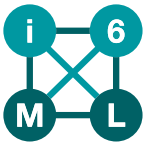


Chair of Machine Learning and Reasoning (Computer Science 6)

RWTH AACHEN UNIVERSITY

This thesis was submitted to the Chair of Machine Learning and Reasoning.

Open-World Robotic Assembly with Physically Feasible Planning

Bachelor Thesis

Presented by

Wenbo Ma
433351

Supervised by Daniel Swoboda, M.Sc.

1st Examiner Prof. Hector Geffner, Ph.D.

2nd Examiner Prof. Dr. rer. nat. Alexander Ferrein

Aachen, May 28, 2026

Abstract

Autonomous robotic assembly requires robots to perceive parts, determine feasible assembly orders, and execute contact-rich manipulation under geometric and physical constraints. This thesis studies this problem in LEGO Duplo assembly, where discrete stud-grid geometry makes task specification reproducible while still requiring accurate grasping, alignment, insertion, and intermediate-structure stability.

The proposed system integrates assembly-sequence planning, RGB-D perception, and collision-aware execution in a structured robotic pipeline. Given a target structure specification, assembly orders are generated using the Assembly-by-Disassembly principle and are filtered through grasp-reachability, collision, and press-stability checks based on a voxel representation of the assembly. The perception module combines open-vocabulary detection, instance segmentation, and CAD-based 6D pose estimation to localize task-relevant bricks, while the execution module performs grasping, placement, insertion, and release through a state-machine controller.

The system is evaluated in a MuJoCo-based simulation benchmark constructed using LIBERO tools and conventions, with increasing task complexity and comparison against a vision-language-action baseline. It is also validated on a physical UR5 setup with an Intel RealSense camera and a Robotiq 2F-style gripper. In simulation, the structured pipeline achieves higher success rates, lower stability violation rates, and more reliable insertion performance, especially for tasks with dense spatial constraints and longer assembly horizons. Real-robot experiments show that the same pipeline can transfer to physical LEGO assembly, while also exposing practical limitations caused by calibration accuracy, LEGO friction, gripper force, safety-limited robot motion, and contact-sensitive insertion. Overall, the results demonstrate that explicit geometric reasoning and physically grounded execution remain important for robust long-horizon robotic assembly.

Contents

1 Introduction

1.1 Motivation

Autonomous robotic assembly is an important research area in robotics and intelligent manufacturing, aiming to enable robots to perceive parts, plan assembly orders, and execute manipulation actions in physically constrained environments. Representative research scenarios include furniture assembly, industrial component assembly, and tabletop object construction tasks [13, 24, 40].

These application scenarios require robots not only to manipulate objects, but also to perceive the environment and generate physically executable assembly plans. During the assembly process, the robot must continuously reason about spatial relations, contact interactions, and structural dependencies between different parts. In many tasks, the robot must determine feasible assembly orders, select suitable grasp configurations, align components precisely, avoid collisions with surrounding structures, and maintain stable intermediate assemblies during insertion operations. Even small geometric errors or incorrect assembly decisions may lead to failed insertions or unstable structures. Consequently, robotic assembly is a representative long-horizon manipulation problem that requires the coordinated integration of perception, task planning, motion generation, and feedback control [13, 40]. Compared with simpler robotic manipulation settings, assembly tasks typically involve stronger geometric constraints, richer contact interactions, and more complex dependencies between sequential manipulation actions.

In practical scenarios such as flexible manufacturing, tabletop manipulation, and service robotics, robots must operate under cluttered scenes, uncertain object configurations, and previously unseen objects [3, 13, 20, 27]. These requirements also motivate recent vision-language-action (VLA) systems, which attempt to map visual observations and language instructions directly to robot actions [4, 5, 18, 51]. However, contact-rich assembly still requires strong geometric consistency, precise insertion, and stable intermediate structures. As a result, modern robotic assembly systems increasingly require the integration of autonomous planning, environment perception, collision-aware manipulation, and constrained motion generation under kinematic and geometric constraints.

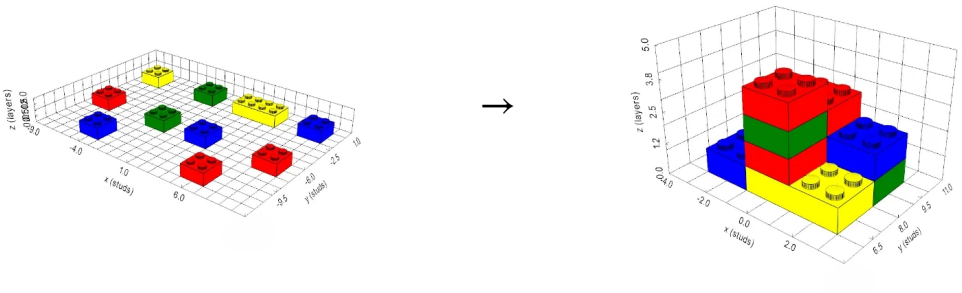


Figure 1.1: Example robotic assembly task used in this work. The left image represents the initial configuration with separated LEGO Duplo bricks, while the right image represents the target assembled structure.

1.2 Problem Setting and Challenges

LEGO-style assembly has become a commonly used platform in robotic assembly research because it provides a compact, reproducible, and scalable environment while still preserving many fundamental challenges of real-world assembly tasks [2, 31]. The discrete stud-grid structure simplifies geometric representation, and the modularity of LEGO components enables task generation with varying structural complexity. In this work, LEGO Duplo bricks are used as the assembly object category because their size is compatible with parallel-jaw robotic grasping while still requiring precise alignment and press-in insertion. At the same time, the sequential and interdependent nature of LEGO construction introduces substantial planning complexity. Since each assembly action may constrain future grasp accessibility, insertion directions, and structural support conditions, robotic systems must determine physically feasible assembly orders before execution.

Assembly Sequence Planning (ASP) is a computationally difficult combinatorial search problem [36, 46]. As the number of assembly components increases, the search space of feasible assembly orders grows rapidly, making exhaustive forward search increasingly impractical. Moreover, robotic assembly planning requires not only determining a valid assembly order, but also verifying geometric and physical feasibility during execution, including grasp reachability, collision-free insertion, and intermediate-structure stability. In contact-rich assembly tasks such as LEGO insertion, the robot must continuously reason about insertion directions, neighboring obstacles, structural dependencies, and support conditions.

Besides high-level planning, robotic assembly also depends heavily on robust perception capabilities. In long-horizon assembly tasks, the spatial configuration of the workspace continuously changes as new parts are inserted into the structure. Partial occlusions, self-occlusions, varying object orientations, and dense object arrangements further increase the difficulty of accurate object localization and pose estimation. Accurate 6D pose estimation is particularly important

in robotic assembly because even small translational or rotational errors may propagate into insertion failures during contact-rich manipulation [43, 45, 48].

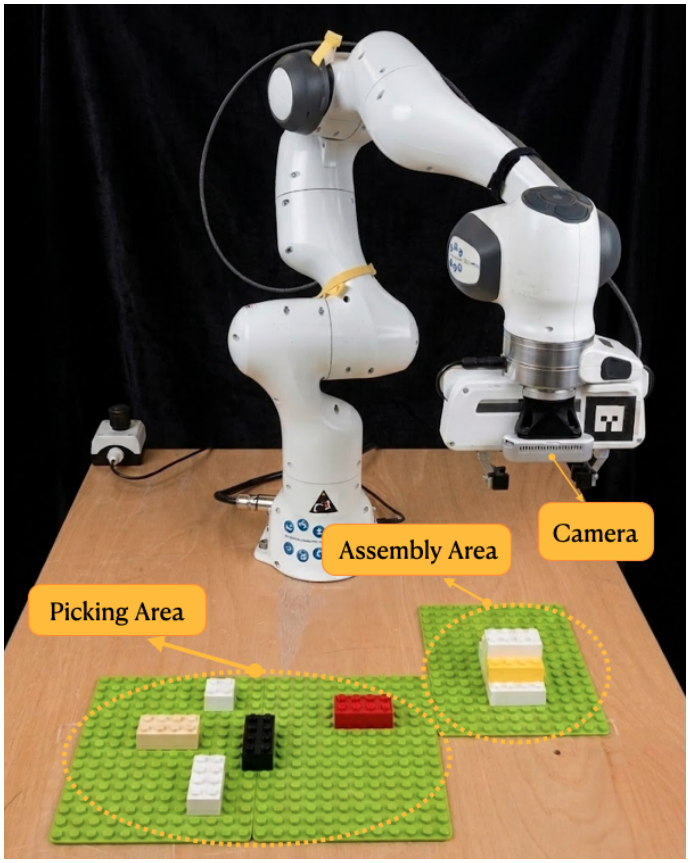


Figure 1.2: Example robotic assembly workbench illustrating the task setting, consisting of a robot arm, an RGB-D camera, separated LEGO Duplo bricks, and a target construction region.

Although recent end-to-end visuomotor policies and VLM/VLA-based systems have demonstrated promising generalization capabilities [4, 5, 18, 51], purely implicit reasoning may still struggle to maintain reliable geometric consistency and physical feasibility in contact-rich assembly tasks. Therefore, combining symbolic assembly reasoning with explicit geometric verification provides a practical compromise between generalization capability and physical reliability. This motivates the development of structured robotic assembly pipelines that combine high-level assembly reasoning with explicit geometric verification and physically grounded execution modules.

These challenges become particularly significant in LEGO-style assembly. Successful LEGO insertion requires precise alignment between studs and receiving holes while simultaneously maintaining collision-free motion and stable intermediate structures. During execution, the robot must continuously reason about insertion direction, neighboring obstacles, grasp accessibility, and structural support conditions. Small pose estimation errors may accumulate across

multiple assembly stages and eventually lead to insertion failure or structural instability. Consequently, explicit geometric reasoning and physically grounded planning remain important for robust robotic assembly systems.

Motivated by the RoboCup Smart Manufacturing League (SML) assembly workbench setting, this thesis studies a LEGO Duplo robotic assembly task under semi-structured tabletop environments with randomized initial object configurations. Figure 1.2 illustrates an example robotic assembly workbench for this task setting. The example setup consists of a robot arm, a parallel-jaw gripper, an RGB-D camera mounted above the workspace, initially separated LEGO Duplo bricks, and a target construction region where the structure is built.

As illustrated in Figure 1.1, the robot is required to construct a target structure from initially separated parts. The task requires end-to-end robotic autonomy, including generating executable assembly sequences, identifying and localizing target parts, selecting collision-free grasp configurations, planning feasible trajectories, and executing accurate placement and stud-engagement insertion operations.

1.3 Proposed Approach

The proposed system follows a structured planning–perception–execution pipeline. Given a target structure specification, the system derives executable assembly sequences using Assembly-by-Disassembly reasoning together with grasp reachability analysis and press-stability verification. RGB-D observations are processed using GroundingDINO, Segment Anything (SAM), and FoundationPose to perform open-vocabulary object detection, instance segmentation, and CAD-based 6D pose estimation [20, 27, 45]. The estimated object poses are subsequently used by a collision-aware execution module integrated with ROS 2, MoveIt 2, and OMPL-based motion planning [6, 7, 38] to perform grasping, alignment, insertion, and press-in assembly operations.

The main contributions of this work are summarized as follows:

- We propose a structured robotic assembly framework for LEGO Duplo assembly tasks that integrates Assembly-by-Disassembly planning, open-vocabulary perception, and collision-aware robotic execution within a unified pipeline. The proposed framework further combines voxel-based grasp reachability analysis, press-stability verification, and collision-aware motion planning to improve the physical feasibility and execution robustness of long-horizon robotic assembly tasks.
- We conduct a systematic evaluation in a MuJoCo-based simulation benchmark constructed using LIBERO tools and conventions, and compare the proposed system against a VLM/VLA-based baseline. Experimental results show that explicit geometric reasoning

and structured assembly planning improve execution robustness, insertion stability, and manipulation efficiency in contact-rich robotic assembly scenarios.

- We validate the proposed pipeline on a physical UR5 setup with an Intel RealSense camera and a Robotiq 2F-style gripper, and analyze real-world transfer effects including safety-limited robot motion, friction-aware gripper control, color-based perception filtering, and practical pose-estimation or grasp-force failure modes. The simulation and real-robot experiments use different manipulators because the simulation follows the LIBERO/MuJoCo Franka Panda benchmark configuration, while the physical validation uses the available UR5 hardware platform to test whether the pipeline transfers to a real setup.

2 Background

Robotic assembly can be viewed as a structured process that bridges high-level task planning and low-level physical execution. To understand how the proposed system operates, this section reviews the foundational concepts involving assembly sequence planning, perception, and grasp feasibility, followed by the computational tools used in this work.

2.1 Assembly-Sequence Planning and Symbolic Reasoning

Assembly Sequence Planning (ASP) is one of the central problems in robotic assembly and intelligent manufacturing [36, 46], since the order in which parts are assembled directly affects both feasibility and execution efficiency. In robotic assembly tasks, the assembly sequence not only determines precedence relationships between parts, but also influences whether subsequent operations can be successfully executed under kinematic reachability, collision avoidance, and grasp/placement constraints. Many assembly tasks exhibit strong structural and geometric dependencies, where certain components must first provide supporting structures before later parts can be correctly inserted or stably placed [19, 34]. Therefore, an effective assembly plan must ensure not only logical consistency, but also physical realizability at every intermediate assembly state.

Traditional assembly planning methods commonly employ symbolic representations to describe part states and assembly relationships [32, 36]. In symbolic planning, the environment is abstracted into discrete symbolic states and symbolic actions that transition between states. Each action is typically associated with preconditions and effects, describing when the action can be executed and how the world state changes after execution. For example, in the classical BlocksWorld formulation [8], the assembly process is represented as transitions between discrete block configurations, while grasping and placement operations modify the symbolic structure state. Such approaches are effective for modeling task dependencies and logical ordering, and have therefore been widely adopted in task planning and robotic reasoning research.

However, purely symbolic planning methods often simplify physical constraints present in real robotic systems [19, 40]. An assembly action that appears valid at the symbolic level may still fail during execution due to collisions, insufficient grasping clearance, restricted insertion directions, or structural instability [34, 46]. Consequently, assembly sequences generated purely through symbolic reasoning are not always executable in practical robotic environments. To improve physical feasibility, modern robotic assembly systems typically incorporate geometric

feasibility, kinematic constraints, and structural stability analysis directly into the planning process [40].

One widely used assembly planning strategy is Assembly-by-Disassembly (ABD), which derives an assembly order by first reasoning about how a completed structure can be disassembled and subsequently reversing the resulting disassembly sequence to obtain an executable assembly sequence [29, 46]. Compared with forward assembly search from the initial state, ABD naturally reduces the search space because, at any given stage, only parts that are currently removable are considered. Parts blocked by neighboring structures, unreachable for grasping, or critical for maintaining structural stability are automatically excluded from the candidate set.

A key advantage of ABD is that removability implicitly satisfies many geometric and structural feasibility constraints [29, 46]. If a part can be removed from the current structure without collision, the reversed operation usually corresponds to a feasible insertion path during assembly. As a result, reversing a valid disassembly sequence often produces an assembly order that respects precedence relations while avoiding obviously infeasible transitions.

Beyond symbolic ordering, assembly planning must additionally consider physical feasibility constraints during execution. Even if a sequence is logically valid, execution may still fail when a part cannot be grasped, oriented, or inserted as required. Geometric constraints include collision avoidance, sufficient gripper clearance, and the existence of feasible insertion or approach directions. Stability constraints ensure that intermediate structures remain physically supported throughout the assembly process.

A typical example occurs in LEGO-style assembly, where parts are connected through downward press-in motions to engage studs and complete the connection. A brick may appear valid in the symbolic assembly order, yet the required pressing operation can destabilize or topple the partially assembled structure. Therefore, when evaluating whether a part can be selected as the next disassembly candidate, the planner must additionally check press stability during the reversed assembly process. Consequently, ABD provides a natural framework for integrating geometric feasibility, grasp reachability, insertion constraints, and structural stability into assembly planning, and has therefore been widely adopted in robotic assembly research [2, 26, 40].

2.2 Perception and Part Identification

Computer vision is a fundamental component of automated robotic assembly, since robots are required to detect, localize, and reason about parts under varying degrees of uncertainty. Unlike traditional industrial assembly settings, where object positions and categories are fixed and known in advance, assembly tasks in open or semi-structured environments are characterized by unknown initial configurations, cluttered scenes, and partial occlusions. Consequently,

robotic systems must continuously perceive and interpret the workspace state to provide reliable environmental information for downstream grasping, placement, and motion planning.

In robotic assembly tasks, the perception module typically consists of three major components: open-vocabulary perception, instance segmentation, and 6D pose estimation. Together, these components determine how effectively the robot can understand the scene and directly influence the performance of downstream planning and execution modules. Figure 2.1 illustrates the difference between task-specific LEGO part perception and more general open-vocabulary scene grounding.

(a) LEGO part perception (b) Open-vocabulary scene perception

Figure 2.1: Examples of open-vocabulary perception in cluttered environments. The right image is adapted from [35].

2.2.1 Open-Vocabulary Perception

Open-vocabulary perception aims to detect and identify objects using semantic descriptions instead of relying on a fixed set of predefined categories [27, 33]. Traditional object detection systems are usually trained on closed-category datasets, where the model can only recognize object classes that appear during training. This assumption is often too restrictive for robotic assembly tasks, where target parts may be specified by task-level descriptions, such as color, shape, or part type, and where new object categories may appear in different tasks.

Recent vision-language detection models address the closed-category limitation by aligning visual regions with natural language descriptions [27, 33]. Instead of predicting only fixed class labels, these models take text prompts as input and localize image regions that correspond to the given semantic query. This makes it possible to search for objects such as “red 2x4 brick” or “blue cube” without training a dedicated detector for every possible part category.

GroundingDINO extends transformer-based object detection with language-guided grounding [27]. It combines image features and text features through cross-modal fusion, allowing the detector to associate words in a prompt with relevant image regions. The output of the model is a set of bounding boxes together with confidence scores for the queried objects. In

robotic assembly, such text-conditioned detection is useful because part specifications can be derived directly from task descriptions or structured product files.

In this work, open-vocabulary perception provides the first stage of the perception pipeline. It identifies candidate object regions based on semantic part descriptions, which are then refined by instance segmentation and used for downstream 6D pose estimation. This allows the perception system to remain flexible with respect to new part descriptions while still providing spatial information required for grasping and assembly execution.

2.2.2 Instance Segmentation

While object detection provides coarse object localization, robotic assembly tasks often require more precise geometric information for grasp synthesis, contact reasoning, placement verification, and downstream pose estimation. Instance segmentation addresses this need by providing pixel-level object masks and object boundaries [12, 20].

In robotic assembly, such masks are useful because assembly operations frequently involve close object interactions, partial occlusions, and constrained manipulation spaces. As the assembly structure gradually grows, visible object regions may become increasingly limited, making coarse bounding-box representations insufficient for reliable manipulation. Accurate segmentation masks help estimate object contours, contact regions, and occlusion boundaries for grasp generation, collision checking, and insertion planning [12, 20].

Recently, general-purpose segmentation frameworks such as the Segment Anything Model (SAM) [20] have demonstrated strong segmentation performance across diverse visual tasks. SAM can generate high-quality pixel-level segmentation masks from prompts such as points, bounding boxes, or coarse masks. In robotic perception pipelines, SAM is commonly integrated with open-vocabulary object detectors to form Grounded-SAM-style perception systems [35]. In such pipelines, language-guided detectors such as GroundingDINO [27] first localize target objects using semantic prompts, and the predicted bounding boxes are subsequently used as prompts for SAM to generate more accurate instance masks.

Compared with object detection methods based only on bounding boxes, instance segmentation provides richer geometric information for robotic assembly. The resulting masks can be further used for point cloud extraction, contact analysis, pose estimation refinement, and collision-aware manipulation planning [20, 35].

2.2.3 6D Pose Estimation

To enable robots to grasp and manipulate objects under varying spatial configurations, robotic assembly systems require accurate 6D object pose estimation, including three-dimensional translation and rotation information [43, 48]. In robotic manipulation and assembly tasks, 6D

pose estimation provides essential spatial information for grasp generation, object alignment, placement computation, collision checking, and motion planning.

Traditional pose estimation approaches often estimate object poses using geometric reconstruction from depth observations and point cloud analysis [14]. By combining depth images with instance segmentation masks, robotic systems can reconstruct partial object geometry and estimate object orientations from geometric features. However, such geometry-based methods are often sensitive to partial occlusions, sensor noise, sparse observations, and object symmetries, which frequently occur in robotic assembly environments. In addition, many traditional geometric registration and correspondence-based methods [14] require iterative optimization or dense point cloud matching, leading to relatively high computational cost and slow inference speed in complex scenes.

To improve robustness, recent robotic manipulation systems increasingly employ learning-based and CAD-based 6D pose estimation methods that explicitly incorporate visual features, depth observations, or known object geometry during pose inference [43, 45, 48]. Such approaches are particularly suitable for robotic assembly tasks because assembly parts usually have predefined geometric models and well-defined structural shapes. By matching RGB-D observations against known CAD models, the system can estimate more accurate object poses even in partially occluded scenes.

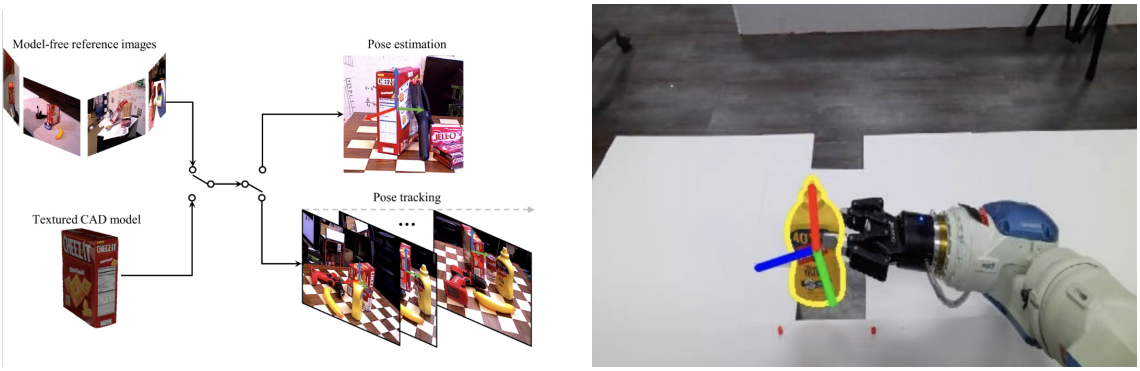
FoundationPose is a representative CAD-based pose estimation framework that estimates object poses from RGB-D observations together with object CAD models [45]. The framework supports both 6D pose estimation and pose tracking, while also allowing model-based and model-free object representations. As illustrated in Figure 2.2a, FoundationPose receives either textured CAD models or a small number of reference RGB images as object representations and estimates the 6D pose of the target object from the current RGB-D observation.

The relevance of FoundationPose for this thesis is practical rather than architectural: LEGO Duplo parts have known CAD geometry, and the robot requires metric object poses rather than only image-space detections. CAD-based pose estimation therefore provides the transformation needed for grasp planning, placement computation, and collision-aware execution.

Figure 2.2b illustrates the resulting pose output, where the predicted object coordinate axes are aligned with the detected object instance in the scene.

2.3 Execution and State-Machine Control

In robotic assembly systems, symbolic assembly plans must eventually be transformed into executable robot motions and physical interactions [6, 7, 38]. Unlike high-level task planning, execution involves continuous motion generation, collision avoidance, grasp control, insertion alignment, and interaction with partially assembled structures. Therefore, robotic assembly



(a) FoundationPose framework supporting both model-based and model-free pose estimation.

(b) Example of 6D pose estimation with predicted object coordinate axes.

Figure 2.2: Overview of FoundationPose and example 6D pose estimation results. Left: unified model-based and model-free pose estimation framework. Right: example predicted object pose with aligned coordinate axes. Adapted from [45].

execution is commonly modeled as a sequential manipulation process composed of multiple execution stages [37].

Finite State Machines (FSMs) are used in robotic manipulation and assembly because they provide a structured way to organize sequential robot behaviors and coordinate transitions between different execution stages [37]. In assembly tasks, manipulation actions such as grasping, transport, alignment, insertion, and release often require different motion constraints and control strategies. State-machine-based execution therefore allows robotic systems to decompose complex assembly procedures into smaller execution units while maintaining explicit transition logic between different manipulation stages.

In this thesis, each assembly operation is divided into phases such as approach, grasping, lifting, transport, alignment, insertion, and release. Different stages may require different motion constraints and control strategies. For example, free-space transport is performed using collision-aware trajectory planning, while insertion operations require constrained Cartesian motions and more precise geometric alignment. As a result, the execution pipeline is organized as a structured sequence of manipulation states rather than a single atomic action.

In a state-machine framework, each manipulation stage is represented as an individual execution state together with explicitly defined transition conditions between states. The robot switches from one state to another according to the current execution result and workspace condition. For example, after reaching a pre-grasp pose, the system transitions to the grasping state; after successful grasping, the robot enters the transport state; and after reaching the placement location, the execution switches to insertion and release operations.

State-machine execution provides several advantages for this assembly pipeline. First, it decomposes complex manipulation tasks into smaller execution units, making the overall

system easier to organize and debug. Second, different control strategies can be assigned to different states. For example, free-space transport motions are executed using collision-aware trajectory planning, while insertion operations use constrained Cartesian motions [16] and more accurate geometric alignment. Third, explicit state transitions support execution monitoring and failure recovery [37].

Failure handling is particularly important in robotic assembly because errors frequently occur during grasping, alignment, or insertion [19]. Grasping may fail due to inaccurate pose estimation or insufficient clearance, while insertion may fail because of geometric misalignment or unexpected contact interactions. In a state-machine framework, the system can locally retry failed actions, switch to alternative grasp configurations, or trigger replanning procedures without restarting the entire task.

2.4 ROS and MoveIt

Modern robotic assembly systems typically require the integration of perception, planning, motion generation, and low-level robot control within a unified software framework. Robot Operating System 2 (ROS 2) has become one of the most widely adopted middleware frameworks in robotics because it provides standardized communication interfaces and modular software composition for distributed robotic systems [28]. In ROS 2, different functional modules such as perception, task planning, motion planning, and execution control can operate as independent nodes while exchanging data through topics, services, and actions. This modular architecture enables robotic assembly systems to flexibly integrate heterogeneous sensing, planning, and control components.

MoveIt is a widely used robotic motion planning framework built on top of ROS [6, 7]. It provides motion planning, kinematics computation, collision checking, and trajectory execution functionalities for robotic manipulation tasks. Given a target robot pose or joint configuration, MoveIt generates collision-free trajectories while considering robot kinematics, joint limits, and workspace constraints. To achieve this, MoveIt maintains a continuously updated planning scene that contains robot states, workspace geometry, and collision objects, allowing the planner to evaluate motion feasibility in dynamically changing environments.

Motion planning in MoveIt is commonly performed using the Open Motion Planning Library (OMPL) [38], which provides sampling-based motion planning algorithms for high-dimensional robotic systems. The resulting trajectories can subsequently be time-parameterized and executed by robot controllers. In robotic assembly tasks, collision-aware motion planning is particularly important because the robot must avoid collisions not only with the workspace and surrounding objects, but also with partially assembled structures that gradually increase the complexity of the environment during execution.

In this thesis, ROS 2 and MoveIt provide the software infrastructure for connecting perception outputs, planned assembly targets, collision-aware motion planning, and robot execution in both simulation and the real-robot setup.

3 Related Work

This chapter reviews prior work related to the main components of the proposed system: assembly-sequence planning, physics-aware feasibility evaluation, perception for robotic assembly, low-level execution, and integrated robotic assembly pipelines. The emphasis is on methods that connect high-level assembly reasoning with geometric or physical constraints, since this connection is central to reliable LEGO Duplo assembly.

3.1 Assembly-Sequence Planning

Prior work on assembly-sequence planning can be broadly grouped into symbolic and geometric planning methods, physics-aware planning methods, and learning-based approaches. This section follows that progression and highlights how each line of work handles the gap between a valid assembly order and a physically executable robot action.

Assembly-sequence planning has long been studied as a central problem in robotic assembly because the assembly order directly affects both execution feasibility and computational complexity. Early approaches formulated the problem using symbolic assembly relations, blocking graphs, and geometric precedence constraints to determine feasible assembly paths [11, 29, 36, 46]. In these methods, assembly feasibility is represented through directional or non-directional blocking relationships between parts. A component can only be assembled or removed if a collision-free motion exists along a valid direction or through a feasible path.

Homem de Mello and Sanderson [29, 30] proposed one of the earliest Assembly-by-Disassembly (ABD) formulations, where feasible assembly orders are obtained by reversing valid disassembly sequences. Their work demonstrated that disassembly reasoning can naturally encode geometric precedence constraints while reducing the branching factor compared with forward assembly search. Halperin et al. [11] introduced non-directional blocking graphs to analyze geometric interference relationships between parts and reduce the assembly planning problem into graph reasoning over feasible disassembly motions.

However, these early approaches were typically restricted to relatively simple geometries and idealized assembly conditions because the search complexity increases rapidly as the number of assembly components grows [11, 46]. Most methods primarily focused on symbolic or purely geometric feasibility while assuming simplified contact interactions, fixed insertion directions, or gravity-free settings.

To handle more complex assemblies, later work incorporated geometric collision checking, continuous motion planning, and physics-based reasoning into the planning process [23, 39, 41, 50]. Instead of considering only symbolic ordering constraints, these methods additionally evaluate collision-free assembly paths, insertion directions, object interactions, and robot kinematic constraints during sequence generation.

Sundaram and colleagues [39] integrated geometric motion constraints directly into assembly planning by evaluating feasible translational motions in configuration space. Subsequent approaches further introduced sampling-based motion planning and constraint-based search strategies to generate executable assembly trajectories under narrow-clearance conditions [50]. Rather than treating assembly as a purely symbolic search problem, these methods explicitly reason about continuous spatial feasibility during each assembly transition.

Physics-based assembly planning has also attracted increasing attention in recent years. Tian et al. [41] combined physical simulation with assembly sequence generation to evaluate contact-rich interactions and structural feasibility during intermediate assembly states. Such methods attempt to capture effects including gravity, friction, support stability, and dynamic contact forces that are difficult to model using purely symbolic or geometric reasoning alone.

In real robotic assembly tasks, gravitational stability and structural support relations must be considered throughout the planning process [19, 34, 41]. A sequence that is symbolically valid may still fail during execution if intermediate structures become unstable, unsupported, or physically unreachable. However, accurately evaluating the physical feasibility of complex assemblies remains computationally expensive because each candidate assembly transition may require repeated collision checking, motion planning, contact analysis, and stability verification.

To improve planning efficiency, several approaches employ hierarchical feasibility evaluation strategies, where simpler static constraints are evaluated before more expensive motion-planning or physics-based constraints are applied [19, 23, 41]. This design is also relevant to the proposed system: inexpensive voxel and grasp-clearance checks are applied during search, while full robot motion planning is deferred to the execution stage.

3.1.1 Physics-Based Simulation for Assembly

Beyond purely geometric analysis, physics-based simulation has increasingly been adopted to evaluate the physical feasibility of assembly and disassembly sequences. Unlike symbolic or purely geometric planners, physics-based methods explicitly model contact interactions, gravity, friction, and structural stability during manipulation. This allows robotic systems to reason not only about whether a motion is geometrically feasible, but also whether the resulting intermediate assembly states remain physically stable throughout execution.

Early work incorporated physics engines into disassembly planning primarily for stability analysis. Aleotti and Caselli [1] employed physics-based animation and rigid-body simulation to evaluate the stability of intermediate disassembly states and guide sequence generation. Similarly, Rakshit and Akella [34] analyzed how motion paths and assembly sequences affect the stability of assembled structures under gravitational constraints. However, these studies mainly focused on relatively simple block-stacking scenarios and did not consider large-scale multi-part robotic assembly under realistic manipulation constraints.

Later work explored the integration of physics simulation with search-based planning to improve computational efficiency and physical realism. Kim and Likhachev [19] proposed simulation-aided physical reasoning for assembly planning under uncertainty, where physically plausible state graphs were constructed to accelerate feasibility search. Their framework was evaluated on insertion-oriented tasks such as box-on-table placement and peg-in-hole assembly. Rather than exhaustively validating every candidate action using full simulation, the planner selectively applied physics reasoning only to promising transitions, reducing computational cost.

Physics simulation has also become an important component in reinforcement-learning-based robotic assembly systems. Several studies combined simulation environments with reinforcement learning to learn insertion and manipulation policies for contact-rich tasks [47, 49]. RoboAssembly [49], for example, introduced a multi-robot contact-rich simulation environment for learning furniture assembly policies. Other approaches learned force-guided insertion skills and demonstrated assembly behaviors directly in simulation before transferring them to real robotic platforms. However, many of these methods focus primarily on low-level manipulation primitives such as peg-in-hole insertion, lap-joint assembly, or local alignment correction rather than long-horizon multi-part assembly planning.

More recently, physics-aware assembly systems have used richer geometric and simulation representations to model contact-rich interactions during assembly [41, 42]. Compared with coarse convex-hull approximations, more detailed geometry and contact models can better capture frictional contact, interlocking shapes, and continuous collision response during narrow-clearance insertion. Nevertheless, despite progress in physically realistic simulation, many existing approaches still focus on isolated insertion skills or simplified manipulation settings rather than generating complete assembly sequences that are simultaneously collision-free, structurally stable, and executable on real robotic systems [41, 49].

3.1.2 Learning-Based Assembly Planning

Learning-based assembly planning aims to reduce the combinatorial search cost by learning assembly decisions from data [9, 21, 44]. Instead of enumerating all possible part orders, these

methods train models to predict promising assembly or disassembly actions from the current assembly state.

Early methods used self-supervised data collection, where valid and invalid assembly transitions were obtained through repeated trial-and-error interactions. These data-driven strategies can reduce manual rule design, but they were mostly applied to simplified shape-matching or kit-assembly tasks, where geometry and contact conditions are relatively simple [44].

Graph-based methods are especially suitable for assembly planning because an assembly can be represented as a relational graph. Parts are modeled as nodes, while edges encode contact relations, support dependencies, blocking relations, or geometric constraints. A graph neural network can then learn structural dependencies between parts and predict which component should be assembled or disassembled next. Watanabe and Inada [44] used previously collected assembly sequence data to guide sequence search, reducing the need for exhaustive enumeration. Kitz and Thomas [21] proposed neural dynamic assembly sequence planning, where the model predicts assembly decisions from changing intermediate assembly states rather than from a fixed final structure alone.

Reinforcement learning formulates assembly planning as a sequential decision-making problem [9, 47]. The state usually encodes the current partial assembly, the available parts, and sometimes geometric or contact information. The action corresponds to selecting the next part, pose, or assembly operation. The reward is designed to encourage successful assembly progress and penalize invalid transitions, collisions, or unstable states. Funk et al. [9] combined graph-based reinforcement learning with mixed-integer programming for 3D robot assembly discovery, using graph representations to reason about part relations while optimization constraints help enforce feasible assembly decisions.

Although learning-based methods can improve inference efficiency and capture structural patterns from data, they often require large training datasets or extensive simulation [13, 49]. They may also have difficulty strictly enforcing hard geometric constraints, such as collision-free insertion paths, gripper clearance, and intermediate structural stability [9, 40]. For this reason, learning-based planners are often combined with explicit feasibility checks when reliable robotic execution is required.

3.2 Perception for Robotic Assembly

Perception is a fundamental component of robotic assembly because robots must reliably identify, localize, and track assembly parts under cluttered and partially occluded conditions. Compared with traditional pick-and-place tasks, robotic assembly often requires significantly higher geometric precision since small localization errors can directly lead to grasp failures, insertion collisions, or unstable assemblies.

3.2.1 Classical Vision-Based Assembly Perception

Early robotic assembly systems mainly relied on handcrafted vision pipelines based on color segmentation, edge extraction, template matching, or geometric feature detection. Such approaches were widely used in structured industrial environments where object appearance, lighting, and camera viewpoints were tightly controlled. Classical feature-based methods further employed SIFT, SURF, or contour matching together with RGB-D sensing and point-cloud registration to estimate object poses [14]. Although these methods can provide accurate geometric alignment in constrained settings, they are highly sensitive to occlusions, illumination changes, reflective surfaces, and sensor noise.

3.2.2 Deep Learning-Based Perception

With the development of deep learning, convolutional neural network-based detectors and segmentation models such as Mask R-CNN became widely adopted for robotic manipulation and assembly perception [12]. These methods significantly improved robustness under cluttered tabletop environments by learning semantic object representations directly from large-scale datasets. In robotic assembly, object detection is commonly combined with instance segmentation to obtain more accurate object boundaries for grasp synthesis and collision reasoning [2, 35]. For example, segmentation-based systems can segment assembly parts before performing pose estimation and grasp planning in industrial manipulation tasks.

More recent work has adopted real-time segmentation-based detectors such as YOLOv8-seg for robotic assembly environments. Barghi et al. [2] employed a YOLOv8m-seg model for LEGO brick localization and segmentation in collision-aware LEGO reconstruction tasks. Their system fine-tuned the detector on a custom dataset containing LEGO bricks with multiple orientations and viewpoints to improve robustness under cluttered assembly scenes. However, they observed that coarse bounding-box localization alone is insufficient for reliable grasp generation because grasp accuracy strongly depends on stud alignment and brick orientation estimation.

To improve localization precision, several works combine deep visual detection with geometric reasoning [2]. In the system of Barghi et al. [2], OpenCV-based circle detection was applied to detect LEGO studs after instance segmentation. The detected stud coordinates were subsequently used to estimate brick orientation and grasp positions. This type of geometry-aware perception is relevant to precision assembly tasks because accurate contact alignment is required during insertion and placement.

3.2.3 6D Pose Estimation

Beyond 2D localization, robotic assembly systems usually require accurate 6D object pose estimation. Classical model-based pose estimation methods commonly rely on geometric

matching between RGB-D observations and CAD models [14]. However, these methods often degrade under partial occlusions or sparse depth observations. To address these limitations, learning-based pose estimation frameworks such as PoseCNN [48], DenseFusion [43], and FoundationPose [45] have been proposed. DenseFusion combines RGB features and depth point-cloud features through iterative fusion to improve robustness in cluttered scenes, while FoundationPose directly leverages CAD geometry together with RGB-D observations for robust pose estimation and tracking during manipulation.

Recent advances in vision-language models have further improved perception flexibility in robotic assembly. Open-vocabulary detectors such as GroundingDINO [27] can localize objects directly from natural-language prompts rather than fixed object categories. Combined with SAM [20], these systems enable Grounded-SAM-style perception pipelines that jointly perform semantic grounding and pixel-level segmentation. Such methods are particularly useful for robotic assembly because target parts are often specified through semantic task descriptions instead of predefined detector labels.

However, robotic assembly perception remains challenging because assembly tasks frequently involve severe occlusions, contact-rich interactions, reflective materials, dense object arrangements, and strict geometric tolerances [2, 14]. Even small segmentation or pose-estimation errors may propagate into downstream planning and execution failures. Consequently, modern robotic assembly systems increasingly combine deep visual perception, geometric reasoning, and physical feasibility constraints to improve robustness in long-horizon assembly tasks [2, 3, 40].

3.3 Low-Level Execution and Task Control

Low-level execution converts assembly plans into physically executable robot motions under geometric, kinematic, and contact constraints [6, 7, 38]. In robotic assembly tasks, execution is often organized into sequential phases such as approach, grasp, lift, transport, alignment, insertion, press-in, release, and retreat. Different phases require different control strategies. Free-space transport mainly depends on collision-free trajectory generation, while insertion and alignment require accurate Cartesian motion, contact handling, and small pose-error correction.

3.3.1 Classical Robotic Assembly Execution

One common execution strategy is to use structured state-machine control [37]. In these systems, the assembly task is decomposed into explicit execution states such as PREGRASP, GRASP, LIFT, PREPLACE, INSERT, PRESS, and RETREAT. Each state defines its own controller, motion constraints, transition conditions, and recovery rules. For example, transport phases are commonly executed using joint-space trajectory planning, while insertion phases use

constrained Cartesian motions along predefined insertion directions. State-machine execution also supports local failure recovery, such as retrying grasps, switching to alternative grasp candidates, or replanning failed insertion actions.

Collision-aware motion planning is another important component of low-level assembly execution. Sampling-based motion planners such as OMPL and MoveIt are widely used to generate robot trajectories under kinematic limits, joint constraints, and planning-scene collision constraints [6, 7, 38]. During assembly, the planning scene must continuously update the workspace geometry, loose parts, and partially assembled structures. As the assembly grows, feasible motion space becomes smaller and narrow-clearance manipulation becomes increasingly difficult.

3.3.2 Visual Servoing and Contact Control

Precise insertion additionally requires closed-loop feedback control. Visual servoing methods are commonly divided into image-based visual servoing (IBVS) and position-based visual servoing (PBVS) [16]. IBVS minimizes errors directly in image feature space, while PBVS estimates the relative target pose and minimizes Cartesian pose errors. Force or impedance control is also often used during insertion to regulate contact forces and avoid excessive contact or jamming.

Besides classical motion-planning pipelines, recent work increasingly explores learning-based low-level execution [47, 49]. Reinforcement learning methods learn insertion, alignment, and contact-rich manipulation policies directly from interaction data. In these approaches, policies map observations and robot states to continuous actions, while reward functions are designed around insertion success, contact stability, pose error, or force minimization. Such methods can learn compliant behaviors that are difficult to model analytically, but they usually require large amounts of simulation data and often generalize poorly under unseen object geometries or friction conditions.

3.3.3 Vision-Language-Action Models

More recently, Vision-Language-Action (VLA) models have been proposed for end-to-end robotic manipulation [4, 5, 18, 51]. Unlike classical robotic systems that explicitly separate perception, planning, and control, VLA systems directly predict robot actions from multimodal observations including RGB images, language instructions, and robot states. Most VLA architectures are transformer-based and learn visuomotor policies from large-scale robot demonstration datasets.

RT-1 [5] introduced a transformer-based robot policy trained on large-scale real-world robot data by tokenizing observations and actions into a sequence prediction problem. RT-2 [51] further incorporated vision-language representations from internet-scale pretrained models,

enabling semantic reasoning from natural-language instructions. More recent systems such as OpenVLA [18] and π-style visuomotor policies [4] extend this idea using autoregressive or diffusion-based action generation. These models directly generate continuous robot actions such as end-effector motions or gripper commands without explicitly computing symbolic assembly sequences, grasp poses, or insertion trajectories.

A key characteristic of π-style policies is that manipulation strategies are learned implicitly from demonstration trajectories rather than manually engineered through symbolic planning rules [4]. The model attempts to infer grasping, alignment, and insertion behaviors directly from multimodal training data. Diffusion-based action decoders are particularly useful because they can generate smooth continuous trajectories and represent multimodal action distributions during contact-rich manipulation.

Although VLA systems demonstrate strong generalization ability for open-world manipulation tasks [18, 51], robotic assembly remains challenging for purely end-to-end visuomotor execution. Assembly tasks involve long-horizon sequential dependencies, narrow-clearance insertion, contact-sensitive interactions, and strict geometric tolerances [13, 40]. Small pose errors during insertion can accumulate across multiple steps and lead to failure. In addition, many VLA systems do not explicitly enforce collision checking, insertion feasibility, or structural stability during inference, which may produce physically invalid actions even when the generated behavior appears semantically reasonable.

For this reason, recent robotic assembly systems increasingly combine learning-based execution with explicit geometric planning and physical feasibility verification [2, 3, 40]. In such hybrid systems, learned policies provide semantic priors or coarse manipulation guidance, while collision-aware motion planning, grasp feasibility analysis, and state-machine execution ensure reliable physical assembly behavior.

3.4 Integrated Robotic Assembly Pipelines

Recent robotic assembly research has increasingly focused on hybrid frameworks that integrate high-level planning with low-level feasibility verification [2, 3, 40]. Instead of relying only on a symbolic task planner or an end-to-end policy, these systems typically combine discrete task reasoning, geometric motion planning, physical simulation, and execution monitoring within a unified pipeline. This design is especially relevant for assembly tasks, where a high-level plan may be logically valid but still fail during execution because of collisions, unreachable grasps, unstable intermediate structures, or insertion errors.

3.4.1 LLM-Based Task Planning

Some recent approaches use Large Language Models (LLMs) to generate high-level task plans from natural language instructions or scene descriptions [15, 17]. In such systems, the LLM proposes an ordered sequence of actions, while feasibility is checked later through affordance models, motion planning, collision checking, or physics simulation. This approach improves task flexibility and semantic reasoning, but the generated plans still require explicit grounding in robot kinematics, object geometry, and physical constraints.

3.4.2 Vision-Language Reasoning for Assembly

Vision-Language Models (VLMs) extend this idea by incorporating visual feedback into the planning process [18, 51]. These systems can use images or RGB-D observations to identify objects, reason about spatial relations, and guide plan refinement. In assembly tasks, VLMs can help select target parts, detect execution failures, or provide semantic feedback for replanning. However, VLMs usually operate at a semantic or object-relation level and often do not directly guarantee collision-free motion, grasp feasibility, or contact-stable insertion.

3.4.3 Symbolic-Geometric Hybrid Planning

Another line of work combines symbolic or graph-based planning with motion planning and physics simulation [2, 3, 40]. In these hybrid symbolic-continuous architectures, a planner first generates a discrete state graph or action sequence. Each symbolic action is then grounded into continuous robot motions through motion planning and validated using geometric collision checking or physical simulation. If grounding fails, the system can trigger symbolic replanning or local motion replanning. This structure allows the system to combine the clarity of symbolic reasoning with the physical feasibility checks required for real robotic execution.

State-machine execution is also commonly integrated into such frameworks to manage low-level task progress [2, 3, 37]. After a task plan has been generated and grounded, execution is divided into states such as approach, grasp, transport, align, insert, release, and retreat. Each state can use its own controller, success condition, and failure handling logic. This makes it easier to monitor execution and recover from local failures without restarting the entire assembly task.

Overall, existing hybrid frameworks show that reliable robotic assembly requires coordinated integration of symbolic planning, geometric reasoning, grasp feasibility analysis, motion planning, physical validation, and low-level execution control [2, 3, 40]. Different systems emphasize different modules, but robust assembly performance generally depends on connecting high-level task reasoning with continuous feasibility verification and execution feedback.

4 Methods

This chapter describes how the proposed system converts a target LEGO Duplo structure into executable robotic assembly actions. It first introduces the workspace, task representation, and system-level data flow. It then presents the physical feasibility planner, the open-vocabulary perception pipeline, and the collision-aware execution controller used to realize the planned assembly sequence.

4.1 System Overview and Setup

This section gives a concise overview of the task setting and the data passed between the main system modules. The detailed planning, perception, and execution procedures are described in the following sections.

Our approach constructs a structured robotic assembly pipeline that transforms a high-level product specification into executable assembly actions. We consider a vision-guided tabletop assembly setting with a 6-DoF manipulator, a parallel-jaw gripper, and an RGB-D camera. The tiered benchmark is evaluated in MuJoCo simulation with a Franka Emika Panda model, ROS 2, and MoveIt 2, while the real-world validation is performed on a UR5 robot equipped with a Robotiq 2F-style parallel gripper and an RGB-D camera.

The robotic assembly workspace used in this work is organized into two functional zones: one for initially separated LEGO Duplo bricks and one for target structure construction. Both zones are mounted on LEGO-compatible baseplates to provide consistent geometric alignment during manipulation and insertion. An RGB-D camera observes the workspace and provides color and depth observations for perception and pose estimation. The manipulator is positioned between the two zones, allowing the robot to transport bricks from the separated-brick region to the construction region while maintaining collision-free motion within the reachable workspace. Figure 4.1 shows the real UR5-based workspace used for the final validation experiments.

The assembly goal is specified as a structured YAML product description. Each brick entry contains the brick identifier, brick type, color, spatial position, and orientation within the final target structure. This representation defines the desired final geometry before planning starts and provides a compact intermediate format shared between planning, perception, and execution modules.

Figure 4.2 illustrates the overall three-stage pipeline. The planner receives the target structure, generates an assembly sequence with grasp and stability constraints, and passes the next

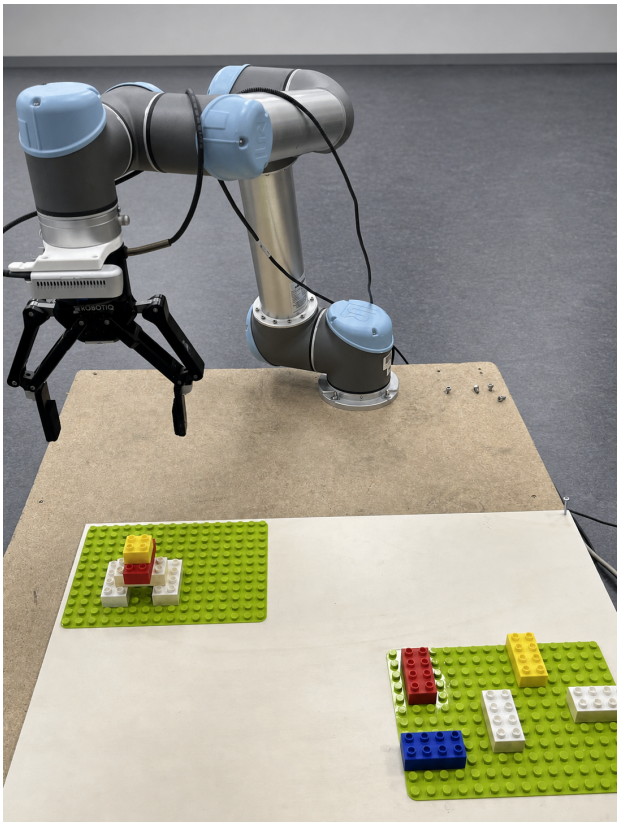


Figure 4.1: Real-robot validation workspace. Loose LEGO Duplo parts are detected in the picking area, transported by the UR5 manipulator with a parallel-jaw gripper, and placed onto the LEGO-compatible assembly area according to the planned sequence.

required part to the perception module. The perception module localizes that physical part from RGB-D observations and estimates its 6D pose. The execution module then converts the planned target placement and perceived object pose into robot motions.

Listing 1 shows an example structured specification in YAML format. Based on this representation, the planning module reconstructs the voxel-aligned target state, infers structural dependencies between bricks, and generates a physically executable assembly sequence through ABD reasoning. The positions are represented in a grid-aligned metric coordinate system consistent with the LEGO stud layout.

4.2 Assembly-Sequence Planning with Physical Feasibility

Assembly-by-Disassembly Planning The planning module receives a target product description from the structured YAML specification and converts it into an executable assembly sequence. The target structure is represented as a discrete voxel state aligned with the LEGO stud grid. Each voxel cell stores both occupancy and brick identity information, so the planner

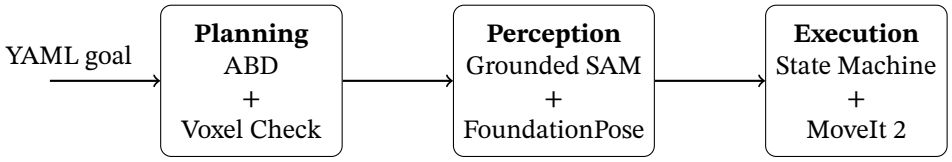


Figure 4.2: System data flow. A structured YAML assembly goal is first converted into an ABD-based assembly sequence. For each planned part, the perception module localizes the corresponding physical brick and estimates its 6D pose. The execution module then uses this pose and the planned target placement to generate collision-aware pick, transport, insertion, and retreat motions.

```yaml
blocks:
  - name: "2x2_brick_1"
    type: "brick_2x2"
    color: "yellow"
    pos: [0.0, 0.148, 0.0]
    rotation: [0, 0, 90]

  - name: "2x2_brick_2"
    type: "brick_2x2"
    color: "green"
    pos: [0.032, 0.148, 0.0]
    rotation: [0, 0, 0]

  - name: "4x2_brick_3"
    type: "brick_4x2"
    color: "yellow"
    pos: [0.016, 0.116, 0.0]
    rotation: [0, 0, 0]

  - name: "2x2_brick_4"
    type: "brick_2x2"
    color: "blue"
    pos: [-0.032, 0.116, 0.0]
    rotation: [0, 0, 0]

  - name: "2x2_brick_5"
    type: "brick_2x2"
    color: "red"
    pos: [0.0, 0.116, 0.0191]
    rotation: [0, 0, 0]

  - name: "2x2_brick_7"
    type: "brick_2x2"
    color: "blue"
    pos: [0.032, 0.148, 0.0191]
    rotation: [0, 0, 0]

  - name: "2x2_brick_9"
    type: "brick_2x2"
    color: "blue"
    pos: [0.0, 0.148, 0.0191]
    rotation: [0, 0, 90]
```

Listing 1: Example YAML task specification for a LEGO Duplo assembly. The planner reads the list of block entries, reconstructs the target structure from their type, position, and orientation fields, and uses the color and identity fields to connect planned parts to perception queries during execution.

Algorithm 1 ABD reasoning with grasp reachability and press-stability checks. Input: current structure state S. Output: a disassembly sequence π, or FAIL if no candidate order satisfies the feasibility checks. The final assembly sequence is obtained by reversing π.

```
function DISASSEMBLE(S)
    if S = ∅ then
        return [ ]
    C ← TOPDOWNCANDIDATES(S)
    for all p ∈ C do
        if REACHABLE(p, G(p), w, S) and PRESSSTABLE(S \ {p}) then
            if ISSUBASSEMBLY(p, S) then
                A ← BUILDSUBASSEMBLY(p, S)
                π_sub ← DISASSEMBLE(A)
                if π_sub = FAIL then
                    continue
                π_rest ← DISASSEMBLE(S \ A)
                if π_rest = FAIL then
                    continue
                return π_sub + π_rest
            else
                π_rest ← DISASSEMBLE(S \ {p})
                if π_rest = FAIL then
                    continue
                return [p] + π_rest
    return FAIL
```

can reason about which space is occupied, which brick owns each occupied cell, and how different bricks are connected inside the target structure. This representation is particularly suitable for LEGO Duplo assembly because the regular stud-grid geometry makes spatial relations, support relations, and local collision checks easier to evaluate than in an unstructured mesh representation.

We formulate assembly-sequence generation as a recursive Assembly-by-Disassembly (ABD) search. Instead of constructing the product forward from an empty assembly area, the planner starts from the completed target structure and reasons about how it can be taken apart. The resulting disassembly sequence is then reversed to obtain the assembly sequence used during execution. This reduces the branching factor compared with forward assembly search, since the planner only considers components that are currently removable from the completed or partially disassembled structure.

Algorithm 1 takes the current structure state S as input and returns a disassembly sequence π, represented as an ordered list of bricks. The final assembly sequence is obtained by reversing π. At each recursive search state S, the planner first calls TOPDOWNCANDIDATES to generate a set of removable candidates C. In this thesis, TOPDOWNCANDIDATES denotes a deterministic candidate-filtering rule that scans the current voxel structure from higher layers to lower

layers and selects bricks whose removal is not immediately blocked by bricks above them or by direct local support dependencies. This candidate generation follows the physical structure of LEGO assembly: parts that are higher in the structure or not blocked by other bricks are considered before lower supporting components. For each candidate brick $p \in C$, the combined REACHABLE and PRESSSTABLE condition evaluates whether removing p is geometrically and physically feasible. A candidate is rejected if it is blocked by neighboring bricks, if no feasible grasp configuration is available, or if the remaining structure would fail the press-stability check used for the reversed assembly action.

The first feasibility condition is grasp reachability. For each candidate brick, a finite set of predefined grasp candidates is generated according to the brick geometry and the parallel-jaw gripper configuration. The planner checks whether at least one grasp candidate provides enough free space for the gripper and admits a collision-free approach motion in the current voxel state. In this way, the planner does not only ask whether a brick is symbolically removable, but also whether the robot can physically reach and grasp it during execution.

The second feasibility condition is press-stability, evaluated by PRESSSTABLE. Although the search is performed as disassembly, the final execution is the reversed assembly process. Therefore, when the planner considers removing a brick p, it also checks whether the remaining partial structure $S \setminus \{p\}$ would be stable when p is later inserted by a downward press-in motion. This is important for LEGO Duplo assembly because stud engagement typically requires a vertical pressing force. If the corresponding press operation would topple or destabilize the partially assembled structure, the candidate is rejected even if it is geometrically removable.

If both grasp reachability and press-stability are satisfied, the candidate p is accepted as the next disassembly action. The subassembly branch handles cases where the accepted candidate should not be treated as an isolated brick. If ISSUBASSEMBLY detects that p is part of a connected local structure, BUILDSUBASSEMBLY constructs this component $A \subseteq S$ by collecting the bricks that are connected through the same local support relation. The sequence π_{sub} denotes the local disassembly order inside this component, while π_{rest} denotes the disassembly order for the remaining structure. If either recursive call fails, the procedure continues evaluating the remaining candidates in C. If no candidate satisfies all checks, the function returns FAIL.

Figure 4.3 illustrates this ABD reasoning process on an example target structure. The numbered bricks indicate the inferred disassembly order π, and the black markers show candidate grasp-point pairs evaluated during planning. Algorithm 1 summarizes the recursive procedure. By filtering infeasible candidates during disassembly, the planner generates an assembly sequence that already accounts for grasp reachability, local collision constraints, and press-stability requirements before execution.

The algorithm first handles the recursion base case: an empty structure has an empty disassembly sequence. It then generates the candidate set C using TOPDOWNCANDIDATES, which

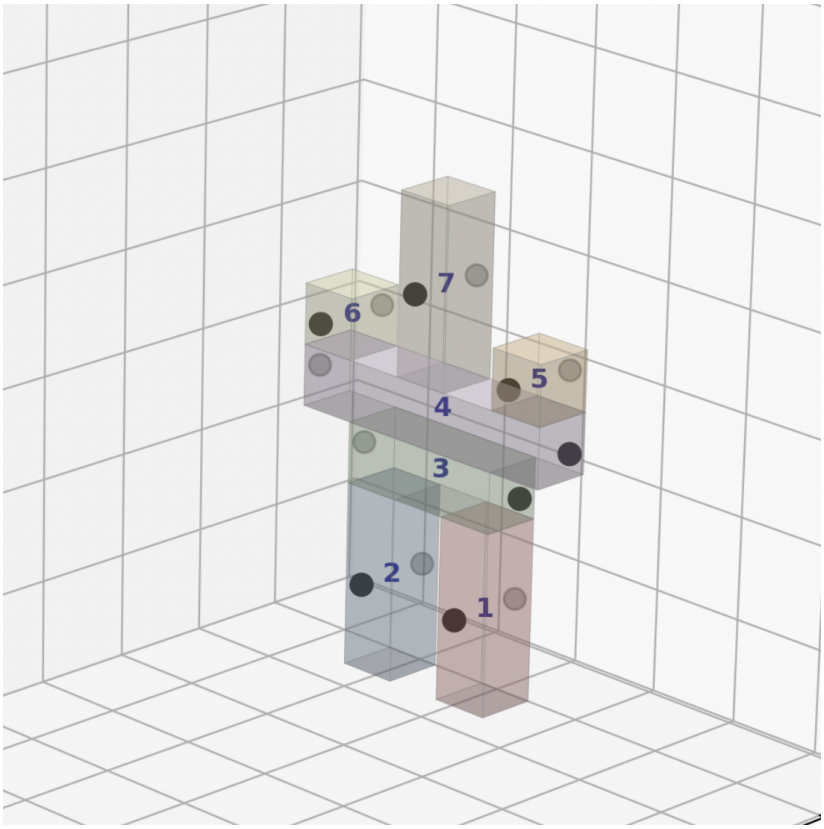


Figure 4.3: Example of ABD reasoning on a target structure. The planner first evaluates removable top-level candidates, checks the marked grasp-point pairs against voxel occupancy, and records the accepted removal order with the numbered labels. Reversing this order gives the assembly sequence executed by the robot.

prioritizes exposed upper-layer bricks before lower support bricks, and tests each candidate against grasp reachability and press-stability. If the candidate belongs to a connected local component, the subassembly branch constructs A, computes π_{sub}, and then computes π_{rest}. Otherwise, the single-brick branch removes only p and recursively plans the remaining structure. The return statements record the order in which parts are removed.

Grasp Reachability Analysis For each candidate block p, the planner evaluates whether the grasp action can be executed under the current workspace configuration. Instead of testing only a single grasp pose, the system defines a finite set of candidate grasp-point pairs $\mathcal{G}(p) = \{g_1, g_2, \dots, g_n\}$, where each g_i corresponds to a pair of opposing contact points compatible with the parallel-jaw gripper. These grasp candidates are generated according to the brick geometry and represent different side-grasp directions around the target brick.

To efficiently approximate manipulation feasibility during planning, each grasp candidate is associated with a gripper occupancy region representing the workspace volume occupied by the gripper fingers and gripper body during grasp execution. Rather than performing expensive

mesh-level collision checking, the occupancy region is discretized directly in the voxel-state representation.

In our implementation, the gripper occupancy region is modeled as a fixed 2×2 voxel area aligned with the LEGO stud grid, corresponding to four occupied grid cells surrounding the target grasp location. Candidate grasps are generated mainly from opposing long-side contact pairs because these grasps match the parallel-jaw gripper geometry and leave the top surface available for vertical press-in operations. Short-side or top grasps are excluded in the current implementation because they either provide less stable contact for elongated Duplo bricks or interfere with the planned insertion direction. This representation provides a conservative approximation of the minimum free space required for collision-free grasp execution.

During evaluation, the planner checks whether the occupancy region overlaps with neighboring bricks, surrounding objects, or partially assembled structures under the current voxel occupancy state. If any overlap exists, the corresponding grasp candidate is rejected because the gripper would collide with the environment during grasp execution. Consequently, a grasp candidate is considered feasible only when sufficient free clearance exists around the target brick.

Figure 4.4 illustrates two examples of grasp feasibility evaluation under voxel-based occupancy constraints. The dashed regions visualize the approximated gripper occupancy area considered during planning. In the left example, the occupancy region remains collision-free, allowing the grasp candidate to be accepted. In the right example, the occupancy region intersects with neighboring bricks, making the grasp physically infeasible under the current scene configuration.

During recursive ABD planning, grasp reachability acts as an execution-aware feasibility constraint. A candidate block p is considered removable only if at least one grasp candidate $g_i \in \mathcal{G}(p)$ satisfies the collision-free occupancy condition. Otherwise, the candidate is rejected and excluded from the recursive disassembly search.

Press-Stability Check In addition to grasp feasibility, the planner evaluates whether the partial assembly remains stable during the reversed insertion process. This check is motivated by the physical characteristics of LEGO-style assembly, where bricks are typically connected through a downward press-in motion to engage studs and complete the attachment. Although a candidate part may be geometrically insertable, the insertion force itself can destabilize the partially assembled structure and cause toppling during execution.

Instead of performing computationally expensive multi-step physics simulation during recursive ABD search, we employ a lightweight quasi-static geometric approximation for stability evaluation. This approximation is particularly suitable for LEGO Duplo assembly because the insertion direction is predominantly vertical and the stud-grid geometry provides well-defined support relationships.

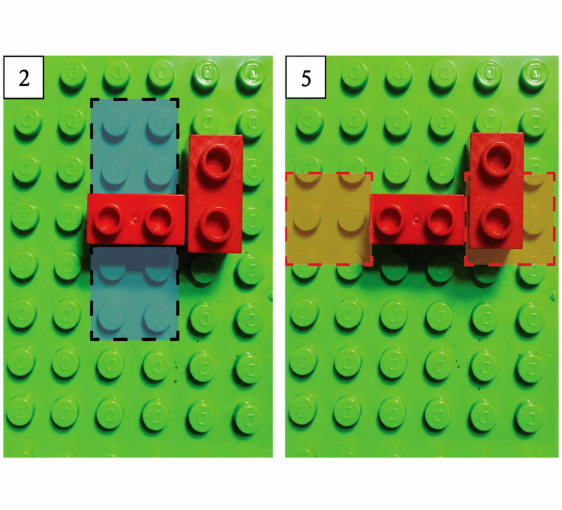


Figure 4.4: Grasp reachability evaluation under voxel-based gripper occupancy constraints. The dashed 2×2 voxel regions approximate the free space required by the gripper. Candidates on the long sides are evaluated because they match the parallel-jaw gripper geometry and keep the top surface available for vertical insertion. Short-side and top grasps are not used in the current planner because they provide less stable contact for elongated Duplo bricks or interfere with the downward press-in direction. Left: feasible candidate with sufficient clearance. Right: infeasible candidate where the gripper occupancy region collides with neighboring bricks.

Let S denote the current partial assembly structure. We define the support region $\mathcal{P}$ as the planar support area generated by all ground-contacting bricks in the current structure. In our implementation, $\mathcal{P}$ is approximated from the projected footprints of all bottom-layer bricks on the supporting plane, followed by a convex-hull construction.

For a candidate insertion operation, the projected press location is denoted as p_{proj}, which represents the planar location where the downward insertion force is applied. A candidate insertion is considered press-stable only if $p_{\text{proj}} \in \mathcal{P}$. This condition ensures that the downward press-in force acts within the support boundary, reducing the risk of generating an overturning moment during insertion. If the projected press location falls outside the support region, the corresponding candidate action is rejected because the insertion may destabilize the partially assembled structure during the press-in operation.

Although simplified, this geometric stability model provides an efficient execution-aware feasibility filter during recursive ABD planning. By incorporating press stability directly into candidate evaluation, the planner avoids assembly sequences that are geometrically valid but physically unstable during the final press-in operation.

Subassembly Reasoning Subassembly reasoning is an internal rule of the ABD planner rather than a separate execution module. Its purpose is to prevent the recursive search from treating every brick as fully independent when several connected bricks should be handled

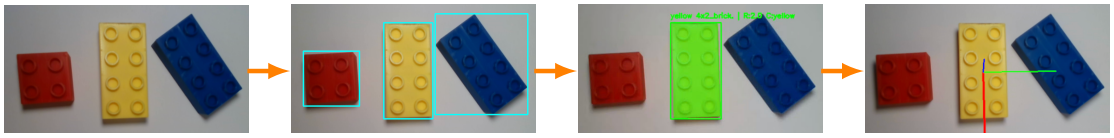
as one local structural component. This situation can occur in hierarchical LEGO structures, where an upper brick depends on a lower support brick and where the local support relation matters for the reversed assembly order.

During recursive ABD search, the planner evaluates candidate bricks from the current removable candidate set. Each candidate p is first tested against the same two feasibility conditions as every other candidate: grasp reachability and press-stability. If the candidate fails either check, it is skipped. If it passes both checks, IsSubassembly tests whether p belongs to a connected local component that should be planned together instead of being removed as a single independent brick.

When such a component is detected, BuildSubassembly constructs the subassembly $A \subseteq S$ from bricks connected through the same local support relation. The ABD procedure is then applied recursively inside A to compute π_{sub}, and separately to the remaining structure $S \setminus A$ to compute π_{rest}. The planner returns the concatenated disassembly order $\pi_{\text{sub}} + \pi_{\text{rest}}$. After reversal, this produces an assembly order in which lower supporting bricks are placed before the dependent upper bricks, preserving the intended support relationship during execution.

This mechanism is used as part of the complete planner in the Tier 3 and Tier 4 structures, where multi-layer dependencies occur. However, the current evaluation does not include a separate ablation study that isolates subassembly reasoning from the other planning checks. Therefore, the reported results should be interpreted as evaluating the full ABD planner with grasp-reachability, press-stability, and subassembly handling together, rather than as independently quantifying the benefit of the subassembly rule alone.

4.3 Perception: Open-Vocabulary Visual Pose Estimation



(a) RGB-D workspace input (b) GroundingDINO box (c) SAM instance mask (d) FoundationPose 6D pose

Figure 4.5: Perception pipeline for active part localisation in a LEGO Duplo workspace. From left to right: RGB-D workspace input, GroundingDINO box, SAM instance mask, and FoundationPose 6D pose.

To support robotic grasping, placement, and insertion during assembly execution, the system must continuously localize target LEGO Duplo parts and estimate their spatial poses from RGB-D observations. In this work, we implement a task-oriented open-vocabulary percep-

tion pipeline that combines GroundingDINO, SAM, and FoundationPose for semantic object localization, geometric refinement, and CAD-based 6D pose estimation.

The overall perception workflow is illustrated in Figure 4.5. The system receives aligned RGB and depth observations from an RGB-D camera. The depth stream is aligned with the RGB image so that each image region corresponds to metric depth measurements. Camera intrinsics are represented using the intrinsic matrix K, while the calibrated camera-to-base transformation is loaded from a YAML configuration file for downstream robot execution.

During execution, the current assembly task is first loaded from a structured YAML task specification containing the target brick identity and placement configuration. Unlike conventional closed-set detectors trained on fixed object categories, our system adopts open-vocabulary detection to support flexible task-level object specification. GroundingDINO is therefore queried using semantic prompts such as “lego block” to localize candidate LEGO parts inside the workspace.

The detector outputs candidate bounding boxes together with confidence scores. Since different LEGO bricks have characteristic geometric proportions, the system further performs a lightweight geometric verification step before pose estimation. Each candidate region is refined using SAM-based instance segmentation, and the resulting mask is used to estimate the object aspect ratio through oriented bounding-box fitting. Candidates with inconsistent geometric proportions are rejected directly, significantly reducing false detections before downstream pose estimation.

In the real-robot setup, the final candidate selection also applies strict color and size filtering before pose estimation. This is useful when the possible LEGO parts are known in advance, and it reduces practical false positives observed during execution, such as specular highlights being detected as white brick regions by the visual model.

After filtering, the remaining candidates are ranked according to both detection confidence and geometric consistency, and the highest-scoring candidate is selected for 6D pose estimation.

The system subsequently performs CAD-based pose estimation using FoundationPose. Since LEGO Duplo geometries are predefined, the corresponding STL mesh is dynamically loaded according to the current task. Before registration, the mesh is automatically normalized and recentered. Large meshes are rescaled from millimetres to metres when necessary, and the mesh centroid is shifted to the origin so that the object coordinate frame is aligned with the geometric center of the brick.

FoundationPose receives the RGB image, depth image, camera intrinsics, segmentation mask, and CAD mesh as input. Surface points are sampled directly from the CAD geometry and used for iterative RGB-D pose registration. During optimization, the framework aligns rendered CAD observations with the segmented RGB-D observation to estimate the full object pose

$T_{\text{cam}}^{\text{obj}} \in \mathbb{R}^{4\times4}$, which represents the rigid transformation from the object frame to the camera frame.

Compared with classical ICP-based registration methods, FoundationPose provides significantly improved robustness under partial occlusions, sparse depth observations, varying viewpoints, and cluttered tabletop environments. An additional advantage of FoundationPose is that it supports both pose estimation and online pose tracking within a unified framework. This is particularly useful for robotic assembly because the robot continuously manipulates and reorients objects during execution.

To improve estimation stability, the system continuously tracks the selected object for several seconds and accumulates multiple pose predictions instead of relying on a single-frame estimate. During tracking, the estimated object coordinate axes are projected back into the RGB image for online visualization and debugging. The final stable pose estimate is subsequently transformed into the robot base frame using the calibrated camera extrinsic matrix: $T_{\text{base}}^{\text{obj}} = T_{\text{base}}^{\text{cam}} T_{\text{cam}}^{\text{obj}}$. The transformed object pose is then used for downstream robotic manipulation. The system extracts the object yaw angle from the estimated rotation matrix and normalizes it according to LEGO symmetry constraints. Based on the current assembly strategy, the perception module computes grasp orientations, placement orientations, and insertion alignment targets for the robot controller.

Finally, the resulting object pose, grasp orientation, and placement target are exported through a shared YAML interface to the robot execution node. In this way, the perception module serves as the geometric bridge between RGB-D scene understanding and collision-aware robotic assembly execution.

4.4 Execution: Collision-Aware State Machine

The execution module transforms the planned assembly targets and perceived object poses into physically executable robot motions. In our implementation, execution is realized as a ROS 2 and MoveIt 2 based state-machine controller. The controller receives one active assembly task at a time from the perception module through a shared YAML interface and executes a fixed pick-and-place routine for the corresponding LEGO Duplo brick.

Each active task contains the brick name, the estimated brick pose, the robot pick orientation, and the target placement pose. The brick position is obtained from the perception module, while the pick and place orientations are computed according to the detected brick yaw, the selected grasp strategy, and the target assembly blueprint. This separation allows the system to distinguish between the physical pose of the brick in the scene and the end-effector pose required for grasping.

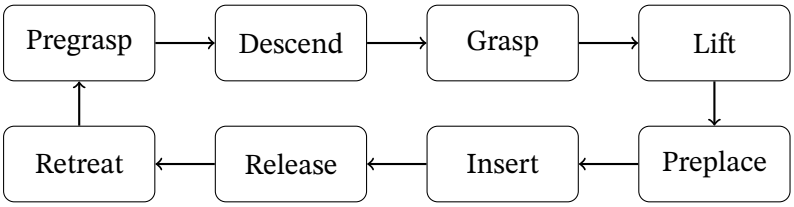


Figure 4.6: Execution state machine for one pick-and-place assembly step. The controller moves from free-space transport to slow insertion, release, and retreat before the next planned part is executed.

The execution node continuously monitors the task file. Once a new task is detected, the node parses the YAML file, selects the corresponding STL mesh according to the brick type, and inserts the brick model into the MoveIt planning scene. In our implementation, both 2×2 and 2×4 LEGO Duplo meshes are supported. The mesh geometry is loaded from an STL file and scaled from millimetres to metres before being used as a collision object in the planning scene.

Before execution, the workspace table is also added as a collision object. The planning scene therefore contains the robot model, the table, the target brick, and the relevant workspace geometry. This allows MoveIt 2 to generate collision-aware motions while considering the robot arm, gripper, object mesh, and surrounding environment.

The manipulation routine is organized into the following execution states:

- PREGRASP
- DESCENDTOGRASP
- GRASP
- LIFT
- PREPLACE
- DESCENDTOPLACE
- RELEASE
- RETREAT

The overall execution flow is illustrated in Figure 4.6. First, the robot moves to a pre-grasp pose above the detected brick. This pose is computed by adding a vertical hover offset to the grasp pose, taking into account the gripper height and a predefined safety clearance. The motion to the pre-grasp pose is executed as a point-to-point motion using MoveIt 2. For these large free-space motions, the system uses relatively high velocity scaling to reduce execution time while still relying on planning-scene collision checking.

4.4.1 Grasping and Placement Execution

After reaching the pre-grasp pose, the robot opens the parallel-jaw gripper and performs a slow vertical Cartesian descent toward the brick. MoveIt first generates the geometric Cartesian path through `computeCartesianPath`. Time-Optimal Trajectory Generation (TOTG) [22] is

then used to assign timestamps, velocities, and accelerations along this path while respecting the configured velocity and acceleration limits. In the real-robot experiments, these limits are deliberately reduced so that the downward approach and contact with the LEGO brick remain slow and conservative.

Once the gripper reaches the grasp pose, it closes through the ROS 2 `GripperCommand` action interface. In the real-robot experiments, gripper opening is executed in two stages and closing velocity is reduced to avoid collisions with nearby bricks and to account for real Duplo friction and gripper-pad compliance. After a successful grasp, the brick is attached to the robot hand in the MoveIt planning scene using the gripper links as touch links, and the robot lifts it vertically with a Cartesian motion.

The robot then moves to a pre-place pose above the target assembly location using a MoveIt point-to-point motion. The final placement descent is executed as a slow Cartesian motion because stud alignment is more sensitive to pose error than free-space transport.

After the brick reaches the target placement pose, the system detaches the object from the robot hand, opens the gripper, and retreats vertically to a safe height. The active YAML task file is removed after completion so that the next action is not started before the current perception and execution step has finished.

4.4.2 Real-Robot Perception Scheduling

The real-robot implementation adds an asynchronous perception–execution schedule to reduce idle time between consecutive pick-and-place steps. In the basic synchronous loop, the robot returns to an observation pose, waits for the vision module to detect and estimate the next target part, executes the corresponding motion, and then repeats the same sequence for the following part. This is robust but inefficient because the vision pipeline remains inactive while the robot is moving.

In the deployed UR5 system, the first RGB-D observation is captured from a fixed observation pose and reused as a frozen picking-area observation for subsequent target queries. While the robot executes the current pick-and-place action, the perception process estimates the 6D pose of the next target from this frozen observation. After a part has been selected, the corresponding image region is stored and excluded from later detections. This prevents repeated selection of the same physical brick when several identical bricks are present, which is especially important for real-robot tasks containing multiple white 2x4 Duplo bricks. The next action can therefore start once the current robot motion finishes and the next perception result is already available.

This scheduling strategy does not change the assembly order or the geometric feasibility checks. It only changes how perception and execution are coordinated in time. It is appropriate for the real-robot validation because all loose bricks are visible in the initial picking-area observation

and the picking area is treated as a known initial scene. More dynamic scenes would require repeated observations or closed-loop re-detection after each manipulation step.

4.4.3 Collision-Aware Motion Planning and Velocity Scheduling

Velocity control is phase-dependent. Free-space motions such as PREGRASP, LIFT, and PREPLACE use higher velocity scaling, while contact-sensitive phases such as DESCENDTOGRASP, DESCENDTOPLACE, and PRESSIN use lower Cartesian velocity and acceleration limits. This scheduling keeps transit motions efficient while reducing impact and alignment errors during grasping and placement.

The real-robot implementation uses more conservative velocity settings than the simulation pipeline. This is mainly due to safety constraints around the physical UR5 workspace, the need to avoid high-impact contact with the LEGO baseplate, and the stronger influence of real friction during grasping and press-in insertion. As a result, real-world manipulation time is expected to be substantially longer than the corresponding simulated manipulation time even when the same high-level assembly sequence is executed successfully.

The execution node is implemented as a ROS 2 multi-threaded execution pipeline. A multi-threaded ROS 2 executor continuously updates robot joint states, TF transformations, and MoveIt planning-scene information while the main task loop simultaneously waits for incoming YAML assembly tasks. This design ensures that motion planning and trajectory execution always operate on up-to-date robot states and environment geometry. To further improve runtime stability, the MoveGroup interface and planning-scene objects are maintained within a controlled execution scope so that MoveIt resources are released safely during shutdown without leaving active planning objects or controller handles.

4.4.4 Failure Handling

Execution failures are monitored explicitly during runtime. A failure condition is triggered when no valid motion plan can be generated, when the executed pose deviates beyond a predefined tolerance, or when grasping or stud engagement does not succeed. Since each manipulation phase is represented as an independent execution state inside the state-machine framework, failed actions can be retried locally without restarting the entire assembly sequence.

For grasping failures, the state-machine structure can incorporate gripper-state feedback for explicit grasp verification. One possible strategy is to monitor the final gripper width after the closing command has been executed. If the measured finger distance differs significantly from the expected grasp width, the system can infer that the brick was not successfully grasped or slipped during closure. The controller can then retry the grasp using alternative grasp candidates generated during the planning stage.

Similarly, insertion failures can be detected through incomplete Cartesian execution, abnormal contact behavior, or excessive pose deviation during the final placement stage. In such situations, the execution module can perform local recovery behaviors such as repeating the insertion at lower velocity, applying small corrective alignment motions, or returning to a nearby pre-place pose before re-executing the insertion trajectory.

5 Evaluation

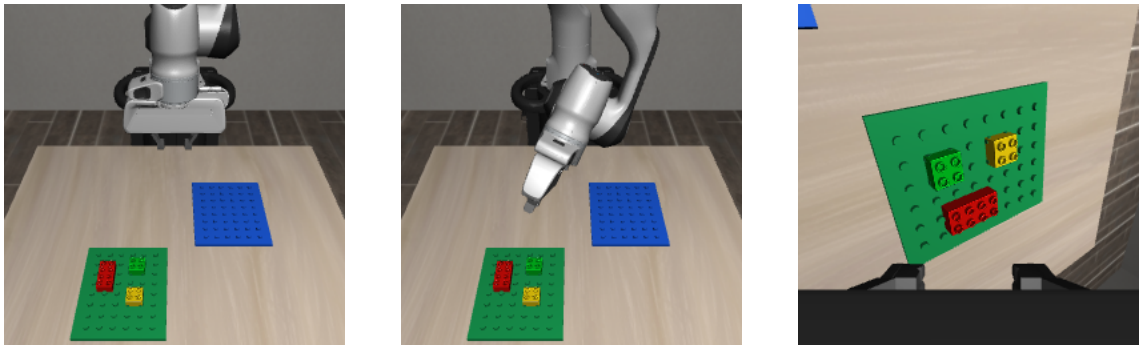
We evaluate the proposed robotic assembly system using both a tiered simulation benchmark and real-robot validation experiments. The evaluation is organized around four questions: whether the benchmark produces increasing task difficulty, how the proposed perception–planning–execution pipeline performs as complexity grows, how it compares with a representative VLM/VLA baseline, and how well the same pipeline transfers to a physical UR5-based setup.

5.1 Experimental Setup

Simulation Environment All simulation experiments are conducted in a custom MuJoCo environment constructed using LIBERO tools and task conventions [25, 42]. This means that the benchmark follows the LIBERO-style simulation workflow and task interface, but the LEGO Duplo assets, baseplates, contact settings, and assembly tasks are defined specifically for this thesis rather than taken from the standard LIBERO task suite. The environment models a Franka Emika Panda robot arm with a parallel-jaw gripper, a tabletop workspace, LEGO Duplo bricks, and LEGO-compatible assembly plates. The setup follows the workspace configuration introduced earlier in this thesis, with initially separated parts represented on one side of the workspace and target structures constructed on LEGO-compatible plates. Figure 5.1 shows example simulation observations from the benchmark.

The scene is defined using MuJoCo XML descriptions. LEGO bricks are loaded from STL meshes and scaled to metric units. To separate visual rendering from physical interaction, each brick uses a visible mesh geometry together with simplified collision primitives. The brick body is approximated using box colliders, while studs are modeled using small cylindrical collision geometries, improving contact stability compared with full mesh-based collision handling. The assembly base plate is modeled as a thin rectangular plate with cylindrical studs aligned to the LEGO Duplo grid, enabling stud-based insertion contact and providing a fixed geometric reference for the assembled structure. Contact parameters including friction, `solref`, and `solimp` are tuned to achieve stable quasi-static interactions during grasping, placement, and press-in execution.

Perception Setup The perception pipeline operates on RGB-D observations from two camera viewpoints: a fixed external camera for global scene perception and a wrist-mounted camera for local manipulation observations. The external camera supports scene-level object detection and



(a) Workspace overview (b) External view (c) Wrist view

Figure 5.1: Example MuJoCo simulation observations used in the tiered assembly benchmark. The simulated setup contains a Franka Emika Panda model, separated LEGO Duplo parts, and a LEGO-compatible target assembly area. During evaluation, both the external workspace view and the wrist-camera view are available as visual observations for perception and for the VLM/VLA baseline input.

pose estimation, while the wrist view improves robustness during close-range manipulation and partially occluded assembly stages.

For all experiments, CAD models of the LEGO parts are provided beforehand and used for CAD-based 6D pose estimation with FoundationPose [45]. The perception pipeline combines GroundingDINO [27] for open-vocabulary object detection, SAM [20] for instance segmentation, and FoundationPose for RGB-D-based pose estimation. The resulting object poses are transformed into the robot coordinate frame and used by the execution module for grasping and placement.

Task Tiers The benchmark contains four task-complexity tiers, as illustrated in Figure 5.2. Increasing tiers contain more assembly parts, denser spatial occupancy, longer assembly horizons, taller structures, and more constrained insertion operations. Tier 1 contains single-brick relocation or placement tasks, Tier 2 evaluates multi-brick vertical stacking, Tier 3 focuses on shape assembly with tighter placement constraints, and Tier 4 contains densely arranged or interlocking structures with more challenging grasping and insertion conditions.

Evaluation Protocol For each tier, $N = 30$ assembly tasks are randomly sampled. Both the proposed system and the VLM/VLA baseline operate under identical workspace conditions and receive the same target structure specifications and initial object configurations. Target structures are represented using structured YAML files that define part identities, assembly poses, and structural relations. During evaluation, the planner derives an executable assembly order, the perception module estimates object poses, and the execution module performs grasping, transportation, alignment, insertion, press-in, and release operations using collision-aware motion planning [6, 7].

(a) Tier 1 (b) Tier 2 (c) Tier 3 (d) Tier 4

Figure 5.2: Examples of the four task complexity tiers used in the simulation evaluation. From left to right: single-brick relocation or placement (Tier 1), multi-brick vertical stacking (Tier 2), shape assembly with tighter placement constraints (Tier 3), and complex interlocking structures (Tier 4).

All experiments are conducted on a workstation equipped with an NVIDIA RTX 4000 Ada Generation GPU.

Real-Robot Validation Setup In addition to the MuJoCo benchmark, the proposed pipeline is validated on the real workspace shown in Figure 4.1. The physical platform consists of a UR5 manipulator, a Robotiq 2F-style parallel gripper, an Intel RealSense RGB-D camera, LEGO Duplo bricks, and two LEGO-compatible baseplates for picking and assembly. The real-robot experiments use the same planning–perception–execution decomposition as the simulation pipeline, with slower robot and gripper motions, strong color-and-size filtering, and conservative collision margins for physical execution. The physical evaluation uses ten representative examples selected from each corresponding simulation tier for Tier 1, Tier 2, and Tier 3, so the tier definitions remain consistent between simulation and real-robot validation. Runtime is measured as complete pipeline wall-clock time, including perception, planning, execution, communication, and intermediate information transfer.

5.2 Baseline Method

Baseline Architecture To provide a representative end-to-end comparison, we implement a vision-language-action (VLA) baseline that combines a vision-language model (VLM) for high-level task decomposition with a pretrained visuomotor policy for low-level manipulation execution [4, 18, 51]. The baseline is designed to represent a modern language-conditioned robotic manipulation pipeline rather than a manually simplified comparison system.

Baseline Inputs The baseline receives the same target assembly specification and visual observations as the proposed method. The target structure is provided in YAML format, while the visual input consists of the RGB observations from the external workspace camera and the wrist-mounted camera, corresponding to the simulation views shown in Figure 5.1. No privileged voxel occupancy state, grasp candidates, or stability labels are provided to the baseline.

Language Planner Starting from the structured target specification, the VLM generates assembly instructions expressed in natural language. In our implementation, we use Gemini 2.5 Flash accessed through an API interface [10]. The prompt contains the final target structure specification together with task-level assembly requirements and manipulation constraints.

Based on the target specification, the VLM predicts a sequence of manipulation instructions, including the target part, placement relation, relative alignment requirement, insertion direction, and press-in action description. The prompts are template-guided and include target positions, relative spatial relationships, insertion directions, and vertical press-in requirements to reduce ambiguous language outputs [15, 17].

Policy Execution The generated language instructions are executed by a pretrained visuomotor manipulation policy inspired by the π_0 family of VLA models [4]. The policy operates directly on RGB observations and language instructions without explicit symbolic planning or geometric reasoning. During execution, the policy predicts continuous robot control commands in a closed-loop manner, including end-effector motion updates and gripper open–close actions.

Comparison Conditions Both the proposed system and the VLA baseline operate under the same workspace configuration, target structures, and initial object layouts. The comparison is not intended to isolate learned policy quality alone: the proposed planner-based system uses additional structured information, including the YAML assembly specification, CAD models, known brick dimensions, camera calibration, and explicit geometric feasibility checks. In contrast, the VLA baseline receives the target specification and RGB observations but does not explicitly maintain a structured representation of assembly feasibility during execution. Collision avoidance, insertion alignment, grasp selection, and manipulation ordering are therefore handled only through the generated language instructions and the learned policy representation.

5.3 Simulation Evaluation

5.3.1 Metrics

We evaluate both systems using task-level metrics related to assembly correctness, execution robustness, and computational efficiency. Table 5.1 summarizes the evaluation metrics used throughout the experiments.

5.3.2 Simulation Results

Success and Accuracy Table 5.2 summarizes the quantitative performance of both methods across all task tiers. The proposed structured perception–planning–execution pipeline achieves

Table 5.1: Task-level evaluation metrics.

Metric	Definition
Assembly Success	If every required brick $v \in V$ is placed at its target pose in the assembly area, attached to the structure, and no longer touched by the robot.
Execution Accuracy	Fraction of brick-to-brick connections whose realised relative pose matches the target. A brick mounted with an offset or not fully pressed down is counted as inaccurate.
Planning Time	Wall-clock time spent on perception, reasoning, and motion planning, excluding physical robot motion.
Manipulation Time	Wall-clock duration of robot motion, from the first command to release of the assembly.
Stability Violation Rate	Fraction of bricks in the assembly area that are not connected to the main structure.

higher success rates and execution accuracy than the VLM/VLA baseline in every tier. In Tier 1 and Tier 2, the proposed system reaches success rates of 96.67% and 90.00%, respectively, while maintaining high execution accuracy and low stability violation rates.

Difficulty Progression As task complexity increases, performance decreases for both methods, but the proposed system remains substantially more robust. In Tier 3 and Tier 4, it still achieves success rates of 70.00% and 66.67%, whereas the VLM/VLA baseline drops to 23.33% and 6.67%. The transition from Tier 2 to Tier 3 introduces a clear increase in structural difficulty: Tier 1 mainly evaluates single-brick placement, Tier 2 focuses on vertical stacking, and Tier 3 introduces spatial shape assembly, denser neighboring structures, and narrower insertion clearances. This explains why the performance drop between Tier 2 and Tier 3 is larger than between the earlier tiers. The stability violation rate increases more strongly from Tier 3 to Tier 4 because Tier 4 includes target shapes whose intermediate assembly process is not automatically balanced; without explicit reasoning about how to preserve stability during construction, especially in the VLM/VLA baseline, plausible placements can still leave parts unsupported or disconnected from the stable main structure.

Runtime The reported planning time increases with task complexity, rising from 7.13 seconds in Tier 1 to 46.82 seconds in Tier 4 for the proposed system. This runtime includes symbolic assembly reasoning as well as perception and feasibility checks, including open-vocabulary detection, instance segmentation, CAD-based 6D pose estimation, grasp-feasibility evaluation, motion-planning queries, and press-stability verification. The increase is therefore mainly an implementation-level trend caused by more parts, more candidate poses, and more feasibility checks. It could be reduced by caching unchanged pose estimates and updating only the affected parts of the voxel state.

Table 5.2: Quantitative comparison across different task tiers. Higher is better for execution accuracy and success rate. Lower is better for planning time, manipulation time and stability violations. For the proposed system, planning time includes perception, symbolic and geometric reasoning, feasibility checks, and motion-planning queries; for the VLM/VLA baseline, it includes language-level instruction generation and policy-side planning overhead before robot motion.

Metric	Tier 1	Tier 2	Tier 3	Tier 4	Overall
Ours (Structured Pipeline)					
Execution Accuracy (%)	**93.33**	**87.22**	**68.13**	**66.55**	**78.81**
Planning Time (s)	7.13	13.66	44.78	46.82	28.10
Manipulation Time (s)	**8.28**	**14.08**	**51.34**	**53.66**	**31.84**
Stability Violations (%)	**3.33**	**5.00**	**10.80**	**25.07**	**11.05**
Success Rate (%)	**96.67**	**90.00**	**70.00**	**66.67**	**80.84**
VLM/VLA Baseline					
Execution Accuracy (%)	66.67	48.82	19.34	13.54	37.09
Planning Time (s)	**2.18**	**2.29**	**3.15**	**3.66**	**2.80**
Manipulation Time (s)	15.46	48.82	164.45	178.98	101.93
Stability Violations (%)	38.33	55.02	82.97	87.73	68.51
Success Rate (%)	73.33	63.33	23.33	6.67	41.67

Manipulation time follows a similar trend because higher-tier tasks require more grasping, transport, placement, insertion, and retreat operations. The proposed system remains faster than the baseline in complex tiers because explicit geometric checks reduce failed attempts and recovery motions. The baseline average is also inflated by timeout cases: each simulation trial is stopped when it reaches the predefined trial-level execution limit of 200 s, and this timeout is counted as a failed trial with the elapsed time included in the reported average.

Baseline Behavior The VLM/VLA baseline shows a much sharper performance degradation as task complexity increases. Although it can complete some simple assembly tasks, its success rate decreases from 73.33% in Tier 1 to only 6.67% in Tier 4, while execution accuracy decreases and stability violations increase substantially. A major limitation is that the baseline does not explicitly model geometric and physical feasibility. Its generated language-level instructions may be semantically plausible but physically infeasible during execution. For example, the baseline may select a reasonable target part or placement relation while failing to account for grasp clearance, neighboring obstacles, collision-free insertion trajectories, or the downward press required for LEGO stud engagement.

In addition, the VLM/VLA baseline is sensitive to prompt design and instruction quality. Small changes in language phrasing, task decomposition, or spatial descriptions can lead to different manipulation behaviors during execution. Template-guided prompting improves consistency

to some extent, but the generated instructions still do not explicitly enforce voxel occupancy constraints, grasp reachability, collision-free insertion, or press stability. This explains the high rate of failed insertions, unstable intermediate assemblies, and collisions in higher-tier tasks.

Planner Behavior The proposed planner differs most clearly from the baseline in cases where local structure affects the next feasible action. The subassembly branch is activated when the selected candidate belongs to a connected local component rather than behaving as an isolated brick. This occurs mainly in higher-tier examples, such as bridge-like structures or compact multi-brick groups, where removing or placing one brick changes the feasibility of neighboring connected parts. In these cases, the planner groups the local component, reasons about its internal order, and then continues with the remaining structure. This observation is not a separate ablation study of the subassembly mechanism, but it explains where the mechanism appears during the full-pipeline evaluation. The broader qualitative failure modes are discussed later in the error analysis.

The baseline failure cases are qualitatively different. It often produces a plausible semantic order but does not detect that the selected placement would require an obstructed grasp, a narrow insertion path, or a downward press that destabilizes the partial structure. As a result, higher-tier failures commonly appear as collision-prone approaches, incomplete insertions, or unstable placements rather than simply choosing the wrong target object.

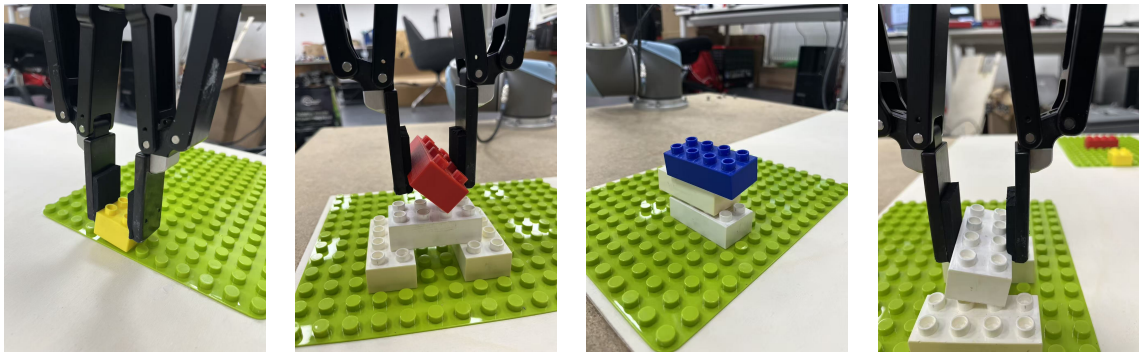
5.4 Real-Robot Validation

Real-World Transfer The real-robot experiments evaluate the pipeline on a physical UR5-based assembly setup. The system uses the planned assembly sequence to request target parts, applies open-vocabulary and color- and size-aware perception to localize the active brick, estimates its 6D pose from RGB-D observations, and executes the corresponding grasp and placement action with the collision-aware state machine. Compared with simulation, the real setup introduces additional uncertainty from camera calibration, depth noise, gripper compliance, LEGO friction, and small manufacturing or placement tolerances. These factors make the real-robot trials a useful validation of the physical assumptions made in the planner and execution module.

Physical Trial Results Table 5.3 summarizes the physical validation results. The real robot completes 9 out of 10 Tier 1 trials, 9 out of 10 Tier 2 trials, and 7 out of 10 Tier 3 trials. The corresponding average full-pipeline runtimes are 55.77 s, 167.98 s, and 183.33 s. Tier 4 was not attempted on the physical setup and is left for future real-robot validation. The Tier 3 success rate decreases because the task has tighter placement tolerances and less margin for pose-estimation error, tilted grasps, or baseplate contact offsets.

Table 5.3: Real-robot validation results on the UR5 setup. Runtime is measured as full pipeline wall-clock time, including perception, planning, execution, communication, and intermediate information transfer.

Tier	Physical task	Steps	Success	Runtime (s)
Tier 1	Single-brick relocation to a fixed target pose	1	9/10	55.77 ± 4.29
Tier 2	Multi-brick vertical stacking	2–4	9/10	167.98 ± 8.96
Tier 3	Shape assembly, e.g., bridge structure	4–6	7/10	183.33 ± 7.38



(a) Small-part tilt (b) Off-center closure (c) Inaccurate stack (d) High grasp point

Figure 5.3: Representative real-robot failure cases. (a) Smaller parts are more sensitive to pose and grasp errors, which can lead to tilted placement. (b) Off-center gripper closure applies asymmetric force and can lift one side of the brick. (c) Perception or execution offsets can accumulate into an inaccurate final stack. (d) A grasp point that is too high on the brick can rotate the object and produce a tilted insertion pose.

Runtime Gap The real-robot runtime is longer than the simulated manipulation time because it measures the complete deployed pipeline, not only robot motion. It includes perception, planning, execution, robot communication, and intermediate information transfer. The UR5 is also operated with conservative safety limits, the Robotiq 2F-style gripper moves more slowly to handle LEGO friction and contact uncertainty, and the perception stage performs additional color-and-size filtering. These choices increase runtime but reduce collisions, brick displacement, and failed grasps.

Error Analysis The simulation and real-robot experiments show related error patterns, but the physical setup makes several of them more visible. In simulation, most remaining failures in Tier 3 and Tier 4 are caused by perception uncertainty, insertion sensitivity, narrow gripper clearance, accumulated placement error, and press instability in tall or weakly supported structures. Small pose errors can produce residual misalignment during stud insertion, while a sequence that is feasible under the voxel approximation can still become difficult when the available clearance is narrow or when earlier placements have accumulated small offsets.

The real-robot trials exhibit the same underlying sensitivities, but with additional physical causes from calibration error, depth noise, gripper compliance, LEGO friction, and contact force. Figure 5.3 summarizes representative physical examples. An off-center gripper closure applies asymmetric force to the brick and can lift one side of the object instead of producing a stable side grasp. This can then lead to tilted placement, especially for smaller parts, because a small angular error occupies a larger fraction of the part footprint and makes stud alignment more sensitive. A grasp point that is too high on the brick can also rotate the object during lifting or transport, producing a tilted insertion pose. In other cases, perception or execution offsets accumulate across steps and lead to an imprecise final stack even if every individual action is approximately correct.

Gripper force and contact material are also important in the real system. If the gripper closes too strongly, it can over-constrain or drag the brick; if it closes too weakly or the robot lifts too quickly, low friction between the fingers and the plastic surface can cause the brick to slip or fail to lift. During vertical stacking, however, the downward press for upper layers can sometimes correct a small pose error in a lower layer by pushing the partially seated brick back toward the stud grid. These observations suggest that robust real-world LEGO assembly requires not only accurate pose estimation, but also appropriate gripper-pad material, carefully tuned closing force, slower lift motions after grasping, and explicit grasp-success verification before transport.

Implications The real-robot results support the main conclusion of the simulation benchmark while also clarifying the remaining sim-to-real gap. Explicit sequence planning and geometric feasibility checks are still useful because they reduce obviously infeasible orders, blocked grasps, and unstable press operations before execution. However, real-world robustness also depends on low-level sensing and contact handling. In particular, pose estimation accuracy, gripper-force tuning, color- and size-aware part selection, and closed-loop insertion refinement become more important once the system is deployed on physical hardware. A task-specific fingertip or gripper geometry that applies downward pressure more directly could improve LEGO placement accuracy, but the current Robotiq 2F-style gripper remains more general because it can grasp a broader range of object shapes beyond the LEGO-specific setup.

Overall, the results show that explicit assembly-sequence planning, voxel-based grasp reachability analysis, collision-aware motion planning, and press-stability reasoning provide clear advantages over purely language-driven end-to-end policies for structured robotic assembly. At the same time, the remaining errors suggest that future work should incorporate closed-loop visual servoing or other feedback-based insertion refinement mechanisms during final alignment and press-in execution.

6 Conclusion

This thesis presented a geometry-aware robotic assembly framework for LEGO Duplo assembly tasks. The proposed system integrates structured assembly planning, open-vocabulary perception, and collision-aware execution within a unified robotic pipeline that transforms high-level product specifications into executable robotic manipulation actions. The system was evaluated on a hierarchical LEGO Duplo assembly benchmark designed to analyze robotic assembly performance under progressively increasing geometric and structural complexity, and it was additionally validated on a physical UR5-based assembly setup.

To generate physically executable assembly sequences, this work introduced an Assembly-by-Disassembly (ABD) planning strategy combined with voxel-based geometric reasoning. The framework incorporates grasp reachability analysis, collision-aware feasibility verification, and press-stability reasoning to handle constrained assembly operations under structural and workspace limitations. In addition, the perception and execution modules enable open-vocabulary object localization, CAD-based 6D pose estimation, and collision-aware robotic manipulation within cluttered assembly environments.

Experimental results demonstrate that the proposed pipeline consistently outperforms the VLM/VLA baseline across all task tiers. In particular, explicit geometric reasoning and physically grounded planning improve execution robustness, insertion stability, and manipulation efficiency in long-horizon assembly tasks. The experiments further show that purely language-driven end-to-end policies frequently generate semantically plausible but physically infeasible assembly actions when geometric constraints, grasp reachability, collision constraints, and structural stability are not explicitly modeled. The real-robot validation further shows that the pipeline can be executed on physical hardware, completing 9/10 Tier 1 trials, 9/10 Tier 2 trials, and 7/10 Tier 3 trials on the UR5 setup. However, practical deployment introduces additional runtime and robustness challenges from safety-limited robot speeds, real LEGO friction, gripper-force tuning, color-based part disambiguation, communication overhead, and calibration-sensitive pose estimation.

The results additionally suggest that hybrid robotic systems combining semantic perception, explicit geometric reasoning, and physically grounded execution currently provide a practical compromise between generalization capability and execution reliability for robotic assembly. While recent VLM/VLA approaches demonstrate strong semantic reasoning and open-world interaction capability, reliable contact-rich robotic assembly still requires explicit modeling of geometry, physical feasibility, and interaction constraints.

At the same time, the experiments reveal several remaining limitations of the current system. In higher-complexity assembly tasks, system performance becomes increasingly sensitive to perception uncertainty and insertion precision. Error accumulation becomes particularly significant in long-horizon assembly sequences, where small inaccuracies in 6D pose estimation or insertion alignment may propagate across multiple manipulation stages and eventually lead to insertion failure or unstable contact behavior. The real-robot failure cases show this limitation directly: an off-center pose estimate can cause asymmetric gripper closure, tilted lifting, brick displacement, or failed insertion. In addition, the current press-stability analysis still relies on simplified geometric approximations and does not yet model more complex physical interaction effects such as force propagation, contact dynamics, gripper compliance, or friction-dependent brick behavior during insertion.

Future work will focus on incorporating closed-loop visual servoing and feedback-based refinement during the final alignment and insertion stages. Such feedback mechanisms could compensate for residual pose-estimation errors and further improve insertion robustness in highly constrained assembly scenarios. Future planning methods may also incorporate more accurate press-stability analysis, force-aware insertion reasoning, and richer contact modeling. Additional future directions include larger-scale real-robot evaluation, automatic gripper-force adaptation, dynamic replanning under uncertain or partially observable environments, and grasp-candidate generation for non-LEGO objects whose geometry is not well represented by the current stud-grid and long-side grasp assumptions. These directions aim to improve both geometric robustness and adaptive interaction capability during contact-rich robotic assembly.

Beyond the specific LEGO Duplo setting, the proposed framework suggests a possible path toward more usable structured tabletop assembly systems. This claim should be interpreted with the current assumptions in mind: the planner and pose-estimation pipeline still rely on structured task specifications, known object geometry, CAD models, and calibrated RGB-D observations. Therefore, the most direct extension is not unconstrained household manipulation, but broader blocks-world and tabletop assembly scenarios where object geometry can be modeled and where contact constraints remain important. Future work can gradually relax these assumptions by improving object modeling, grasp-candidate generation, closed-loop correction, and handling of partially known geometries.

Overall, this work demonstrates the importance of combining geometric reasoning, physically grounded planning, and collision-aware execution for reliable long-horizon robotic assembly in structured contact-rich environments.

List of Figures

List of Tables

List of Algorithms

List of References

[1] J. Aleotti and S. Caselli, "Efficient planning of disassembly sequences in physics-based animation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2009, pp. 87–92. DOI: 10.1109/IROS.2009.5354401.

[2] A. Barghi, N. Pasiar, M. T. Masouleh, and A. Kalhor, "From bricks to bots: Automated collision-aware sequence planning for LEGO reconstruction with a two-finger gripper," in *2024 10th International Conference on Control, Instrumentation and Automation (ICCIA)*, 2024, pp. 1–6. DOI: 10.1109/ICCIA65044.2024.10768173.

[3] D. Beßler, M. Pomarlan, A. Akbari, Muhayyuddin, M. Diab, J. Rosell, J. Bateman, and M. Beetz, "Assembly planning in cluttered environments through heterogeneous reasoning," in *KI 2018: Advances in Artificial Intelligence*, Springer, 2018, pp. 201–214. DOI: 10.1007/978-3-030-00111-7_18.

[4] K. Black et al., *π_0: A vision-language-action flow model for general robot control*, 2024. arXiv: 2410.24164 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2410.24164.

[5] A. Brohan et al., "RT-1: Robotics transformer for real-world control at scale," in *Proceedings of Robotics: Science and Systems*, 2023. DOI: 10.15607/RSS.2023.XIX.025. arXiv: 2212.06817 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2212.06817.

[6] S. Chitta, I. A. Sucan, and S. Cousins, "MoveIt!" *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012. DOI: 10.1109/MRA.2011.2181749.

[7] D. Coleman, I. A. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A MoveIt! case study," *Journal of Software Engineering for Robotics*, vol. 5, no. 1, pp. 3–16, 2014. DOI: 10.6092/JOSER_2014_05_01_p3.

[8] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, vol. 2, no. 3–4, pp. 189–208, 1971. DOI: 10.1016/0004-3702(71)90010-5.

[9] N. Funk, S. Menzenbach, G. Chalvatzaki, and J. Peters, "Graph-based reinforcement learning meets mixed integer programs: An application to 3d robot assembly discovery," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022. arXiv: 2203.04120. [Online]. Available: https://arxiv.org/abs/2203.04120.

[10] Google Cloud, *Gemini 2.5 flash*, Accessed: 2026-05-27, 2026. [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-flash.

[11] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3–4, pp. 577–601, 2000. DOI: 10.1007/S004539910025.

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322. [Online]. Available: https://openaccess.thecvf.com/content_iccv_2017/html/He_Mask_R-CNN_ICCV_2017_paper.html.

[13] M. Heo, Y. Lee, D. Lee, and J. J. Lim, "Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation," *The International Journal of Robotics Research*, vol. 44, no. 10–11, pp. 1863–1891, 2025. DOI: 10.1177/02783649241304789.

[14] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," in *Asian Conference on Computer Vision (ACCV)*, Springer, 2012, pp. 548–562. DOI: 10.1007/978-3-642-37331-2_42.

[15] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, T. Jackson, N. Brown, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," in *Proceedings of the 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 205, PMLR, 2023, pp. 1769–1782. [Online]. Available: https://proceedings.mlr.press/v205/huang23c.html.

[16] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, 1996. DOI: 10.1109/70.538972.

[17] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu, "Do as i can, not as i say: Grounding language in robotic affordances," in *Proceedings of the 6th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 205, PMLR, 2023, pp. 287–318. [Online]. Available: https://proceedings.mlr.press/v205/ichter23a.html.

[18] M. J. Kim et al., *OpenVLA: An open-source vision-language-action model*, 2024. arXiv: 2406.09246 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2406.09246.

[19] S.-K. Kim and M. Likhachev, "Parts assembly planning under uncertainty with simulation-aided physical reasoning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 4074–4081. DOI: 10.1109/ICRA.2017.7989463.

[20] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 4015–4026. [Online]. Available: https://openaccess.thecvf.com/content/ICCV2023/html/Kirillov_Segment_Anything_ICCV_2023_paper.html.

[21] K. Kitz and U. Thomas, "Neural dynamic assembly sequence planning," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 2063–2068. DOI: 10.1109/CASE49439.2021.9551620.

[22] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, Jul. 2012. DOI: 10.15607/RSS.2012.VIII.027.

[23] D. T. Le, J. Cortés, and T. Siméon, "A path planning approach to (dis)assembly sequencing," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2009, pp. 286–291. DOI: 10.1109/COASE.2009.5234177.

[24] Y. Lee, E. S. Hu, Z. Yang, A. Yin, and J. J. Lim, *IKEA furniture assembly environment for long-horizon complex manipulation tasks*, 2019. arXiv: 1911.07246 [cs.RO]. [Online]. Available: https://arxiv.org/abs/1911.07246.

[25] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, "LIBERO: Benchmarking knowledge transfer for lifelong robot learning," in *Advances in Neural Information Processing Systems*, vol. 36, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

[26] R. Liu, K. Deng, Z. Wang, and C. Liu, *Stablelego: Stability analysis of block stacking assembly*, 2024. arXiv: 2402.10711 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2402.10711.

[27] S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang, J. Yang, Q. Jiang, C. Li, J. Yang, H. Su, J. Zhu, and L. Zhang, "Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection," in *European Conference on Computer Vision (ECCV)*, 2024. arXiv: 2303.05499. [Online]. Available: https://arxiv.org/abs/2303.05499.

[28] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022. DOI: 10.1126/scirobotics.abm6074.

[29] L. S. H. de Mello and A. C. Sanderson, "AND/OR graph representation of assembly plans," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 2, pp. 188–199, 1990. DOI: 10.1109/70.54734.

[30] L. S. H. de Mello and A. C. Sanderson, "A correct and complete algorithm for the generation of mechanical assembly sequences," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 2, pp. 228–240, 1991. DOI: 10.1109/70.75905.

[31] L. Nägele, A. Hoffmann, A. Schierl, and W. Reif, "Legobot: Automated planning for coordinated multi-robot assembly of LEGO structures," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 9088–9095. DOI: 10.1109/IROS45743.2020.9341428.

[32] P. Pu and M. Reschberger, "Assembly sequence planning using case-based reasoning techniques," *Knowledge-Based Systems*, vol. 4, no. 3, pp. 123–130, 1991. DOI: 10.1016/0950-7051(91)90001-I.

[33] A. Radford et al., "Learning transferable visual models from natural language supervision," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 8748–8763. [Online]. Available: https://proceedings.mlr.press/v139/radford21a.html.

[34] S. Rakshit and S. Akella, "The influence of motion paths and assembly sequences on the stability of assemblies," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 615–627, 2015. DOI: 10.1109/TASE.2014.2345569.

[35] T. Ren, S. Liu, A. Zeng, J. Lin, K. Li, H. Cao, J. Chen, X. Huang, Y. Chen, F. Yan, Z. Zeng, H. Zhang, F. Li, J. Yang, H. Li, Q. Jiang, and L. Zhang, *Grounded SAM: Assembling open-world models for diverse visual tasks*, 2024. arXiv: 2401.14159 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2401.14159.

[36] A. C. Sanderson, L. S. H. de Mello, and H. Zhang, "Assembly sequence planning," *AI Magazine*, vol. 11, no. 1, pp. 62–81, 1990. DOI: 10.1609/aimag.v11i1.824. [Online]. Available: https://www.sigmod.org/publications/dblp/db/journals/aim/aim11.html.

[37] SMACH Contributors, *SMACH: Task-level architecture for rapidly creating complex robot behavior*, Accessed: 2026-05-27, 2023. [Online]. Available: https://docs.ros.org/en/jazzy/p/smach/.

[38] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012. DOI: 10.1109/MRA.2012.2205651.

[39] S. Sundaram, I. Remmler, and N. M. Amato, "Disassembly sequencing using a motion planning approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, 2001, pp. 1475–1480. DOI: 10.1109/ROBOT.2001.932818.

[40] Y. Tian, K. D. D. Willis, B. A. Omari, J. Luo, P. Ma, Y. Li, F. Javid, E. Gu, J. Jacob, S. Sueda, H. Li, S. Chitta, and W. Matusik, "ASAP: Automated sequence planning for complex robotic assembly with physical feasibility," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 4380–4386. DOI: 10.1109/ICRA57147.2024.10611595. arXiv: 2309.16909. [Online]. Available: https://asap.csail.mit.edu/.

[41] Y. Tian, J. Xu, Y. Li, J. Luo, S. Sueda, H. Li, K. D. D. Willis, and W. Matusik, "Assemble them all: Physics-based planning for generalizable assembly by disassembly," *ACM Transactions on Graphics*, vol. 41, no. 6, pp. 1–11, 2022. DOI: 10.1145/3550454.3555525.

[42] 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*, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[43] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "Densefusion: 6d object pose estimation by iterative dense fusion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3343–3352. DOI: 10.1109/CVPR.2019.00348. [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2019/html/Wang_DenseFusion_6D_Object_Pose_Estimation_by_Iterative_Dense_Fusion_CVPR_2019_paper.html.

[44] K. Watanabe and S. Inada, "Search algorithm of the assembly sequence of products by using past learning results," *International Journal of Production Economics*, vol. 226, p. 107 615, 2020. DOI: 10.1016/j.ijpe.2020.107615.

[45] B. Wen, W. Yang, J. Kautz, and S. Birchfield, "Foundationpose: Unified 6d pose estimation and tracking of novel objects," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. arXiv: 2312.08344. [Online]. Available: https://arxiv.org/abs/2312.08344.

[46] R. H. Wilson and J.-C. Latombe, "Geometric reasoning about mechanical assembly," *Artificial Intelligence*, vol. 71, no. 2, pp. 371–396, 1994. DOI: 10.1016/0004-3702(94)90048-5.

[47] J. D. Winter, I. E. Makrini, G. V. de Perre, A. Nowé, T. Verstraten, and B. Vanderborght, "Autonomous assembly planning of demonstrated skills with reinforcement learning in simulation," *Autonomous Robots*, vol. 45, no. 8, pp. 1097–1110, 2021. DOI: 10.1007/s10514-021-10020-x.

[48] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes," in *Proceedings of Robotics: Science and Systems*, 2018. [Online]. Available: https://research.nvidia.com/labs/srl/publication/xiang-2018-pose-cnn/.

[49] M. Yu, L. Shao, Z. Chen, T. Wu, Q. Fan, K. Mo, and H. Dong, *Roboassembly: Learning generalizable furniture assembly policy in a novel multi-robot contact-rich simulation environment*, 2021. arXiv: 2112.10143 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2112.10143.

[50] X. Zhang, R. Belfer, P. G. Kry, and E. Vouga, "C-space tunnel discovery for puzzle path planning," *ACM Transactions on Graphics*, vol. 39, no. 4, 104:1–104:14, 2020. DOI: 10.1145/3386569.3392468.

[51] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, Q. Vuong, V. Vanhoucke, H. Tran, R. Soricut, A. Singh, J. Singh, P. Sermanet, P. R. Sanketi, G. Salazar, M. S. Ryoo, K. Reymann, K. Rao, K. Pertsch, I. Mordatch, H. Michalewski, Y. Lu, S. Levine, L. Lee, T.-W. E. Lee, I. Leal, Y. Kuang, D. Kalashnikov, R. Julian, N. J. Joshi, A. Irpan, B. Ichter, J. Hsu, A. Herzog, K. Hausman, K. Gopalakrishnan, C. Fu, P. Florence, C. Finn, K. A. Dubey, D. Driess, T. Ding, K. M. Choromanski, X. Chen, Y. Chebotar, J. Carbajal, N. Brown, A. Brohan, M. G. Arenas, and K. Han, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proceedings of the 7th Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 229, PMLR, 2023, pp. 2165–2183. [Online]. Available: https://proceedings.mlr.press/v229/zitkovich23a.html.