A Composable Framework for Policy Design, Learning, and Transfer Toward Safe and Efficient Industrial Insertion

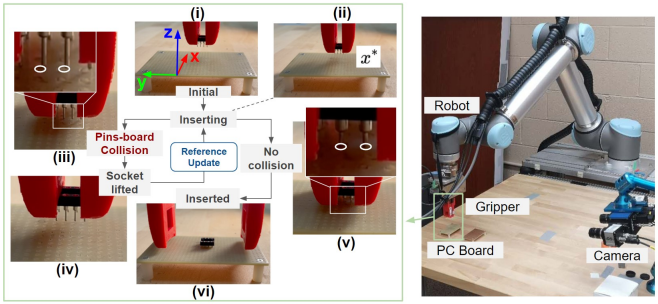
Rui Chen, Chenxi Wang, Tianhao Wei, Changliu Liu

***Abstract*— Delicate industrial insertion tasks (e.g., PC board assembly) remain challenging for industrial robots. The challenges include low error tolerance, delicacy of the components, and large task variations with respect to the components to be inserted. To deliver a feasible robotic solution for these insertion tasks, we also need to account for hardware limits of existing robotic systems and minimize the integration effort. This paper proposes a composable framework for efficient integration of a safe insertion policy on existing robotic platforms to accomplish these insertion tasks. The policy has an interpretable modularized design and can be learned efficiently on hardware and transferred to new tasks easily. In particular, the policy includes a safe insertion agent as a baseline policy for insertion, an optimal configurable Cartesian tracker as an interface to robot hardware, a probabilistic inference module to handle component variety and insertion errors, and a safe learning module to optimize the parameters in the aforementioned modules to achieve the best performance on designated hardware. The experiment results on a UR10 robot show that the proposed framework achieves safety (for the delicacy of components), accuracy (for low tolerance), robustness (against perception error and component defection), adaptability and transferability (for task variations), as well as task efficiency during execution plus data and time efficiency during learning.**

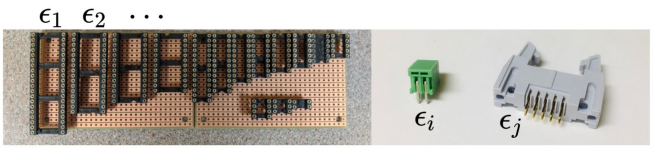
I. Introduction

In modern industrial assembly, robotic arms are widely used in many tasks. However, it remains challenging for robots to be applied to delicate multiple-contact insertion where the components can be easily bent or damaged. Specifically, this paper considers the task to insert integrated circuit (IC) chips to printed circuit (PC) boards where the IC chips have multiple delicate pins and have low error tolerance during insertion. Our goal is to develop a robotic insertion framework that (a) ensures safety by preventing physical damage to either the chip or the board, (b) achieves high performance in terms of accuracy and efficiency, (c) handles perception errors and component defection, (d) is able to transfer to different tasks and hardware, and (e) is easy to interpret and maintain. To meet these goals, we choose to design the framework to be composable, which consists of modules that can be selected and assembled in various combinations.

Challenges to design such a framework arise from both task specifications and hardware limitations. Regarding the insertion task, the pins on the IC chips to be assembled can be easily bent upon collision with the PC board. Since the defects can hardly be detected, it is difficult for the robot to determine whether the defective chips are still valid for insertion and if so, how to adjust the insertion routine. Another challenge is that insertion tasks can vary widely in terms of component size and insertion tolerance. It is non-trivial to transfer the framework to various tasks without tedious tuning for each task instance. Besides task-related obstacles, the hardware implementation is also challenging. First, considering the detection of the workspace, the perception system might not be accurate enough to localize the PCB holes within the error tolerance for insertion. Second, there are communication delays in different modules in the robotic system, which imposes additional difficulty on real-time hardware control.



(a) Industial insertion task.



(b) Variation of components.

Fig. 1: Illustration of (a) a delicate industrial insertion task and (b) variation of components to insert. (a) (right) shows the system setup. (a) (left) shows the insertion procedure. We marked target hole locations in white in (iii) and (v). We define the Cartesian coordinate system with X, Y, and Z axis as shown in (i). (b) shows a sample of target components with various specifications ϵ.

This paper presents the design and hardware implementation of a composable framework that tackles the above challenges. To handle task variations as well as unperceivable defects, our prior work [1] proposed a tolerance-guided policy learning method, which 1) parameterizes the policy using the tolerance to ensure transferrability of the policy to new components and 2) maintains a tolerance-guided probablistic model to infer the best insertion pose based on failed insertion attempts in order to ensure adaptability of the policy to defective components. The method has been validated in simulation but yet to be implemented on real hardware. The challenges regarding hardware limitations and collision-induced pin deformation remain unsolved, which

[1] R. Chen, C. Wang, T. Wei, and C. Liu are with Carnegie Mellon University, Pittsburgh, PA, USA. ruic3, chenxiwa, twei2, cliu6@andrew.cmu.edu,

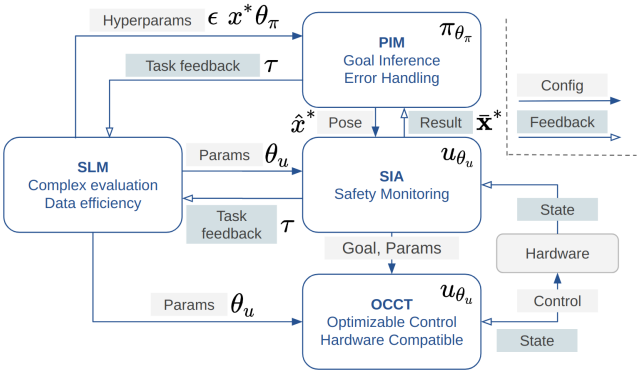


Fig. 2: Our insertion framework consists of four modules. Flow of module inputs and feedbacks are depicted in solid and hollow arrows respectively. Variables that are passed between modules are also marked (see section III for detailed description). The probabilistic inference module (PIM) generates insertion routines and send to the safe insertion agent (SIA). The SIA guides the insertion process by publishing Cartesian goals and handling failures. The optimal configurable Cartesian tracker (OCCT) directly communicates with the robot to perform optimization-based Cartesian space control. Finally, the safe learning module (SLM) optimizes the whole framework for task-specific objectives.

motivate our design of the proposed composable framework. In summary, we contribute to (a) an optimal configurable Cartesian tracker (OCCT) to better interface with hardware under communication delay, (b) a safe insertion agent (SIA) to handle the collision, (c) an extention of probabilistic inference module (PIM) in [1] to handling perception error, and (d) a novel safe learning module (SLM) using evolutionary methods to overcome the sim2real gap [2] and enable safe and data efficient learning on hardware. The performance of the proposed method is evaluated on a UR10 robot.

II. Related Work

Policy learning for robotic insertion tasks: Recently, learning-based methods have been applied to robotic insertion on hardware [3], [4], [5]. These approaches achieve high-precision assembly, but only consider robust components with limited variations. This paper considers delicate multiple-contact insertion with variations in more aspects such as component geometry and insertion tolerance.

Safe and data efficient learning on hardware: In this paper, we perform learning directly on hardware, where one immediate challenge is the data efficiency and safety. In literature, deep reinforcement learning-based approaches demonstrate limited data efficiency [6]. Even though the data efficiency can be improved [7], there is no guarantee on safety. On the other hand, principled logic-based policies can guarantee safety, which are commonly used in industrial applications. Since these policies usually have much fewer parameters than deep neural networks, directly learning with these policies can ensure data efficiency as well as safety. These learning methods are usually referred to as automatic parameter tuning [8]. Nonetheless, these policies are not always differentiable, hence unable to learn with gradient-based methods. This paper introduces evolutionary algorithms [9] to learn these parameters.

Goal inference under uncertainties: When there are perception errors and component defects, the actual insertion goal (as the relative position between the perceived board and the component) may deviate from the desired nominal insertion goal, and is not directly perceivable. To identify goal locations without visual sensors, existing methods (mostly in single peg-hold-insertion) utilize blind search, which first rub the peg across the surface to search for a partially-inserted pose [10], [11], [12], then performs goal identification through pattern matching [13], [14], [15]. These methods do not apply to delicate multi-contact insertion since the delicate components have little tolerance for shear forces during surface rubbing. Instead, we perform the goal inference from failed insertion attempts probabilistically [1].

III. Problem Specification and System Architecture

A. Problem Specification

We consider delicate industrial insertion tasks where a robotic arm inserts multi-pin components into PC boards with an attached gripper as illustrated in Fig. 1. The definition of the Cartesian coordinates is shown in Fig. 1(a). The Z axis points upwards vertically while the X and Y axes are parallel to the PC board. We denote the Cartesian pose of the gripper as $x := [t_x^\top, \theta_x^\top]^\top \in \mathbb{R}^6$ where $t_x \in \mathbb{R}^3$ is the translation vector and $\theta_x \in \mathbb{R}^3$ is the rotation vector in angle-axis representation. Each insertion task is encoded as a tuple $\langle \epsilon, x^* \rangle \sim \mathcal{E} \times \mathcal{X}^*$ where ϵ refers to the component specifications (e.g., geometries, insertion tolerance, etc.) and x^* refers to the nominal insertion pose calculated from task specification and workspace perception. For example, each component in Fig. 1b would correspond to a different ϵ and Fig. 1a(ii) shows one possible nominal insertion pose x^*. Notably, different component instances from the same task ϵ can have undetectable variations in the physical form due to manufacturing defect. Besides, the workspace perception is prone to errors. As such, the nominal insertion pose x^* might not be feasible for every component instance. This requires our insertion policy to detect whether x^* is feasible for each particular component and infer an alternative insertion pose $\hat{x}^*$ if necessary. Hence, we learn two functions: (a) an adaptation policy $\pi_{\theta_\pi} : \bar{\mathbf{x}}^* \times x^* \mapsto \hat{x}^*$ that proposes the next insertion pose $\hat{x}^*$ from the insertion history $\bar{\mathbf{x}}^*$ (as a list of states that result in collision) and the nominal goal x^* and (b) an insertion policy $u_{\theta_u} : \hat{x}^* \mapsto \tau$ that generates arm trajectories to track $\hat{x}^*$. θ_π and θ_u are adjustable policy parameters. Our objective is to maximize the expected reward R based on the executed trajectory τ over the task distribution $\mathcal{E} \times \mathcal{X}^*$:

$$\max_{\theta_\pi, \theta_u} \mathbb{E}_{\langle \epsilon, x^* \rangle \sim \mathcal{E} \times \mathcal{X}^*} \Big[\mathbb{E}_{\hat{x}^* \sim \pi_{\theta_\pi}(\bar{\mathbf{x}}^*, x^*)} \big[R(\tau)|_{\tau = u_{\theta_u}(\hat{x}^*)} \big] \Big]. \tag{1}$$

The inner expectation in (1) quantifies the insertion performance on a specific component insertion task $\langle \epsilon, x^* \rangle$ where the actual insertion pose $\hat{x}^*$ is adapted by π_{θ_π} and tracked by controller u_{θ_u}. The reward function R can be customized to

evaluated the executed trajectory τ based on various performance indices such as insertion time, whether a collision happens, and maximum contact force on the component. Finally, the outer expectation in (1) allows the insertion framework to be optimized over the task distribution $\mathcal{E} \times \mathcal{X}^*$. Specifically, $\mathcal{X}^*$ covers all possible insertions poses on the PC board (see Fig. 1a(ii) for an example) and $\mathcal{E}$ includes all possible components (see Fig. 1b for examples).

There are several sub-goals we aim to achieve when solving problem (1): 1) minimizing the learning efforts for the robot by incorporating as much prior knowledge as possible in the policy design; 2) minimizing the manual integration and tuning efforts by instantiating automatic parameter tuning on hardware; and 3) accounting for limitations on resources by enabling flexible scheduling of computation resources for efficient hardware execution.

B. Framework Architecture

The proposed composable industrial insertion framework is shown in Fig. 2. As an overview, the optimization objective in (1) is evaluated together by three modules: the probabilistic inference module (PIM), the safe insertion agent (SIA), and the optimal configurable Cartesian tracker (OCCT). The PIM constitutes the adaptation policy π_{θ_π} while the SIA and OCCT together consitute insertion policy u_{θ_u}. Then, the overall optimization (1) is solved by a safe learning module (SLM). Next, we elaborate on each of these modules and describe how they interact to solve (1).

Given an insertion task $\langle \epsilon, x^* \rangle$, the PIM π_{θ_π} proposes an insertion pose $\hat{x}^*$ and passes it to the insertion policy u_{θ_u}. Inside u_{θ_u}, the SIA first generates a sequence of sub-goals which starts at current gripper pose x and ends with the insertion pose $\hat{x}^*$. The OCCT then controls the arm to track each sub-goal in the sequence while monitoring collision. When the component collides with the board, the SIA notifies the PIM to adjust $\hat{x}^*$ and repeats the insertion attempts. An insertion succeeds when the Z position of $\hat{x}^*$ is reached without collision and XY position errors are within tolerance. An insertion fails when a maximum number of failures occur or when the component is determined as un-insertable if the maximal probability for a successful insertion drops below a threshold. One insertion loop in SIA terminates when either the insertion succeeds or it fails. The executed trajectory τ is then evaluated by a customizable reward function R. Finally, the SLM may optimize SIA, OCCT, and PIM to maximize R. The SLM is able to learn both offline and online. During offline training tasks, the SLM learns general configurations of these modules. During online execution, the SLM can continue to fine-tune those configurations based on newly received tasks.

The composable framework achieves the sub-goals in the following way. First, the framework is able to incorporate prior knowledge in SIA regarding how to perform insertion (to be elaborated in section IV-A). Second, the system parameters can be automatically learned using SLM (to be discussed in section IV-D). Third, the modules to can scheduled to run at different frequency to maximize the usage of computation resources.

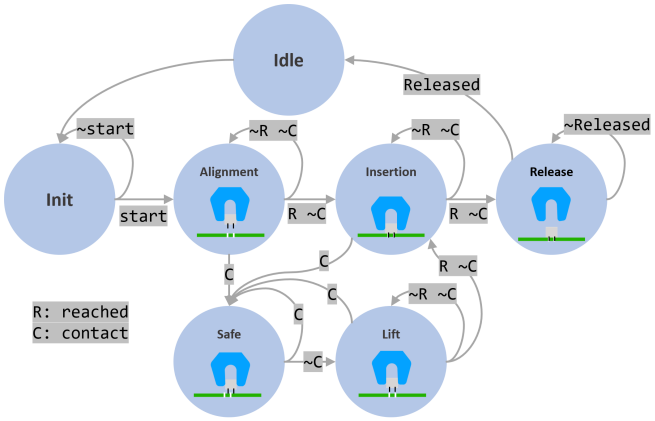


Fig. 3: State machine of SIA.

IV. Module Design

This section provides a detailed discussion on each module in our insertion framework in terms of their functionalities and the challenges that they address. We also discuss the transferability of our framework to other manipulation tasks in terms of efforts required.

A. Safe Insertion Agent

We design the safe insertion agent (SIA) to generate and execute insertion routines and handles component-board collisions. In the insertion task shown in Fig. 1, it is helpful to apply a logic-based policy for robustness and intepretability. One rule we summarize from observing human demonstrations is that in order to insert a multi-pin chip, one needs to first align the pins above the holes and then move downwards while keeping all the pins perpendicular to the board. Another rule is that when the pins make contact with the board, we lift the component to prevent damage and minimize the shear force. We incorporate those two rules as a logic-based policy which guides the insertion routine in SIA (see Fig. 3 for the state machine). When a state transition is triggered either by collision or by fulfillment of goal reaching, the SIA sends the next tracking reference to the OCCT.

The logic-based policy in SIA requires an external specification of two key poses: the *alignment pose* x^a and the *insertion pose* $\hat{x}^*$. x^a differs from $\hat{x}^*$ only in the Z component, i.e., $x^a = \hat{x}^* + \Delta z \, \vec{Z}$ where Δz is the *alignment offset*. We assume that the component is already grasped by the gripper when the insertion task starts. The SIA first enters *alignment* state and commands x^a. Then, the SIA enters *insertion* state and commands insertion pose $\hat{x}^*$. We use a force torque sensor mounted on the gripper to monitor the pin-board contact force. If a collision (force exceeding a threshold) is detected during the movements, implying that x^a and $\hat{x}^*$ need corrections, the SIA enters *safe* state and lifts the component. The collision is reported to an external module (the PIM in our case) which then corrects x^a and $\hat{x}^*$. Then, the SIA returns to *insertion* state and re-tries the

insertion with updated goals. This loop terminates under the conditions mentioned in section III-B. After that, the SIA releases the gripper, broadcasts the execution statistics, and reset to the *init* state.

Our SIA design enables motion-wise behavior configuration. The alignment state closes the major gap between initial position and target position with negligible chance of collision with the PC board, thus can be executed in high speed with generous tolerance on tracking error. The insertion state performs final insertion, thus should be optimized for stable and accurate control in lower speed. Such flexibility helps the framework to achieve both safety and efficiency.

B. Cartesian Tracking Control for Robot Arms

The optimal configurable Cartesian tracker (OCCT) controls the robot arm via the Robot Operating System (ROS) control interface [16] to track a Cartesian reference commanded by the SIA. We design OCCT as an optimization-based controller to achieve two major objectives: (a) robustness to hardware communication delays and (b) optimality of the generated trajectories. We desire the robot to track references in Cartesian space via least possible effort (e.g., torques) within time limits. In this work, we use a UR10 robot and such optimality is not guaranteed by current drivers such as UR ROS driver[1] and Moveit[2].

Communication delay exists in any ROS-based system. The delay is intractable and unpredictable since it depends on real-time network conditions. Thus, it is difficult to implement a reactive OCCT control loop with high sampling frequency. Nonetheless, a 125Hz[1] control loop is required for smooth arm motion. This motivates us to isolate OCCT and the arm driver. Instead of doing reactive control, we plan and send a whole trajectory that goes to the target pose specified by SIA. The trajectory is then executed by the arm driver[1] with high sampling frequency. To maintain responsiveness to insertion events as discussed in section IV-A, OCCT refreshes at 10Hz, the same rate as SIA. However, it will only modify an existing trajectory when the target pose specified by SIA changes or when there is a large tracking error. Note that the second case never occurs since the tracking error is always small to be shown in the experiments in section V-A.

To achieve trajectory optimality, we first generate the Cartesian trajectory by solving a T-step discrete linear-quadratic regulator (LQR) problem and then solve inverse kinematics (IK) for the corresponding joint trajectory. In the LQR problem, the Cartesian state $x_k := [t_{x_k}^\top, \theta_{x_k}^\top]^\top$ evolves according to a second-order dynamics under control input $u_k = \ddot{x}_k$ with time interval Δt. The objective is to drive x_k to the reference which is assumed to be the origin for simplicity. We define the run-time cost as $l_k(x_k, u_k) = \frac{1}{2}[x_k^\top, \dot{x}_k^\top] Q [x_k^\top, \dot{x}_k^\top]^\top + \frac{1}{2} u_k^\top R u_k$ and the terminal cost as $l_N(x_T) = \frac{1}{2}[x_T^\top, \dot{x}_T^\top] S [x_T^\top, \dot{x}_T^\top]^\top$ where $Q = v_Q \cdot \mathbf{I}$, $S = v_S \cdot \mathbf{I}$, and $R = \mathbf{I}$. $\mathbf{I}$ refers to identify matrices. There is an input constraint $u_k \in [-b_u, b_u]$ (to bound acceleration), an orientation constraint $\theta_{x_k} = \mathbf{0}$ (to ensure that the end-effector points downward vertically), a velocity constraint $\dot{x}_k \in [-b_v, b_v]$, and a terminal state constraint $x_T = \mathbf{0}$. The corresponding optimization problem is defined as

$$\begin{aligned} \min_{u_{0:T-1}} \ & \sum_{k=0}^{T-1} l_k(x_k, u_k) + l_T(x_T) \qquad (2) \\ \text{s.t.} \ & \begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \Delta t \cdot \mathbf{I} \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} + \begin{bmatrix} 0.5 \Delta t^2 \cdot \mathbf{I} \\ \Delta t \cdot \mathbf{I} \end{bmatrix} u_k \\ & u_k \in [-b_u, b_u], \ \dot{x}_k \in [-b_v, b_v], \ \theta_{x_k} = \mathbf{0}, \ x_T = \mathbf{0} \end{aligned}$$

Solving (2) yields the optimal planned trajectory $\mathbf{x} := \{x_0, \dot{x}_0, x_1, \dot{x}_1, \cdots, x_T, \dot{x}_T\}$. Denote the robot joint state as $q \in [-\pi, \pi]^{|q|}$ where $|q|$ is the number of joints. We then solve IK to convert $\mathbf{x}$ to joint space as $\mathbf{q} := \{q_0, \dot{q}_0, q_1, \dot{q}_1, \cdots, q_T, \dot{q}_T\}$ using the ICOP algorithm [17]. Notably, the IK is solved for Cartesian poses within a short range from the current state. As such, we can avoid singularity of IK by choosing insertion poses far from singularities. The OCCT finally sends the joint trajectory command $\mathbf{q}$ to the UR10 ROS driver for execution.

All OCCT parameters can be optimized by SLM to achieve user-defined goals. For example, the choices of planning horizon T and time interval Δt depends on the computation resources and scale of the task environment. A larger T improves reachability of the trajectory at the cost of longer planning time that grows exponentially, while a smaller Δt improves trajectory smoothness at the cost of reduced reachability. Since an optimal OCCT configuration can hardly be chosen manually, it is sensible to learn them using SLM.

C. Probabilistic Inference Module

The insertion goal has uncertainty due to possible component defection and perception errors. In our previous work [1], we proposed a model to handle component defection. The model infers the adapted insertion goal $\hat{x}^* = \pi_{\theta_\pi}(\bar{\mathbf{x}}^*, x^*)$ that the model believes to have the best chance to succeed based on the insertion history $\bar{\mathbf{x}}^*$ and the nominal goal x^*. In that case, we assume no perception error and the perception x_t^* exactly matches x^*, the true insertion goal with which a defect-free component can be inserted. Our prior model solves the following optimization:

$$\max_{\hat{x}^*} \ \mathbb{E}_{\langle \epsilon, x^* \rangle \sim \langle \mathcal{E}|_{x^*}, x^* \rangle} \Big[\mathbf{1}(\epsilon, \hat{x}^*) \mid \bar{\mathbf{x}}^* \Big], \qquad (3)$$

where $\mathcal{E}|_{x^*}$ represents the distribution of possible defects for that particular component which is known in the manufacturing process and $\mathbf{1}(\epsilon, \hat{x}^*)$ indicates whether the adapted insertion goal $\hat{x}^*$ is feasible. The above optimization constitutes the mapping $\pi_{\theta_\pi} : \bar{\mathbf{x}}^* \times x^* \mapsto \hat{x}^*$.

This paper considers perception error which can cause the perceived nominal goal x_t^* to deviate from the true nominal goal x^*. Thus, we extend the probabilistic inference to incorporate perception-induced goal uncertainty. It is assumed that the true nominal goal follows a Gaussian distribution given the perceived goal: $x^* \sim \mathcal{N}(x_t^*, \Sigma)$ where

[1] https://github.com/UniversalRobots/Universal_Robots_ROS_Driver

[2] https://github.com/ros-planning/moveit

the covariance Σ approximates the perception uncertainty and is tunable by SLM. This paper only considers the variance in X and Y directions denoted as Σ_x and Σ_y. Given the perceptual uncertainty, we need to construct a new policy $\pi_{\theta_\pi} : \bar{\mathbf{x}}^* \times x_t^* \mapsto \hat{x}^*$ that solves the following optimization:

$$\max_{\hat{x}^*} \mathbb{E}_{\langle \epsilon, x^* \rangle \sim \langle \mathcal{E}|_{x^*}, x^* \rangle, x^* \sim \mathcal{N}(x_t^*, \Sigma)} \Big[\mathbf{1}(\epsilon, \hat{x}^*) \mid \bar{\mathbf{x}}^* \Big]. \quad (4)$$

We calculate the expectation in (4) via sampling. The sample size N_{smp} influences both the estimation accuracy and the computational effort. Large sample size results in more accurate estimation but also takes longer time to compute. Similar trade-off also applies to the optimization process. Since we utilize CMA-ES [18] to optimize for the best $\hat{x}^*$, the generation size N_{gen}, population size N_{pop}, and elite size N_{elite} all influence the accuracy of the outcome $\hat{x}^*$ and the computational effort. In summary, θ_π contains the following parameters: N_{smp}, N_{gen}, N_{pop}, and N_{elite}, Σ_x and Σ_y. These parameters can all be learned by SLM instead of selected manually. We will empirically show that PIM leads to less insertion attempts until success than random search in section V-B and leave formal analysis of success rate as future work.

D. Safe Learning Module

The safe learning module (SLM) is designed to optimize the safety and efficiency of the insertion framework. The framework has many deeply coupled parameters that are difficult for hand tuning. Moreover, these parameters are not directly differentiable which precludes gradient-based learning. We choose CMA-ES as our learning algorithm since it is derivative free [18]. Briefly speaking, to solve (1), CMA-ES iteratively samples a set of candidate parameters θ_π and θ_u and keep only those who have a high *fitness* score based on the evaluation of the objective in (1). During training, we keep the environment configuration (e.g., robot, camera, etc.) the same since the parameters learned by CMA-ES would be invalid if the environment changes. We leave learning under changing environments as future work.

To further improve the efficiency of learning, we introduce the following two approaches: decomposition of the parameter space and augmentation of hardware learning with learning in simulation. Due to the curse of dimensionality, the learning complexity and the learning time grow exponentially with the number of parameters. Therefore, to improve the efficiency of learning, we divide the parameters to learn into groups where strongly coupled parameters are in the same group. Our composable framework design naturally suggests the division of parameters into two groups: one containing parameters of OCCT and SIA and the other containing parameters of PIM. The two groups are decoupled since the first group fully controls the execution of insertions (as u_{θ_u} in (1)) while the second only controls goal generation (as π_{θ_π} in (1)).

Besides parameter decomposition, we also augment hardware learning with learning in simulation to improve data efficiency. The motivation is that, it is impractical to train part of the modules directly on hardware due to intrinsic needs for large amounts of data. For example, learning PIM parameters requires insertions with diverse defective components. In addition, switching between components is time consuming and requires massive human labor. To mitigate this issue, we learn PIM parameters in simulation first, and then fine-tune the learned parameters on hardware. Notably, transferring policies learned in simulation to hardware is difficult due to the sim2real gap, i.e., the simulation cannot accurately model the real-world physics. With our composable framework, however, the sim2real gap is mitigated since we can train each module separately and use different learning platforms. Specifically, we perform simulation learning only on the PIM since it does not require high-fidelity physics simulation. On the other hand, we train OCCT and SIA directly on hardware to avoid the sim2real gap.

E. Transfer to Other Manipulation Tasks

The composable design of our framework enables transfer to other manipulation cases with appropriate modifications. Specifically, both SIA and PIM need to be re-designed to accommodate different manipulation requirements. The methods presented above is specific to multi-pin component insertion. On the other hand, both OCCT and SLM are task-agnostic can be directly applied to other manipulation tasks. OCCT provides a general robot control interface that facilitates policy execution on various hardware, while SLM is able to optimize the policy in different environments as long as an objective is available like in (1).

V. Experiments

This section presents hardware experiments with IC sockets on a UR10 robotic arm to demonstrate how our composable design satisfies user specifications under various constraints. The reason why we chose IC sockets is that the pins of IC chips are extremely delicate and are bent "out of the box". The remedy that human operators have to take is to align and flatten them before inserting them. We slightly simplified the problem by focusing on IC sockets, which are more robust but can still bend under large contact force. Since the shape of the IC socket body is very similar to that of the chip, it is not an oversimplification of the insertion task. In our experiments, we use a Ensenso X36 stereo camera kit to perceive the PC board and components to insert. The camera kit generates high-resolution point clouds that lead to localization standard deviations within 0.1 mm. Moreover, we record actual robot trajectories from the UR10 ROS driver which we consider as the ground truth.

Section V-A shows how OCCT accommodates limited hardware control API and achieves high precision motion execution. Section V-B demonstrates how SIA and PIM interact to handle perception errors and component defects during insertion, which also validate the safety of the system in handling collisions. Section V-C shows that SLM can efficiently optimize other modules and provide insights on how different parameters are coupled and how they affect the insertion performance. The results in this section represent

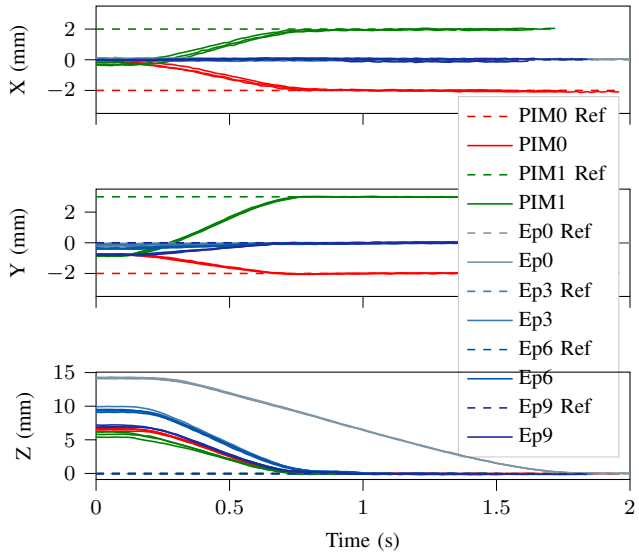


Fig. 4: Gripper trajectories in X, Y and Z positions during four training insertions (from different SLM learning epoches), i.e., Ep0, Ep3, Ep6, and Ep9, and two PIM-guided insertions, i.e., PIM0 and PIM1. The dotted lines are the target positions set by SIA, while the solid curves are the executed trajectories. Three repeated insertions are conducted for each task and are plotted in the same color to show repeatability. In training tasks, all goal poses share the same X and Y component. In PIM-guided tasks, desired XY positions are changed by PIM due to goal position inference.

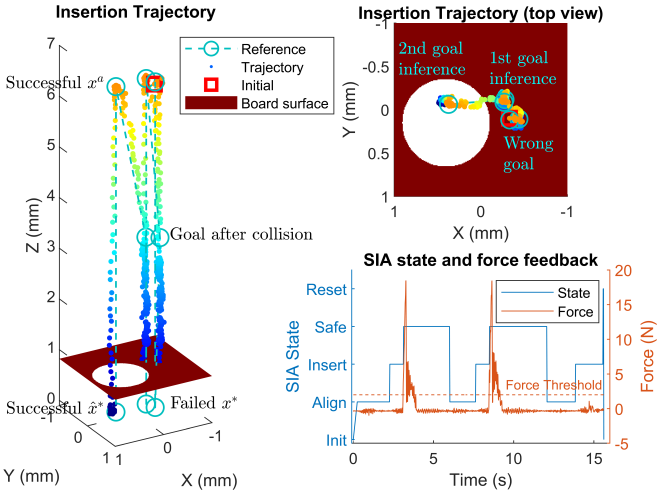


Fig. 5: Insertion with with inaccurate nominal insertion pose x^*. The left figure shows the gripper trajectory in configuration space with a top-down view on the top right. We also plot the SIA state and contact forces.

insertion of arbitrary components with vertical insertion trajectories (see Fig. 1b). The adaptability and transferrability of the method to new tasks is not directly shown unlike in [1], but can be supported by the short amount of time it needs to fine-tune the system on hardware.

A. Tracking in Cartesian Space

This section shows the static reference tracking of OCCT under several settings from two scenarios. First, we demonstrate that OCCT can generate optimal trajectories during the learning process where the OCCT parameters change. Then, we demonstrate that OCCT can generate optimal trajectories under different goals $\hat{x}^*$ proposed by the PIM. When solving the Cartesian trajectory $\mathbf{x}$, we applied constraints $\ddot{x}_{max} = b_u = 0.02\ \mathrm{m/s^2}$ and $\dot{x}_{max} = b_v = 0.02\ \mathrm{m/s}$. All trajectories are shown in Fig. 4. We repeated each experiment three times to demonstrate repeatability of the trajectories. During learning, the OCCT parameters and insertion heights Δz were changed in each epoch by SLM. We can see that in all epochs (from grey to blue curves), the OCCT generated and tracked smooth trajectories while maintaining linear motions (maximum 10^{-4} m horizontal error along X and Y). Thus, OCCT was able to solve optimal trajectories while guaranteeing safety constraints. Besides, when guided by PIM, the OCCT was also able to track XY references smoothly (red and green curves). In all experiments, OCCT showed repeatable trajectories, meaning that the OCCT well separated the generation of task trajectories from actual robot hardware execution.

B. Single Component Insertion with Multiple Attempts

This section demonstrates how SIA and PIM guided the insertion of an IC socket. We used an IC socket with 2×4 pins (circle pins with 0.6 mm) and applied our framework to insert it into a PC board with 1.0 mm diameter holes. Other types of sockets are also applicable since PIM can adapt to different components as described in [1]. To demonstrate error handling, we arbitrarily chose an inaccurate nominal insertion pose x^* (see Fig. 5) and bent one of the socket pins slightly. This simulated how perception errors and component defection fail insertions. During the insertion, we monitored the contact force on the pins via a force torque sensor mounted on the gripper. A collision would be alarmed when the contact force exceeds 2 N. This threshold could be adjusted depending on the robustness of the pins and the board. When collision happened, SIA entered the *safe* state, lifted the socket, and notified the PIM. The PIM then inferred the next insertion goal based on failed insertions and updates $\hat{x}^*$ according to (3). After receiving the update, the SIA made the next insertion attempt. Figure 5 illustrates the gripper trajectory and the SIA states throughout the process. In that example, after two collisions, the PIM located the real insertion position within insertion tolerance and the insertion succeeded. We see that even with erroneous goal specifications, SIA and PIM could detect insertion failures and make proper adjustments to accommodate those errors. Notably, the maximum instantaneous pin-board contact force was less than 20 N and any force larger than 5 N lasts only for very short amount of time. This was within safety range since we did not observe collision-induced pin deformation. In addition, we compared PIM to a random policy which sampled new insertion locations from a Gaussian distribution centered at x^*. Results showed that PIM took 2.8 insertions on average with an advantage of 0.9 over the random policy.

C. Automatic Parameter Tuning

In this experiment, we used SLM to optimize the parameters of OCCT, SIA, and PIM to improve the insertion efficiency and safety. Benefit from our composable design, the parameters to learn (see table I) can be divided into a low-level group and a high-level group. Each group can

Module	Parameters	Default	Range	Result
OCCT	v_Q	10	[0, 20]	13.102
OCCT	v_S	100	[90, 110]	91.605
OCCT	T_{OCCT}	1	[0.75, 3]	0.75
OCCT	$1/\Delta t$	10	[5, 20]	9.278
SIA	Δz	0.015	[0.005, 0.025]	0.0056
PIM	N_{smp}	500	[5, 1000]	334
PIM	N_{gen}	10	[2, 20]	2
PIM	N_{pop}	50	[3, 100]	44
PIM	N_{elite}	15	[3, 30]	15
PIM	Σ_x	0.0003	[0.00001, 0.0005]	0.000405
PIM	Σ_y	0.0003	[0.00001, 0.0005]	0.000363

TABLE I: Parameters to learn.

be learned separately. The low-level set consists of OCCT control parameters and insertion distance. The high-level set consists of PIM parameters which controls the goal inference procedure. To ease the learning process on hardware, we first learned each parameter set separately and then fine-tuned them jointly. Next, we describe the SLM optimization objectives and analyze the learning results to answer the following questions: (a) how much can SLM improve the performance, (b) how are the module parameters coupled or decoupled and how do they affect the insertion efficiency and safety, and (c) how data-efficient and scalable is SLM.

1) OCCT & SIA Learning on Hardware: We designed the following reward to jointly optimize the parameters: planned execution time $T_{\text{OCCT}} = T \cdot \Delta t$, planning frequency $1/\Delta t$, LQR cost v_Q, v_S, and alignment offset Δz:

$$R = -t_{\text{insert}}^2 - 10\ C - 0.1\ F - 100\ \mathbf{1}_{\text{infeas}} - 100\ \mathbf{1}_{\text{fail}}, \quad (5)$$

where t_{insert} is the insertion time, C is the collision count, F is the maximal contact force, $\mathbf{1}_{\text{infeas}}$ indicates failure of planning, and $\mathbf{1}_{\text{fail}}$ indicates failure of insertion. t_{insert} is the total time SIA spends in the insertion state for reaching a single $\hat{x}^*$ which includes the computation time of OCCT trajectory, execution time of the trajectory on hardware, and communication delays. The goal is to reduce the cycle time as much as possible while keeping the insertion safe and feasible. The learning results are shown in Fig. 6 (a), which shows that both the force and the time decrease with iterations. As for the parameters shown in table I, v_Q is increased, while v_S, T_{OCCT}, $1/\Delta t$, Δz are decreased.

2) PIM Learning in Simulation: The parameter θ_π in PIM needs to be learned to balance the accuracy of the inferred insertion pose and its computation time. It worth noting that we have a good initial estimation of the uncertainty Σ_x and Σ_y, which are directly obtained from hardware specifications. So we skipped them in this phase to reduce the learning time, and left them for the fine-tune step. We tested the PIM parameters on 100 randomly generated defected chip insertion poses. If PIM infers the correct insertion pose within 20 trials, we consider the insertion successful. The reward function is defined as:

$$R = -100\ \mathbf{1}_{\text{fail}} - 2\ L - t_{\text{comp}}, \quad (6)$$

where t_{comp} is the computation time of PIM, and L is the number of insertion trials. The learning results are shown in Fig. 6 (b), which shows that the average insertion trial

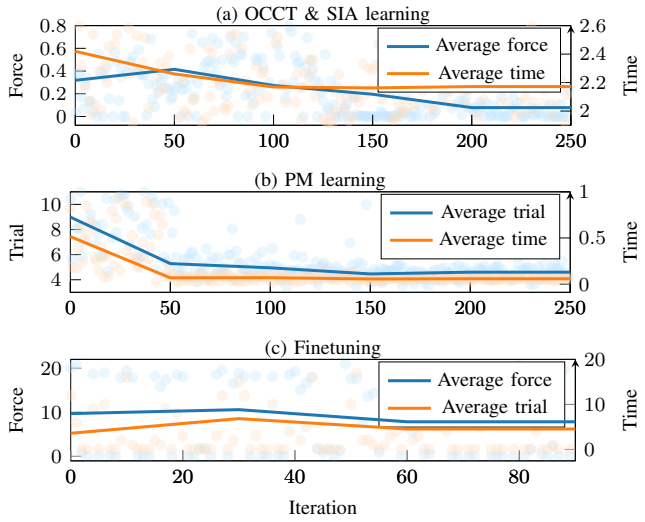


Fig. 6: Figures from top to bottom show curves for: OCCT & SIA learning, PIM learning, and fine-tuning. Every scatter point corresponds to a set of parameters. Infeasible data points are not counted in the average. We did not consider perception error in OCCT & SIA learning, so there was no collision and the forces were caused by robot movements. However, larger forces in fine-tuning were caused by collisions.

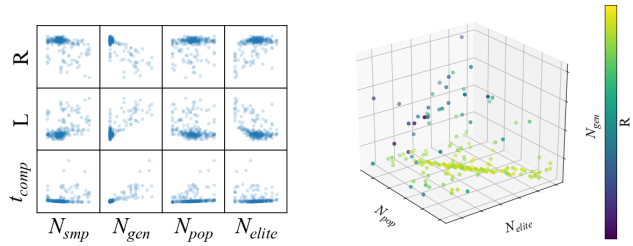


Fig. 7: Illustration of parameter coupling. Left: (a) Correlation analysis between parameters and metrics in PIM learning. Parameters are listed on the X axis, while metrics are listed on the Y axis. Each scatter point corresponds to one experiment. Most parameters do not show strong correlations to metrics. Right: (b) Reward distribution with regard to N_{gen}, N_{pop}, and N_{elite}. The reward shows a strong correlation to the elite-population ratio.

and average computation time were both reduced. As for the parameters shown in table I, N_{gen} was greatly reduced by the learning. This suggests a low N_{gen} is enough for accurate pose estimation. N_{smp} and N_{pop} were slightly decreased to reduce t_{comp}, and N_{elite} remained unchanged.

3) Fine-tuning on Hardware: We jointly fine-tuned all the parameters to verify our sim-to-real transfer and examine undiscovered coupling between different parameter groups. If *the reality gap* exists, the parameters learned in simulation can be further improved by fine-tuning on real hardware. Similarly, if there is undiscovered coupling between parameters from different groups, the performance will also be improved. To increase fine-tuning efficiency, the exploration range is reduced to around 20% of the range in learning since the parameters are already near optimal. We used the same reward function defined in eq. (5), because this reward reflects the whole insertion task performance. As shown in Fig. 6 (c), the performance does not show a significant change, which indicates that the parameters learned in simulation were successfully transferred to reality,

Exp.	Time
OCCT & SLA learning (hardware)	30 min
PIM learning (simulation)	180 min
Fine tuning (fine-tune)	60 min

TABLE II: Learning time.

and the parameters were well decoupled across groups. Notably, the force readings in OCCT & SIA learning phase was smaller than those during finetuning. This is because in the first case, we only provided feasible x^* to ensure collision-free insertions. As such, the force readings came from acceleration of the arm. In the finetuning phase, we deliberately fed infeasible x^* to cause collisions, resulting in contact forces in the 10 N to 20 N range.

4) Parameter analysis: We inspected the coupling among parameters by analyzing their correlations to different metrics in R. From Fig. 7 (a), we can see that the correlations between individual parameters and metrics are not obvious. But there are strong correlations when we visualize their joint distribution as shown in Fig. 7 (b). This suggests that the coupling exists and parameter tuning must be conducted jointly. Some parameters show a strong correlation to certain metrics, such as N_{gen} and t_{comp}. Hence, we can learn N_{gen} separately from others to further reduce learning time.

5) Learning efficiency: The learning time is shown in table II. PIM learning was performed on a PC with AMD Ryzen threadripper 3960x 24-core processor. No parallel computing was used. But PIM learning can be easily sped up by parallel computing. Other experiments were performed on a UR10 robot with a PC host equipped Intel Xeon(R) E5-2609 v4 1.7GHz 8 core CPU.

VI. Conclusion

This paper presented a composable framework for safe and efficient industrial insertion. The framework features four individual modules, each addressing different challenges: the OCCT for optimal insertion trajectory generation under constraints, the SIA for collision handling, the PIM for goal inference under component defection and perception errors, and the SLM for learning on hardware. We applied our framework on a delicate socket insertion task with a UR10 robotic arm. We showed that the framework was able to generate and execute optimal insertion trajectories under safety constraints, handle failed insertions and infer true insertion positions, and automatically tune module parameters for safety and efficiency on hardware. For future work, we will theoretically analyze the robustness of the proposed framework.

ACKNOWLEDGMENT

This work is supported by Efort Intelligent Equipment Co., Ltd. The authors would like to thank Brad Lisien, Dan Troniak, Yueqi Li, Weiye Zhao, Charles Noren, and Ruixuan Liu for their help in experiments, and Joseph Giampapa for his help in polishing the paper.

References

[1] C. L. Boshen Niu, Chenxi Wang, "Tolerance-guided policy learning for adaptable and transferrable delicate industrial insertion," in *4th Conference on Robot Learning*, Cambridge MA, USA., Nov. 2020.

[2] S. Koos, J.-B. Mouret, and S. Doncieux, "The transferability approach: Crossing the reality gap in evolutionary robotics," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 1, pp. 122–145, 2012.

[3] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 819–825.

[4] Y. Fan, J. Luo, and M. Tomizuka, "A learning framework for high precision industrial assembly," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 811–817.

[5] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, A. M. Agogino, A. Tamar, and P. Abbeel, "Reinforcement learning on variable impedance controller for high-precision robotic assembly," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3080–3087.

[6] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. Aparicio Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5548–5555.

[7] M. Vecerik, O. Sushkov, D. Barker, T. Rothörl, T. Hester, and J. Scholz, "A practical approach to insertion with variable socket position using deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 754–760.

[8] Y. Chen, "Automatic parameter tuning and its implementation for robot arms," in *1988 IEEE International Conference on Robotics and Automation Proceedings*, Apr 1988, p. 691–698 vol.2.

[9] S. Doncieux, J.-B. Mouret, N. Bredeche, and V. Padois, *Evolutionary Robotics: Exploring New Horizons*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2011, vol. 341, p. 3–25.

[10] H. Park, J. Park, D.-H. Lee, J.-H. Park, M.-H. Baeg, and J.-H. Bae, "Compliance-based robotic peg-in-hole assembly strategy without force feedback," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 8, pp. 6299–6309, 2017.

[11] M. W. Abdullah, H. Roth, M. Weyrich, and J. Wahrburg, "An approach for peg-in-hole assembling using intuitive search algorithm based on human behavior and carried by sensors guided industrial robot," *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 1476–1481, 2015.

[12] I. F. Jasim, P. W. Plapper, and H. Voos, "Position identification in force-guided robotic peg-in-hole assembly tasks," *Procedia Cirp*, vol. 23, pp. 217–222, 2014.

[13] S. R. Chhatpar and M. S. Branicky, "Particle filtering for localization in robotic assemblies with position uncertainty," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3610–3617.

[14] R. C. Luo, A. Chang, and C. Li, "A novel peg-in-hole approach based on geometrical analysis for inclined uncertainty," in *2017 IEEE international conference on advanced intelligent mechatronics (AIM)*. IEEE, 2017, pp. 891–896.

[15] Y.-L. Kim, B.-S. Kim, and J.-B. Song, "Hole detection algorithm for square peg-in-hole using force-based shape recognition," in *2012 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2012, pp. 1074–1079.

[16] S. Chitta, E. Marder-Eppstein, W. Meeussen, V. Pradeep, A. R. Tsouroukdissian, J. Bohren, D. Coleman, B. Magyar, G. Raiola, M. Lüdtke, *et al.*, "ros_control: A generic and simple control framework for ros," *The Journal of Open Source Software*, vol. 2, no. 20, pp. 456–456, 2017.

[17] W. Zhao, S. He, C. Wen, and C. Liu, "Contact-rich trajectory generation in confined environments using iterative convex optimization," in *ASME Dynamical Systems and Control Conference*, 2020.

[18] N. Hansen, "The CMA evolution strategy: A tutorial," *arXiv:1604.00772*, 2016.