UNIVERSITY OF LEEDS

This is a repository copy of *To ask for help or not to ask: A predictive approach to human-in-the-loop motion planning for robot manipulation tasks*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/189268/

Version: Accepted Version

Proceedings Paper:

Papallas, R orcid.org/0000-0003-3892-1940 and Dogar, MR (2022) To ask for help or not to ask: A predictive approach to human-in-the-loop motion planning for robot manipulation tasks. In: 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022), 23-27 Oct 2022, Kyoto, Japan. IEEE , pp. 649-656. ISBN 978-1-6654-7927-1

https://doi.org/10.1109/IROS47612.2022.9981679

This is an author produced version of a conference paper published in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), made available under the terms of the Creative Commons Attribution License (CC-BY), which permits unrestricted use, distribution and reproduction in any medium, provided the original work is properly cited.

Reuse
This article is distributed under the terms of the Creative Commons Attribution (CC BY) licence. This licence allows you to distribute, remix, tweak, and build upon the work, even commercially, as long as you credit the authors for the original work. More information and the full terms of the licence here: https://creativecommons.org/licenses/

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose
university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

To ask for help or not to ask: A predictive approach to human-in-the-loop motion planning for robot manipulation tasks

Rafael Papallas and Mehmet R. Dogar

***Abstract*— We present a predictive system for non-prehensile, physics-based motion planning in clutter with a human-in-the-loop. Recent shared-autonomous systems present motion planning performance improvements when high-level reasoning is provided by a human. Humans are usually good at quickly identifying high-level actions in high-dimensional spaces, and robots are good at converting high-level actions into valid robot trajectories. In this paper, we present a novel framework that permits a single human operator to effectively guide a fleet of robots in a virtual warehouse. The robots are tackling the problem of Reaching Through Clutter (RTC), where they are reaching onto cluttered shelves to grasp a goal object while pushing other obstacles out of the way. We exploit information from the motion planning algorithm to predict which robot requires human help the most and assign that robot to the human. With twenty virtual robots and a single human-operator, the results suggest that this approach improves the system's overall performance compared to a baseline with no predictions. The results also show that there is a cap on how many robots can effectively be guided simultaneously by a single human operator.**

I. Introduction

We consider the problem of a fulfilment centre where a large number of semi-autonomous robots are reaching through clutter on shelves to grasp a goal object. Since the shelves are cluttered, the robots use non-prehensile manipulation to push movable obstacles away while reaching for the goal object.

The robots start to tackle the problems autonomously. As this is a kinodynamic problem, it requires physics-based search in a high-dimensional state space and, therefore, some instances of the problem are extremely difficult for autonomous robots. In these cases, a human operator is available to provide help with high-level reasoning to alleviate the computational burden for the robots. The human, through a graphical user interface, possibly remotely, inspects the scene and suggests a high-level action (for example, push this object to this region). The robot leverages the input and integrates it in its motion planning algorithm and continues autonomously.

An interesting question is, *when should the robots ask for human help*? Can different solutions to this problem lead to better performance in a multi-robot guidance setting?

A. *When to ask for human help?*

In our previous works [1], [2] we explored two possible answers to this question. The most naive solution is to always ask for help before planning [1]. This can help robots in all scenarios, since the human will engage with all problems, but if the human is unavailable, possibly helping other robots, robots will be waiting unnecessarily. A better solution is to let robots solve problems autonomously and fallback to a human only when they cannot [2]. The obvious advantage is that the robots will solve trivial problems autonomously and only fallback to a human when they are faced with non-trivial problems. A disadvantage, however, is that, sometimes, it can take a while for an autonomous planner to fail and asking for human help earlier can improve the robot's performance.

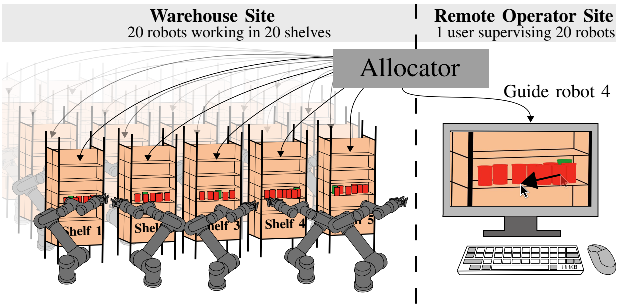


Fig. 1: Twenty robots guided in parallel by a human in a virtual warehouse. Robots autonomously reach for the green object, but might require human help. Each robot estimates "how much" it will benefit from human help, and the "Allocator" allocates the human to the robot that will benefit the most. The human provides a high-level input ("push this object here"), the robot leverages the input and disengages with the human.

In this paper, we are proposing a new solution that combines the desired property of the first solution, employing human input early, and the desired property of the second solution, allowing robots to tackle trivial problems autonomously. We propose a predictive approach that much before the autonomous planning algorithm gets stuck, predicts whether the planning will fail, and, therefore, employs the human without losing productive time. This requires predicting the performance of the autonomous planner in a given scene. An important question is *what to predict*?

B. *What to predict?*

One answer is to predict a binary value; given a scene, predict the success or failure of the autonomous planner, making this a classification problem. Although this could work, in this work we are interested in the problem where *tens of robots* are all working in parallel in a warehouse under the supervision of a single human operator. At some point,

This research has received funding from the UK Engineering and Physical Sciences Research Council under grants EP/V052659/1, EP/N509681/1 and EP/P019560/1.

For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) license to any Author Accepted Manuscript version arising.

Authors are with the School of Computing, University of Leeds, United Kingdom {`r.papallas, m.r.dogar`}`@leeds.ac.uk`

a classification algorithm could suggest a number of robots that are "failing", and therefore requiring human help. Which one should the human help? Therefore, a better approach is to frame this as a regression problem, and predict/estimate *how much the robot will benefit from human help*. This way, the estimate can be used to sort the requests and assign the human to robots that will benefit the most.

Consider Fig. 1 as an example. There are twenty robots, all of which are solving Reaching Through Clutter (RTC) problems. They attempt to reach and grasp the green object, while pushing the red ones out of the way. Most of the robots are able to solve the tasks autonomously, however, the system predicts that robot 2, robot 4 and robot 8 would significantly benefit from human help. The system identifies which robot will benefit the most from human help (robot 4) and assigns it to the operator. The human operator, over a remote site, inspects the scene on a screen and quickly identifies a high-level action (e.g., "push object 1 to that region"), the robot integrates the high-level input into its motion planning algorithm, the human disengages with robot 4 and becomes available for other robots.

C. Challenges and Motivation

Reaching Through Clutter (RTC) is a challenging problem for multiple reasons, as shown in the Amazon Picking Challenge [3]. In this competition, even with relaxed assumptions, autonomous systems struggled to solve hard RTC problems.

The difficulty of the task arises mainly for the following reasons: First, the robot needs to reason about its own state as well as the state of objects in the environment in a high-dimensional state-space. Second, this is a problem that requires the use of physics simulation to predict the state of the system after a robot action. Physics simulation in such cluttered scenes is computationally expensive to run. Additionally, the RTC problem is NP-Hard [4]. Finally, this is an under-actuated system, as the robot needs to find the correct controls to change the state of movable objects.

Despite these challenges, RTC problems arise frequently in real-world environments like our houses (e.g., retrieve an object from a fridge or a shelf) or in e-commerce applications (e.g., robots in warehouses reaching onto shelves to grasp objects), and, therefore, have the potential for near-term impact to home/service and warehouse robotics.

This paper is structured as follows. First, in Sec. II we review related work and in Sec. III we formulate the problem. We describe the proposed framework, Predictive Guided Framework (PGF), in Sec. IV. In Sec. V we describe the prediction models, and finally, in Sec. VI we evaluate PGF in a simulated warehouse comprised of up to thirty robots guided in parallel by a single human operator.

The contributions of this work are (1) a predictive framework that allows a robot to realise the difficulty of a problem before productive time is wasted and (2) the integration of this predictive approach in a motion planning algorithm with a human-in-the-loop. Finally, we provide public access to the source code as well as to the scene configurations[1].

II. RELATED WORK

An important area, related to this paper, is *sliding-autonomy*. In this line of work, a system can incorporate human intervention when needed and adjust its level of autonomy automatically [5]–[8]. Swamy et al [5], for example, proposed a shared-autonomy approach that tries to learn to assign to the human the robot that they would have chosen to operate. The approach works by observing the human operating a small number of robots and then trying to fit a model to explain the user's choices based on an internal scoring function. This approach addresses simple tasks. Learning a scoring model for real-world problems, like RTC, might be challenging. We take a similar approach but trying to predict the optimisation cost.

At the core of this paper are motion planning algorithms. They can generally be divided into three classes. Sampling-based planners [1], [9]–[14], trajectory optimisation-based planners [2], [15]–[20] and learning-based planners [21]–[26]. In this work, we are particularly interested in trajectory optimisation-based planners. In these algorithms, motion planning is formulated as an optimisation problem, where a cost function is defined over the trajectory and the planning is framed as the optimisation of its cost. Specifically, we use a similar motion planner to the one in our previous work [2] which is based on STOMP [16].

Although autonomous motion planning algorithms have seen success to various problems, for hard instances of the problem (i.e., high-dimensional kinodynamic state spaces) they struggle to find a solution. Human-in-the-loop systems demonstrated to be effective for various tasks, including geometric path planning [27], perception [28], [29], and grasping [30], [31]. This is mainly because humans are good at high-level reasoning [1], [2], [31]–[33], and is hard to derive one autonomously, in high-dimensional spaces.

Other works looked into failure recovery systems. Sankaran et al. [34], present a state machine-based failure recovery system with shared autonomy that can query a user when other autonomous solutions are exhausted. Although this is practical, in cases of trajectory optimisation-based motion planning, it can take a while for a robot to fail. Islam et al. [35] take a similar approach, where a Multi-Heuristic A* motion-planning algorithm is used to autonomously solve a motion planning problem. They propose a heuristic-based approach to estimate when a robot is in a "stagnation" region so they can query a human for guidance.

All the aforementioned works present interesting ways to deal with similar problems, however, all the robots act individually, ignoring the fact that the human is supervising multiple robots.

III. PROBLEM FORMULATION

We consider a *system* in a fulfilment warehouse centre. The system comprises r robots reaching onto r shelves to fulfil a total of m Reaching Through Clutter (RTC) tasks within T seconds. Once the robot completes an RTC task, it moves to the next one, until T seconds elapsed. The system also consists of a remote human operator that is available to provide high-level guidance to the r robots. At any time $t \in T$, the robots might require help from the human. Furthermore, the human high-level input impacts the underlying motion planner.

---

[1] `https://github.com/rpapallas/PGF`

We measure the performance of the system based on the success rate and the average planning time it takes to complete the tasks. As the number r increases, at some $t \in T$, there will be an increase in the number of robots requiring human help. We assume that out of these robots, one will benefit the most from immediate human help. Therefore, at any given time, we want to assign to the human operator the robot that will benefit the most from his input.

To do this, we would like to *predict* which robots are likely to fail, if any, and estimate a value that indicates how useful human input will be for each. We refer to this value as the *robot gain from human input*, or simply the *robot gain*.

Since human input is directly integrated to the motion planning algorithm, the prediction needs to happen at the motion planning level.

A. *Motion planning formulation*

Each robot environment comprises n_{obj} movable objects $\mathcal{O}$ and other static obstacles (e.g., walls). 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. Since the robot is working through clutter and cannot reach from the top, we assume that the robot motion is constrained to the plane and hence we simplify its configuration space to $SE(2)$.

Similarly, we denote the state of a movable obstacle $j \in \{1, \dots, |\mathcal{O}|\}$ with $x^j = (q^j, \dot{q}^j, \ddot{q}^j) \in \mathcal{X}^j$ where $q_j \subset SE(2)$ is the object's configuration and $\dot{q}^j$ and $\ddot{q}^j$ 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 comprises the linear and angular robot velocities denoted by $u_t \in U$ applied at time t for a fixed duration Δt. The state of the environment at time t is given by $x_t = \{x^r, x^1, \dots, x^{|\mathcal{O}|}\} \in \mathcal{X}^E$. The discrete time dynamics of the system are given by $x_{t+1} = f(x_t, u_t)$ where f is implemented by a physics simulator.

We assume a *trajectory optimisation-based* motion planner, and we have a trajectory $\tau = \langle u_0, u_1, \dots, u_{n-1} \rangle$ of n steps. 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 $s = \langle x_0, \dots, x_n \rangle$. We also have a cost function $\mathcal{C}(s)$ to compute the cost of the state sequence. The trajectory optimisation-based motion planner using the cost function, $\mathcal{C}$, will be reducing the cost at every iteration by updating the control sequence. The cost function has different terms to compute the cost of the trajectory say $c^i_x, c^i_y, \dots, c^i_z$. We define the *total* cost at iteration i as c^i which is the sum of the cost terms of the cost function: $c^i = c^i_x + c^i_y + \cdots + c^i_z$.

B. *Prediction formulation*

We denote the optimisation state O_{state} as $\langle n_{\text{obj}}, \langle c^{i-p}_x, c^{i-p}_y, \dots, c^{i-p}_z \rangle, \dots, \langle c^i_x, c^i_y, \dots, c^i_z \rangle \rangle$. Where n_{obj} is the number of objects in the environment and $\langle c^{i-p}_x, c^{i-p}_y, \dots, c^{i-p}_z \rangle$ represents the cost breakdown at iteration $i-p$, *before* human input. O_{state} essentially represents the cost breakdown *history* over the last p iterations. O_{state} captures the complexity of a given optimisation problem in the last p iterations.

We define two prediction models ϕ_{human} and ϕ_{auto} that predict the cost of the optimisation. The former, given the state of the optimisation algorithm, O_{state}, at iteration i, it predicts the total cost c^{i+l}_{human} representing the predicted cost of the optimisation l iterations in the future, assuming *human input* was provided. The latter, ϕ_{auto}, given the state of the optimisation algorithm, O_{state}, at iteration i, it predicts the total cost c^{i+l}_{auto} representing the predicted cost of the optimisation l iterations in the future, assuming *no human input* was provided. The *robot gain* is simply the difference between the two predictions. These two prediction models attempt to estimate how the optimisation would evolve in future iterations if human help is provided and if human help is *not* provided.

IV. THE PREDICTIVE GUIDED FRAMEWORK (PGF)

In this section, we introduce the Predictive Guided Framework (PGF). First, in Alg. 1, we describe the overall framework based on trajectory optimisation with a human-in-the-loop. In Sec. IV-A we describe how the human input is captured. In Sec. IV-B, we describe the cost function of the trajectory optimisation-based solver and in Sec. IV-C we describe how we construct the initial trajectories. Finally, in Sec. IV-D, we describe the allocator function that allocates a robot to a human.

Algorithm 1 Predictive Guided Framework

```
 1: procedure PGF
 2:     $x_0$ ← initial state of the system
 3:     $\tau$ ← init trajectory as a straight line to the goal
 4:     while $\tau$ not feasible do
 5:         $s \leftarrow R(x_0, \tau)$
 6:         $c^i_d, c^i_f, c^i_b \leftarrow \mathcal{C}(s)$
 7:         $g \leftarrow$ PREDICT($O_{\text{state}}$)
 8:         humanAssigned ← ALLOCATOR($g$)
 9:         if humanAssigned then
10:             input ← obtain input from human
11:             update cost function $\mathcal{C}$ based on input
12:             update $\tau$ based on input
13:             continue to line 5
14:         sample $k$ noisy trajectories from $\tau$
15:         rollout each of the $k$ trajectories from $x_0$ using $R$
16:         obtain cost for each rollout using $\mathcal{C}$
17:         $\tau$ ← trajectory with the lowest cost
18:     execute $\tau$
```

Alg. 1 describes the algorithm each robot is using to solve RTC problems. Each robot starts with observing the initial state (line 2). It then generates τ, a straight line trajectory to the goal object in line 3 (more details in Sec. IV-C). Then, in a while loop, until a *feasible* trajectory is found (or until time-limit exceeded), the solver starts to optimise the trajectory.

Initially, the solver rolls out the initial trajectory to obtain a sequence of states (line 5). It then obtains the current cost breakdown of the rollout (line 6). In line 7 we use the optimisation state, O_{state} (see Sec. III-B), to predict the robot gain g (more details in Sec. V-C). The predicted gain, g, is then given to the ALLOCATOR function (line 8) which decides if the robot will get human help. If the allocator decides that the current robot should be assigned to the human (line 9), the solver stops and the robot asks for human help. The human is prompted for input (line 10) and the input is integrated in the optimiser (lines 11 and 12). We describe how the input is integrated from Secs IV-A to IV-C. Then we continue to line 5 (line 13).

If human input is not required, then we sample k noisy trajectories from τ (line 14). 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 (line 15) and obtain the cost for each (line 16) and keep the trajectory with the lowest cost (line 17).

A. *Human Input*

In Alg. 1 between lines 10 and 12, we integrate the provided human input to the cost function and we also update the trajectory, τ. The high-level input, provided by the human operator, suggests a particular object, o_j, to be pushed to a particular region on the plane. We can formalise the human input as the triple (o_j, x_j, y_j), where $o_j \in \mathcal{O}$ is an object and (x_j, y_j) is a point on the plan that o_j needs to be pushed to. The user provides the input (Alg. 1 line 10) using a Graphical User Interface (GUI). The GUI is depicted in Fig. 1. The user is presented with a view of the scene in the simulator (that replicates the real-world configuration). The user uses the mouse pointer to first select o_j and then the point on the plane. Now that we defined the human input, we can describe PGF's cost function.

B. *Cost Function*

The cost function is used in Alg. 1 in line 6 and is also updated in line 11.

1) No human input provided: If no human-input is provided, the cost function for a state sequence s is defined as $\mathcal{C}(s) = c_d + c_f + c_b$:

- $c_d = w_d \cdot \mathrm{d}(\mathbf{q^{ee}}, \mathbf{q^{g}})$: The Euclidean distance from the robot's end-effector to the goal object o_g.
- $c_f(x_i) = \sum_{j=1}^{|\mathcal{O}|} w_f \cdot F(x_i^j)$: 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. In our implementation, F is provided by a physics simulator.
- $c_b(x_i) = \sum_{j=1}^{|\mathcal{O}|-1} w_b \cdot \mathbb{1}_b(\mathbf{x_n}, \mathbf{q^{ee}}, \mathbf{q^{i}})$: We check if there is a blocking obstacle in the robot's end-effector in the last state of the state sequence (x_n). $\mathbb{1}_b$ is an indicator function that returns 1 if the robot has an obstacle in the hand, 0 otherwise.

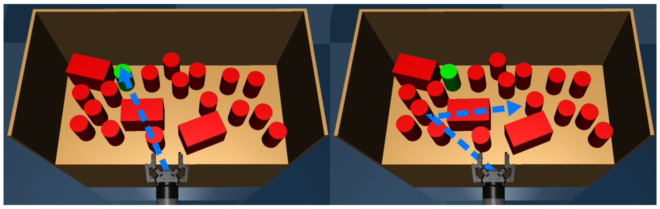

(a) Reaching goal object (b) Pushing an object

Fig. 2: Initial trajectories. The arrow illustrates the trajectory. In (b) the object to be pushed is the box the arrow penetrates.

The first cost term, c_d, is defining how close the robot is to grasp the desired object and is, therefore, indicating if the robot is grasping the goal object at the end of the trajectory. To avoid trajectories that forcefully push objects against static obstacles, we use the second term, c_f. The third term, c_b encourages the planner to find trajectories that do not get blocking obstacles in the robot's hand. This cost function (including their parameters, which we report in the results section) was experimentally chosen and provided a good cost function for the problem of Reaching Through Clutter.

2) Human input provided: If human input is provided (Alg. 1, line 10), we update the cost function (line 11) to integrate the input in the optimisation. The update is two-fold, we first push the operator's indicated object to the desired position using $\mathcal{C}(s) = c_h + c_f + c_b$ and then we reach for the goal object using $\mathcal{C}(s) = c_d + c_f + c_b$.

$$c_h = w_h \cdot \mathrm{d}(\mathbf{q_n^i}, \mathbf{q^i_{\text{desired}}}) \tag{1}$$

For a high-level input (o_i, x_i, y_i), c_h is the weighted Euclidean distance of o_i position at the final state, q_n^i, with the user's provided position, q^i_{desired}, of that object. This cost term will encourage the solver to explore trajectories where the object indicated by the human is pushed towards the desired position.

C. *Initial trajectories*

In Alg. 1 line 3, we initialise a straight-line trajectory. We use straight-line trajectories because they are cheap to compute (no physics simulation is required) and they are effective for RTC problems.

We depict two such initial trajectories in Fig. 2. The first trajectory, Fig. 2a, is the initial trajectory for reaching the goal object and is a straight line trajectory from the robot's current position to the position of the goal object. The second trajectory, Fig. 2b, is the initial trajectory for pushing an object to its desired position (that we update in Alg. 1, line 12). This trajectory is following a straight line from the object's current position to its desired position.

We have now defined the components of the motion planner, and next we will outline how PGF decides when to ask for human help.

Algorithm 2 Allocator

```
1: function ALLOCATOR($g_i$)
2:     if window not created then
3:         create window for $T_{\text{window}}$ seconds
4:     $P \leftarrow P \cup \{g_i\}$
5:     if $T_{\text{window}}$ not elapsed then
6:         wait until $T_{window}$ seconds elapsed
7:     return $g_i == \max(P)$
```

D. Allocator

In Alg. 1 in line 8, we make a call to ALLOCATOR to check if the robot can ask for human help. This function is described in Alg. 2. The state of the allocator function is shared with all robots, and it creates timed windows where robots join a pool of robots that are "bidding" for human time. If a window is not currently active, the system creates a window for T_{window} seconds (line 3). Robots join the active window, and they add their estimated robot gain, g_i to the P set (line 4). If the human is currently busy, guiding a robot for example, the robots *do not* join the window, and they continue autonomously until the human becomes available again. Once a robot joins a window, it waits for T_{window} seconds (line 6) for other robots to join. Finally, the robot with the highest robot gain is assigned to the human (line 7). Ties are resolved randomly. To ensure fairness in human time allocation and avoid starvation, when the robot's planner hits local minima, its gain is set to the highest value to guarantee that the robot will get human time.

So far, we introduced the PGF framework that employs a predictive approach to decide when to ask for human help and the allocator that uses the predicted values to allocate the appropriate robot to the human. In the next section, we outline implementation details of the predictors.

V. PREDICTION MODELS

In this section, we describe the implementation of the prediction models. For the problem of RTC we train two Deep Neural Networks (DNNs) and use them as our prediction models.

A. The prediction algorithm

Algorithm 3 Predictor

```
1: function PREDICT($O_{\text{state}}$)
2:     $c^{i+l}_{\text{human}} \leftarrow \phi_{\text{human}}(O_{\text{state}})$
3:     $c^{i+l}_{\text{auto}} \leftarrow \phi_{\text{auto}}(O_{\text{state}})$
4:     $g \leftarrow c^{i+l}_{\text{auto}} - c^{i+l}_{\text{human}}$        ▷ Predicted robot gain
5:     return $g$
```

In Alg. 1 in line 7, we make a call to a PREDICT function. This function is described in Alg. 3. Using two prediction models ϕ_{human} and ϕ_{auto}, we predict two future costs. In line 2 we predict the total cost of the optimisation l iterations in the future, assuming human input. In line 3 we predict the total cost of the optimisation l iterations in the future, assuming *no* human input. The algorithm then estimates the robot gain, g (line 4). In line 5, we return g to Alg. 1 and line 7.

The predictors take as input the optimiser state, O_{state} (Sec. III-B). This optimiser state comprises the number of movable objects, n_{obj}, in the environment as well as the *cost history* of the optimiser up to the current iteration. We found that using the cost term from the last iteration alone, as input to the model, was not enough to capture the complexity of a problem, and instead we found it more successful to look at the history of the cost.

It is noteworthy that the learned gain does not assume that human input will *always* improve the problem (i.e., decrease the cost). The gain will also capture when *not* to ask for human help (i.e., learn the problems where human input is not effective).

B. Structure of the datasets

We generated two different set of data. One dataset for training ϕ_{auto} ($\mathbb{X}_{\text{auto}}$, the autonomous dataset) and another dataset for training ϕ_{human} ($\mathbb{X}_{\text{human}}$, the human dataset).

The dataset for ϕ_{auto} consists of the optimisation cost breakdown per scene. We generated a number of distinct scenes (different rearrangements of the objects on the shelf) and ran the PGF optimiser (without human intervention) to solve them and collect data. We have the following autonomous training set:

$$\mathbb{X}_{\text{auto}} = \langle \ldots, \langle n_{\text{obj}}, \langle c^0_d, c^0_f, c^0_b \rangle, \ldots, \langle c^m_d, c^m_f, c^m_b \rangle \rangle, \ldots \rangle$$

That is, for each scene, n_{obj} is the number of objects in that scene, and $\langle c^0_d, c^0_f, c^0_b \rangle$ is the cost breakdown at iteration 0 and m is the last iteration of the optimisation.

Similarly, the dataset for ϕ_{human} consists of the optimisation cost breakdown per scene before and after human input. We ran PGF over the scenes. We have the following human training set:

$$\begin{aligned}\mathbb{X}_{\text{human}} = \langle \ldots, \langle n_{\text{obj}}, \langle c^0_d, c^0_f, c^0_b \rangle, \ldots, \langle c^k_d, c^k_f, c^k_b \rangle,\\ \langle c^{k+1}_d, c^{k+1}_f, c^{k+1}_b \rangle, \ldots, \langle c^m_d, c^m_f, c^m_b \rangle \rangle, \ldots \rangle\end{aligned}$$

That is, for a scene, $\langle c^0_d, c^0_f, c^0_b \rangle, \ldots, \langle c^k_d, c^k_f, c^k_b \rangle$ is the cost breakdown from iteration 0 to iteration k *before human input*. $\langle c^{k+1}_d, c^{k+1}_f, c^{k+1}_b \rangle, \ldots, \langle c^m_d, c^m_f, c^m_b \rangle$ is the cost breakdown from iteration $k+1$ to iteration m *after human input*. n_{obj} is the number of objects in the scene and m is the last iteration of the optimisation. The only difference here is that we have a key iteration (iteration $k+1$) when human input was provided and we treat the cost breakdown differently before and after that point (i.e., the change in the cost after the human input).

C. Network architecture

The two sets, $\mathbb{X}_{\text{auto}}$ and $\mathbb{X}_{\text{human}}$, are used to train ϕ_{human} and ϕ_{auto}. We used Tensorflow for implementing both models [36]. ϕ_{auto} and ϕ_{human} are two feed-forward deep neural networks with the same architecture. After experimentation with different architectures, the current architecture consists of an input layer, three hidden layers and the output layer (the cost l iterations in the future). The hidden layers contain 64, 32 and 8 neurons with reLU activation function respectively.

As mentioned in Sec. III, O_{state}, is the optimisation state which comprises the cost breakdown in the last p iterations. In our implementation O_{state} is defined as $\langle n_{\text{obj}}, \langle c_d^{i-p}, c_f^{i-p}, c_b^{i-p} \rangle, \ldots, \langle c_d^{i}, c_f^{i}, c_b^{i} \rangle \rangle$. Therefore, to train these DNNs, we group the iterations in the $\mathbb{X}_{\text{auto}}$ and $\mathbb{X}_{\text{human}}$ datasets into groups of p iterations per scene to predict the total cost at iteration $i+l$. The input layer for the two models, therefore, is the optimisation state O_{state}. That is, the p iterations and the cost breakdown for each. The output layer for the two models is the total cost at iteration $i+l$. In our implementation, $l=10$, that is, the cost we predict is 10 iterations in the future.

VI. EXPERIMENTS & RESULTS

In this section, we evaluate the performance of PGF in a virtual warehouse with a different number of robots for the problem of Reaching Through Clutter. As we explained in Sec. I-A, there are three possible approaches to integrating human high-level guidance to robotic planning:

1) Robots asking for guidance at the beginning of *every* problem [1]. This has the disadvantage that it spends valuable human operator time unnecessarily, even for problems that can be solved autonomously.
2) A robot asking for guidance only if its autonomous planner gets stuck. We implemented this approach in our previous work [2], and named it OR-HITL. In that work, our results showed that this approach outperforms (1) above, showing similar planning performance, but with significantly reduced human involvement. However, OR-HITL waits for the autonomous planner to get stuck, which can take a while.
3) The approach we present in this paper, PGF, which tries to predict the benefit of human help, and, therefore, recruit human for help earlier, before the autonomous planner actually gets stuck.

Therefore, below, we compare PGF to OR-HITL, to test our hypothesis that PGF will lead to using human guidance in a similar amount with OR-HITL, but the overall planning times (and therefore the overall number of tasks that are completed by the robots in the warehouse) improve due to PGF's predictive power. PGF uses the predictive approach to estimate which robot will benefit the most from human help and assign that robot to the human first, while OR-HITL queries the human when stuck in local minima and the human is allocated to the first robot that gets stuck.

Next, we describe the experimental setup. Then, in Sec. VI-B, we explain how we train the two prediction models, ϕ_{human} and ϕ_{auto}. Finally, in Sec. VI-C, we evaluate the proposed Predictive Guided Framework in a virtual warehouse with a different number of robots.

A. Experimental setup

We trained the networks on a computer with Intel© Core™ i7-4790 CPU @ 3.60GHz, 16GB RAM. We use TensorFlow 2.0 [36] with Keras. The training batch size was 50 and use Keras' EarlyStopping callback for adaptive number of epochs. We use the "Adam" optimiser with Root Mean Squared Error (RMSE) as the loss function. T_{window} for ALLOCATOR was 2 seconds. We use a randomiser to generate problems. Each problem consists of a scene with 20 movable objects on a shelf and the robot. The randomiser shuffles the movable obstacles in collision-free places including the goal object but ensures that the goal object is always at the back row of the shelf to create more challenging problems. We use MuJoCo [37] and its C++ API to implement the system dynamics. We use $k = 8$ 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 and the simulator's integration step size was 0.0015. Finally, the cost function's parameters are: $w_d = 4000, w_f = 30, w_b = 450$ and $w_h = 300$. All the details can also be found in the provided code repository.

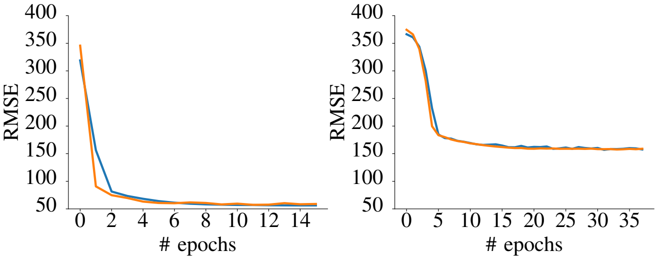


Fig. 3: ϕ_{auto} (left) and ϕ_{human} (right). The blue line is the loss function (Root Mean Squared Error) on the training set, and the orange line is the loss function on the validation set.

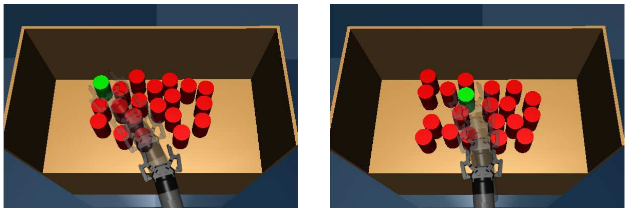
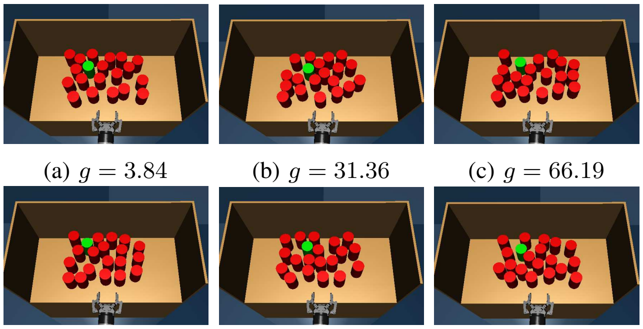


Fig. 4: Two instances of an RTC problem. The problem on the left is challenging, and the robot is very likely to ask for help, while the problem on the right is less challenging and the robot might be able to solve it autonomously.

B. Training ϕ_{human} and ϕ_{auto}

For the human dataset, we collected data from 4 participants (one is an author of the paper) by running the algorithm in 1,500 distinct scenes. The data collected were used to train ϕ_{human}. For the autonomous dataset, we collected data from 4,000 distinct scenes to train ϕ_{auto}. In these 4,000 problems, the robot tried to solve RTC problems fully autonomously, without any human intervention. These datasets were split 80-20% for training and validation, respectively.

Figure 3 shows the training results. The error for the autonomous predictor is much lower due to the availability of more data points. The RMSE of the autonomous predictor is 58.56 and for the human predictor is 157.49. To put these numbers into a perspective, for the two scenes in Fig. 4 the cost of the initial trajectory for the left and right problems is 749.56 and 0 (initial trajectory was feasible) respectively.



(a) $g = 3.84$ (b) $g = 31.36$ (c) $g = 66.19$

(d) $g = 66.64$ (e) $g = 49.52$ (f) $g = 51.06$

Fig. 5: Six robots (out of the twenty) requesting human help at the same time. c_{gain} is their predicted gain. Robot (d) has the highest gain and, therefore, gets the human.

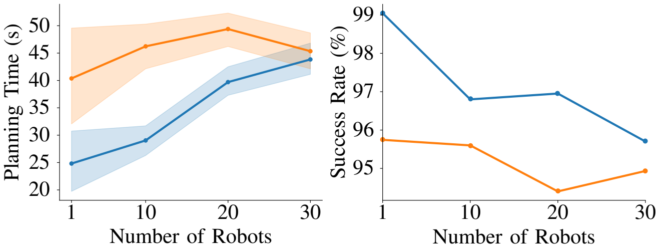


Fig. 6: Planning times (left) and success rate (right) for PGF (blue) and OR-HITL (orange) as the number of robots under the supervision of a single human-operator increases. Error shadow indicates the 95% Confidence Interval.

In the next section, using these prediction models, we evaluate the complete system in a virtual warehouse with a number of robots.

C. Coping with a fleet of demanding robots

Next, we set up a virtual warehouse[2] of 1-30 robots picking objects from shelves. We perform experiments with 1, 10, 20 and 30 robots under the human supervision, and compare how the performance is impacted as we increase the number of robots under the human supervision. Each robot was assigned with 50 random RTC tasks to solve, except the experiments with the single robot, where the robot was assigned to 100 random RTC tasks.

Some demonstration scenes are presented in Fig. 5. The figure presents six robots, out of the twenty in the virtual warehouse, that joined the same PGF window. g is the predicted robot gain for each. High gain means that human input could be more beneficial, while low robot gain means that the robot could solve the problem autonomously. Among the six robots, the robot in scene (d) was the one with the highest predicted gain and the one assigned to the human. The baseline, OR-HITL, solved scene (a) autonomously in 4 iterations, while it solved autonomously scene (b) in 8 iterations. For scene (c), OR-HITL hit a local minimum and queried the human on the 7th iteration and for scene (d), failed to solve the problem altogether. For scene (e), OR-HITL solved it autonomously in 20 iterations and in (f) hit a local minimum on the 4th iteration and queried the human. In (c) the gain is higher than (f) but the robot in (f) actually hits a local minimum on the 4th iteration as opposed to the 7th iteration in (c). This could be interpreted as that (f) is harder than (c). However, it is important to highlight that (1) the gain captures also when human input is not going to be useful (i.e., it might be that the human input in (f) is not going to be as beneficial) and (2) since this is stochastic optimisation, the iteration when a robot hits local minima can vary between runs.

Next, we measure the performance of these systems and present results in Fig. 6. Fig. 6-Left shows the average total planning time each robot spent on an RTC problem. The data suggest that the planning time of the robots increases as the number of robots under the human supervision increases as well, however, there is improvement in planning times when a human supervises from 1 to 20 robots using the PGF system. In these cases, the system correctly queried the human earlier and, therefore, allowed the robots to receive high-level human input earlier. The data suggest that once the number of robots increases from 20 to 30 this improvement shrinks. A possible explanation is that as the number of robots increases, there are more robots waiting for help, but they never get the human and, therefore, perform similar to OR-HITL (where help comes much later or not at all).

The success rate is shown in Fig. 6-Right. The success rate is the success percentage over all RTC problems each group of robots encountered. If a robot do not find a solution for a problem within 3 minutes, we consider the run a failure. The data shows a similar trend, but in general the success rate of both approaches is relatively high given the task difficulty.

Considering the total experiments, OR-HITL queried the human on the 25th iteration on average, while PGF queried the human on the 6th iteration on average. Finally, considering the experiment with the group of 20 robots, out of the 1,000 total problems the 20 robots encountered, PGF assigned the human to 307 of those (31% of the problems), while OR-HITL assigned the human to 175 of those (18% of the problems). Although there is an increase in human involvement with more problems, this is still better than the naive approach [1] that queried the human for 100% of the problems.

VII. CONCLUSIONS AND FUTURE WORK

We introduced the Predictive Guided Framework (PGF) which, by incorporating predictions, is able to query the human for help earlier, before valuable planning time is wasted. The predictions allow to sort robot requests and assign to the human the robot that will benefit the most. In a virtual warehouse with 20 robots, the system shows significant improvements compared to a baseline without predictions. The results also show that there is a cap on how many robots can effectively by guided by a single human.

Finally, the work can be improved in several ways. For example, although we propose a way to predict the robot gain using deep learning, we do not consider our contribution to be in this domain. Our framework demonstrates that a good prediction system is useful, but we acknowledge that there could be better ways to learn the robot gain; here we present one possibility. In the future, we will look into how we can improve the predictions using different learning methods and using different inputs to the prediction models.

---

[2] Using a powerful EC2 instance on Amazon Web Services ©: c5.24xlarge with 96 vCPUs and 192GiB of RAM

REFERENCES

[1] R. Papallas and M. R. Dogar, "Non-prehensile manipulation in clutter with human-in-the-loop," in *IEEE International Conference on Robotics and Automation*, 2020.

[2] R. Papallas, A. G. Cohn, and M. R. Dogar, "Online Replanning with Human-in-The-Loop for Non-Prehensile Manipulation in Clutter — A Trajectory Optimization based Approach," *IEEE Robotics and Automation Letters*, 2020.

[3] 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 *Robotics: Science and Systems*, 2016.

[4] G. Wilfong, "Motion planning in the presence of movable obstacles," *Annals of Mathematics and Artificial Intelligence*, vol. 3, no. 1, pp. 131–150, 1991.

[5] G. Swamy, S. Reddy, S. Levine, and A. D. Dragan, "Scaled autonomy: Enabling human operators to control robot fleets," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 5942–5948.

[6] M. B. Dias, B. Kannan, B. Browning, E. Jones, B. Argall, M. F. Dias, M. Zinck, M. M. Veloso, and A. Stentz, "Sliding autonomy for peer-to-peer human-robot teams," in *Proceedings of the international conference on intelligent autonomous systems*, 2008, pp. 332–341.

[7] D. J. Bruemmer, D. D. Dudenhoeffer, and J. L. Marble, "Dynamic-autonomy for urban search and rescue." in *AAAI mobile robot competition*. Menlo Park, CA, 2002, pp. 33–37.

[8] B. Sellner, R. Simmons, and S. Singh, "User modelling for principled sliding autonomy in human-robot teams," in *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*. Springer, 2005, pp. 197–208.

[9] J. A. Haustein, J. King, S. S. Srinivasa, and T. Asfour, "Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3075–3082.

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

[11] A. Krontiris and K. E. Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3924–3931.

[12] J. E. King, J. A. Haustein, S. S. Srinivasa, and T. Asfour, "Nonprehensile whole arm rearrangement planning on physics manifolds," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2508–2515.

[13] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, "Fast and resilient manipulation planning for target retrieval in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3777–3783.

[14] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3940–3947.

[15] C. Park, J. Pan, and D. Manocha, "Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of the international conference on automated planning and scheduling*, vol. 22, 2012.

[16] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[17] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494.

[18] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9. Citeseer, 2013, pp. 1–10.

[19] N. Kitaev, I. Mordatch, S. Patil, and P. Abbeel, "Physics-based trajectory optimization for grasping in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3102–3109.

[20] D. Koert, G. Maeda, R. Lioutikov, G. Neumann, and J. Peters, "Demonstration based trajectory optimization for generalizable robot motions," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp. 515–522.

[21] 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.

[22] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4238–4245.

[23] 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 *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2018, pp. 1–9.

[24] W. Bejjani, M. R. Dogar, and M. Leonetti, "Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[25] 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.

[26] A. Bajcsy, D. P. Losey, M. K. O'Malley, and A. D. Dragan, "Learning robot objectives from physical human interaction," in *Conference on Robot Learning*. PMLR, 2017, pp. 217–226.

[27] 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.

[28] B. Pitzer, M. Styer, C. Bersch, C. DuHadway, and J. Becker, "Towards perceptual shared autonomy for robotic mobile manipulation," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 6245–6251.

[29] D. J. Butler, S. Elliot, and M. Cakmak, "Interactive scene segmentation for efficient human-in-the-loop robot manipulation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2572–2579.

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

[31] A. E. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. ACM, 2012, pp. 1–8.

[32] C. Bringes, Y. Lin, Y. Sun, and R. Alqasemi, "Determining the benefit of human input in human-in-the-loop robotic systems," in *2013 IEEE RO-MAN*, 2013, pp. 210–215.

[33] R. Papallas, "Human-in-the-loop planning and control for non-prehensile manipulation in clutter," Ph.D. dissertation, University of Leeds, 2021.

[34] B. Sankaran, B. Pitzer, and S. Osentoski, "Failure recovery with shared autonomy," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 349–355.

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

[36] M. Abadi, A. Agarwal *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[37] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.