacles	✓	✓	✓
S3	5 Obstacles	✓	✓	✓
S4	Wide Object	✓	✓	
S5	Tunnel	✓		✓
S6	Shared Workspace	✓		

TABLE I: We evaluate PokeRRT* in 6 different scenarios and demonstrate its superiority to pushing and pick-and-place in S4, S5, and S6. See Figure 3 for visualizations of these test conditions.

reproducibility of results. Simulation results are averaged over 250 trials and real-world results are averaged over 10 trials.We set up experiments using our motion planners to test the following hypotheses:

- **H1**: *PokeRRT* expands the size of the manipulator workspace and performs successfully where push-planning and grasping fail.
- **H2**: *PokeRRT* allows for faster planar manipulation of the target object than push-based planners.

Collectively, our results demonstrate that poking indeed enables fast and successful manipulation of objects in conditions where pushing and grasping fail.

A. Experimental Setup

1) Scenarios: We test multiple algorithms for poking, pushing, and grasping manipulation in the six test scenarios outlined in Table I and visualized in Figure 3. Our motivating application is the concurrent operation of two robots located in adjacent workcells on a factory floor. Reachable regions for each robot may or may not overlap. Therefore, the objects being manipulated must be accessible by both robots to enable successful collaboration. Given this motivation, each scenario in our experiment setup operates in a planar workspace shared by two robots and represents behaviors that are inherently different in nature. Scenarios are designed to test the flexibility of the planner in generating plans for manipulating the object from a fixed start pose to a goal region, which represents the workspace of a second robot. Scenarios 1-3 (S1-S3 in Figure 3) are designed to test baseline manipulation capability through uncluttered (S1) and cluttered (S2, S3) environments. S2 and S3 contain 2 and 4 obstacles, respectively, at fixed poses in the robot workspace. The object being manipulated has size $[9, 14, 5]$ cm and mass $m = 87$ g. Poking, pushing, and grasping will all work in these scenarios since the goal region overlaps with the first robot's reachable workspace. Scenario 4 (S4 in Figure 3) contains a bigger cuboid object of size $[11, 18, 11]$ cm and mass $m = 112$ g in an obstacle-free workspace. Poking and pushing will work in this scenario, however pick-and-place will not since gripper width (10 cm) is smaller than the minimum dimension of the cuboid. This scenario represents situations where object properties are incompatible with robot kinematics, i.e., if the object of interest is too heavy or too wide to be grasped or if the end-effector is malfunctional, and the robot needs to formulate an alternate manipulation plan.

Scenario 5 (S5 in Figure 3) presents two work cells with a tunnel in the center of the table. The first robot cannot push the object to the second robot because the end-effector will collide with the divider along its action path. However, poking will succeed due to the short duration of robot–object contact. The first robot is able to grasp the object over the divider in our setup but for larger dividers such as a screen, pick-and-place will fail. Scenario 6 (S6 in Figure 3) contains an obstacle-free workspace with non-overlapping reachable regions for each robot. The first robot is able to poke the target object to the second robot's reachable workspace, but cannot push or pick-and-place due to limited reachability. Since pushing operates under the quasistatic assumption, it requires constant contact between the robot end-effector and the object being manipulated and therefore, manipulation is limited by robot kinematics and reachability. This scenario represents cases where task success is limited by the robot's kinematic characteristics, i.e., if the goal pose is outside the robot's reachable workspace.

2) Parameters: The action vector $\boldsymbol{a_{valid}} = \langle \boldsymbol{p_c}, \|\vec{\boldsymbol{v}}_{\boldsymbol{EE}}\| \rangle$ for our proposed motion planners is generated at runtime. Contour points p_c close to object corners are ignored for clean pokes. The velocity magnitudes $\|\vec{v}_{EE}\|$ are sampled in the $[0.3, 1.0]$ m/s range to increase likelihood of robot joints achieving the commanded operational space velocities. Goal region is defined as the workspace of the second robot, determined as the frequency of inverse kinematics poses achievable on discretized xy-locations on the planar support surface of the object. The replanning threshold is set at $5cm$ and 10° since a higher threshold will cause the end-effector to miss contact with the object.

	Task Time [seconds]						Success Rate
Planner	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S1 - S6**
PokeRRT*	**46.43 (21.67)**	212.74 (100.01)	232.42 (118.63)	70.47 (38.82)	132.62 (98.27)	**130.01 (84.05)**	0.87 (0.31)
PokeRRT	49.69 (21.11)	**196.54 (124.61)**	**167.55 (138.63)**	**64.14 (52.45)**	**116.35 (89.72)**	171.77 (103.21)	**0.88 (0.31)**
Low-Impulse PokeRRT*	169.87 (71.28)	378.90 (122.21)	346.70 (111.71)	165.91 (44.87)	N/A	N/A	0.53 (0.28)
Low-Impulse PokeRRT	123.71 (36.72)	328.66 (107.18)	322.28 (138.05)	135.80 (32.91)	N/A	N/A	0.63 (0.18)
Two-Level Push Planner	122.68 (47.07)	284.78 (104.40)	249.32 (68.30)	117.58 (67.05)	N/A	N/A	0.44 (0.26)
Pick-and-Place	16.29 (3.76)	17.71 (3.81)	16.14 (3.25)	N/A	19.15 (4.99)	N/A	0.67 (0.00)

TABLE II: Task times for various planning algorithms in simulation are presented across multiple scenarios. Results are presented as *mean (stddev)*. Success rates are aggregated across all 6 scenarios. Overall, poking is faster than pushing and leads to higher success rate than pushing or grasping. Low-impulse poke planners and *Two-Level Push Planner* are unsuccessful in S5 and S6 due to action path collisions (S5) or goal region being outside the robot's reachable workspace (S6). The robot is unable to pick or place object in S4 and S6 due to limitations in robot mechanics.

3) Algorithms: *PokeRRT* and *PokeRRT** are compared against several baseline approaches. To evaluate pushing, we use the *Two-Level Push Planner* presented in [11]. This work integrates simulation-based forward modeling with sampling-based motion planning to explore the space of feasible pushing actions required to get an object from start to goal. Our baseline approaches, *Low-Impulse PokeRRT* and *Low-Impulse PokeRRT**, are designed to show that poking is a more fundamental manipulation skill and encompasses pushing if the applied impulse magnitudes are kept small. They operate similarly to *PokeRRT* and *PokeRRT** but with $\|\vec{v}_{EE}\| = 0.2$ m/s to simulate pushing. Lastly, *Pick-and-Place* is performed in an open-loop manner with predefined grasps for known objects. An experimental trial fails if the planner does not find a valid plan to the goal region in 240 seconds or if the object falls off the table during execution.

B. Simulation Experiments

Table II shows the task times and success rates for *Pick-and-Place* and five non-prehensile manipulation planners—*PokeRRT**, *Low-Impulse PokeRRT**, *PokeRRT*, *Low-Impulse PokeRRT*, and *Two-Level Push Planner*. Results are averaged over 250 trials. Poking is successful in all scenarios while pushing fails in S5 and S6 and grasping fails in S4 and S6, supporting **H1**. *Two-Level Push Planner* has a low overall success rate (44%) for two main reasons: i) pushing must satisfy the quasistatic assumption by maintaining constant contact between the robot end-effector and the object—therefore, pushing does not work in S5 due to collision with the workspace divider and does not work in S6 due to limited robot reachability, and ii) the pushing action, inherently longer than the instantaneous poking, causes the end-effector to collide with obstacles in narrow spaces in S2 and S3.

Poking task times are lower than pushing task times across all scenarios. Standard deviations are high due to the sampling-based nature of our planners. The task time for S4 is greater than for S1 since the object used in S4 is not only bigger in size but also larger in mass than the object used in S1, therefore poke displacements are lower given the same contact force. *Pick-and-Place* has the lowest task time because it does not involve planning in the object configuration space—the robot moves to object pose, grasps, and moves to goal pose, so only a single action is executed. Low-impulse poke planners have large search trees due to shorter displacements of the object. This results in longer planning times and also lower success rates. Task times are higher for obstacle scenarios because the robot end-effector is more likely to run into obstacles and collision checking is a computationally expensive procedure.

	Number of Actions in Execution Path					
Planner	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**
PokeRRT*	**3.93 (0.98)**	**4.62 (1.04)**	**4.49 (1.02)**	**4.76 (1.20)**	**3.71 (1.02)**	**3.65 (1.08)**
PokeRRT	6.10 (1.96)	8.21 (2.95)	7.94 (2.08)	6.77 (2.35)	4.56 (1.27)	8.17 (2.46)
Low-Impulse PokeRRT*	16.43 (2.28)	22.27 (2.26)	19.00 (1.54)	18.54 (1.44)	N/A	N/A
Low-Impulse PokeRRT	19.08 (1.51)	19.62 (1.64)	20.74 (2.13)	19.38 (1.59)	N/A	N/A
Two-Level Push Planner	8.72 (1.42)	8.61 (1.71)	7.68 (1.19)	8.53 (1.51)	N/A	N/A
Pick-and-Place	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	N/A	1.00 (0.00)	N/A

TABLE III: Number of executed actions in planned path for various planning algorithms in simulation across multiple scenarios. Both *PokeRRT** and *Low-Impulse PokeRRT** lead to fewer actions than their respective RRT-based counterparts due to the online path smoothing approach inherent to RRT*.

Pick-and-Place always succeeds in simulation for S1, S2, S3, and S5 because there is no uncertainty in object pose so manipulation boils down to moving to a predefined grasp configuration and moving to the goal location. We intentionally set up the start and goal configurations for *Pick-and-Place* in order to create an upper bound for comparison with our planners. As expected, it fails for S4 because the object being manipulated is too wide for the gripper and it fails for S6 because the goal pose for the object is out of robot reach. Low-impulse poking has higher success rates than pushing because the shorter robot trajectories result in fewer obstacle collisions than poking. Low-impulse poking fails in S5 and S6 because the pokes are not strong enough to pass the workspace divider or cross into the second robot's reachable workspace.

Table III presents the number of executed actions in the

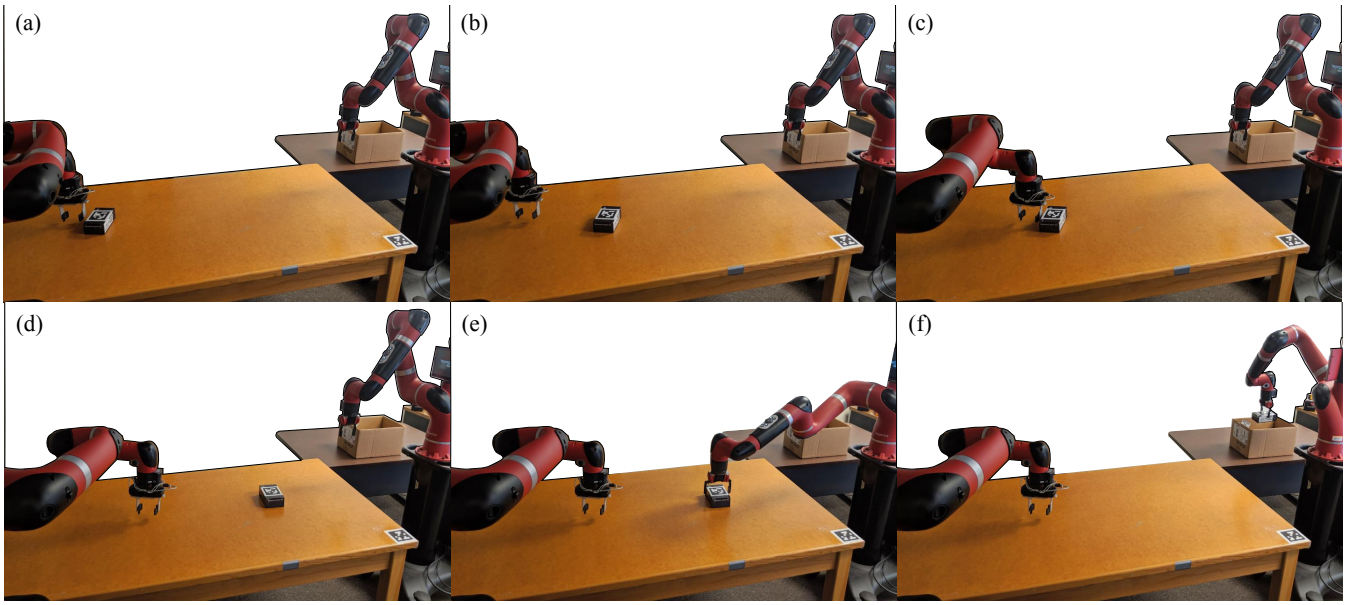

Fig. 4: Two robots with non-overlapping reachable regions are shown (Scenario 6). Robot A (left) applies 2 pokes to manipulate the object to Robot B's (right) reachable workspace (a-d). Robot B then grasps the object and places it in a bin that is not reachable by Robot A (e-f).

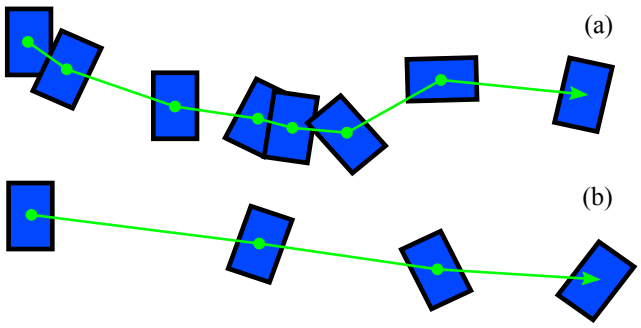

Fig. 5: Plans generated by *PokeRRT* (a) involve more pokes than those generated by *PokeRRT** (b), which indicates that *PokeRRT** is better able to exploit poking's key characteristic of covering large distances.

final planned path for multiple planning algorithms across several scenarios. It indicates that the number of executed actions is lower for poking compared to pushing, showing that poking can displace the object further with fewer actions in less time, hence supporting **H2**. Notably, the number of actions is lowest for *PokeRRT**, thereby supporting our claim that minimizing the number of poking actions exploits the large displacement property of poking manipulation (Figure 5). However, the task time is similar to that of *PokeRRT* since *PokeRRT** introduces the choose parent and rewiring steps to *PokeRRT*, which increases the number of actions sampled while planning. In general, *PokeRRT* is preferable in scenarios with obstacles—even though the number of pokes for *PokeRRT** is lower (i.e., faster overall execution), a greater percentage of time is spent on resampling actions in *PokeRRT**, most of which lead to object–obstacle collisions. Task times for low-impulse poke planners are comparable to pushing, supporting our claim that pushing is a limiting case of poking where applied impulses are low in magnitude.

C. Real-World Experiments

We evaluate *PokeRRT**, *PokeRRT*, *Two-Level Push Planner*, and *Pick-and-Place* for a subset of the scenarios listed in Table I in the real-world. As in the simulation setup, a Sawyer arm is positioned in front of a table of size $[91, 183]$ cm. We run 10 trials for each planner and each scenario (Table IV). The task times and the number of actions executed in the real world are slightly higher than in simulation. This is expected because the plan is generated in simulation and simulation does not fully capture the complexities of the real-world environment. Therefore, the number of replans are higher and thus the task times are higher. Real-world results align with the results from simulation. In S4, poking is 1.7 times faster than pushing. In S5, pushing fails because of collision with obstacles and grasping fails because the object is too large for the gripper. Both pushing and grasping fail in S6 because the goal region is unreachable. Figure 4 depicts a failure recovery case for S6 where failure to grasp to the second robot's reachable workspace does not result in task failure—the first robot pokes the object to the second robot's reachable workspace, therefore, allowing the second robot to successfully manipulate the object. Additionally, while grasping in the real-world, like in simulation, is a faster form of manipulation with fewer actions than poking or pushing, it has a lower overall success rate (27%) than pushing (33%) due to perception inaccuracies. Similar to simulation results, *PokeRRT** has the fewest number of executed actions.

V. Conclusion and Discussion

In this work, we demonstrate poking manipulation as a fundamental motion primitive that complements grasping and encompasses pushing in terms of capability. We show qualitative and quantitative results for multiple test conditions to demonstrate the flexibility and robustness of

	Task Time [seconds]			Number of Actions			Success Rate
Planner	**S4**	**S5**	**S6**	**S4**	**S5**	**S6**	**S4 - S6**
PokeRRT*	**69.58 (26.15)**	**127.45 (57.80)**	182.03 (122.77)	**5.27 (2.00)**	**5.12 (1.05)**	**6.11 (1.91)**	**0.9 (0.3)**
PokeRRT	83.55 (29.59)	148.55 (86.23)	**157.03 (94.41)**	6.20 (1.60)	6.67 (1.56)	6.88 (2.15)	**0.9 (0.3)**
Two-Level Push Planner	120.21 (47.09)	N/A	N/A	6.00 (1.26)	N/A	N/A	0.33 (0.47)
Pick-and-Place	N/A	20.31 (3.21)	N/A	N/A	1.00 (0.00)	N/A	0.27 (0.44)

TABLE IV: Real-world task times, number of actions in executed path, and success rates are presented in this table. Results are averaged across 10 trials. Pushing has the highest success rate and fastest task times. PokeRRT* generates plans with the fewest number of actions.

PokeRRT. We present the task times, number of executed actions, and success rates of our proposed motion planners and four baseline algorithms across six different scenarios. The results demonstrate the strengths of the poking motion primitive: poking is not limited by robot reachability, robot end-effector design, object properties, and inaccuracies due to perception. Success rates are higher for poking-based planners than for the push planner indicating that poking expands the size of reachable workspace by its ability to execute longer object displacements using shorter robot end-effector trajectories. Task times for computed plans are significantly lower for poking than for pushing, indicating that poking allows fast object manipulation because it does not face the same constant-contact restriction as pushing. *PokeRRT* and *PokeRRT** also demonstrated the qualitative behaviors outlined in Table I.

Exploring additional applications of instantaneous contact and extending this idea to other non-prehensile manipulation skills are interesting avenues for future work. Moreover, by utilizing multiple skills (e.g. grasping and poking) and planning around the key strengths of each motion primitive, robots will be able to more efficiently manipulate objects.

REFERENCES

[1] M. T. Mason, "Toward robotic manipulation," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 1–28, 2018.

[2] I. M. Bullock and A. M. Dollar, "Classifying human manipulation behavior," in *2011 IEEE International Conference on Rehabilitation Robotics*. IEEE, 2011, pp. 1–6.

[3] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, no. 6446, 2019.

[4] K. M. Lynch and M. T. Mason, "Dynamic nonprehensile manipulation: Controllability, planning, and experiments," *The International Journal of Robotics Research*, vol. 18, no. 1, pp. 64–92, 1999.

[5] W. Huang, "Impulsive manipulation," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, August 1997.

[6] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *arXiv preprint arXiv:1903.11239*, 2019.

[7] J. Shi, J. Z. Woodruff, P. B. Umbanhowar, and K. M. Lynch, "Dynamic in-hand sliding manipulation," *IEEE Transactions on Robotics*, vol. 33, no. 4, pp. 778–795, 2017.

[8] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Advances in Neural Information Processing Systems*, 2016, pp. 5074–5082.

[9] J. Z. Woodruff and K. M. Lynch, "Planning and control for dynamic, nonprehensile, and hybrid manipulation tasks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4066–4073.

[10] M. Bauza, F. Alet, Y.-C. Lin, T. Lozano-Pérez, L. P. Kaelbling, P. Isola, and A. Rodriguez, "Omnipush: accurate, diverse, real-world dataset of pushing dynamics with rgb-d video," *arXiv preprint arXiv:1910.00618*, 2019.

[11] C. Zito, R. Stolkin, M. Kopicki, and J. L. Wyatt, "Two-level RRT planning for robotic push manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 678–685.

[12] A. Kloss, M. Bauza, J. Wu, J. B. Tenenbaum, A. Rodriguez, and J. Bohg, "Accurate vision-based manipulation through contact reasoning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 6738–6744.

[13] J. K. Li, W. S. Lee, and D. Hsu, "Push-Net: Deep planar pushing for objects with unknown physical properties." in *Robotics: Science and Systems*, 2018.

[14] M. T. Mason, "Mechanics and planning of manipulator pushing operations," *The International Journal of Robotics Research*, vol. 5, no. 3, pp. 53–71, 1986.

[15] M. Bauza and A. Rodriguez, "A probabilistic data-driven model for planar pushing," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3008–3015.

[16] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," in *5th International Conference on Learning Representations (ICLR 2017)*, 2017.

[17] J. Zhou, R. Paolini, J. A. Bagnell, and M. T. Mason, "A convex polynomial force-motion model for planar sliding: Identification and application," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 372–377.

[18] J. Wu, J. J. Lim, H. Zhang, J. B. Tenenbaum, and W. T. Freeman, "Physics 101: Learning Physical Object Properties from Unlabeled Videos," in *Proceedings of the British Machine Vision Conference*, vol. 2, no. 6, 2016, p. 7.

[19] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum, "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 127–135.

[20] A. Kloss, S. Schaal, and J. Bohg, "Combining learned and analytical models for predicting action effects," *arXiv preprint arXiv:1710.04102*, 2017.

[21] A. Ajay, J. Wu, N. Fazeli, M. Bauza, L. P. Kaelbling, J. B. Tenenbaum, and A. Rodriguez, "Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3066–3073.

[22] C. Song and A. Boularias, "Learning to slide unknown objects with differentiable physics simulations," *arXiv preprint arXiv:2005.05456*, 2020.

[23] C. Zhu, Y. Aiyama, T. Arai, and A. Kawamura, "Frictional sliding motion in releasing manipulation," *Advanced Robotics*, vol. 19, no. 2, pp. 141–168, 2005.

[24] E. Coumans and Y. Bai, "PyBullet, a Python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[25] J. Collins, D. Howard, and J. Leitner, "Quantifying the reality gap in robotic manipulation tasks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6706–6712.

[26] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 928–935.

[27] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[28] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.