IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Online Replanning with Human-in-The-Loop for Non-Prehensile Manipulation in Clutter — A Trajectory Optimization based Approach

Rafael Papallas, Anthony G. Cohn, and Mehmet R. Dogar

***Abstract*— We are interested in the problem where a number of robots, in parallel, are trying to solve reaching through clutter problems in a simulated warehouse setting. In such a setting, we investigate the performance increase that can be achieved by using a human-in-the-loop providing guidance to robot planners. These manipulation problems are challenging for autonomous planners as they have to search for a solution in a high-dimensional space. In addition, physics simulators suffer from the uncertainty problem where a valid trajectory in simulation can be invalid when executing the trajectory in the real-world. To tackle these problems, we propose an online-replanning method with a human-in-the-loop. This system enables a robot to plan and execute a trajectory autonomously, but also to seek high-level suggestions from a human operator if required at any point during execution. This method aims to minimize the human effort required, thereby increasing the number of robots that can be guided in parallel by a single human operator. We performed experiments in simulation and on a real robot, using an experienced and a novice operator. Our results show a significant increase in performance when using our approach in a simulated warehouse scenario and six robots.**

I. Introduction

Consider the scenario where a large number of robots are working in a warehouse, reaching for items on cluttered shelves. We investigate whether a human-in-the-loop can improve the performance of such a system, and we propose a planning and control method to enable that.

Take the example in Fig. 1, where different robots are illustrated in each row. The horizontal axis illustrates time. Each robot has the task of reaching onto a shelf to grasp an object (in green), by moving obstacle objects out of the way. There is one human who can provide help by suggesting to move certain objects. Initially, in frame 1 of each row, all robots try to generate a plan, using trajectory optimization. Some of the robots, e.g. robots $r1$ and rn, quickly generate a feasible trajectory and start autonomous execution without requiring any human help. Since there is uncertainty in how objects move, the robots perform online replanning (similar to model predictive control), where they re-optimize and execute the trajectory at each time step. Robot $r2$, however, decides to ask for human help in frame 1, and prompts the user. In frame 2, the human engages with robot $r2$, quickly inspects the scene, uses an interface to provide high-level input (white arrow in the figure), and disengages. In frame 4, robot $r2$ tries to generate a trajectory again, this time making use of the human provided input, and then proceeds with autonomous execution using online replanning. In the meantime, after a duration of autonomous execution, the objects in rn's environment move very differently from the planner's expectations, resulting in rn requiring human help. The human is prompted for input, and execution proceeds.

Such a system has certain advantages. One advantage is the availability of human help in *planning* non-prehensile reaching through clutter motions. A variety of autonomous planners have recently been proposed to solve this problem [1]–[11], though difficult instances of the problem still result in low success rates or long planning times in the order of tens of seconds or minutes. This is partly the reason why current industrial applications are limited to scenes where the robot can *directly* reach for the object to be picked, without interacting with any other objects. In this work, we investigate the performance increase that can be achieved by using a human-in-the-loop providing guidance, particularly by minimizing the human effort required, thereby increasing the number of robots that can be guided in parallel by a single human. Such high-level guidance is usually easy and natural for a human to provide and can dramatically accelerate the performance of planners in difficult scenes.

Another advantage of the system described above is the use of *online replanning*. When a robot executes a non-prehensile plan, objects in the real world move differently to the model's predictions, which makes it necessary to update the plan. Trajectory optimization based planning approaches are particularly effective in such settings, because a new optimization cycle can be warm-started with the previous solution, and convergence can be achieved in few optimization iterations. In this respect, a key novelty of our method is the integration of human-interaction into this online-replanning architecture, enabling the system to use human help at any point in time during execution, not only as an initial input to the planning problem. Our previous work uses human guidance to solve reaching through clutter problems [12], but this approach requires human input to be provided before planning. It also uses a sampling-based planning approach (as opposed to the trajectory optimization approach we use in this paper). However, trajectory-optimization lends itself more easily to online-replanning, through warm-starting the optimization with the trajectory from the previous iteration.

Therefore, a key novel feature of our proposed system is the use of trajectory optimization and performing trajectory

This research has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grants agreement No. 746143, the AI4EU project (agreement No. 825619), from the UK Engineering and Physical Sciences Research Council under grant EP/N509681/1, EP/P019560/1 and EP/R031193/1, and a Turing Fellowship to the second author.

The authors are with the School of Computing, University of Leeds, United Kingdom (e-mail: r.papallas@leeds.ac.uk; a.g.cohn@leeds.ac.uk; m.r.dogar@leeds.ac.uk)

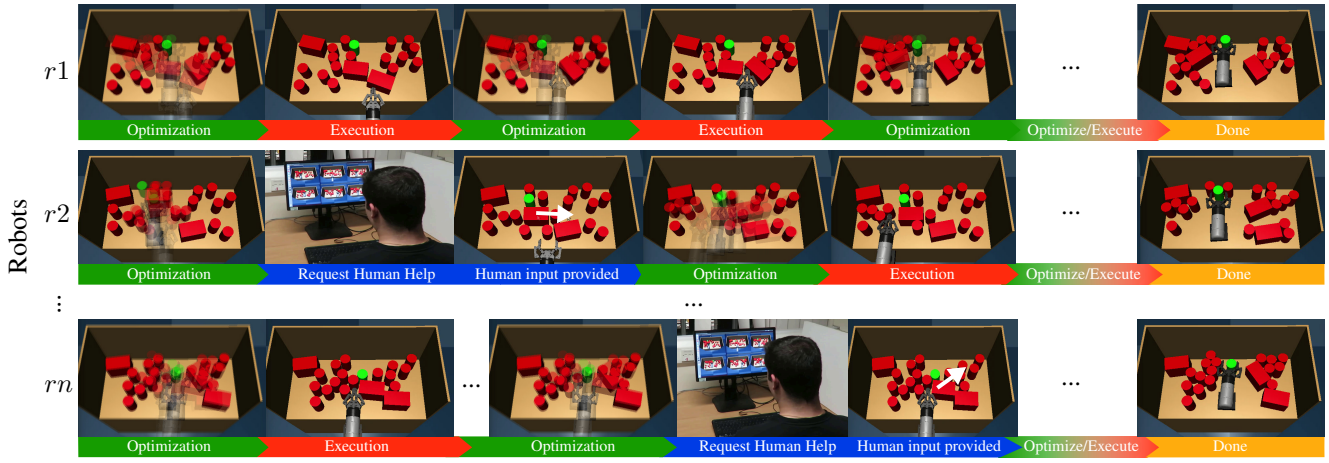


Fig. 1: Proposed System. Each row shows a different robot working in parallel. Human input is requested only when needed (*blue* color). Human high-level input is shown with a white arrow. Planning is shown with *green* and execution with *red* color.

optimization with human input. To achieve this, we propose to make the human input a part of the objective/cost function, minimized by the optimizer. This enables the human input to be easily integrated into the optimization performed at each step of the online-replanning process.

A final novel feature of our proposed system is the *efficient use of human time*. Our previous work [12] requires human help irrespective of whether an autonomous planner can solve the problem efficiently or not. We propose that the human should be recruited for help only if and when an autonomous planner is expected to fail or when human help is expected to speed up planning significantly. Such a system should be at least as good as the state-of-the-art fully autonomous system with the addition that, when needed, a human is in the loop to help. We propose two different approaches to realize this. The first approach will ask for human help if the planning fails to generate a plan within a fixed amount of time. The second approach, better integrated with trajectory optimization, will ask for human help if the optimization gets stuck at a local minimum.

To evaluate the proposed system we ran a number of different experiments. First, in Sec. V-B we evaluated two approaches of asking for human help and how they compare with two autonomous approaches. We conducted experiments with both an experienced and a novice user. In Sec. V-C we conducted experiments in simulation with artificial uncertainty, and on a real-robot (Fig. 2) to check the robustness of our online replanning execution mechanism. Finally, in Sec. V-D we test the entire system, in simulation, with a fleet of six robots trying to solve a number of planning problems simultaneously with a single human-operator and we compare this with parallel fully autonomous planners.

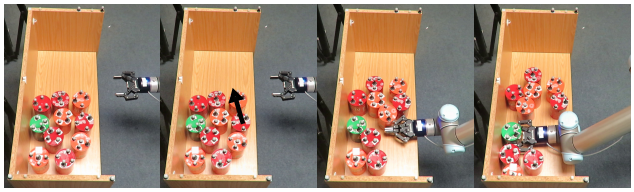


Fig. 2: Robot tries to reach for the goal object (green). Arrow indicates human input.

The novel contributions of this work are thus as follows: (1) the integration of human-input to an optimization problem for non-prehensile manipulation, (2) our framework allows a human operator to manage a fleet of robots at the same time for non-prehensile manipulation (3) we use an adaptive approach to decide when to ask for human input based on the problem difficulty and (4) we present a robust execution method to address real-world physics uncertainty.

II. Related work

The Amazon Picking Challenge [13] was a competition that encouraged fully-autonomous solutions to reaching for objects in cluttered environments. This competition gained particular attention for this potential near-term impact of robotic manipulation to warehouse robotics. The competition showed that these problems, even with relaxed assumptions, are difficult for a fully-autonomous system, particularly when the target object is behind clutter.

The autonomous solutions to the reaching through clutter problem can be categorized into three groups: There are *sampling-based planning* approaches [5], [6], [9], *trajectory optimization* based approaches [3], [14], and *learning-based* approaches [4], [7], [15], [16]. While these approaches show varying degrees of success, the difficult instances of this problem are still challenging for autonomous systems, due to the problem being high-dimensional and under-actuated, and also due to real-world physics uncertainty. We take the trajectory optimization approach, inspired by STOMP and CHOMP [17], [18], but we apply it to non-prehensile physics-based manipulation in clutter and we extend it by investigating how human input can be integrated with trajectory optimization in an online-replanning manner, and how a trajectory optimization process can decide to ask for help.

Williams et al. [19] describe a Model Predictive Path Integral (MPPI) approach to follow an already optimized trajectory. They run *one* iteration of optimization after *each* step execution in the real-world. We take a similar approach, but in a non-prehensile manipulation setting. Moreover, each of our optimization steps have the constraint to reach a goal state (e.g., having the goal object in the robot's hand), instead of simply optimizing a soft cost.

The use of high-level inputs is related to recent work in robotic hierarchical planning [2], [20]–[23] and task-and-motion planning (TAMP) [24]–[26]. This line of work shows that with a good high-level plan for a task, the search of the low-level motion planner can be guided to a relevant but restricted part of the search space, making the planner faster and more successful. Motivated by existing work in human-in-the-loop systems [27]–[40], in this letter we investigate the potential of using a human operator to suggest such high-level plans. However, most of the above existing work in human-in-the-loop planning focuses on providing clues to a planner to guide it through the collision-free space. Instead, we focus on non-prehensile manipulation. Our previous work [12], proposes a planner for non-prehensile manipulation using human-guidance, but the current work has significant differences in terms of the planning method (trajectory optimization versus sampling-based planning), the execution (closed-loop online re-planning versus open-loop execution), and most importantly how human input is requested (human input requested only when needed versus human input requested for all problems).

III. Problem Formulation

Our environment comprises a robot r, $\mathcal{O}$ movable obstacles that the robot can interact with and other static obstacles that the robot should not interact with. We also have the goal object to reach, $o_g \in \mathcal{O}$.

The state of the robot is denoted by $x^r = (q^r, \dot{q}^r) \in \mathcal{X}^r$. $q^r \subset SE(2)$ is the robot's configuration and $\dot{q}^r$ is the robot's velocities. Similarly, we denote the state of a movable obstacle $i \in \{1, \ldots, |\mathcal{O}|\}$ with $x^i = (q^i, \dot{q}^i, \ddot{q}^i) \in \mathcal{X}^i$ where $q_i \subset SE(2)$ is the object's configuration and $\dot{q}^i$ and $\ddot{q}^i$ the object's velocities and accelerations respectively. The state space of the entire system, $\mathcal{X}^E$, is given by the Cartesian product: $\mathcal{X}^E = \mathcal{X}^r \times \mathcal{X}^1 \times \mathcal{X}^2 \times \cdots \times \mathcal{X}^{|\mathcal{O}|}$.

The robot's control space is denoted by U and is comprised of the linear and angular robot velocities denoted by $u_t \in U$ applied at time t for a fixed duration Δt. We also have a trajectory $\tau = \langle u_0, u_1, \ldots, u_{n-1} \rangle$ of n steps. We use $\tau_{[0,n-1]}$ to denote a subsequence of controls where $[0, n-1]$ is a closed interval indicating the start and end of the subsequence. For example, for the trajectory $\tau = \langle u_0, u_1, u_2, u_3 \rangle$, $\tau_{[0]}$ refers to the first element of the trajectory (u_0), while $\tau_{[1,3]}$ refers to the subsequence $\langle u_1, u_2, u_3 \rangle$.

The state of the environment at time t is given by $x_t = \{x^r, x^1, \ldots, x^{|\mathcal{O}|}\} \in \mathcal{X}^E$. The discrete time dynamics of the system are given by $x_{t+1} = f(x_t, u_t) + \zeta$ where ζ is the system stochasticity. We do not explicitly represent the system stochasticity ζ. Instead, we take an online-replanning approach, which replans a new trajectory at every step during execution using the deterministic model f.

We say that we *rollout* a trajectory τ using f from an initial state $x_0 \in \mathcal{X}^E$ using a rollout function $R(x_0, \tau)$ to obtain a sequence of states $S = \langle x_0, \ldots, x_n \rangle$. We also have a cost function $\mathcal{C}(S)$ to compute the cost of the state sequence.

Given an initial state $x_0 \in \mathcal{X}^E$ and a goal object o_g, the goal is for the robot to execute a sequence of controls to move the robot from x_0 to a state where is grasping o_g while handling real-world physics uncertainty.

Algorithm 1 OR-HITL Framework

```
1:  function OR-HITL(τ, C)
2:      x^world ← observe current real-world state
3:      if x^world reached the goal then stop
4:      do
5:          result ← SOLVE(x^world, τ, C)
6:          if result is "human input required" then
7:              input ← obtain input from human
8:              update cost function C based on input
9:              update τ based on input
10:     while result is not "success"
11:     execute τ_[0] in real-world
12:     τ ← τ_[1,n−1] and expanded with u_togoal
13:     return OR-HITL (τ, C)
```

IV. Online Replanning with Human-In-The-Loop

We use an optimization-based approach that integrates human input to solve the problem of reaching through clutter. Our system starts tackling the problems fully autonomously and decides to ask for human help only when needed. In this way our system is capable of solving trivial problems fully autonomously without any human intervention, where possible. Our system integrates optimization and execution in a unified online-replanning framework that constantly optimizes and executes solution in the real-world robustly.

In Sec. IV-A we describe the proposed framework, OR-HITL. In Sec. IV-B we describe a stochastic optimizer that supports the optimization part of the OR-HITL framework. In Sec. IV-C we define the user-input and how we capture this input. In Secs IV-D and IV-E we define the cost function and how we compute the initial trajectories used in the optimization. Finally, in Secs IV-F and IV-G we describe two approaches to *decide* when to ask for human help.

A. Framework Overview

Alg. 1 describes OR-HITL that unifies optimization and execution in one framework and alternates between the two to increase real-world execution robustness.

The algorithm starts with an initial trajectory τ for reaching the goal object (see Fig. 3a) and a cost function $\mathcal{C}$ with the optimization objectives (Sec. IV-D). In line 2 we observe the real-world state and if this state is a goal state, we stop and declare success (line 3). If not, then we proceed with the *optimization* part of the framework.

Algorithm 2 Trajectory Optimization-based Solver

1: **function** SOLVE($x_0, \tau, \mathcal{C}$)
2: $S \leftarrow$ rollout τ from x_0 using $R(x_0, \tau)$
3: obtain the cost of rollout using $\mathcal{C}(S)$
4: **while** not successful **do**
5: **if** humanHelpRequired() **then**
6: **return** "human input required"
7: sample k noisy trajectories from τ
8: rollout each of the k trajectories from x_0 using R
9: obtain cost for each rollout using $\mathcal{C}$
10: $\tau \leftarrow$ trajectory with the lowest cost
11: **return** "success"

Optimization: In line 5 using the SOLVE function (Sec. IV-B) we pass to the solver the initial state x^{world}, the current trajectory τ and the current cost function, $\mathcal{C}$. The solver will optimize for some duration and then return a *result*. The result can either be "human input required" or "success". If the solver returns "success", it also updates τ with the new trajectory. If the solver decides that human input is required (we describe how this decision is taken in Secs IV-F and IV-G), then in line 7 we obtain a high-level input from a human operator (Sec. IV-C). This high-level input includes information to update the cost function (line 8) and to instantiate a new initial trajectory τ (line 9). We repeat these steps until *result* is "success" (line 10). Once the optimization is successful, we proceed with the *execution* part of the framework.

Execution: To cope with physics uncertainty when executing a trajectory in the real-world, we propose an Online-Replanning approach. In line 11 we execute the first control of the trajectory in the real-world. We then update our trajectory τ in line 12 to be the remaining τ trajectory, expanded with a control towards the goal (u_{togoal}). This padded control at the end of the trajectory provides freedom to the trajectory to be further optimized in the future. Once we update the trajectory, we recurse in line 13 and we get the real-world state in line 2 that might be different to the one predicted by the simulator (physics uncertainty). The optimization in line 5 *warm-starts* with the trajectory from the previous iteration, and therefore is likely to be successful in the current iteration, requiring little or no additional work from the solver. This closed-loop execution and alternation from optimization to execution allows our system to cope with execution uncertainty and correct the trajectory early on.

B. Stochastic Optimization

In Alg. 1, in line 5, we make a call to a solver. Alg. 2 describes this solver. The solver accepts an initial state, x_0, an initial trajectory to optimize, τ, and a cost function $\mathcal{C}$ for computing the cost of the trajectories.

We begin by rolling out the trajectory τ from the initial state x_0 (line 2) and then using the provided cost function, $\mathcal{C}$, we compute the cost of the trajectory (line 3). If the solver is successful (line 4), then we return "success" straightaway (line 11). To decide if the solver is successful we check if the robot has reached the goal and that the cost is minimized below some threshold. If the solver is not successful we continue with the optimization.

First we check if human help is required in line 5. We describe ways to make this decision in Secs IV-F and IV-G. If human help is required, we return a signal that human help is required (line 6).

If human help is not required, we sample k noisy trajectories from τ (line 7). To create these k trajectories, we add Gaussian noise to the controls of τ using $\mathcal{N}(0, v)$ where $\mathcal{N}$ is the Gaussian distribution and v is the variance for a degree-of-freedom of the robot. We then rollout each of the k trajectories from x_0 using the rollout function R (line 8) and then obtain a cost for each of the trajectories (line 9). In line 10, we update τ to be the trajectory with the lowest cost. It is possible that all k trajectories have higher cost than the current τ, in this case τ does not change. We repeat these steps until the solver is successful (line 4).

C. User Input

A user's high-level action suggests a particular object o_i to be pushed to particular point on the plane. We formalize this high-level action with the triple (o_i, x_i, y_i), where $o_i \in \mathcal{O}$ is an object, and (x_i, y_i) is a point of on the plane that o_i needs to be pushed to.

To capture the user's high-level action we developed a simple user interface. The operator is presented with a window showing the environment and the robot. The operator, using a mouse pointer, provides the input by first clicking on the desired object and then a point on the plane (Alg. 1 line 7).

Using the human input we can now define the cost function and the initial trajectory that makes use of the human input in the next subsections.

D. Cost Function

The cost function, $\mathcal{C}$, is used in Alg. 2 (lines 3 and 9) but is also updated in Alg. 1 (line 8) to integrate the human input.

No human-input provided: If no human-input is provided, then the cost function for a state sequence S is defined as $\mathcal{C}(S) = \mathcal{C}_1 + \mathcal{C}_2$. Where $\mathcal{C}_1$ is the cost for reaching the goal object:

$$\mathcal{C}_1 = w_g \cdot \mathrm{d}(\mathbf{q^{ee}}, \mathbf{q^g}) \tag{1}$$

$\mathcal{C}_1$ is the weighted Euclidean distance from the robot's end-effector to the goal object, o_g. $\mathcal{C}_2$ defines three cost terms with their corresponding weights:

$$\mathcal{C}_2 = \sum_{i=1}^{n} c_e(x_i) + c_f(x_i) + c_s(x_i) \tag{2}$$

- $c_f(x_i) = \sum_{j=1}^{|\mathcal{O}|} w_f \cdot F(x_i^j)$: For a state x_i we penalize any movable object that applies high forces to any other movable or static obstacle. F is a function that given a state of an object x_i^j returns the contact forces of that object.

(a) Reaching goal object (b) Pushing an object

Fig. 3: Initial trajectories. The arrow illustrates the trajectory. In (b) the object to be pushed is the box the arrow penetrates.

- $c_e(x_i) = \sum_{j=1}^{|\mathcal{O}|} w_e \cdot \mathbb{1}_e(q_i^j)$: For a state x_i we penalize any movable object $o_j \in \mathcal{O}$ that is geometrically outside the configuration space. $\mathbb{1}_e$ is an indicator function that returns 0 if the objects is geometrically inside the configuration space, 1 otherwise.
- $c_s(x_i) = w_s \cdot \mathbb{1}_s(x^r)$: For a state x_i we penalize the robot for colliding with a static obstacle. $\mathbb{1}_s$ is an indicator function that returns 1 if the robot, r, collides with any of the static obstacles or 0 otherwise.

Human-input provided: If human-input is provided (Alg. 1, line 7) the cost function is updated (Alg. 1, line 8). This update is two-fold, we first push that object to the human indicated position using $\mathcal{C}(S) = \mathcal{C}_3 + \mathcal{C}_2$, and then we reach for the goal object using $\mathcal{C}(S) = \mathcal{C}_1 + \mathcal{C}_2$.

$$\mathcal{C}_3 = w_p \cdot \mathrm{d}(\mathbf{q_n^i}, \mathbf{q_{desired}^i}) \tag{3}$$

For a high-level input (o_i, x_i, y_i), $\mathcal{C}_3$ is the weighted Euclidean distance of o_i position at the final state, q_n^i, with the user's provided position, $q_{desired}^i$, of that object. This cost term in the optimization will encourage the solver to explore solutions where the object indicated by the human is pushed towards the desired position.

E. Initial trajectories

When we start the optimization, we need to provide the solver with an initial trajectory. We use straight-line trajectories because they are cheap to compute (no physics simulation).

We depict two such initial trajectories in Fig. 3. The first trajectory, Fig. 3a, is the initial trajectory for reaching the goal object and is a straight line trajectory from the robot's current position, q^r, to the position of the goal object, q^g. The second trajectory, Fig. 3b, is the initial trajectory for pushing an object to its desired position. This trajectory is following a straight line passing from the object's current position and ending at the object's desired position.

Next, in Secs IV-F and IV-G we describe two ways to decide when to ask for human input. This decision is taken in Alg. 2 line 5 and we describe two different approaches to take this decision.

F. Asking for human help with a fixed timeout

A straightforward way to decide if human input is required, is based on a fixed timeout limit. The solver tries to find a solution for some fixed time limit and if a solution is not found it will time-out and request human input.

We denote this time-limit with Fixed z where $z \in \mathbb{Z}^+$. Therefore, in Alg. 2 in line 5 when using Fixed OR-HITL, `humanHelpRequired` will return true if z seconds are reached. For example, for Fixed 20, `humanHelpRequired` will return true every 20 seconds if the solver cannot find a solution.

Although this is a simple and straightforward approach, it can be problematic, in some cases. For example, if the solver is able to solve a problem in 25 seconds but the Fixed Time limit is set to 20 seconds, then the solver will ask for human help although it could have found a solution if there were 5 more seconds available. Similarly, for a hard problem, giving more time to time-out could make the system waste time unnecessarily before asking for human help.

This shortcoming of the Fixed Timeout inspired us to introduce an adaptive approach that decides when to ask for human help depending on the problem at hand.

G. Adaptively asking for human-help

One property of trajectory optimization is that the convergence rate of the cost of a problem from iteration to iteration can indicate whether the solver can explore new solutions (i.e., more time is needed) or if the solver is stuck at a local minimum (i.e., immediate human input could be beneficial). We leverage this property to adaptively decide when to ask for human help based on the problem at hand. If at some point during the optimization we find that the solver hits a local minimum, then we send a signal that human input is required.

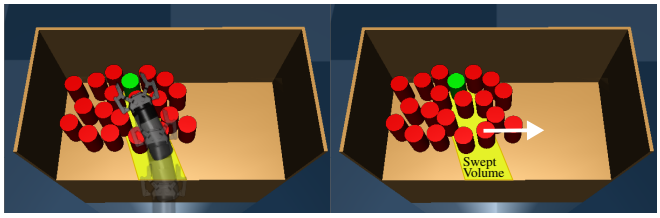
To decide if the solver is stuck at a local minimum, we look at the absolute difference between the previous iteration's cost, $c_{previous}$, and at the current iteration's cost, $c_{current}$. If this difference is lower than a threshold for a number of consecutive iterations, then we say that we are stuck at a local minimum. Since this is a stochastic optimization, we need to check this for some consecutive iterations to conclude that we are stuck at a local minimum because it is likely that in an iteration the cost does not improve, but this is not an indication that we are stuck at a local minimum.

Therefore, in Alg. 2 in line 5 when using Adaptive OR-HITL, `humanHelpRequired` will return true if we hit a local minimum for some consecutive iterations. In our implementation we stop if local minimum is found for 2 consecutive iterations.

V. Experiments & Results

We focus on three main questions for our evaluation. First, in Sec. V-B we evaluate the proposed framework performance with human-in-the-loop and compare with several baselines. Second, in Sec. V-C we evaluate the replanning aspect of our framework in presence of physics uncertainty. Third, in Sec. V-D we evaluate our system in a simulated warehouse scenario with a single human operator. Some of these experiments are shown in the accompanying video[1]. We employed two

[1]Also available at `https://youtu.be/t3yrx-J8IRw`.



(a) (b)

Fig. 4: Heuristic: Yellow area is the robot's swept volume and white arrow is the suggested high-level plan.

operators in these experiments. An experienced user, who is also an author of the paper, who conducted all the human experiments throughout this section except the experiments marked with "Novice" where a novice operator was employed instead. The novice had no prior experience with robotic systems or robotics research.

A. Experimental Setup

For all the experiments we used MuJoCo[2] [41] to implement the system dynamics. We used a randomizer to generate random simulation scenes. The randomizer placed the goal object first at the back of the shelf and then the remaining objects in collision-free positions within a radius of 30cm around the goal object. The total time limit for every experiment was 180 seconds, after which the robot was stopped and the run was marked a failure. This time-limit includes combined optimization times as well as human interaction time.

Optimization Parameters: The optimization parameters were: $k = 15$ noisy trajectories at each iteration, variance of 0.04 m/s for the robot's linear velocities and 0.04*rad*/s for the robot's angular velocity. We had 3-seconds long trajectories with 8 steps each. To rollout a 3-second trajectory with an integration step size of 0.0015 it took on average 1 second on our computer. To execute a 3-second trajectory in simulation took around 3 seconds. The cost function's parameters are: $w_g = 2000$, $w_f = 50$, $w_e = 300$ and $w_s = 300$. Finally, the success threshold is 70.0 (Alg. 2, line 4).

Robot: We use a 2-finger Robotiq gripper on a UR5 robot mounted on a Clearpath Ridgeback. We controlled the hand in simulation (3 DOF: 2 linear and 1 angular). The gripper has 1 DOF but we do not consider it in the optimization, instead we close the gripper at the beginning of the optimization and we open it before the last action of the trajectory. When executing the solutions in the real-world we mapped the gripper velocities to Ridgeback velocities.

B. Framework Evaluation

We compare Fixed 5, Fixed 20 and Adaptive with an experienced and a novice user with two autonomous planners. The novice user had no prior knowledge of the system or the problem. We trained the novice user for around 45 minutes.

[2]On a computer with Intel Core i7-4790 CPU @ 3.60GHz, 16GB RAM.

TABLE I: Framework evaluation. Error indicates the 95% CI.

	Success (%)	Planning Time (s)	Human Time (s)	Total Time (s)
Experienced User				
Fixed 5	90.0	38.1 ± 15.6	9.6 ± 4.1	68.8
Fixed 20	93.3	44.2 ± 13.8	7.0 ± 1.8	63.2
Adaptive	**96.6**	31.0 ± 12.8	$\mathbf{2.5 \pm 0.9}$	**42.5**
Novice User				
Fixed 5	86.6	$\mathbf{27.8 \pm 11.7}$	19.8 ± 8.8	62.6
Adaptive	90.0	33.6 ± 16.5	5.5 ± 1.0	45.1
No Human				
Autonomous	74.6	79.8 ± 11.2	–	82.8
Adaptive Heuristic	82.5	86.4 ± 13.5	–	98.4

We generated 30 scenes for *each* planner. We generated a different set of problems for each planner where a human was involved to avoid the chance where a pattern of the problem is memorized by the users. For the Autonomous and the Heuristic planners, since there is no learning, we evaluate them over all the scenes.

We compare two autonomous planners here. First, "Autonomous" is a planner that uses the solver highlighted in Alg. 2 but aims to reach for the goal object without considering any high-level plan. We also implemented a straight-line heuristic to replace the human from the OR-HITL framework. The robot moves on a straight line to the goal object to find the first blocking obstacle and to capture the robot's swept volume. It then finds a collision-free position outside the swept volume for this obstacle. The object and the new position is then returned to the framework (i.e., substituting the human input entirely) to plan and push the object to the new position. We illustrate this straight-line heuristic in Fig. 4. We call this planner "Adaptive Heuristic".

Table I presents the simulation results. Planning time is the average planning time per problem. Human time is the average time spent by a user providing guidance. Total time includes the planning time, human time (if applicable) and execution time, providing an overall average time to solve a problem. The results show that the planners with a human-in-the-loop were more successful than the autonomous planners and they had dramatically improved the planning times. Adaptive requested less human intervention with an average of 2.5 seconds of human time per problem. Fixed 5 requested human intervention more frequently and shows that human engagement with this approach is considerably high. The novice user confirmed that the interaction with the Fixed 5 was more tedious as he was prompted too often and he found it challenging to provide high-level input using the Fixed 5 because the 5 seconds timeout forced him to provide high-level actions that are easy optimization problems. The novice participant found Adaptive more intriguing and comfortable to use and he enjoyed the fact that the robot managed to solve more problems on its own.

C. Handling uncertainty

Our framework is designed to handle physics uncertainty during execution. To evaluate this property we performed two different experiments. First, we evaluated the system in

TABLE II: Simulated uncertainty. Errors indicate 95% CI.

	Adaptive	Autonomous	OL Autonomous
Success	58 / 60	35 / 60	23 / 60
Optimization Failures	1 / 60	16 / 60	23 / 60
Execution Failures	1 / 60	9 / 60	14 / 60
Optimization Time (s)	62.2 ± 10.5	118.2 ± 17.2	121.0 ± 18.1
Replanning Iterations	2.8 ± 1.2	4.2 ± 3.0	11.9 ± 3.1
Human Time (s)	5.4 ± 0.8	-	-
Total Time (s)	79.6	124.2	124.0

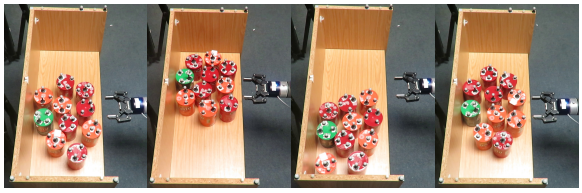


Fig. 5: Real-world scenes

simulation with artificial uncertainty added to the objects' motion, during execution. Second, we also evaluated the system with a *real robot* manipulating objects on a real shelf.

Simulation: We configured the simulation environment in the following way. At each *simulation step* (during execution, not during optimization) we observe the velocities of all the movable objects and if they are beyond some threshold[3] we add a small Gaussian noise to their velocities (changes direction and magnitude). We compare Adaptive, Autonomous and we also implemented an Open-Loop Autonomous (OL Autonomous) planner as a baseline. OL Autonomous replans only at the end of the trajectory execution instead of the end of each action. Table II presents the results. Execution Failures indicate the number of times the planner found an initial solution but failed to execute and replan. Replanning Iterations indicate the number of optimization iterations required to successfully replan a trajectory that failed during execution. Total time is the average time to complete the task (planning time, human time and execution time). It is clear that Adaptive performed better than Autonomous and OL Autonomous in success rate and planning time. Success rate for Adaptive was 97%, 58% for Autonomous and 38% for OL Autonomous. During replanning Adaptive and Autonomous required on average 3 and 4 iterations respectively to correct the trajectory while the OL Autonomous required on average 12 iterations.

Real-world: In a real-world setting we evaluated Adaptive and Open-Loop Adaptive (OL Adaptive). OL Adaptive replans at the end of the trajectory instead of the end of each action. We performed 30 real-world experiments, 15 for each planner in 15 different scenes. The robot was asked to reach for the green object in a small shelf among other 9 obstacles. Some of the scenes are depicted in Fig. 5. To avoid damage of the physical robot or of the objects in the environment, we stopped the robot if it collided with any static obstacle or forcefully pushed an object against the shelf and we declared the attempt

[3]To avoid adding uncertainty to objects that have not moved since the previous step.

TABLE III: Real-world results

	Adaptive	OL Adaptive
Success	13 / 15	8 / 15
Optimization Failures	1 / 15	1 / 15
Execution Failures	1 / 15	6 / 15

TABLE IV: Warehouse. Errors indicate 95% CI.

	Adaptive	Autonomous
Success	37 / 50	16 / 50
Failures	13 / 50	34 / 50
Optimization Time (s)	94.7 ± 15.1	149.7 ± 15.9
Human Time (s)	5.5 ± 1.0	-
Total Time (s)	112.2	152.7

as an Execution Failure due to violation of these safety rules. There are some demonstrations of these experiments in the accompanying video.[4] We present the results in Table III. The success rate for Adaptive is 86% while for the OL Adaptive is 53%. Adaptive failed once during planning and once during execution (due to safety rule violation). OL Adaptive failed once during planning and six times during execution. The main cause of failures for the OL Adaptive planner was that physics uncertainty caused the robot to violate some of the safety rules. The OL Adaptive, since there was no replanning at each step, was in general faster at executing trajectories, but when it failed it required more replanning time. Adaptive on the other hand, replanned for some easy instances of the problem, but the replanning was always very short because we warm-started the optimization.

D. Warehouse Problem

We consider a scenario where there are 50 orders to pick objects from cluttered shelves in a simulated warehouse. The warehouse operates six robots at the same time. Our objective is to increase the success rate of these robots and fulfill the 50 orders as quickly as possible.

In simulation, we compare six autonomous robots (Autonomous) working in parallel trying to fulfill these 50 orders, with our system with a *single* human-operator (Adaptive) guiding these six robots simultaneously. Some of these scenes are shown in Fig. 1. Each robot attempts to solve the assigned problem within 180 seconds. Once a robot finishes with a problem it is assigned with the next available one. We conducted this experiment on a more powerful computer[5] as 6 simultaneous instances of the physics simulator requires extensive CPU power and memory usage.

The results (Table IV) show that our approach was more successful (74% success rate compared to 32% for the Autonomous) and faster in planning solutions per problem, almost a minute faster. The average number of actions for Adaptive was four and hence the execution time (which is fixed to 3 seconds) added up further 12 seconds per problem

[4]"Adaptive" execution is slower than "OL Adaptive" due to delays with the perception system called after each action execution.

[5]Intel Xeon E5-2650 v2 CPU @ 2.60GHz, 64GB RAM.

on average. On the other side, since Autonomous is planning straight to the goal, there is only one trajectory and hence there are only 3 seconds of execution time per problem.

VI. Conclusions

We have proposed OR-HITL, an online-replanning framework with Human-In-The-Loop based on trajectory optimization. Our approach starts solving the problem fully autonomously and decides to ask for human input only when the problem is estimated to be too difficult to be solved autonomously. Our system uses an adaptive approach (Adaptive) to take this decision based on the problem difficulty. We demonstrated that this adaptiveness is useful in a simulated warehouse setting where a single human operator manages a fleet of robots at the same time. Finally, our framework showed increased robustness in real-world execution due to the online-replanning strategy we implemented in the framework.

References

[1] M. Dogar, K. Hsiao, M. Ciocarlie, and S. Srinivasa, "Physics-based grasp planning through clutter," in *RSS*, 2012.

[2] G. Havur, G. Ozbilgin, E. Erdem, and V. Patoglu, "Geometric rearrangement of multiple movable objects on cluttered surfaces: A hybrid reasoning approach," in *ICRA*. IEEE, 2014, pp. 445–452.

[3] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel, "Physics-based trajectory optimization for grasping in cluttered environments," in *ICRA*. IEEE, 2015, pp. 3102–3109.

[4] W. Bejjani, M. R. Dogar, and M. Leonetti, "Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning," in *IEEE/RSJ IROS*. IEEE, 2019.

[5] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *ICRA*. IEEE, 2015, pp. 3075–3082.

[6] M. ud din, M. Moll, L. Kavraki, J. Rosell *et al.*, "Randomized physics-based motion planning for grasping in cluttered and uncertain environments," *IEEE RA-L*, vol. 3, no. 2, pp. 712–719, 2018.

[7] W. Bejjani, R. Papallas, M. Leonetti, and M. R. Dogar, "Planning with a receding horizon for manipulation in clutter using a learned value function," in *IEEE-RAS Humanoids*. IEEE, 2018, pp. 1–9.

[8] C. Nam, J. Lee, Y. Cho, J. Lee, D. H. Kim, and C. Kim, "Planning for target retrieval using a robotic manipulator in cluttered and occluded environments," *arXiv preprint arXiv:1907.03956*, 2019.

[9] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *IEEE ICRA*. IEEE, 2016, pp. 3940–3947.

[10] E. Huang, Z. Jia, and M. T. Mason, "Large-scale multi-object rearrangement," in *IEEE ICRA*. IEEE, 2019, pp. 211–218.

[11] K. Kim, J. Lee, C. Kim, and C. Nam, "Retrieving objects from clutter using a mobile robotic manipulator," in *2019 16th International Conference on Ubiquitous Robots (UR)*. IEEE, 2019, pp. 44–48.

[12] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," in *IEEE ICRA*. IEEE, 2020.

[13] C. Eppner, S. Höfer, R. Jonschkowski, R. Martín-Martín, A. Sieverling, V. Wall, and O. Brock, "Lessons from the amazon picking challenge: Four aspects of building robotic systems." in *RSS*, 2016.

[14] W. C. Agboh and M. R. Dogar, "Real-time online re-planning for grasping under clutter and uncertainty," in *IEEE-RAS Humanoids*. IEEE, 2018, pp. 1–8.

[15] J. A. Haustein, I. Arnekvist, J. Stork, K. Hang, and D. Kragic, "Learning manipulation states and actions for efficient non-prehensile rearrangement planning," *arXiv preprint arXiv:1901.03557*, 2019.

[16] W. Yuan, K. Hang, D. Kragic, M. Y. Wang, and J. A. Stork, "End-to-end nonprehensile rearrangement with deep reinforcement learning and simulation-to-reality transfer," *Robotics and Autonomous Systems*, vol. 119, pp. 119–134, 2019.

[17] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *IEEE ICRA*. IEEE, 2011, pp. 4569–4574.

[18] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *IJRR*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[19] G. Williams, A. Aldrich, and E. A. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 344–357, 2017.

[20] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour, "Manipulation planning among movable obstacles," in *IEEE ICRA*. IEEE, 2007, pp. 3327–3332.

[21] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *IEEE ICRA*, 2011, pp. 1470–1477.

[22] M. Dogar and S. Srinivasa, "A framework for push-grasping in clutter," *Robotics: Science and systems VII*, vol. 1, 2011.

[23] G. Lee, T. Lozano-Pérez, and L. P. Kaelbling, "Hierarchical planning for multi-contact non-prehensile manipulation," in *IEEE/RSJ IROS*. IEEE, 2015, pp. 264–271.

[24] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *IJRR*, vol. 33, no. 14, pp. 1726–1747, 2014.

[25] A. Wells, N. Dantam, A. Shrivastava, and L. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robotics and Automation Letters*, 2019.

[26] J. Lee, Y. Cho, C. Nam, J. Park, and C. Kim, "Efficient obstacle rearrangement for object manipulation tasks in cluttered environments," *arXiv preprint arXiv:1902.06907*, 2019.

[27] Y. K. Hwang, K. R. Cho, S. Lee, S. M. Park, and S. Kang, "Human computer cooperation in interactive motion planning," in *ICAR*. IEEE, 1997, pp. 571–576.

[28] O. B. Bayazit, G. Song, and N. M. Amato, "Enhancing randomized motion planners: Exploring with haptic hints," *Autonomous Robots*, vol. 10, no. 2, pp. 163–174, 2001.

[29] M. Taïx, D. Flavigné, and E. Ferré, "Human interaction with motion planning algorithm," *Journal of Intelligent & Robotic Systems*, vol. 67, no. 3-4, pp. 285–306, 2012.

[30] J. Denny, J. Colbert, H. Qin, and N. M. Amato, "On the theory of user-guided planning," in *IEEE/RSJ IROS*. IEEE, 2016, pp. 4794–4801.

[31] F. Islam, O. Salzman, and M. Likhachev, "Online, interactive user guidance for high-dimensional, constrained motion planning," *arXiv preprint arXiv:1710.03873*, 2017.

[32] J. Denny, R. Sandström, and N. M. Amato, "A general region-based framework for collaborative planning," in *Robotics Research*. Springer, 2018, pp. 563–579.

[33] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning from physical human corrections, one feature at a time," in *ACM/IEEE HRI*, 2018, pp. 141–149.

[34] D. Koert, G. Maeda, R. Lioutikov, G. Neumann, and J. Peters, "Demonstration based trajectory optimization for generalizable robot motions," in *IEEE-RAS Humanoids*. IEEE, 2016, pp. 515–522.

[35] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning robot objectives from physical human interaction," *Proceedings of Machine Learning Research*, vol. 78, pp. 217–226, 2017.

[36] G. Swamy, S. Reddy, S. Levine, and A. D. Dragan, "Scaled autonomy: Enabling human operators to control robot fleets," *arXiv preprint arXiv:1910.02910*, 2019.

[37] A. E. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *ACM/IEEE HRI*. ACM, 2012, pp. 1–8.

[38] S. Muszynski, J. Stückler, and S. Behnke, "Adjustable autonomy for mobile teleoperation of personal service robots," in *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, 2012, pp. 933–940.

[39] T. Witzig, J. M. Zöllner, D. Pangercic, S. Osentoski, R. Jäkel, and R. Dillmann, "Context aware shared autonomy for robotic manipulation tasks," in *IEEE/RSJ IROS*. IEEE, 2013, pp. 5686–5693.

[40] M. Ciocarlie, K. Hsiao, A. Leeper, and D. Gossow, "Mobile manipulation through an assistive home robot," in *IEEE/RSJ IROS*. IEEE, 2012, pp. 5313–5320.

[41] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ IROS*. IEEE, 2012, pp. 5026–5033.