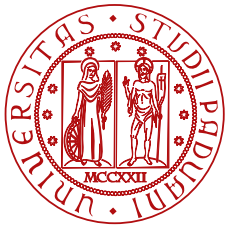


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in
Computer Engineering - AI & Robotics

A FOUNDATION MODEL FOR ROBOTIC MANIPULATION: GAM, GRASP-ANYTHING-MODEL

Master Candidate

Matteo Villani

Student ID 2090299

Supervisor

Prof. Emanuele Menegatti

University of Padova

Co-supervisor

Dott. Alberto Bacchin

University of Padova

Academic Year
2024/2025

"If you can't take risks you can't make a future!"

- Monkey D. Rufy

Abstract

This thesis introduces the *Grasp-Anything Model (GAM)*, a modular, interpretable, and ROS2-native grasping pipeline designed to enable zero-shot manipulation of arbitrary objects in unstructured environments. Built for the Tiago family of service robots developed by PAL Robotics, GAM integrates prompt-grounded perception, single-view 3D reconstruction, grasp pose generation, and task-aware motion planning into a unified framework. Unlike monolithic or dataset-bound systems, GAM embraces decoupled service-oriented modules—including *Grounded-Segment-Anything (GSAM)*, *Multiview Compressive Coding* (*MCC*), and *MoveIt Task Constructor* (*MTC*)—to ensure component reusability, architectural transparency, and diagnostic granularity.

The system is evaluated across both simulated and real-world settings using a representative set of objects varying in geometry, size, and occlusion. Empirical results demonstrate GAM's ability to generalize grasp strategies without object-specific retraining, achieve stable motion execution in partially observed scenes, and operate under CPU-only constraints with total runtime under one minute per grasp. Planning latency is significantly reduced via *MTC*'s structured action pipeline, while grasp success rates exceeded 70% on average across diverse object categories. Nonetheless, limitations in sensor quality, segmentation under ambiguity, and grasp planning near constrained surfaces highlight opportunities for further optimization.

By prioritizing modularity over end-to-end speed, this work contributes a foundational manipulation framework suitable for deployment, extension, and research in real-world robotic systems. GAM is not presented as a final grasping solution, but as a robust platform upon which future advances in data-driven manipulation, adaptive behaviour logic, and system-level autonomy can be built.

Contents

List of Figures

List of Tables

List of Code Snippets

List of Acronyms

GAM Grasp Anything Model

GPD Grasp Pose Detection

SAM Segment Anything Model (v1)

GSAM Grounded SAM

MCC Multiview Compressive Coding

YOLOe You Only Look Once Everything

MVBB Minimum Volume Bounding Box

MVS Multi View Stereo

NeRF Neural Radiance Fields

SDF Signed Distance Functions

RGB-D RGB-depth camera sensor

PCL Point Cloud Library

LLM Large Language Model

ROS Robotic Operating System

MTC MoveIt Task Constructor

BT Behavior Tree

IK Inverse Kinematics

DoF Degree of Freedom

CoM Center of Mass

PoV Point of View

AP Average Precision

mAP mean Average Precision

PSNR Peak signal-to-noise ratio

GUI Graphical User Interface

SoTA state-of-the-art

RQ Research Question

DNNs Deep Neural Networks

1 Introduction

Grasping arbitrary objects in dynamic, unstructured environments remains a fundamental challenge in service robotics, particularly when deployed on platforms with limited computational resources. Existing approaches often assume powerful GPU hardware [6] [40], rely on fixed object datasets [9] [39], or fail under sensor noise and occlusion [70]. This thesis addresses these limitations by introducing the **Grasp-Anything Model (GAM)**, a lightweight, modular, and CPU-feasible grasping pipeline tailored to the Tiago robot family developed at **PAL Robotics**. GAM integrates prompt-driven perception, single-view 3D reconstruction, 6-Degrees-of-Freedon (DoF[1]) grasp synthesis, and task-aware motion planning into a unified ROS2 architecture. Unlike monolithic or data-hungry pipelines [40], GAM is designed to generalize to novel objects, maintain real-time responsiveness, and operate under real-world deployment constraints.

1.1 Background

Robotic grasping remains one of the most complex and actively researched challenges in autonomous manipulation. From warehouse automation to assistive robotics, the ability to identify, reconstruct, and manipulate everyday ob-

[1] **Degrees-of-Freedom** define the number of independent movements or parameters required to fully describe a system's configuration. In robotics, **6-DoF** refers to an object's ability to move freely in 3D space: three translational (x, y, z axes) and three rotational (roll, pitch, yaw) movements. For example, a 6-DoF robotic gripper can position and orient itself arbitrarily to grasp objects.

jects in unstructured environments is essential for the deployment of robots in the real world. Historically, grasping systems often rely on hand-crafted heuristics and analytical models [34], or extensive prior knowledge of object geometries (e.g. detailed CAD-like object models) to compute stable grasps [9] [70]. These approaches, while effective in constrained environments, often **failed to generalize** beyond pre-modeled or geometrically simple objects.

In recent years, the availability of **RGB-D sensors** and advances in deep learning have led to the rise of data-driven and vision-based grasping pipelines. These pipelines typically integrate modules for object detection, 3D reconstruction, and grasp synthesis, using large datasets to learn grasp affordances [9] [39]. However, while **is powerful under ideal conditions**, these approaches often lack modular fault tolerance, transparency, and reusability. Many of these systems assume access to high-end GPU resources, require fixed training datasets, or fail to account for occlusions, lighting variation, and sensor noise — all of which degrade performance **in real-world scenarios** [70] [9].

This research is carried out in the dynamic and innovative environment of PAL Robotics, in Barcelona, during my internship. **PAL Robotics** is a leader in the development of mobile and service manipulators, with a strong focus on real-world applications. It is a company comprising over 150 members, coming from more than 25 nationalities: with more than 20 years of expertise in developing service robots to enhance people's quality of life in numerous scenarios like healthcare, university, research, industries, retail, etc… The target sectors of the robots that are being produced at PAL range from **Mobile Interaction**, where we can find robots like ARI and Tiago family (e.g. Tiago, Tiago++, Tiago-Pro): that are mainly products and services for industries and research; **Legged**, that involves robots like Reem-C, Kangaroo and Talos, developed for state-of-the-art humanoid research; to Intralogistics systems that automate the transportation of goods and perform inventory tracking.

This work focuses mainly on the **manipulator robots**: in particular, on Tiago, Tiago++ and Tiago-Pro, presented in Fig. 1.1. All platforms are equipped with RGB-D Orbbec Astra camera [43], either one or two 7-DoF arms with an interchangeable end-effector including parallel-jaw gripper, 2F-140, 2F-85 and ePick. In this setting the Tiago robots serve as a practical platform for testing and vali-

dating advanced grasping techniques.

(a) Tiago (b) Tiago Dual (Tiago++) (c) Tiago-Pro

Figure 1.1: Tiago Family

This thesis builds upon that modular perspective by integrating state-of-the-art perception and planning components into a flexible, interpretable, generalizable, and resource-aware grasping system. The design targets manipulation platforms, that operate under computational and sensory constraints, with the goal of enabling real-world object grasping through grounded, prompt-driven interaction, improving the grasping capabilities of the Tiago robots.

1.2 Context

My work on the Grasp-Anything Model (GAM) is deeply rooted in the challenges encountered in industrial automation and service robotics manipulation. At PAL Robotics, the objective is not only to achieve high-fidelity reconstructions but also to ensure that the system operates in real time— delivering robust performance even under noisy sensor conditions and in cluttered environments. The research aims to integrate state-of-the-art perception modules, efficient 3D reconstruction methods, grasp pose generation algorithms and collision-free trajectory planning & execution into a unified pipeline. This modular approach ensures that each component can be **optimized independently** while contributing to an overall system capable of generalizing to a wide variety of objects' shapes and dimensions.

While prior **ROS1**-based implementations were restricted to predefined object geometries due to middleware limitations, the **migration to ROS2**—a technical imperative for this work—provides the architectural foundation necessary to generalize across diverse object shapes and dimensions. By leveraging ROS2's enhanced real-time communication, distributed computing, and modular service orchestration, GAM addresses the scalability demands of industrial and service robotics, ensuring reliable operation in cluttered, dynamic environment.

By addressing the practical challenges of detection, localization, reconstruction, and real-time processing, this research seeks to bridge the gap between the existing traditional grasping methods and generalized object grasping strategies. The ultimate goal is to improve the grasping capabilities of Tiago robots (Fig. 1.1), enabling them to handle grasps reliably and efficiently in everyday environments optimized for unseen objects.

1.3 Purposes and Objectives

The primary aim of this research is to develop a robust, lightweight, and efficient grasping pipeline to enhance the manipulation capabilities of Tiago robots. GAM is designed to overcome practical deployment constraints by prioritizing **modularity**, **CPU feasibility**, and **zero-shot generalization**, in contrast to existing approaches that rely on GPU hardware, fixed object datasets, or rigid integration [6, 9].

To this end, the research addresses core challenges identified in the background: operating under **sensor uncertainty**, handling **arbitrary object geometries**, and enabling **end-to-end autonomy** without relying on task-specific training. The specific technical and methodological objectives are as follows:

- **Perception Module:** Develop a perception pipeline that leverages advanced detection and segmentation techniques to reliably detect and segment the desired object/s from cluttered scenes. This module addresses the limitations of prior pipelines that struggle with visual occlusion and require dataset-specific object classes.
- **3D Reconstruction:** Implement a single-view 3D reconstruction strategy robust to noisy RGB-D inputs. Unlike traditional systems that rely on full object models or multi-view input, GAM's reconstruction module must infer occluded geometry using only partial views, compensating for hardware limitations.

- **Grasping Pose Detection:** Integrate a 6-DoF grasp pose prediction module that computes stable, feasible grasp candidates based on the reconstructed shape and robot constraints. This component moves beyond heuristic or CAD-based approaches by leveraging data-driven inference that generalizes across object geometries.
- **Trajectory Planning and Execution:** Develop a task-aware motion planning module that selects high-quality grasp poses and executes a collision-free trajectory, leveraging *MoveIt Task Constructor (MTC)* for modularity and robustness. This aims to replace the sequential, monolithic action chains common in earlier systems.
- **Execution and validation:** Rigorously evaluate the pipeline under various conditions: object variability, occlusion, sensor misalignment, to demonstrate generalization and fault tolerance. This objective reinforces the practical value of the system and reflects its real-world integration at PAL Robotics.

Together, these objectives form a tightly connected response to the limitations described in the background. GAM is **modular** where other systems are monolithic, **zero-shot** where others are data set-bound and **hardware-efficient** where others assume computational abundance. The result, as evidenced by the high-level architecture of GAM in Fig.1.2, is a practical grasping framework tailored for modern robotic platforms operating under real-world constraints.

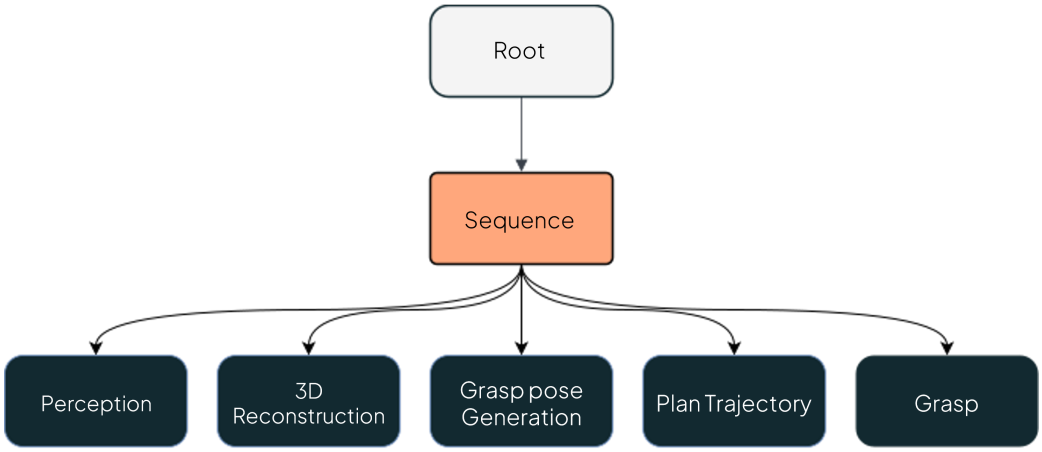


Figure 1.2: High-level architecture of the Grasp-Anything-Model (GAM): how the Behaviour Tree (BT) is structured.

1.4 Significance, Scope and Definitions

The Grasp-Anything Model (GAM) directly assesses two critical limitations in current grasping systems, notably in three key areas:

- **Precision vs Speed trade off:** High-fidelity reconstruction methods often rely on resource-intensive computation, impeding real-time applicability. In contrast, fast pipelines may sacrifice geometric accuracy. GAM bridges this gap by integrating components that maintain reconstruction fidelity while remaining compatible with CPU-only deployment—achieving a practical balance between speed and precision.
- **Scalability and Generalization:** Many existing systems are overfitted to specific object types or constrained lab settings. By leveraging prompt-driven grounding, zero-shot vision modules, and model-agnostic grasp detection, GAM can generalize to objects with varying sizes, textures, and configurations—ranging between 0.1 and 30 cm. This enhances adaptability across tasks, reducing the need for retraining or dataset-specific tuning.
- **Robot Agnosticism:** Conventional pipelines are often tailored to a single robot configuration or end-effector type. GAM is explicitly designed for **cross-platform deployment** within the Tiago family (Tiago, Tiago++, Tiago-Pro). Its modular design ensures that the same pipeline components can be reused and adapted across different robot embodiments with minimal reconfiguration.

This study deliberately focuses grasping **single objects** in controlled, tabletop environments, with varying object types and spatial configurations (Figures 1.3-3.4a-4.1a). While real-world variability is included in the testing (e.g., sensor noise, occlusions, lighting), the primary objective is to **validate an integrated grasping pipeline**—from perception to execution—under manageable conditions. The system is evaluated across a representative range of objects, and prompts but does not address multi-object grasping or cluttered bin-picking scenarios, dynamic interaction with moving targets, post-grasp manipulation tasks such as handovers or object use. While these challenges are critical, they fall outside the current scope and represent natural extensions for future research.

Building on the objectives defined in Section 1.3, the following functional components represents the research focus:

- **Perception:** Integrating State-of-the-Art (SoTA) segmentation and grounding techniques to extract object masks from RGB-D data with minimal supervision.

- **3D Reconstruction:** Estimating full object geometry from partial depth views, enabling the system to infer occluded surfaces from a single observation.
- **Grasp Pose Detection:** Generating feasible, 6-DoF grasp candidates that respect the spatial constraints of the object, gripper, and surrounding surfaces.
- **Motion Planning & Execution:** Computing and executing safe, collision-free trajectories based on the predicted grasp poses, leveraging structured task planning.
- **Robot Integration:** Deploying and validating the entire system across Tiago platforms.

These components collectively define the system's operational boundaries and functional goals, while ensuring modularity and extensibility for future enhancements. To ensure clarity and consistency throughout this thesis, the following key terms are defined:

- **ROS2** A middleware framework for robotic systems that provides tools, libraries, and communication protocols to distributed, real-time applications. Unlike ROS1, ROS2 uses *Data Distribution Service (DDS)* protocol for reliable, low-latency communication, enabling deterministic performance in dynamic environments. It supports microservices architecture, allowing components (nodes) to be developed and tested independently.
- **MoveIt2:** The state-of-the-art **motion planning framework** for ROS2, designed to simplify complex manipulation tasks. MoveIt 2 extends the capabilities of MoveIt 1 by leveraging ROS2's improved middleware offering **collision-aware** motion planning, taking advantage of algorithms like OMPL (Open Motion Planning Library) for trajectory optimization; **real-time adaptability**: supports dynamic reconfiguration of planning parameters during execution. In this research, MoveIt 2 is used to generate collision-free trajectories for the Tiago's manipulator, ensuring safe and efficient grasp execution.
- **RGB-D Image:** Fig.1.3 illustrates a digital image that contains both standard colour information (RGB), visible in the frame titled *POV*, and depth information (D), the points plotted behind the frame. RGBD cameras can do a pixel-to-pixel merging of RGB data and depth information to deliver both in a single frame, typically captured using a structured light or time-of-flight depth sensor. RGB-D images are critical for scene understanding and 3D reconstruction in robotic perception.

- **Point Cloud:** A discrete set of data points representing the surface geometry of an object or scene in three-dimensional space. Each point is typically defined by spatial coordinates (X, Y, Z) and may also contain additional attributes such as RGB colour values or surface normals.

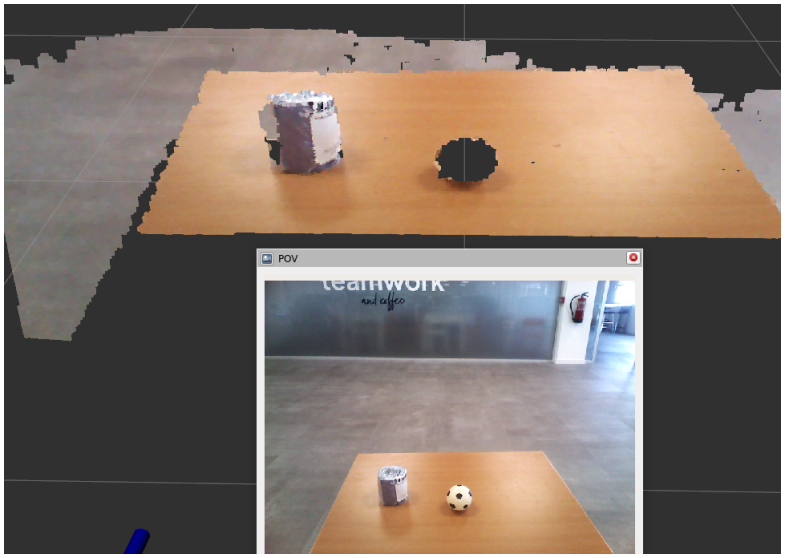


Figure 1.3: Example of RGB-D sensor data. The *POV* frame displays what Tiago sees, above it the depth points (D), aligned with the RGB, are plotted in *RViz*.

- **Diffusion Model:** A generative approach that learns to model complex data distributions by reversing a gradual noise perturbation process. In the context of this study, diffusion models can be used for refining 3D shape reconstruction and improving grasp prediction.
- **Grasping Pose:** Defines the 6 DF pose for which the end-effector consider as a target pose to plan, execute the trajectory in order to grasp the object.

This section establishes the fundamental scope and conceptual framework for the research, ensuring a clear understanding of its objectives, constraints, and technical foundations.

1.5 Thesis Outline

The remainder of this thesis is organized into several chapters that detail the evolution of the research, from literature review and methodology to experimental evaluation and conclusions:

- **Chapter 2: Literature Review:** Reviews the state-of-the-art in robotic grasping, 3D reconstruction, and diffusion-based models. It discusses the advantages and limitations of methods such as *MCC* [63], *PointInfinity* [21], and *Grasp-Pose-Detection (GPD)* [46], and highlights gaps in the current research.
- **Chapter 3: Research Design and Implementation:** Describes the design and implementation of the proposed grasping pipeline. This chapter de-

tails the components of the system—from perception to grasping execution—and explains the rationale behind key design decisions.

- **Chapter 4: Results:** Outlines the experimental framework, including different environments, evaluation metrics, and system configurations. Results are presented, comparing the performance of the proposed pipeline with state-of-the-art approaches under various conditions. Analyses the experimental results, discusses the strengths and limitations of the approach, and explores potential improvements. This chapter also reflects on the trade-offs between accuracy, speed, and robustness observed during testing.
- **Chapter 5: Conclusion and Future Work:** Summarizes the contributions of the research, discusses its impact on the field of robotic grasping, and proposes directions for future research to further enhance the system's capabilities.

2

Literature Review

Robotic grasping is one of the fundamental challenges in robotic manipulation, playing a crucial role in industrial automation, service robotics and assistive technologies. Achieving robust, adaptive, and efficient grasping remains a **non-trivial problem** due to the object variability, environmental clutter and sensor noise. Over the past decades, extensive research has focused on **perception 3D scene representation**, **grasp pose detection**, and **execution strategies** to enhance robotic grasping capabilities.

This chapter reviews the state-of-the-art approaches and methodologies across each of these components, analysing design principles, performance trade-offs, and implementation challenges with a focus on how they inform the **RQs** (Research Questions) posed:

- **RQ1:** *To what extent can prompt-driven perception generalize to real-world object identification without high-performance computing hardware?*
- **RQ2:** *How can hybrid grasp synthesis algorithms balance precision and computational cost for effective manipulation in unstructured environments?*
- **RQ3:** *How can modular motion planning architectures (e.g., MTC, section 2.5.1) improve responsiveness and task completion in dynamic environments like those Tiago operates in?*

These questions emerge from observed gaps in the literature regarding the deployment of integrated robotic grasping systems under real-world constraints. They serve as the basis and task for the proposed Grasp-Anything-Model (GAM) pipeline discussed in the following chapter.

The structure of this chapter is as follows:

- **Section 2.1: Historical Background** – Brief overview of the historical progression during the decades of the Robotic Grasping.
- **Section 2.2: Perception for Robotic Grasping** – Sensor technologies and segmentation techniques essential for detecting graspable objects.
- **Section 2.3: 3D Scene Representation** – Point clouds, voxels, and analytical vs. data-driven approaches for modelling object geometry.
- **Section 2.4: Grasp Pose Detection** – Methods for generating stable 6-DoF grasping configurations.
- **Section 2.5: Motion Planning & Execution** – Trajectory optimization techniques and real-time robot control architectures.
- **Section 2.6: Limitations and Research Gaps** – Identifies gaps in the literature and how this research addresses them.

By synthesizing these areas, this chapter establishes a conceptual framework for the proposed grasping pipeline, emphasizing trade-offs between speed, precision, and generalization in existing methods.

2.1 Historical Background

Robotic grasping has evolved considerably over the past five decades, transitioning from rigid, **rule-based** strategies in the 1970s and 1980s to the **modular, data-driven** pipelines of the 2000s. Early research focused on analytical models grounded in physics and geometry. Classical methods such as force-closure analysis [41] and grasp quality metrics [16] offered mathematically grounded criteria for evaluating grasp stability. While effective in structured environments with known object geometries, these approaches struggled to scale to unstructured, dynamic settings.

The early 2000 marked a major shift with the advent of **RGB-D cameras** which revolutionized robotic grasping by enabling **real-time perception** of depth and colour. Research began integrating **machine learning models** to detect graspable regions directly from visual inputs, moving beyond pure geometry-based heuristics [55]. However, these early learning-based approaches suffered from two critical limitations: reliance on **curated datasets** and **poor generalization** to novel objects.

A crucial milestone came with the release of the **Robotic-Operating-System** (ROS) in 2007-2008, which laid the groundwork for **modular robotic architectures**. Tools like *MoveIt!* (introduced in 2014) further standardized motion planning and integration across different hardware platforms. This modularity became increasingly important as robotic systems began incorporating diverse components such as perception, planning, and control.

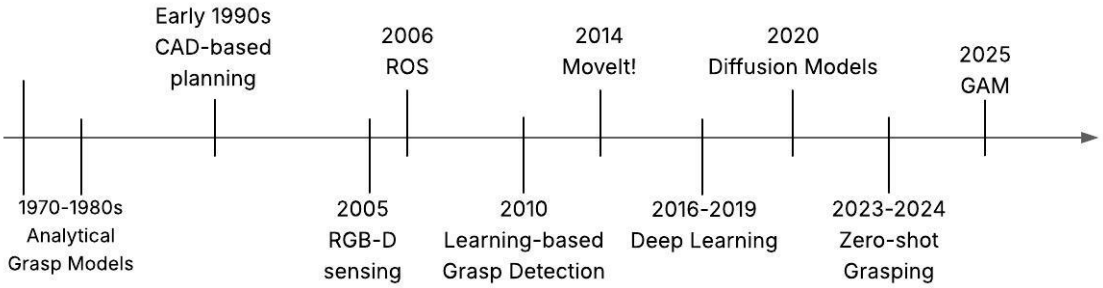


Figure 2.1: Key milestones in the evolution of robotic grasping systems over time.

With the advent of **Dense Neural Networks** (DNNs) and **reinforcement learning (RL)**, data-driven grasp planning saw significant improvements. *Dex-Net* [34] introduced a synthetic grasp dataset with millions of simulated grasp attempts, improving grasp prediction accuracy. Meanwhile, *GraspNet-1Billion* [14] expanded large-scale grasp datasets, enabling 6-DoF grasp planning using DNN models like *GraspNet* [15] and *Contact-GraspNet* [61].

Recent trends emphasize **modular grasping pipelines** that integrate:

- **Perception** modules (e.g., *YOLOe*, *SAM*) for task-aware instance segmentation.
- **3D Reconstruction** methods (e.g., *MCC*) for spatial awareness.
- **Hybrid learning-based grasp synthesis,** balancing adaptability and efficiency.
- **Motion planning frameworks** (e.g., *MTC*) for sequential action planning.

Despite advancements in modular pipelines, a critical trade-off persists between computational complexity and real-time performance. Many deep learning-based methods require significant computational power (e.g. GPUs), making

them impractical for embedded robotic systems. Consequently, hybrid approaches combining physics-based models with data-driven approaches are emerging as a viable solution [25]. While hybrid approaches [25] address computational bottlenecks, they lack modularity—a gap addressed by GAM's ROS2-based pipeline.

The evolution of robotic grasping demonstrates a shift from analytical models to **adaptive**, learning-based pipelines. Despite major advancements, key challenges remain, including handling occlusions, improving real-time efficiency, and enabling grasp generalization to novel objects. We first survey image-based perception (Section 2.2), then 3D scene reconstruction (Section 2.3), grasp synthesis (Section 2.4), how the Planning Scene (Section 2.5) is handled and finally motion planning (Section 2.6). Each section concludes by tying findings to the core research questions (**RQs**), laying the foundation for the system architecture proposed in Chapter 3.

2.2 Perception for Robotic Grasping

Perception forms the foundation of robotic interaction with the environment, enabling a robot to understand, interpret and respond to its surroundings. In the context of service robots (e.g., the Tiago robots), accurate and robust perception is critical for enabling manipulation tasks in dynamic and/or unstructured environments. This stage is inherently challenging due to:

- **Variability** in object geometry and texture: robots must generalize across different shapes, sizes and surface properties.
- **Occlusion** and **cluttered environments**: graspable objects are often partially obscured, necessitating robust segmentation techniques
- **Real-time constraints**: computational efficiency is critical to enable responsive decision-making.

To address these challenges, recent advancements in **vision-language** models and deep learning-based segmentation have significantly enhanced robotic perception capabilities. Various perception models and architectures have emerged to meet these demands, each offering distinct strengths and limitations.

2.2.1 End-to-end Vision Language Model (RT-2)

A significant advancement in perception-based grasping is in *RT-2* (Robotics Transformer 2) model [3]. *RT-2* represents a **vision-language-action** model that seamlessly integrates vision-based perception with high-level reasoning and action generation. Unlike traditional perception models that rely solely on image processing techniques, *RT-2* incorporates **semantic reasoning** from vast internet-scale datasets, enabling the robot to interpret commands in natural language and generate executable motion plans.

RT-2 operates as an **end-to-end** model that translates **RGB input** and **natural language command** into **robot actions**, effectively enabling closed-loop control through high-level reasoning without explicit grasp pose programming.

RT-2 **generalizes to unseen objects** by leveraging web-scale pre-training enabling high-level tasks (e.g. *"Pick up the largest red object and place it next to the green cup"*) through visual-language alignment, without explicit object annotations. The model outputs **continuous action tokens**. However, the model's tokenized action format is rigid, and its reliance on GPU acceleration (~500ms on A100/V100 GPUs), renders it incompatible with CPU-only platforms like Tiago.

This directly informs **RQ1**: While *RT-2* excels in semantic generalization its hardware demands, and fixed output structures limit its deployment in embedded systems. The GAM pipeline addresses this gap by exploring lightweight alternatives better suited for CPU execution.

2.2.2 Segment-Anything-Model

SAM (SAM-v1) [27] is a foundation model that revolutionizes instance segmentation, designed for **promptable segmentation**, **promptable masks**, supporting points, boxes, or text as inputs; capable of generating high-quality object masks. Pre-trained on 1.1B masks (*SA-1B dataset*), *SAM* achieves **zero-shot** transfer to novel objects, meaning that it can identify and isolate objects without task specific fine-tuning; making it ideal for cluttered scenes. While ***SAM-v2*** [50] has recently been released with enhancements targeted toward **video object segmentation and tracking**. Since the scope of this work is confined to **single-frame object segmentation** in static scenes, *SAM (v1)* remains sufficient.

SAM consists of three core components: an **image encoder** which extracts high-level image embeddings; a **prompt encoder** that accepts multiple prompt formats, including points, bounding boxes, and free-form text descriptions; and a **mask decoder** that generates 512x512 masks via cross-attention between image embeddings and prompt vectors.

The mask decoding phase, which can generate around **500 masks** from a single frame enabling exhaustive scene parsing; runs ~5ms on an *A100 GPU* vs ~2s on *Intel I7-13900K CPU*. Furthermore, *SAM*'s text encoder underperforms when prompts lack spatial context (e.g., "*segment the apple*" in cluttered scenes), leading to segmentation failures; also, it struggles with fine structures (e.g., <5px utensil handles) compared to "zoom-in" methods like Mask2Former [7], making it unideal to this study-case.

This connects to **RQ1** by showing that promptable segmentation can operate under minimal supervision. However, its GPU needs again highlight the trade-off between generalization and real-time feasibility. GAM adopts hybrid strategies to mitigate these limitations.

2.2.3 YOLOE

YOLOe (*You Only Look Once everything*) [62] is a recently introduced real-time detection and segmentation framework that extends the capabilities of the YOLO family by supporting **text-prompted**, **visual-prompted**, and **prompt-free** perception within a single unified model architecture.

YOLOe is designed for**open-world perception**, where it can flexibly detect arbitrary objects guided by a natural language or visual hints, or operate without explicit prompting. Includes a feature extraction network, regression and segmentation heads, and a novel object embedding head for handling **open-vocabulary classification**. Designed for real-time robotic applications, it addresses the limitation of closed-set *YOLOv8* variants while maintaining computational efficiency.

While *YOLOe* advances open-world scenarios perception, three key limitations impact its applicability in robotic grasping:

1. **AGPL Licensing:** Derivatives of *YOLOe* inherit the **Affero GPL (AGPL)** license, mandating open-source redistribution of modified code. This **restricts commercial deployment** in proprietary robotic systems, as in the context of PAL Robotics.
2. **Multi-task trade off:** Joint optimization of detection and segmentation reduces performance on frequent object categories, lowering detection **APf** by 0.9% compared to single-task baselines [62].
3. **Vocabulary constraints:** : The built-in 4585-category vocabulary limits generalization to niche industrial objects (e.g., custom 3D-printed tools), unlike LLM-based methods like GenerateU [62] that leverages generative language models.

YOLOe's architectural innovations align with the demand of modular grasping pipelines like GAM: **natural language integration**, resource-constrained deployment, relevant to **RQ1**, and dynamic environment adaptation. To address *YOLOe*'s **mask quality limitations** (e.g., blurred edges on thin structures) and the AGPL licensing limit, GAM integrates *GSAM*, introduced in the following sub-section, for pixel-precise segmentation.

2.2.4 Grounded-Segment-Anything

Grounded-Segment-Anything (GSAM) [52] is a recent SoTA perception framework that addresses one of the most pressing challenges in robotic grasping: **open-vocabulary instance segmentation** under unconstrained, real-world conditions. It achieves this by composing two powerful expert models: ***Grounding DINO*** [33], an open-set **object detector** trained on over 10 million image-text pairs, and ***SAM*** [27], defined in the sub-section 2.2.2.

This aggregate model allows *Grounded-SAM (GSAM)* [52] to segment virtually any object or region referenced through **natural language**: unlike static classifier or fixed detectors, it dynamically adapts to a wide range of objects prompts, enabling human-interpretable interactions in robotic systems. Its grounding mechanism can detect rare or niche objects (e.g., botanical species, industrial components) unseen during training, addressing a major gap in closed-set detectors. The output quality is dependent on prompt specificity. Ambiguous or poorly formulated prompts can lead to suboptimal detection and segmentation.

Grounded-SAM is structured around a two-stage pipeline:

1. **Detection** via ***Grounding DINO:*** Given an RGB input image and a user-defined text prompt (e.g., "*the small green apple*"), *Grounding DINO* performs **open-set object detection**, returning bounding boxes aligned with the semantic reference [33].
2. **Segmentation** via ***SAM:*** These bounding boxes are then passed as spatial prompts to *SAM*, which then generates pixel-accurate instance masks corresponding to the detected regions.

This division of concerns—semantic alignment via *DINO* and special precision via *SAM*—enables *Grounded-SAM* to outperform traditional segmentation pipelines on tasks requiring fine-grained, prompt-specific understanding, facilitating integration into frameworks like ROS2-based systems. It also excels in **zero-shot generalization**, as evidenced by its **48.7 mAP**[1] (mean Average Precision, AP[2]) on the *SegInW* benchmark (Segmentation in the Wild) [67], surpassing methods like *UNINEXT* and *OpenSeeD* [52][69]. These properties align strongly with the goals of modular grasping pipelines like GAM, particularly in **unstructured environments** where novel objects must be identified and manipulated without retraining.

The two-stage nature of the architecture imposes a higher latency (~13s inference on CPU-only systems), which may hinder real-time grasping on resource-constrained platforms. The segmentation accuracy is bounded by the bounding box produced by *DINO*, thus, errors in detection propagates to segmentation.

In the context of this thesis, *Grounded-SAM* excels for **interactive grasping scenarios**. While its computational cost currently does not match the real-time requirements, its accurate semantic-rich output trade-offs the inference speed, thus, inform hybrid strategies in GAM (e.g., offline segmentation for static scenes). This balances flexibility and computational efficiency, informing **RQ1**.

In conclusion, as notable in Table 2.1, perception models vary significantly in **semantic flexibility**, **latency**, and **hardware demands**. *YOLOe* achieves real-time performance but struggles with occlusion, whereas *RT-2*'s zero-shot capa-

[1] **mean Average Precision** (mAP) is the average of AP values across all classes in a dataset. It provides a single metric to evaluate overall a model performance.

[2] **Average Precision** (AP) quantifies the precision-recall trade-off for a single class. It is calculated as the area under the precision-recall curve generated by varying the confidence threshold of predictions. Evaluates how well a model detects the true positives.

Model	Input	Latency	Performance
RT-2	RGB + Text	~500 ms (GPU)	~89% success rate [5]+[1]
SAM	RGB + Points/Boxes	~5 ms (GPU)	44.7 AP [20]
YOLOe	RGB + Text/Visual	102.5 FPS (T4 GPU)	35.9 AP [20]
GSAM	RGB + Text	~14 s (CPU)	48.7 mAP [67]

Table 2.1: Comparison of Perception approaches.

bilities reduce the need for retraining but at the cost of latency, while *GSAM* seems to be suited as it has the best performance/requirements ratio. The GAM pipeline leverages these insights to build a CPU-optimized hybrid system that balances generalization capacity and real-time constraints, directly addressing **RQ1**.

While these models offer reliable object localization, grasping demands a deeper understanding of object geometry. The next section explores methods for reconstructing 3D scenes from RGB-D input to enable effective robotic manipulation.

2.3 3D Scene Representation

Models like *SAM* [27] and *YOLOe* [62], outlined in the previous section, primarily serve the function of perceiving the environment detecting, segmenting and reasoning about objects. However, for successful 3D scene representation and task execution, this perceptual information must be structured and contextualized. Object masks, bounding boxes and labels alone are insufficient; robots require a coherent, up-to-date 3D representation that captures both geometry and semantic. In this section, we explore recent approaches to 3D scene reconstruction, evaluating their suitability for integration into the GAM pipeline, addressing **RQ1** and **RQ2**.

3D Reconstruction is a critical component of robotic grasping, enabling the robot to infer complete object geometry for collision free, stable grips in cluttered environments [22] [65]. However, reconstructing full 3D models from partial sensor data (e.g., single-view RGB-D scan) remains inherently challenging due to occlusions, sensor noise, limited viewpoints and real time computational constraints. These challenges often force robots to compute grasps using incomplete point clouds, leading to suboptimal contact configurations and increased

collision risk. Over the past decade, four dominant paradigms have emerged to tackle these problems:

1. **Volumetric fusion:** Classical methods integrate multiple depth frames into voxel grids using *Truncated Signed Distance Fields* (TSDF) [38], balancing accuracy with computational cost.

2. **Learning based implicit representations** encodes geometry as continuous neural functions (e.g., *DeepSDF*, *Neural Radiance Fields (NeRF)* [45] [35]), enabling high-fidelity completion of partial scans.

3. **Active interactive reconstruction:** Leverages robotic manipulation to reposition objects or sensors, capturing occluded surfaces at the cost of added motion planning.

4. **Neural implicit field methods:** Jointly predict 3D shape and grasp poses in a unified framework enabling near real-time, zero-shot reconstruction [22].

Without explicit shape modelling, robots often compute grasps using incomplete point clouds, leading to suboptimal contact configurations and increased risk of collision [65].

The transition from **flat perception** to **structured scene understanding** is crucial for enabling tasks such as planning, grasping, and interaction. Moreover, real time operation on embedded platforms imposes stringent computational constraints, forcing reconstruction algorithms to balance speed against fidelity. Below, we evaluate four representative approaches in terms of their reconstruction quality, runtime feasibility, and compatibility with real-world RGB-D inputs. These evaluations support the design choices made in the GAM pipeline referring to **RQ2**.

2.3.1 Point-NeRF

Point-NeRF [64] introduces a hybrid paradigm for 3D reconstruction by unifying **Neural Radiance Fields** (NeRFs) with point cloud representations, overcoming the computational inefficiencies and scalability issues of conventional NeRFs [35], that rely on global MLPs to encode scenes. *Point-NeRF* leverages a neural point cloud to model localized radiance fields, enabling rapid initialization via deep multi-view stereo (MVS) networks and efficient per-scene optimization. As a result, it **reduces reconstruction time** by an order of magnitude

(≈ 30×) compared to vanilla NeRF while achieving comparable or superior rendering quality, making it particularly suitable for robotic grasping applications that demand real-time scene understanding [22].

This framework also supports integration with off-the-shelf reconstruction pipelines (e.g., COLMAP [57] [56]) via point pruning/growing, robustly handling noisy or incomplete inputs common in real-world RGB-D scans.

Point-NeRF's main advantages include:

- **Real-Time Feasibility:** Inference at 16k points takes ~2 seconds on an RTX 4090 GPU, with optimized implementations achieving 15 FPS for real-time rendering. This enables rapid scene updates during robotic manipulation [64].
- **High-Fidelity Geometry:** Generates up to 131k-point clouds, capturing fine structures (e.g., thin ropes, texture gradients) critical for collision-aware grasp planning.
- **Robustness to Sparse Inputs:** The point growing mechanism fills occluded regions, reducing grasp failures caused by incomplete geometry. For example, on COLMAP-derived point clouds with 50% sparsity, Point-NeRF achieves **30.18 dB** PSNR[3] (Peak Signal-to-Noise Ratio) after optimization.

Although *Point-NeRF* excels in quality and speed on GPU equipped machines, its reliance on **multi view RGB inputs** and GPU dependency renders it **infeasible** for the Tiago platforms, which capture single RGB-D views. Thus, *Point-NeRF*, falls short of addressing RQ1's goal of generalization from limited perceptual input in real-world robotic systems. Consequently, a single view reconstruction approach is required for integration into the GAM pipeline.

2.3.2 PointInfinity

PointInfinity [21] advances diffusion-based 3D reconstruction by balancing computational efficiency and high-resolution output, critical for robotic grasping. Unlike traditional transformers, which scale quadratically with point count, *PointInfinity* employs a **two-stream** transformer architecture that decouples **surface modelling** from **point generation**. At its core, a fixed-size latent representation (256

[3] **(Peak Signal-to-Noise Ratio)** PSNR is defined as the ratio between the power of the signal and the power of the noise in an image, indicating the quality of the image with higher values representing better quality by reducing noise.

tokens) captures the underlying 3D shape, while a variable-resolution data stream generates point cloud through lightweight read/write cross-attention modules. This design enables training at low resolution (e.g., 1,024 points) while seamlessly generalizing to high-resolution outputs (up to 131k points) during inference, achieving **linear computational scaling** with point count [21].

Increasing the number of points during inference improves fidelity by enhancing information flow between latent and data streams, for example Chamfer Distance[4] drops from 0.227 at 1k points (CD@1k[5]) to 0.181 at 8k points. Its ultra-high-resolution output (e.g., ~181k points) enable direct surface reconstruction via marching cubes, crucial for collision-aware grasp planning [21].

PointInfinity addresses key limitations of prior methods in grasping pipelines:

- **Denser Geometry:** generates 30x more points than *Point-E* [42], capturing fine details (e.g., thin structures, texture, details) that collision risk in cluttered environments.
- **Real-Time feasibility:** Inference at 16k points takes ~2 seconds on an RTX 4090 GPU, making it suitable for dynamic GPU-equipped robotic tasks.
- **Robustness:** trained on noisy CO3D-v2 data [51], it generalizes to real-world RGB-D scans without requiring multi-view inputs.

However, advising **RQ1**, *PointInfinity* is still GPU-dependent and lacks real-time capabilities under CPU-only setups, underscoring the GAM pipeline's need for CPU-feasible alternatives. This supports **RQ2** by demonstrating the value of scalable output resolution in 3D reconstruction.

2.3.3 Deep SDF

DeepSDF, introduced by Park et al. (2019) [45], presents a novel approach to 3D shape representation by learning continuous ***sdf! (sdf!)*** via neural networks. Unlike classical discretized or parametric methods, *DeepSDF* implicitly models surfaces as zero-level sets of a learned volumetric field, enabling high-fidelity reconstruction, interpolation, and completion of complex topologies.

[4]The **Chamfer Distance** (CD) quantifies the difference between two point clouds by averaging the nearest-neighbour distances between their points.
[5]Chamfer Distance evaluated on a set of **1k points**.

This approach achieves SoTA reconstruction fidelity, outperforming voxel-based (Octree Generating Networks) and mesh-based (AtlasNet [19]) methods in CD and Earth Mover's Distance (*EMD*). The continuous *SDF* formulation intrinsically enforces smooth, closed surfaces, avoiding topological rigidity of meshes and sparsity of point clouds. The method robustly completes partial inputs (e.g., single-view depth scans) by optimizing latent codes against observed *SDF* samples, informing **RQ1**; quantitative results show superior completion accuracy over *3D-EPN* (3D Encoder-Predictor networks), with only linear degradation under increasing noise. A 256-dimensional latent code suffices to represent entire shape classes (e.g., chairs, planes), yielding a model size of **7.4MB** an order of magnitude smaller than typical voxel grids.

However, inference requires iterative latent code optimization, resulting in a median inference latency of **9.72s**, over **900x** slower than AtlasNet (0.01s) [19]: this precludes real-time applications. Moreover, *DeepSDF*'s dependency on watertight meshes limits applicability to synthetic datasets (e.g., ShapeNet [4]), referring to **RQ2**; real-world RGB-D data must undergo complex preprocessing to generate SDF samples, contrasting with methods that process raw inputs directly. Finally, its auto-decoder architecture complicates integration with GAM frameworks that rely on adversarial objectives to enforce realism or diversity in generated outputs. Its reliance on precomputed *SDF* samples likewise introduces complexity and latency incompatible with direct sensor-data processing as other approaches grant (e.g.*PointInfinity, MCC*), as noted in Table 2.2.

2.3.4 MCC

Multiview Compressive Coding (MCC), proposed by Wu et al. (2023)[63], introduces a scalable, category-agnostic framework for single-view 3D reconstruction of both objects and scenes. By leveraging transformer architecture and large-scale training on diverse RGB-D video data, *MCC* compresses appearance and geometry into a 3D-aware latent representation, enabling high-fidelity reconstruction from partial observations.

MCC processes RGB-D inputs to predict occupancy and colour for query 3D points. The encoder fuses image and unprojected depth points via dual ViT-based towers while the decoder engages masked attention to avoid cross-query

dependencies Supervision derives from noisy input clouds generated via Structure-from-Motion (SfM) from posed RGB-D frames, eliminating reliance on CAD models, SDFs or precomputed meshes, thus, improving generalization. The model learns to complete occluded regions by reasoning across Multiview consistency. Training on six datasets (e.g., *CO3D* [51], *Hypersim* [54], *DALL-E 2* generations [48]) demonstrates robustness to domain shifts including synthetic-to-real transfer.

MCC demonstrates three key strengths:

- **Generalization:** Achieves zero-shot reconstruction on in-the-wild iPhone captures, AI-generated images, and novel scenes, outperforming NeRF-based methods and explicit designs like *PoinTr* [66].
- **Efficiency:** The lightweight decoder allows dynamic resolution outputs without re-encoding inputs, achieving **56.7 F1-score** on CO3D-v2 and **0.234** CD on CO3D (CD@1k) [51].
- **Noise Robustness:** Correct imperfect depth inputs and scales linearly with dataset size, showing improvements in reconstruction quality

Performance degrades with unreliable depth estimates (e.g., off-the-shelf monocular depth predictors on *ImageNet*) and it struggles with high-frequency details (e.g., text on objects, finer details, texture) due to single-view ambiguity. Unlike *DeepSDF*, which requires watertight meshes for training [45], *MCC* processes raw RGB-D inputs directly, bypassing *CAD* model dependencies, enabling direct integration with Tiago's sensor data.

MCC advances single-view 3D reconstruction by unifying object and scene understanding under a scalable, transformer-based framework. Its ability to generalize across domains and correct noisy inputs makes this approach suited for the GAM pipeline, directly offering a practical solution to both **RQ1** and **RQ2**. For integration into real-time application, future work must address inference latency. But, since the scope of the reconstruction is to let the robot understand the overall shape of the object, finer details (e.g., texture, text on the surface, etc.) are deemed unnecessary. Trading off the granularity that *MCC* outputs the reconstructed point cloud with the inference latency, it is possible to infer on CPU with ~8s on intel i7 CPU to generate a reconstructed object with ~8k points.

Model	Input	Training Resolution	PSNR	CD @1k
PointInfinity	RGB-D	1,024 points	14.31	0.179 [51]
Point-NeRF	Multi-view RGB	Multi-view images	~33.31 [36]	~0.13 [8]
DeepSDF	*SDF* samples	Marching Cubes at 512^3	–	0.204 [4]
MCC	Single RGB-D	RGB-D images	14.03	0.234 [51]

Table 2.2: Comparison of 3D reconstruction models.

In summary, as shown in Table 2.2 3D scene reconstruction methods vary significantly in their trade-offs between fidelity, computational efficiency, and real-time applicability. Classical representations such as point clouds and *SDFs* offer geometric precision and established analytical tools but often fall short in dynamic or noisy environments. Data-driven techniques like *Point-NeRF* and *DeepSDF* enable dense reconstructions and latent encodings of shape but impose high computational demands, limiting their deployment in embedded robotic systems. Hybrid methods—exemplified by *Multiview Compressive Coding (MCC)*—represent a promising middle ground by integrating learning-based regularization with traditional fusion algorithms. These methods allow for scalable scene understanding while preserving interpretability and modularity, both of which are central to the pipeline proposed in this work.

With the 3D structure of the object established, the next step is to generate candidate grasp configurations that are both feasible and robust under the conditions defined by the GAM pipeline. The following section evaluates methods for generating stable 6-DoF grasps under real-world constraints.

2.4 Grasping Pose Detection

Grasping pose generation is the critical bridge between 3D shape understanding and physical execution, translating reconstructed object geometry into candidate end-effector configurations that a robot can feasibly reach and apply stable forces with [28]. This stage must account for the full 6-DoF of the gripper (three translational, three rotational) often inferred from a single-view or multi-view depth data [28]. Methods for grasp pose generation broadly split into analytic and data-driven paradigms, reflecting a trade-off between interpretability and empirical performance [68].

Analytic approaches compute grasps by optimizing force-closure or form-closure criteria over **known object models**, leveraging classical grasp quality metrics to guarantee theoretical stability [53]. In contrast, data-driven methods either **sample** grasp candidates densely in the workspace, then rank them using learned quality predictors (e.g., *GPD, Dex-Net*) [29] or **directly regress** a small set of high-quality grasps using DNNs [30].

Building on data-driven methods, **RL** frameworks, such as *QT-Opt* [24], improve robustness in unstructured environments by iteratively refining grasps through simulated trial-and-error. **Exemplar-based** and **generative** models leverage shape priors, either retrieved from databases or synthesized via autoencoders, to propose grasps consistent with object semantics and affordances. Evaluation spans success rates in simulation/hardware trials and robustness scores from tool like *GraspIt!* [37] or *GQ-CNN* [37], quantifying a candidate's resistance to pose and friction uncertainties. Achieving **real-time** performance on embedded platforms further imposes stringent limits (<*500ms*) on candidate count and inference latency.

To address these challenges, this section critically reviews representative categories of grasp pose generation methods, in line with **RQ2**: analytic optimization, sampling-and-ranking, direct regression, evaluating them in terms of **computational efficiency, generalization to novel objects**, and **compatibility** with the GAM pipeline.

2.4.1 Grasp it Like a Pro 2.0

Palleschi et al. (2023) present *Grasp it Like a pro 2.0 (GLP)* [44], a data-driven grasp planning algorithm that combines **human demonstrations** and **basic shape decomposition** to generate 6-DoF grasps for unknown objects. The authors validated *GLP 2.0* on both a compliant underactuated SoftHand (94% success over 150 grasps on 30 objects) and a rigid two-finger Franka Emika Hand (85% success over 80 grasps on 16 objects). *GLP 2.0* operates in three stages:

1. **Shape decomposition:** An input RGB-D point cloud is segmented into minimum-volume bounding boxes (**MVBBs**) using a logarithmic split criterion, prioritizing regions of high point density and centroid proximity to approximate handle-like geometries [add ref]. This balances granularity by decomposing small objects into fewer boxes and larger/complex shapes into finer approximations (**RQ2**, section 2.6)[44].

2. **Grasp proposal:** A *decision tree regressor* (*DTR*) trained on human demonstrations of grasps performed on cuboid primitives, maps each MVBB to a set of 6-DoF gripper poses and associated interaction wrenches. The *DTR* generalizes to novel box dimensions, enabling grasp synthesis without explicit object models [44].

3. **Quality ranking:** Each grasp candidate is scored by **box score**, the density and proximity to point-cloud centroid; **wrench score**, normalized predicted force/torque magnitude; **alignment score**, the geometric alignment and hand-opening constraints; **collision score**, that is the volumetric collision checks for both entire gripper and finger closure regions [44]

The **top-scoring**, reachable grasp is selected and executed via an inverse-kinematics solver.

GLP 2.0, despite learned from **only 648 demonstrations**, it generalizes across very different shapes and two gripper types, integrating geometric, force-based, and collision criteria into a unified score. Quality hinges on accurate segmentation and **MVBB fitting**; very noisy or sparse scans could degrade performance. Moreover, the model inherits biases of demonstration set, potentially reducing grasp diversity for highly irregular shapes.

GLP 2.0 relates to **RQ2** by demonstrating the feasibility of low-data, high-generalization grasp policies. Once the authors release their code, integrating this approach into the pipeline could enhance grasp robustness while preserving modularity. Until then, its hybrid data-driven and analytical design offers a template for balancing geometric reasoning with learned priors in custom implementations.

2.4.2 Contact-GraspNet

Contact-GraspNet [61] pioneers an **end-to-end** 6-DoF grasp proposal network that directly processes a scene's depth point cloud to predict antipodal **parallel-jaw** grasps in cluttered environments. By treating **visible surface points** as potential contact roots, it simplifies 6-DoF grasps onto their contact points in the observed cloud reducing the grasp representation to a **4-DoF** parameterization—contact point rotation angle and gripper width—enabling efficient learning and inference. Training relies on the *ACRONYM* dataset's 17 million simulated grasps across 8k+ objects [11], which are projected onto rendered point clouds. Sur-

face points within *5mm* of a ground-truth contacts are assigned grasp poses and widths via nearest-neighbour association.

At inference latency, Contact-GraspNet processes either the full scene or a localized region (e.g., 30 cm^3 around a target object) in <**0.3**s on a NVIDIA V100 GPU, making it suitable for reactive, closed-loop grasping [61]. Grasps are filtered by confidence thresholds, kinematic reachability, and collision checks using voxelized occupancy grids, and the highest-scoring proposal is executed without requiring perfect segmentation [61]. In real-robot trials using a Franka Panda and 51 unseen objects arranged in nine cluttered scenes, the method achieves a **90.2%** overall grasp success rate and an **84.3% first-attempt rate**, outperforming prior 6-DoF pipelines (62.7% and 80.4% respectively) while reducing the average number of attempts per object to **1.16** [61].

Despite its many advances, *Contact-GraspNet* is currently tailored to **parallel-jaw grippers**, and its reliance on visible surface points means it may struggle with heavily occluded contacts. Moreover, while noise injection during training aids transfer, extremely sparse or noisy sensor data might still impair contact confidence predictions. Nevertheless, by unifying grasp generation, collision reasoning, and contact visibility into a single, fast network, *Contact-GraspNet* represents a significant step toward reliable, reactive grasping in unstructured, cluttered environments.

This informs both **RQ1** and **RQ2** by emphasizing that learning-based models can handle occlusion robustly. However, their compute demands remain a bottleneck, reinforcing GAM's need for scalable grasp policies compatible with limited hardware.

2.4.3 AnyGrasp

AnyGrasp [13] advances robotic grasping by unifying spatial and temporal perception into a single framework, enabling parallel grippers to achieve human-like robustness and continuity in unstructured environments. The system operates through two core modules:

1. **Spatial Grasp Synthesis:** A geometry processing module ingests a partial point cloud and predicts **dense 7-DoF grasp poses** (position, orientation

and gripper width) in *100ms* via a single forward pass. Unlike sampling-based methods, this module evaluates grasps holistically by analysing global geometric cues, implicitly learning collision avoidance and stability metrics. Key innovations include the **stability score**, which predicts the normalized distance from the gripper plane to the object's estimated Center of Mass (CoM), mimicking human intuition to prioritize balanced grasps; and **collision awareness** that discards grasps lacking space for gripper pre-shaping by integrating implicit obstacle detection into the scoring process.

2. **Temporal Consistency:** A temporal association module that enables smooth tracking of dynamic objects by minimizing the distance between grasps in the object's coordinate system.

Trained on **144 objects across 268 real-world scenes**, *AnyGrasp* achieves a 93.3% success rate on **300 unseen objects**—matching human performance under open-loop conditions.

While its monolithic design and GPU dependency limit its direct adoption in resource-constrained pipelines, linked to **RQ1**. *AnyGrasp* highlights the opportunity and challenge posed by dynamic scenes, related to **RQ2**, but GAM focuses on single-frame inference for real-time viability.

The authors' pending code release of their grasp library could enable future integration, particularly for **dynamic grasping** tasks requiring real-time pose tracking. Until then, its emphasis on real-world training and dense spatiotemporal supervision offers valuable insights for developing robust, human-inspired grasp planners.

2.4.4 GPD

Grasp Pose Detection proposed by ten Pas et al. [46] represents a paradigm shift in robotic grasping by directly localizing 6-DoF grasping poses in 3D point clouds, without requiring prior object segmentation or CAD models. Its primary contributions include **a novel grasp hypothesis generation method**, a multi-view descriptor incorporating surface normals and occlusion data, and a category-aware classification strategy. *GPD* operates through two primary stages:

1. **Grasp candidate Sampling:** *GPD* generates grasp candidates by analyzing local surface geometry (via *Darboux* frames) [46] and principal curvature axes. This avoids reliance on object segmentation enabling detection in dense clutter. A grid search over orientation (e.g., Φ-axis rotations)

ensures diverse candidate poses, optimized for **parallel-jaw** grippers by aligning with minor principal curvature axis [46].

2. **CNN-based Classification:** A multi-view descriptor encodes geometric and occlusion data from three orthogonal planes (x-y, x-z, y-z), combining surface normals, observed voxels, and unobserved regions into a 15-channel input for a *LeNet-style CNN* [46].

Unlike template-based methods (e.g., Herzog et al.), *GPD* samples candidates on any visible surface, improving generalizability [17]. The method's modular architecture, spanning **sampling**, **encoding**, **classification** and **selection**, aligns well with ROS2's design principles, particularly its emphasis on modularity and real-time performance: it has been used also in the **ROS2 MoveIt Deep Grasps** framework.

GPD answers **RQ2** demonstrating practical runtime performance, critical for real-world deployment. Generating **1,000** grasp candidates requires **0.8–1.7** seconds for sampling and **0.3–6.2** seconds for **classification**, depending on the descriptor type. The 3-channel descriptor reduces inference speed by **x4** compared to the 15-channel variant (0.3–0.4 seconds vs. 4.3–6.2 seconds for 1k candidates) with only a marginal accuracy drop (<5%) [46].

Given *GPD*'s qualities it is considered and implemented in the GAM pipeline.

Model	Input	Training Data	Gripper Type	Success Rate
GLP 2.0	RGB-D cloud	648 human grasps	Soft + rigid	94% + 85%
Contact-GraspNet	Point cloud	*ACRONYM* [11]	Parallel	90.2%
AnyGrasp	Partial cloud	Real-world data	Parallel	93.3%
GPD	Point cloud	300k grasps	Parallel	93%

Table 2.3: Comparison of grasp pose detection methods.

Grasp pose detection remains a pivotal component of the robotic manipulation pipeline, bridging the gap between abstract object understanding and executable action. The reviewed approaches, evidenced in Table 2.3, highlight a broad methodological spectrum—from **purely geometric algorithms** (e.g., *GPD*), which leverage local curvature and surface normals, to **deep learning-based models** such as *Contact-GraspNet* and *AnyGrasp*, which excel in cluttered, real-world environments. Hybrid systems like *GLP 2.0* further demonstrate the power

of combining **human priors** and **analytical features** within data-driven frameworks, offering generalization with minimal data. However, a recurring limitation across these models lies in their **hardware specificity** (e.g., parallel-jaw bias), **GPU dependency**, and **difficulty adapting to novel or irregular geometries**. For this thesis, the modular evaluation of these grasping strategies informs a pipeline capable of supporting both computationally intensive learning-based methods and lightweight, geometry-aware alternatives. This flexibility ensures robust performance across varied object categories and robotic hands within the real-time constraints of the Tiago platform.

After generating candidate grasps, the system must execute them via robust motion planning and trajectory control. The following section reviews contemporary strategies in task-based planning.

2.5 Trajectory Planning & Control

Effective trajectory planning is crucial for robotic manipulation, especially in dynamic environments where precision and adaptability are imperative. This component directly supports **RQ3**, which examines how motion planning architectures improve responsiveness in real-world scenarios. This section explores various motion planning frameworks and algorithms, focusing on their applicability to complex tasks and real-time execution.

2.5.1 MoveIt2 & MTC

MoveIt2 is a widely adopted motion planning framework in the ROS2 ecosystem, offering advanced tools for kinematics, collision checking and trajectory generation. Central to its architecture is the Planning Scene, a dynamic and continuously updated model of the robot's environment, including obstacles and object states. This real-time environmental model ensures not only geometric feasibility but also situational awareness necessary for reliable manipulation.

MoveIt Task Constructor (MTC) [47] [18] builds upon MoveIt2 and introduces a modular and hierarchical abstraction for defining and solving complex manipulation tasks. Rather than relying on monolithic trajectory planners, *MTC*

decomposes high-level goals, specified as **Tasks**, into a sequence of interdependent subtasks handled by specialized **Stages** responsible for a distinct subproblem. This approach inherently supports the modularity required by **RQ3**. These Stages can be flexibly arranged respecting directional data flow, to form both linear and branching task pipelines. The order in which stages can be placed is restricted by the direction in which results are passed. These stages, visualized in Fig.2.2, can be categorized into the following functional types:

- **Generators:** Initiate planning by producing potential start states. Compute their results independently of their neighbour stages and pass them in both directions, backwards and forwards. An example is an Inverse Kinematics[6] (IK) sampler, which kinematics solutions used by adjacent motion stages [47].
- **Propagators:** Extend partial solutions by applying motion primitives. They receive input from one neighbour stage, solve a subproblem and then propagate their result on the opposite site. An example is a stage that computes a Cartesian path based on either a start or a goal state [47].
- **Connectors:** Bridge disjointed solution segments without further propagation, such as free-motion plans connecting two static poses [47].

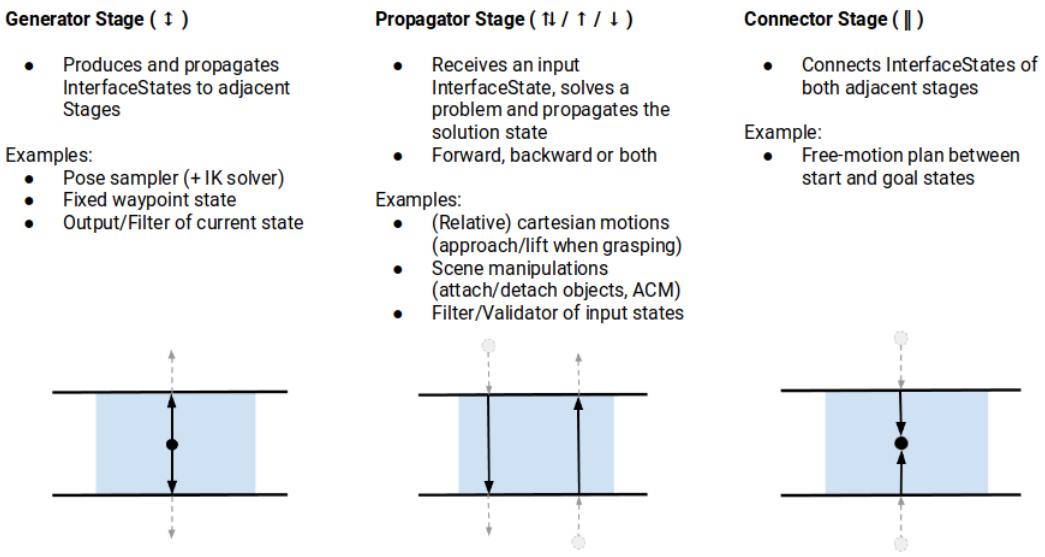


Figure 2.2: Main *MTC* Stages [47].

[6]**Inverse kinematics** (IK) is the process of determining the joint angles required for a robotic arm or other articulated system to reach a specific target position and orientation in space (6-DoF). It essentially works backward from the desired end-effector position to find the corresponding joint configuration.

There are different hierarchy types allowing to encapsulate inferior stages. Stages without subordinate stages are called **primitive stages**, higher-level stages are called **container stages**. There are three container types:

- **Wrappers:** Modify or filter results from other stages. For example, an IK wrapper may constrain solutions to a given pose [47].
- **Serial Container:** Hold a sequence of subordinate stages and only consider end-to-end solutions as results, such as a pick-and-place routine [47].
- **Parallel Containers:** Manage multiple alternatives or fallback solvers, useful for ambiguous tasks or recovery strategies [47].

This stage-based design, rooted in **object-oriented inheritance**, supports flexible state transitions, adaptive task sequencing, and parameterized motion planning.

MTC leverages the Planning Scene to validate geometric and semantic consistency across stages. This integration of perception and planning aligns closely with the architectural philosophy of GAM.

MTC's **modular** architecture, simplifies debugging and testing, its **flexibility** supports complex task definition, including conditional logic and fallback strategies, which are challenging to implement with monolithic planners; and its **transparency**, due to its detailed insights into each planning stage, facilitates better understanding and optimization of the planning process. *MTC* seamlessly incorporates various **planning algorithms** (e.g. *OMPL* [60], *CHOMP* [49], *STOMP* [23], etc.), within its stages, enabling hybrid planning strategies. These features make *MTC* particularly suited for complex manipulation tasks requiring adaptability and precision, connected to **RQ3**.

Beyond motion-level planning, several task-level planning frameworks have been proposed in robotics, **including Hierarchical Task Networks (HTNs)**, **Behaviour Trees**, and the **Flexible Behaviour Engine (FlexBE)** [71]. While these systems provide abstract sequencing and reactive control for complex behaviours, they typically lack integrated motion planning capabilities or require custom interfacing with trajectory solvers. In contrast, the *MTC* natively integrates task decomposition with motion feasibility, offering a unified framework for both **symbolic sequencing and continuous trajectory generation**. While BTs excel

at high-level logic, they often lack direct access to robot state or motion feasibility. *MTC* uniquely enables closed-loop, feedback-aware execution within a single unified planning system, directly contributing to **RQ3**'s focus on real-time responsiveness.

2.5.2 Motion Planning Algorithms

Several motion planning algorithms are integrated within *MoveIt2*, each with distinct characteristics:

- **OMPL** *(Open Motion Planning Library):* Offers a suite of sampling-based planners like RRT [59] and PRM [26]. These planners are probabilistically complete and efficient in high-dimensional spaces but may produce suboptimal or bumpy paths without post-processing [60].
- **CHOMP** *(Covariant Hamiltonian Optimization for Motion Planning):* A gradient-based optimizer that refined an initial trajectory to minimize a cost function enhancing smoothness and **collision avoidance**. While effective in generating smooth paths, *CHOMP* can be sensitive to local minima and requires careful parameter tuning [49].
- **STOMP** *(Stochastic Trajectory Optimization for Motion Planning):* Utilizes stochastic sampling to iteratively improve trajectories, offering robustness against local minima and the ability to handle non-differentiable constraints. *STOMP* often produces smoother trajectories than *CHOMP* and is less sensitive to initial conditions [23].

These methods offer powerful motion planning foundations but generally operate in isolation and lack task-level abstraction. They also require manual tuning and cannot easily accommodate dynamic replanning or conditional logic. In contrast, *MTC* supports these advanced behaviours **natively**, reinforcing the rationale for its adoption in GAM.

Thus, this section supports **RQ3** by evaluating how task-aware modular planning outperforms traditional algorithms in dynamic, real-world scenarios.

2.6 Limitations of Existing Work

The preceding sections have surveyed the state-of-the-art across the primary components of robotic grasping: Perception, 3D Scene Representation, Grasp Synthesis and trajectory planning. While each area has achieved impressive

progress, most existing solutions fall short when it comes to integrated deployment in real-world service robots like the Tiagos. These limitations underscore the **integration gap** between high-performing modules and deployable systems, emphasizing the need for GAM's modular, hardware-aware architecture.

High-performing perception models, such as RT-2 [3] and YOLOe [62] (Section 2.2) while accurate and flexible, rely heavily on GPU acceleration, limiting their use on CPU-only systems, addressing **RQ1**. Grasp synthesis methods, while successful in simulation, often fail to generalize to cluttered or unseen scenes, especially under sensory noise, linked to **RQ2**. Conventional motion planning strategies are monolithic frameworks and inflexible, limiting task-level adaptation and environmental feedback integration, related to **RQ3**.

These limitations reveal a broader issue: a lack of modular, real-time, end-to-end architectures capable of operating under hardware and situational constraints. This integration gap, despite strong performance of individual components, justifies the development of the **GAM pipeline**.

This thesis proposes a solution by introducing a ROS2-native grasping architecture:

- **Prompt-based perception** with ***Grounded-SAM*** [52] enables flexible, interpretable object segmentation under CPU constraints (RQ1).
- **3D Reconstruction** using ***MCC*** [63] facilitates dense, task-relevant spatial awareness with single-view RGB-D inputs (RQ2).
- **Hybrid Grasp pose detection** taking advantage of ***GPD*** [46], combining data-driven and analytical criteria, improving generalizability (RQ2).
- **Modular trajectory planning** via ***MTC*** [18] enables responsive, stage-based control (RQ3).

The theoretical framework guiding this study assumes that effective robotic manipulation in open-world settings must emerge from **tight coupling** between perception, 3D scene representation, and planning, while remaining modular and adaptable. Each module is designed as **swappable**, **testable unit** within a ROS2 architecture, allowing for flexibility in experimentation and iterative improvements.

Thus, this study advances to the field by bridging the gap between high-performance academic models and real-world robotic deployment. Chapter 3 presents the architectural rationale, software integration, and methodological choices underlying each module, laying the foundation for experimental evaluation and performance analysis in subsequent sections.

3

Research Design and Implementation

This chapter presents the research design and implementation of the Grasp-Anything-Model (GAM) pipeline, a **modular architecture** addressing the integration gap (section 2.6) between high-performance academic models and real-world robotic deployment. Building upon the limitations identified in Chapter 2—such as GPU dependency in perception, rigid motion planning, and poor generalization under hardware constraints—this work integrates five sequential stages: Perception, 3D Reconstruction, Grasp Pose Detection, Motion Planning and Execution.

Each stage employs SoTA methods optimized for the Tiago platforms' computational limits, directly addressing RQ1 (**generalization** to real-world perception), RQ2 (**precision-efficiency trade-off**) and RQ3 (**responsiveness**).

Section 3.1 details the architectural synergy between ROS2 and BTs, showcasing how the GAM pipeline balances modularity, reactivity and inference within a unified control structure. Subsequent sections outline each pipeline stage, explaining how theoretical insights from Chapter 2 are translated into concrete implementations. The chapter concludes with a high-level summary setting the stage for experimental validation presented in Chapter 4.

3.1 System Architecture & Control Strategy

The GAM pipeline is designed around the principles of modularity, scalability, and real-time reactivity to suit the constraints of the Tiago robots while meeting the functional requirements identified in Chapter 2. This section provides a top-level view of the system's architecture and execution logic, clarifying how BTs and ROS2 integrate to form the backbone of the GAM orchestration strategy.

The architecture is structured as a **distributed ROS2-based** system, where each core pipeline stage (Perception, 3D Reconstruction, Grasp Pose Detection, Motion Planning, Execution) is implemented as a separate ROS2 service. These services, along with the main action server, remain persistently active after being launched via a centralized ROS2 launch file. This setup ensures that all required models and dependencies are **pre-loaded** and accessible at runtime, minimizing latency during inference.

The GAM pipeline adopts **ROS2 actions** and services in a complementary design. ROS2 actions were selected for orchestrating the overall grasping task due to their inherent support for **asynchronous feedback**, enabling the pipeline to monitor execution progress (e.g., segmentation completion, grasp synthesis status, etc..) and adaptively respond to dynamic environmental changes. Conversely, **ROS2 services** were employed for individual modules (e.g., Perception, 3D Reconstruction, etc..) to enforce deterministic, request-response interactions. This design ensures that each BT node—implemented as a service client—receives a single, atomic response (e.g., a segmentation mask or reconstructed point cloud) without intermediate feedback overhead, aligning with the stateless nature of module-level inference. By decoupling long-term task monitoring (actions) from stateless, atomic computations (services), this hybrid approach enhances system robustness, minimizes latency, and preserves modularity, enabling seamless integration of heterogeneous components. Moreover, ROS2 **services** were **prioritized** over **topics** for their synchronous request-response paradigm, ensuring atomicity in module-level computations (e.g., segmentation, reconstruction) and avoiding race conditions inherent in pub-sub systems.

As suggested by the Fig 3.1, the execution of the pipeline is initiated when a ROS2 action client sends a goal to the GAM action server. The goal, a natural lan-

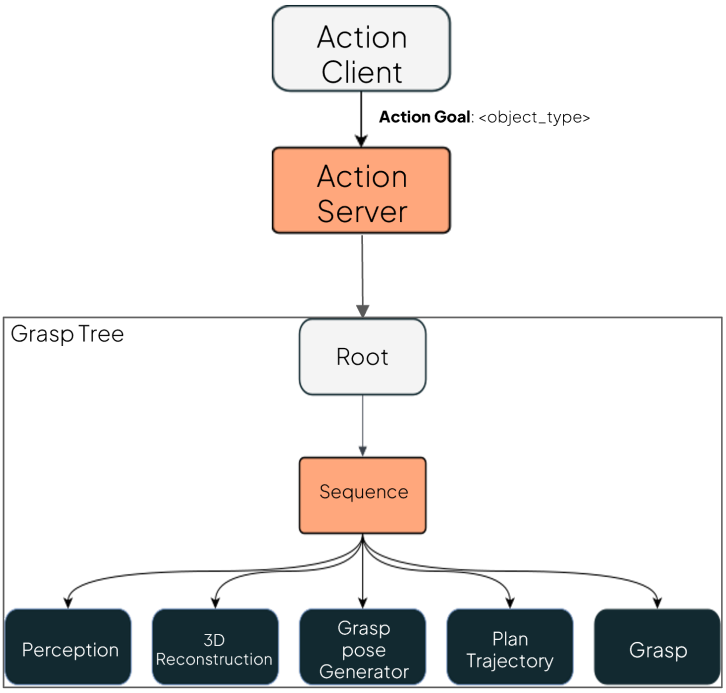


Figure 3.1: ROS2 - BT interaction. The `Action Client` sends the `Action goal`, specifying the *object* to grasp, to the `Action Server`, which starts to *tick* the BT in `Sequence`.

guage prompt (e.g. *"the blue mug"*), is stored on a **centralized BT blackboard**[1], which facilitates global state sharing among the pipeline stages. This shared memory structure allows downstream modules to access contextual information produced by upstream nodes, such as the detected object's mask (from the perception stage) as input to the reconstruction module.

BTs serve as the high-level decision-making framework. Once the action server receives the goal, it triggers a recursive **tick** on the BT. Each BT node corresponds to a **discrete task** within the pipeline and is implemented as ROS2 service client as shown in Fig 3.2. When ticked, the node sends a request to its associated ROS2 service server, passing any required input data retrieved from the blackboard and/or the sensors (e.g., the RGB image, or depth). The server processes the **request** (e.g., runs inference on a perception or reconstructor model) and **returns the service response**, which is subsequently written back to the black-

[1]The **blackboard** acts as a shared workspace, comparable to a collaborative whiteboard, where modules read/write variables (e.g., object masks, grasp poses). It is a simple **key/value storage** shared by all the nodes of the BT.

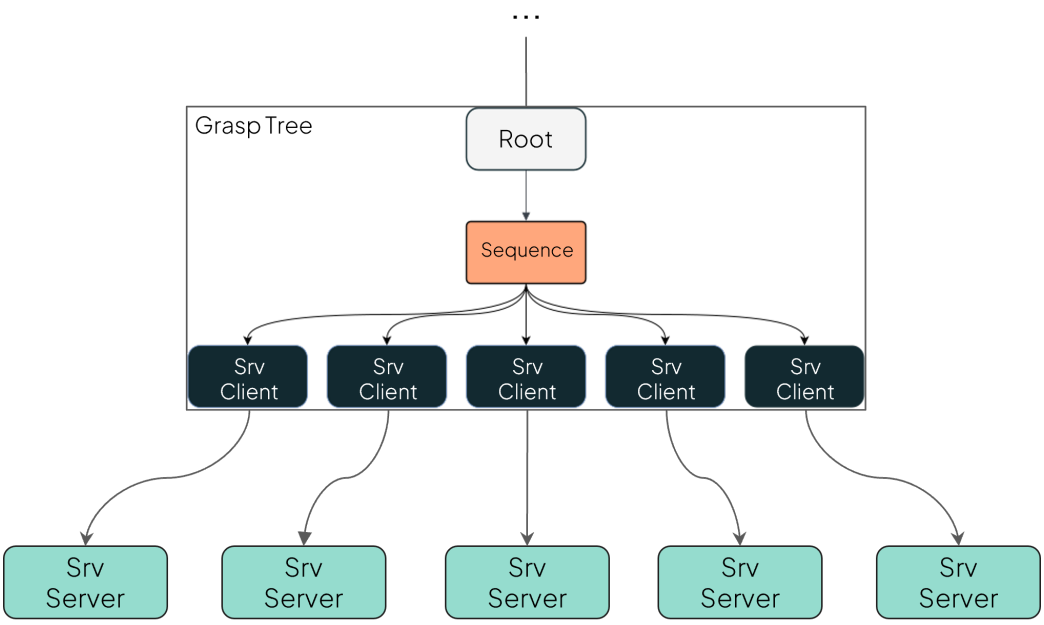


Figure 3.2: BT Nodes implementation as ROS2 service servers.

board, eventually after some post-processing. This data-driven flow ensures tight coupling between the system's perceptual and planning components while maintaining a clear **separation of concerns**.

A key strength of this architecture lies in its **reusability** and **fault tolerance**. Because each node in the tree is both modular and state-aware: failures in one stage (e.g., poor segmentation or grasp detection) can be managed with fallback behaviours or retries, rather than requiring full pipeline termination. Furthermore, BTs allow **asynchronous ticking** and prioritization, enabling potential future extensions where real-time feedback or dynamic replanning is required. For example, for the prompt "*the blue mug*", the Perception node invokes *Grounded-SAM* to generate a binary mask, which is passed to 3D Reconstruction. *MCC* then converts the mask into a ~1k point cloud reconstructing the *blue mug*, enabling *GPD* to propose reliable grasps. If, for example, the grasp synthesis fails, the *GPD* node will be re-executed. Three consecutive failures trigger the failure of the pipeline.

In the following sub-sections will elaborate on the implementation-specific aspects of the ROS2 and BT integration, focusing on the launch configuration, service abstraction, and runtime orchestration.

3.1.1 Research Desgin and Evaluation Strategy

This work adopts a **quantitative system performance evaluation** to validate the GAM pipeline. The system is evaluated as a whole and in part through measurable performance indicators, focusing on real-world deployment constraints such as robustness, responsiveness, and generalization.

The overall methodology is structured around the following components:

- **Dependent Variables:**
  - *Grasp Success Rate:* Percentage of successful grasp attempts, defined by the ability of the robot to lift and transport the object without slippage, in line with **RQ1**.
  - *Pipeline Latency:* Total execution time from prompt input to grasp execution, mapping to **RQ2**.
  - *Planning Time:* Time taken by *MTC* to generate and solve the grasp and post-grasp tasks, inferring **RQ3**

- **Independent Variables:**
  - *Object Diversity:* The system was tested across a range of object shapes, sizes, textures, and physical configurations (e.g., standing, lying down, partially occluded). This variable reflects the system's ability to maintain robustness and generalization across geometrically and visually diverse inputs.
  - *Scene Complexity:* Presence of occlusions, clutter, or partial views.
  - *Prompt Complexity:* Ambiguity or descriptiveness of the object prompt (e.g., "mug" vs. *"blue cylindrical container"*).

The goal of the evaluation is not to test formal statistical hypotheses but to verify whether the system exhibits **functional robustness** under variation in object characteristics, scene layout, and prompt input. Each module is independently designed for zero-shot inference, enabling the overall pipeline to generalize without explicit object-level training. The emphasis is placed on generalization, modular reliability, and qualitative behaviour across edge cases.

Chapter 4 will detail the experimental protocol, including test repetitions, and success/failure thresholds. This design ensures that the engineering outcomes of the GAM pipeline can be interpreted within a reproducible, empirical framework.

3.1.2 Experimental Instruments and Logging

To support performance validation and real-time debugging, GAM integrates several **diagnostic** and **data logging tools**, each mapped to a specific inference module. These instruments serve both for qualitative inspection (e.g., RViz visualizations) and quantitative analysis (e.g., timing, success logs).

Visualization Tools

- ***RViz Topic:***
  - `debug_binary_mask` and `debug_annotated_image`: Visualizes segmentation mask produced by *GSAM*;
  - `transformed_cloud`: Shows the aligned and reconstructed point cloud;
  - `grasp_pose`: Displays candidate grasp poses as `PoseStamped` messages.
  - `grasp_pose`: shows the best found solution if the Task is created successfully.

- ***Simulation & Testing Framework (Gazebo):***
  - The **Gazebo simulator** was used extensively during development to evaluate and tune pipeline components before deployment.
  - It provides a **controlled environment** for iterating on motion planning, perception reliability, and error recovery under repeatable conditions.
  - *Tiago deployment*
  - *ROS2 Logging*:
    - **Pipeline data** (e.g., images, point clouds, planning scenes) was also logged during trials.

Debug Utilities

- BT-based status codes to trace pipeline progress and failures.
- Feedback added to each ROS2 node execution to evaluate each stage's result.

Together, these tools form the experimental backbone of Chapter 4's evaluation, ensuring the GAM pipeline's behaviour can be audited, repeated, and quantitatively assessed across trials.

3.1.3 BT Execution and ROS2 Service Coordination

This section describes how the practical implementation of the GAM pipeline leverages the ROS2 ecosystem for distributed communication and the *BehaviorTree.CPP* library for orchestration. The implementation emphasizes launch-time setup, action interfacing, BT construction, service invocation and modular extensibility.

The architecture is initialized via **two dedicated ROS2 launch files**. The `gam_server.launch.py` script is responsible for deploying all pipeline stages, instantiating, with all the parameters required by each component:

- Detector,
- 3D Reconstructor,
- Grasp Pose Detector,
- Action Server.

Each server loads its model and configuration (e.g., `gsam_params.yaml`, `gpd_config.yaml`) during launch, ensuring readiness for low-latency runtime inference. Once the setup is complete, the **`gam_client.launch.py`** file spawns a client node that retrieves a user-defined object label (`object_type` in script 3.1), **by command line**, and transmits it as a goal to the GAM action server by invoking the `send_goal` function defined in code 3.1.

```
def send_goal(self, object_type: str) -> rclpy.task.Future:
    goal_msg = GamMsg.Goal()
    goal_msg.object_type = object_type

    self.get_logger().info(f"Sending Goal: [{object_type}]\n")
    return self._action_client.send_goal_async(goal_msg)
```

Code 3.1: `send_goal` function implementation

Upon receiving the goal, the GAM action server **stores** the **`object_type`** in the root blackboard of the BT, done by the line at 3.2, and begins "ticking" from the tree's root node in the loop implemented in code 3.3.

Code 3.2: Storing `object_type` on the Blackboard

Feedback messages are published throughout execution, reflecting the pipeline's status node (Code 3.3). If the BT terminates with `SUCCESS`, the result is extracted from the blackboard and returned to the client; if not, the pipeline aborts with a `FAILURE` status.

```
BT::NodeStatus tree_status = BT::NodeStatus::RUNNING;
while (rclcpp::ok() && !BT::isStatusCompleted(tree_status)) {
    tree_status = tree_->tickOnce();
    fb->feedback = "Tree status: " + BT::toStr(tree_status);
    goal_handle->publish_feedback(fb);
}
```

Code 3.3: Ticking of the BT

The tree structure itself is defined in an **external XML file** (e.g., `tree.xml`) following the `BT.CPP` format. Figures 3.1-3.2 illustrate that nodes are organized into a Sequence, with nested fallback and retry strategies using decorators like `RetryUntilSuccessful` with a specified number of maximum attempts (e.g., in this case 3). At runtime, each BT node retrieves its necessary input arguments from the blackboard and sends a service request to its paired ROS2 service server. The server performs the inference or action, then returns the response, which the node writes back to the blackboard. This cyclic exchange ensures continuous synchronization of intermediate results and clean modular separation between stages. An "entry" of the blackboard is a **key/value pair**; an **Input port** can read an entry in the blackboard, whilst an **Output port** can write into an entry. The **key** specifies the name of the stored information, while the **value** can be any type of variable [12].

The **`bt_register_manager.cpp`** dynamically register BT nodes via the `BT_REGISTER_NODES` macro, enabling runtime matching of XML-defined nodes to C++ logic, done by Code 3.4. This allows nodes to be defined in the XML tree with only a symbolic name and matched to the underlying class logic at runtime. This design guarantees **reusability** and **flexibility**: replacing a model or modifying inference behaviour requires only updating the class or XML node name, without altering the core BT execution logic.

```
BT_REGISTER_NODES(factory)
{
  factory.registerNodeType<gam::LookTable>("LookTable");
  factory.registerNodeType<gam::Detector>("Detector");
  factory.registerNodeType<gam::ObjectReconstructor>("
   ObjectReconstructor");
  factory.registerNodeType<gam::DetectGrasp>("DetectGrasp");
  factory.registerNodeType<gam::PlanningSceneHandler>("
   PlanningSceneHandler");
  factory.registerNodeType<gam::MovePreGrasp>("MovePreGrasp");
  factory.registerNodeType<gam::MoveGraspObject>("MoveGraspObject");
  factory.registerNodeType<gam::MovePostGrasp>("MovePostGrasp");
}
```

Code 3.4: Registration of BT nodes

Finally, the `getFactoryInstance()` utility returns a globally accessible BT factory, shown in Code 3.5, used to instantiate trees from XML definitions, where in the script 3.6 is defined as `tree_path`. This factory is initialized once and cached for reuse, accessed in the action server as shown in the implementation 3.6:

```
// Return the ptr of the global factory to allow
// to instanciate the same tree in any file
BT::BehaviorTreeFactory & getFactoryInstance()
{
  static BT::BehaviorTreeFactory factory;
  static bool initialized = false;

  if (!initialized) {
    BT_RegisterNodesFromPlugin(factory);
    initialized = true;
  }
  return factory;
}
```

Code 3.5: Enabling the BT to be globally accessible

```
    auto & factory = gam::utils::getFactoryInstance();
    tree_ = std::make_shared<BT::Tree>(factory.createTreeFromFile(
   tree_path));
```

Code 3.6: Creating the BT from XML definitions

To promote extensibility and accommodate changes in tree structure without duplicating core logic, the GAM action server **inherits from an abstract base class** (`GAMBaseServer`). This templated class encapsulates essential setup routines, including tree loading, parameter declaration, and action lifecycle callbacks. Derived implementations, such as `GAMServer`, override only the **execute()** method to define task-specific behaviours—e.g., how the goal is interpreted and how BT execution status is tracked. This abstraction promotes **separation of concerns**, enabling the reuse of common logic across experiments with alternative tree configurations or manipulation tasks, while leaving the high-level control structure intact.

To ensure maintainability and modularity across the **ROS2 workspace**, the GAM system is organized into **distinct ROS2 packages**. The `gam_server` package encapsulates the action server logic, including the loading and execution of BT, while the `gam_bt` package contains all custom BT node definitions along with their registration logic through the `bt_register_manager` 3.4. This separation reinforces the architecture's modular design by clearly distinguishing control flow orchestration from task-specific execution logic. It also simplifies future extensions—such as integrating new planners or replacing detectors—by localizing changes to specific packages without disrupting the system's core behaviour.

Through this layered and modular orchestration strategy, the GAM pipeline achieves robust coordination of heterogeneous components, decoupling perception, planning, and control into testable service units while maintaining global task coherence via the blackboard and BT execution loop. This architecture supports extensibility (e.g., adding fallback planners), reusability (e.g., swapping the grasp planner node), and fault tolerance (e.g., retrying failed stages), aligning with the system-level requirements of real-world service robotics.

3.2 Perception

Perception is the first and foundational stage of the GAM pipeline. It serves to extract semantically meaningful information from raw sensory inputs—specifically, RGB images. This module bridges the gap between abstract textual input and spatially grounded object representations that downstream modules can use for

planning and manipulation. It directly addresses **RQ1**, which investigates the extent to which prompt-driven perception can generalize to real-world object identification under constrained computational resources.

The perception module transforms abstract textual prompts (e.g., "the apple") into spatially grounded masks using *GSAM*, a hybrid framework combining *Grounding-DINO*'s open-vocabulary detection with *SAM*'s segmentation precision. This addresses **RQ1** by enabling zero-shot generalization to novel object without GPU acceleration. The outputs of this module include:

- A binary **segmentation mask** that localizes the object corresponding to the prompt, used in the following stages,
- The **RGB image** from the robot's perspective at the moment of detection (for debugging purposes),
- A **filtered point cloud** (referred to as the "`full_seen_cloud`") that provides geometric context for 3D reconstruction.

The perception task is executed in a ROS2-compliant pipeline through a modular service-based structure, summarized in Fig 3.3. When the BT node corresponding to object detection is ticked, it triggers a ROS2 service call to the detection module, passing the **RGB image** (robot's POV) and the `object_type` prompt retrieved from the blackboard. Internally, the detector processes this input by executing the *GSAM* model stack and returns both the binary mask and an annotated debug image to assist in verification.

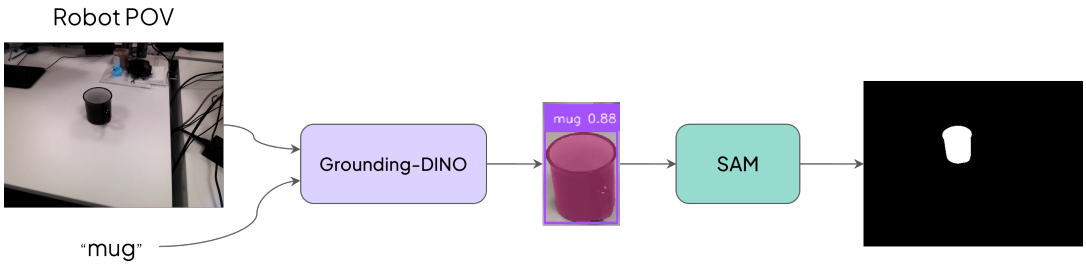


Figure 3.3: **Perception** Pipeline: The robot RGB point of view (POV) and the `object_type` are fed to *Grounding-DINO*, whici will produce the bounding box with the confidence score that serves as input for *SAM*, that produces the binary mask.

Crucially, the perception system is implemented to be resilient and responsive. It employs retry strategies within the BT (via decorators such as `RetryUntilSuccessful`) to handle cases where the object is not initially visible or partially occluded. Moreover, before moving forward in the pipeline, the BT node ensures that the mask aligns with a sufficiently dense point cloud—thus enforcing a consistency check between semantic segmentation and geometric observability.

From an architectural standpoint, the perception module exemplifies GAM's design philosophy: task-driven inference that is modular, reusable, and grounded in real-time ROS2 communication. It transforms abstract task goals into actionable data structures and serves as a semantic anchor for subsequent modules like 3D reconstruction and grasp pose detection.

Section 3.2.1 details the implementation of this perception module, focusing on how the BT node interfaces with ROS2 services, acquires live sensory input, and verifies inference quality before setting results to the BT blackboard.

3.2.1 Perception Module Implementation

The implementation of the perception module in the GAM pipeline leverages a ROS2-compliant structure composed of a BT node acting as a **service client** and a corresponding ROS2 **service server** responsible for running the actual detection and segmentation models. This section outlines the key implementation details of this module, emphasizing modular design, runtime robustness, and integration with the rest of the pipeline.

The BT node responsible for perception, named **Detector**, is implemented as a synchronous action node. When ticked, the node retrieves two inputs from the ROS2 ecosystem: an **RGB image** and a **depth-aligned point cloud** from the topics:

- **`/head_front_camera/rgb/image_raw`**;
- **`/head_front_camera/depth_registered/points`**.

Together, these represents Tiago's current visual (POV) and geometric understanding of the environment. These inputs are stored in local buffers (`table_rgb_` and `full_seen_cloud_`) via callback subscriptions to ensure that the data is current at the time of inference.

The node constructs a **request** message containing the RGB image and the object prompt retrieved from the blackboard. This request is sent to the `detector_service`, a ROS2 service server implemented in Python. The server executes the detection pipeline using the *GSAM* model, comprised of *Grounding-DINO* and *SAM*, and returns a **binary mask** (Fig. 3.4c) corresponding to the **segmented object**, along with an annotated debug image (Fig 3.4b).

(a) Robot POV (b) Annotated image (c) Binary mask

Figure 3.4: *Grounded-SAM* stages.

To ensure data integrity, the BT node **validates** the response before proceeding. Specifically, it checks, for each pixel-point pair, whether the segmented object region (white pixels in the `mask_`) contains sufficient geometric information corresponding to the masked object (points in the cloud aligned with the mask) by evaluating whether the point cloud density within the mask meets a minimal threshold (e.g., 80%)—using the utility function `cloudFillsMask()` used in Code 3.7 which implements the logic in equation 3.1.

$$\text{Density} = \frac{\#\text{ white pixels in the } \texttt{mask\_}}{\#\text{ points in } \texttt{full\_seen\_cloud\_} \text{ aligned with the mask}} \geq 0.80 \quad (3.1)$$

If the mask does not align with a sufficiently populated region of the point cloud, the node retries the inference by **spinning** until satisfactory data is received, as shown in Code 3.7. This check is crucial to prevent propagating incomplete or noisy cloud data to the reconstruction module.

```
while (!utils::cloudFillsMask(this->full_seen_cloud_, this->mask_)) {
    rclcpp::spin_some(this->node_);
};
```

Code 3.7: Assuring consistent point clouds

Successful outputs are then written back to the blackboard for downstream modules to consume. These include:

- The binary **segmentation mask** (`mask_` in code 3.7),
- The **RGB image** at the time of detection (`table_rgb`),
- The validated **full point cloud** used for context and filtering (`full_seen_cloud_` in code 3.7).

On the server side, the detector service is initialized in the `gam_detector` package. It loads the *GSAM* model stack, including relevant configuration parameters such as encoder version and detection thresholds. Upon receiving a request, the service decodes the image, performs detection and segmentation, and packages the result into a ROS2 service response.

To support debugging and monitoring, the server **publishes** the **binary mask** and **annotated image** (Fig.3.4b) to the topics `debug_binary_mask` (Fig.3.4c) and `debug_annotated_image`. These allow developers to visually inspect the quality of detections during runtime.

The perception module is designed to be **modular** and **reusable**. Its implementation in a standalone package with clean ROS2 service interfaces allows it to be reused across different tasks or replaced by alternative detection backends with minimal integration overhead. The BT node can be redirected to a different service name or configured to use alternative sensory topics with trivial adjustments.

This tightly coupled but decoupled-by-design implementation aligns with GAM's architectural goals: it supports generalization to different tasks and scenes (**RQ1**) while maintaining robustness and system coherence under real-time conditions.

The Perception module's output serves as the foundation for 3D Reconstruction. By converting semantic masks into dense geometric representations, this stage directly addresses **RQ2**'s precision-efficiency trade-offs.

3.3 3D Reconstruction

Following the perception stage, the GAM pipeline transitions into 3D reconstruction, where a semantically segmented object is transformed into a dense

3D representation suitable for geometric reasoning. This module takes as input the **output of the perception** system and produces a filtered and transformed point cloud that serves as the basis for grasp pose detection. As such, it directly addresses **RQ2** by enabling efficient, high-fidelity object reconstructions even under limited sensing and computational constraints.

The 3D Reconstruction stage transforms semantic masks into actionable geometric representations using *MCC (Multiview-Compressive-Coding* introduced in the section 2.3.4). By inferring occluded surfaces and aligning outputs to the robot's operational frame, this stage directly addresses **RQ2**'s demand for efficient, high-fidelity reconstructions under hardware constraints. The module executes the *MCC* pipeline by fusing the image, mask, and visible point cloud into a format suitable for reconstruction. *MCC* internally performs preprocessing steps like normalization, masking, cropping, padding, and resampling to prepare consistent input tensors. The output is a dense, coloured point cloud that represents the **reconstructed object surface**, along with a centroid used for spatial alignment. In the Fig. 3.5 is shown the *seen* point cloud of the robot placed in front of a desk with the goal of grasping a "mug". In the picture is evident the absence (grey background) of multiple points that compose the 3D representation of the mug. The *MCC*'s output is visible in Fig. 3.6a: the coloured points represents the 3D reconstruction spatially aligned. Fig. 3.6b, instead, displays the same output with increased **granularity.**

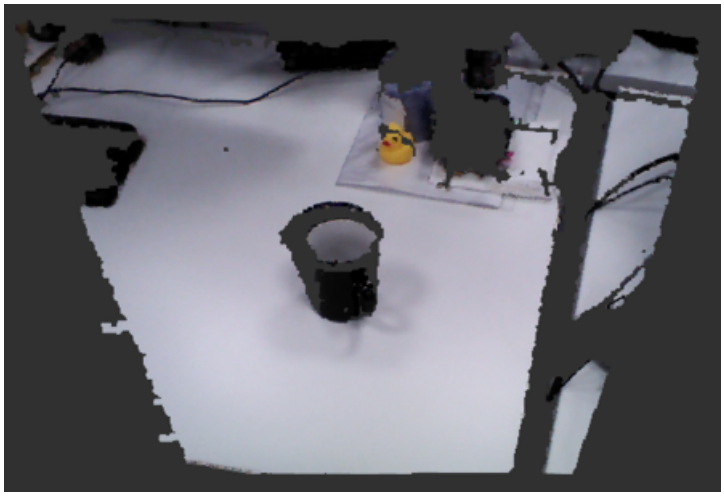
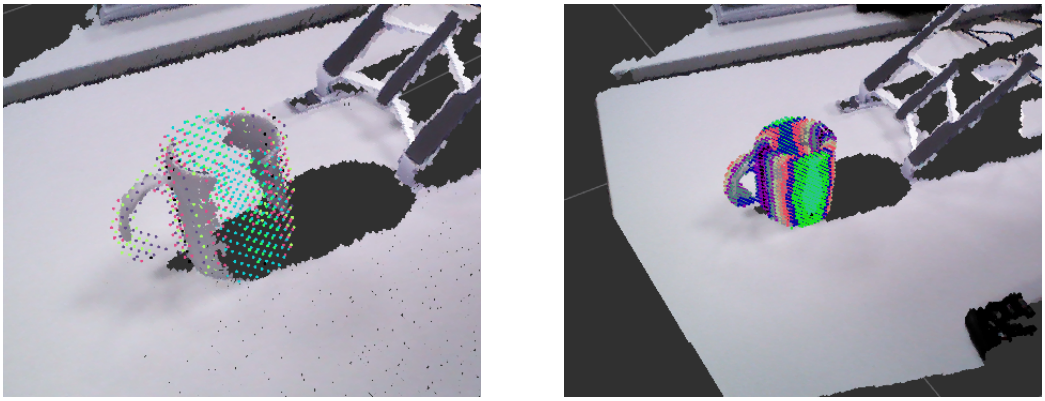


Figure 3.5: *Seen* point cloud.



(a) *MCC*'s output, *granularity=0.05* (b) *MCC*'s output, *granularity=0.1*

Figure 3.6: 3D Reconstructed mug, output examples.

Importantly, this module enforces a **consistency check** between the mask, the image, and the point cloud dimensions before triggering inference. These checks ensure that the reconstruction task is not performed on corrupted or mismatched input, thereby preventing downstream failures. Moreover, the output point cloud is spatially transformed using the centroid of the segmented region to maintain proper alignment with the robot's base frame.

From a system design perspective, this module exemplifies GAM's architecture principles: inference is triggered reactively within the BT, but the logic and model execution are encapsulated in a dedicated ROS2 service. This decouples control from computation, allowing the reconstruction strategy to be swapped or refined independently of the task policy. Section 3.3.1 presents the implementation-level details, highlighting the role of ROS2 service messaging, data validation routines, and the interplay between BT node execution and model inference logic.

3.3.1 3D Reconstruction Module Implementation

The 3D reconstruction module is implemented through a dedicated ROS2 service node and a corresponding BT node that acts as a client. This implementation, housed within the `gam_3d_reconstructor` and `gam_bt` packages respectively, demonstrates modular inference integration in a distributed architecture.

The BT node `ObjectReconstructor` is triggered once the output from the perception stage (mask, RGB image, and point cloud) is available on the black-

board. Upon activation, the node retrieves these data entries and performs an initial validation: it checks that the image, mask, and point cloud dimensions **are aligned** to prevent malformed requests. If the check passes, it constructs a **request** message and invokes the `reconstructor_srv`, a ROS2 service exposed by the Python-based `ReconstructorService` node.

The service server uses a deep learning pipeline based on MCC. The input consists of:

- A **RGB image** (Tiago's POV),
- A **binary mask** (object to grasp),
- A **registered point cloud** aligned to the robot's depth camera (*seen point cloud*).

These inputs are pre-processed by the service to generate a dense point cloud representing the **overall shape** of the object, as shown in the Fig. 3.7. Internally, *MCC* converts the raw sensory data into tensors and normalizes them to account for viewpoint and **scale factor** as shown in Code 3.8. It predicts unobserved surfaces by fusing appearance and geometry cues, ultimately generating a complete 3D object representation.

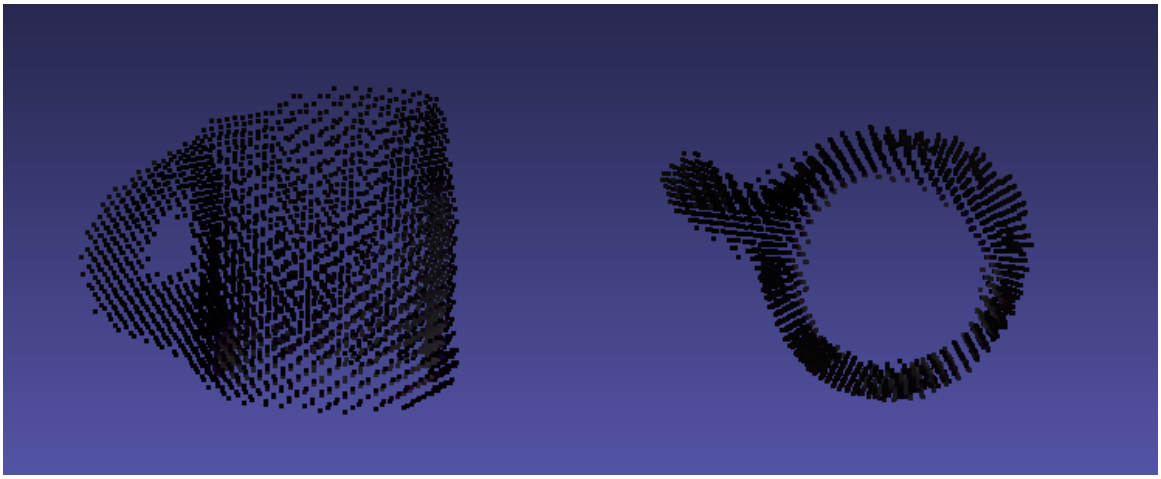


Figure 3.7: 3D Reconstructed shape of a mug. Output of *MCC* having in input the binary mask, the RGB image aligned with the mask, and the *seen* point cloud (as depicted in Fig.3.5).

Once reconstruction is complete, the service returns:

- A **reconstructed point cloud** (as a `PointCloud2` message),
- The **spatial centroid** of the object cloud (`centroid_src` in the 3.8), defining the **object's position** with the respect to Tiago.

```
self.centroid_src = seen_xyz[torch.isfinite(seen_xyz.sum(dim=-1))].
    mean(axis=0)
self.scaling = (seen_xyz[torch.isfinite(seen_xyz.sum(dim=-1))].var(
    dim=0) ** 0.5).mean()
```

Code 3.8: Scaling and centroid factors' computation

On the client side, the BT node applies a geometric transformation to the reconstructed cloud, using the **centroid**, to align it with Tiago's base frame. The centroid is determined by computing the average of all the points' coordinates (x-y-z for each point), resulting in the position of the **object's center** with the respect to Tiago, advertised in Code 3.8. The transformed cloud is then published to a ROS2 debug topic for visualization and written to the blackboard under the key `object_cloud`, making it accessible to subsequent stages.

The BT node also ensures that runtime failures (e.g., missing data, reconstruction errors, transformation issues) result in a `FAILURE` status. Additionally, if the reconstructed point cloud contains fewer than a threshold number of points (e.g., 600), the node retries the inference, as the *MCC* model—due to its non-zero **temperature**[2] parameter—can yield **different reconstructions** across runs. This mechanism is essential for the tree's fallback logic, allowing recovery behaviours to activate if necessary.

This tightly integrated design illustrates GAM's architectural goals: model encapsulation through services, runtime data verification, and modular ROS2 interoperability. By isolating 3D reconstruction logic in a dedicated package and exposing it through a **service interface**, the system ensures clean abstraction boundaries and ease of future upgrades (e.g., swapping *MCC* with an alternative model or fine-tuning).

[2]**Temperature** (T) in *MCC* scales logits to control output randomness across multiple runs: higher values (T >1) soften probability distributions (increasing diversity), while lower (T <1) values sharpen predictions (enhancing confidence).

3.4 Grasping Pose Detection

The grasp pose detection module constitutes the fourth stage in the GAM pipeline, bridging the output of the 3D reconstruction with the motion planning logic that follows. Its primary objective is to **generate** a set of geometrically valid and **executable grasp candidates** based on the object's reconstructed shape and spatial orientation. This stage directly contributes to **RQ2** by addressing how grasp precision and runtime responsiveness can be preserved despite noisy inputs and constrained computational budgets.

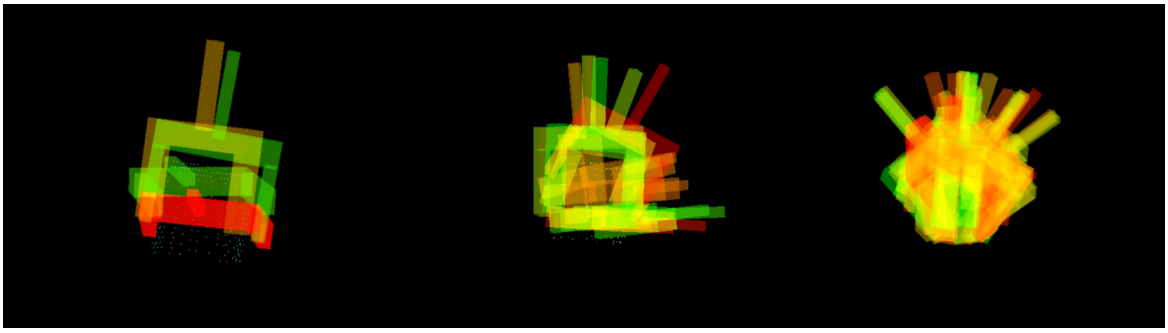
Building upon the **geometric information** reconstructed in the previous stage, this module infers stable, kinematically feasible[3] grasp configurations that the robot can execute. In doing so, it serves as a key link between perception-driven object understanding and motion planning.

Grasp pose detection in GAM is grounded in the use of **geometry-centric** inference, specifically designed to operate on dense, partially reconstructed point clouds. This approach is essential for scenarios where complete object models are unavailable or where the target object may be **novel**, cluttered, or partially occluded. The module aims to identify a **set of 6-DOF grasp poses**—each specifying both position and orientation of the gripper—that maximize the likelihood of a successful grasp, based purely on spatial geometry and contact heuristics.

To achieve this, GAM integrates the ***Grasp Pose Detection*** (*GPD*) algorithm, analysed in the section 2.4.4, a model known for its ability to detect robust grasps directly from raw 3D point clouds. Unlike template-based or model-dependent approaches, *GPD* **does not rely** on predefined object meshes, making it inherently suitable for generalization to unseen items [46]. It analyses local point cloud neighbourhoods to identify geometries conducive to parallel-jaw gripper stability, scoring candidates based on reachability and collision-avoidance metrics. Figure 3.8 illustrates the output of a *GPD* module for a representative "cylinder" object. In this case, Fig. 3.8a depicts a subset of $n = 4$ candidate grasp poses, Fig. 3.8b expands the set to $n = 15$, while Fig. 3.8c visualizes a inference

[3] "Kinematically feasible" refers to a motion, configuration, or action that is physically achievable by a robotic system **without violating** its mechanical constraints.

output with $n = 30$ grasp hypotheses. The predicted grasps are color-coded according to their **confidence scores**, with green representing high-likelihood (high-quality) grasp configurations and red indicating lower-confidence or sub-optimal candidates, as evaluated by the *GPD* model's internal scoring function.



(a) *n=4* (b) *n=15* (c) *n=30*

Figure 3.8: *GPD* output on a reconstructed *cylinder* point cloud.

From a pipeline standpoint, the output of the 3D reconstruction module serves as the input to this stage: a segmented point cloud centred on the object of interest. The grasp detection process then **samples** candidate regions, **evaluates** their surface features, and **ranks** viable poses according to their grasp success probability. These candidates form the semantic foundation for downstream motion planning, providing both **flexibility** (through multiple grasp options) and **adaptability** (by enabling fallback strategies if some poses fail to execute).

Critically, this module reinforces GAM's architectural principles: it is modular, stateless between ticks, and driven by on-demand inference. To ensure robustness, the pipeline supports **fallback mechanisms** within the BT: if no valid grasps are returned, retry logic is triggered automatically. This mechanism allows the robot to reprocess the input cloud or re-run modules to recover from ambiguous or noisy sensing conditions.

As such, the grasp pose detection module exemplifies the GAM philosophy of responsive, task-aware manipulation. It transforms dense spatial data into concrete control objectives while maintaining modularity and generalization capabilities. Section 3.4.1 will present the implementation details, including ROS2 service interfacing, BT integration, and data validation mechanisms employed during runtime execution.

3.4.1 Grasping Pose Detection Module Implementation

The implementation of the grasp pose detection module in the GAM pipeline follows a modular and service-oriented design, consistent with the architectural principles established in previous stages. The module spans two ROS2 packages: **`gam_bt`**, which houses the **`DetectGrasp`** BT node acting as the **service client**, and **`gam_grasp_detector`**, which implements the **service server** responsible for executing the GPD algorithm.

The BT node `DetectGrasp` is triggered once the **reconstructed object point cloud** becomes available on the blackboard. Upon ticking, the node first verifies that the cloud is present and valid. It then constructs a ROS2 service request, populating it with the object cloud produced by the 3D reconstruction module. This request is sent to the `detect_grasp_service`, a dedicated ROS2 service node responsible for executing the grasp detection logic.

On the server side, the `GraspDetectorService` is initialized, as shown in Code 3.9 during system launch and configured with a YAML-based parameter file ("`gpd_config.yaml`" in the script 3.9) defining detection-specific thresholds, model weights, and runtime configurations.

```
1 declare_parameter<std::string>("gpd_params", gam_grasp_detector/
    config/gpd_config.yaml");
2 config_path_ = get_parameter("gpd_params").as_string();
3 gpd_detector_ = std::make_shared<gpd::GraspDtector>(config_path_.
    c_str());
```

Code 3.9: Initialization of the Grasp Detector server

Internally, it wraps the *GPD* library, initializing it with the provided parameters and exposing its functionality via a ROS2 service interface. Upon receiving a service call, the node converts the ROS2 `PointCloud2` message into a *Point Cloud Library*-compatible *(PCL)* format, initializes the *GPD* input cloud, and performs local geometric analysis to detect candidate grasps.

The *GPD* pipeline preprocesses the point cloud, filtering noise and extracting surface normals, before generating a **set of grasp hypotheses**, as seen in Code 3.10. These are evaluated against a set of geometric criteria—such as gripper

clearance, contact surface area (number of contact points), and antipodal grasp quality.

```
std::vector<std::unique_ptr<GpdHand>> gpd_poses = this->gpd_detector_
    ->detectGrasps(*gpd_cloud);
```

Code 3.10: Grasp detection line

Valid candidates are converted into ROS2 `PoseStamped` messages, using the `convertGrasps()` utility function, and returned in the service response. If no candidates are found; some of the poses are not feasible; or some conversion errors is caught, the `result` field is set to false, triggering a `FAILURE` status on the BT side and enabling fallback mechanisms, as laid-out in Code 3.11.

```
// Convert from GPD poses to ROS PoseStamped msgs
if (!gam::utils::convertGrasps(gpd_poses, pose_msgs)) {
    return false;
}
```

Code 3.11: Detected grasps validation

Back in the BT node, the response is parsed to extract the array of valid grasp poses. These are then:

- Written back to the blackboard under the key `grasp_candidates` for downstream access.
- Optionally published to the ROS2 `grasp_pose` topic for real-time **RViz visualization**, assisting in runtime debugging and validation, as illustrated in Fig. 3.9 the grasp hypothesis is represented by the purple arrow as approaching vector with the respect to the Tiago's base frame.

During early integration, the *GPD* module was initially configured to operate solely on the reconstructed object point cloud. However, empirical observations revealed that many predicted grasps exhibited arbitrary approach vectors, including directions from **beneath the table** surface or from the far side of the object—areas that were kinematically unfeasible. To mitigate this, the configuration parameters were tuned to favor approach vectors along a diagonal bias, prioritizing directions from above and in front of the object with respect to the robot's base frame. While this directional constraint improved semantic alignment with the robot's workspace, it did not fully eliminate collisions with the supporting surface, particularly for low-profile or flat objects.

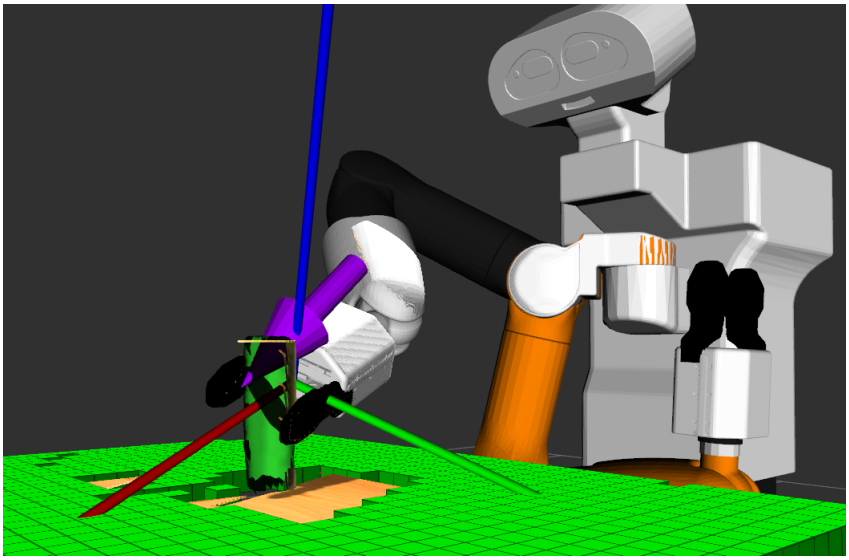
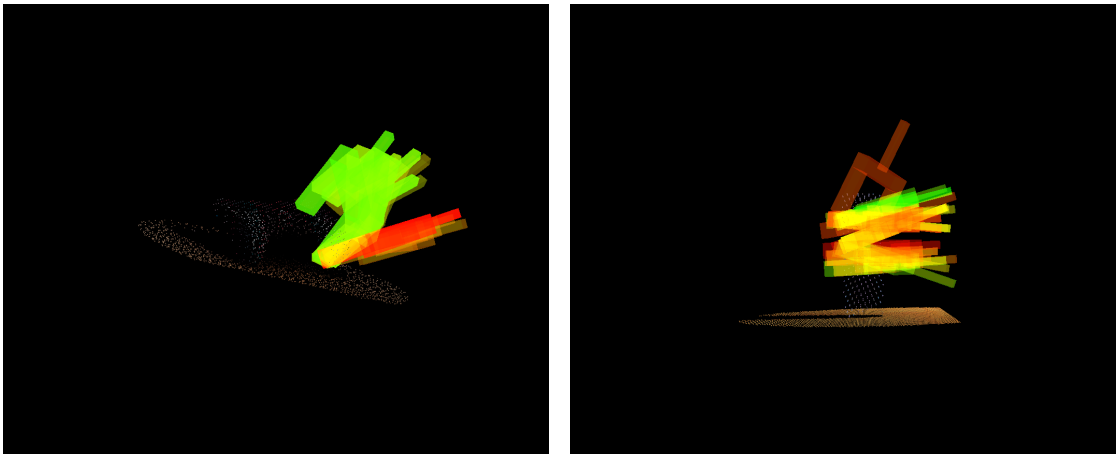


Figure 3.9: RViz representation of a grasp and Tiago aiming to reach the predicted pose with its end-effector. The orange arm resembles the actual position in the simulation, while the arm performing the grasp is the *MTC* plan visualization.

To address this, an additional pre-processing step was introduced: a **local spatial neighborhood** around the object was extracted by selecting all points within a fixed radius *'r'* from the object centroid and appending them to the reconstructed cloud (in Fig.3.10 are seen as *orange* points). This contextual augmentation enabled the *GPD* algorithm to better infer the presence of the tabletop and avoid generating grasp poses from underneath or too close to the support surface. The outcome was a notable improvement in grasp quality and feasibility—grasps were now more consistently aligned with the robot's kinematic constraints and less prone to planning rejections due to collision. As illustrated in Figures 3.10a-3.10b, the detected grasp vectors, in case of grasping a *joystick* and a *cylinder*, exhibit both correct approach orientation and spatial clearance from the table plane, validating the efficacy of this enhancement.

As with previous modules, this node is stateless between ticks and fully encapsulated. It employs internal error handling and timeout monitoring to ensure system responsiveness. Moreover, retry decorators within the BT allow for multiple detection attempts in case of transient failures (e.g., due to sparse or occluded input clouds).



(a) *Joystick* (b) *Cylinder*

Figure 3.10: Predicted grasp poses with improved *GPD*'s input cloud.

This implementation ensures that the grasp detection logic remains **cleanly separated** from both perception and planning modules while maintaining interoperability through standardized ROS2 service interfaces. It reinforces the pipeline's commitment to reusability, runtime robustness, and inference-driven decision-making. As such, the grasp pose detection module not only operationalizes the *GPD* algorithm within the GAM context but does so in a manner that is extensible, interpretable, and aligned with the broader system architecture lying down the foundation for Motion Planning, debated in the following sections.

3.5 Planning Scene Handler

Accurate scene representation is a prerequisite for any reliable motion planning system. In the context of the GAM pipeline, this requirement becomes even more critical: the system is designed to operate in unstructured environments and to generalize to arbitrary, previously unseen objects. Consequently, traditional **voxel-based** approximations—commonly used to populate the planning scene in standard robotic applications—are **inadequate**. They lack the precision required to safely and effectively plan fine-grained manipulation motions around novel object geometries.

To overcome this limitation, the GAM framework explicitly integrates a high-fidelity **object mesh**, as visible in Figure 3.11, into the planning scene prior to

trajectory generation. This design choice strengthens the coupling between perception and planning: it allows the motion planner not only to be informed of the presence of an object but also to "see" its **exact shape** and position, enabling collision-aware, object-specific trajectory synthesis.

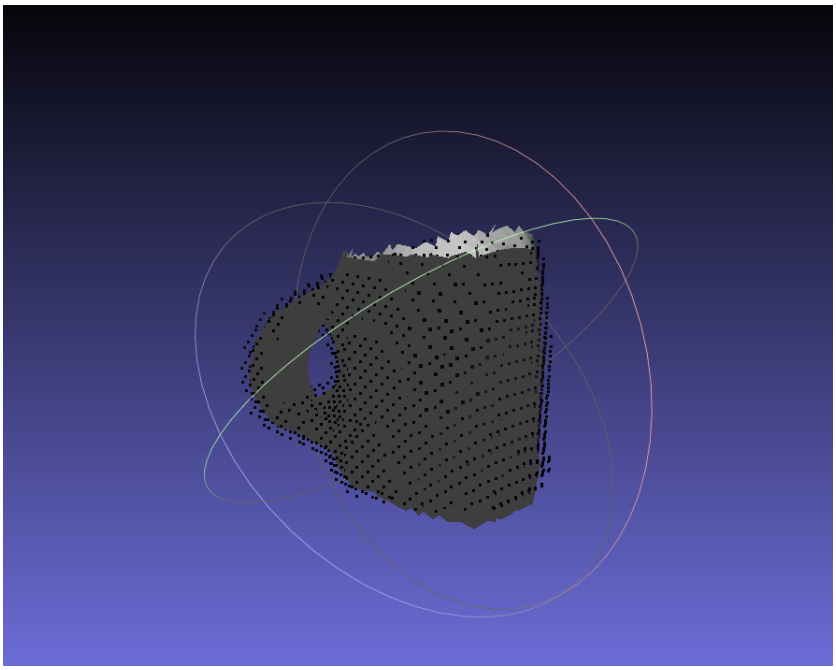


Figure 3.11: Object mesh, generated from a *mug* reconstructed point cloud (black points).

More specifically, the reconstructed point cloud obtained from the 3D Reconstruction module (Section 3.3) is transformed into a watertight **triangular mesh**. Once generated, the mesh is inserted into the MoveIt2 Planning Scene as a **collision-object**. Its pose is anchored using the **centroid** computed during *MCC* pre-processing, ensuring spatial alignment between the virtual mesh and the physical object in the robot's workspace.

Before inserting the object mesh, the Planning Scene is **sanitized**. The pipeline ensures that all previously inserted objects are removed before populating the scene with the **current target**. This is crucial for maintaining temporal coherence and avoiding planning artifacts in the scene—especially when the GAM pipeline is **triggered multiple times** during a multiple grasping-tasks or when successive goals are issued. At any point, the only non-static object explicitly modelled in the scene is the one currently targeted for grasping.

By anchoring the mesh to the *MCC*-computed **centroid** and **sanitizing** the planning scene, GAM ensures **kinematic feasibility** and temporal coherence. This enables *MTC* to generate collision-aware trajectories (e.g., avoiding gripper-object collisions) while sticking to the Tiago's mechanical constraints, operating on an **accurate world model**; as an exaple in Fig. 3.9 is visible a *cylinder* mesh inserted in the Tiago's Planning Scene. This enables not only improved planning efficiency and safety but also enhances the robot's ability to execute grasps in cluttered or constrained environments. Section 3.5.1 will detail how this insertion process is practically implemented and synchronized with the BT execution cycle.

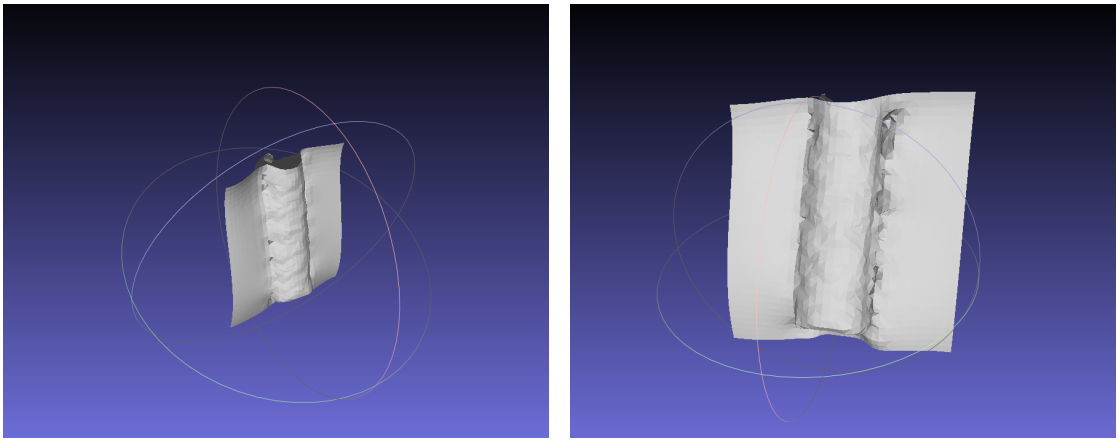
3.5.1 Planning Scene Hnalder Implementation

The practical implementation of the planning scene handler within the GAM pipeline is realized through a dedicated BT node named `PlanningSceneHandler`. This node, implemented in the **`gam_bt`** package, orchestrates the full process of converting the reconstructed object point cloud into a **mesh** and integrating it into the MoveIt2 Planning Scene. The implementation holds strictly to the modular principles of ROS2 and BT, enabling repeatable, verifiable state transitions and extensible scene management.

Upon being ticked, the BT node performs three critical stages: **data retrieval**, **mesh computation**, and **planning scene update**. First, it acquires from the blackboard the inputs required for mesh generation: the reconstructed object cloud (`object_cloud`), its centroid (`object_position`), and the corresponding label (`object_type`). These entries are prerequisites, as the centroid enables spatial alignment and the label ensures unique identification of the object in the Planning Scene.

Poisson Surface Reconstruction (Figure 3.11) was selected as the primary meshing method [2], implemented in Code 3.12. This method was chosen for its robustness in reconstructing watertight meshes from noisy or sparse point clouds. The pipeline also supports alternatives such as **Alpha Shapes** [2] and **Greedy Projection Triangulation** [32] (examples depicted respectively in Fig. 3.12a-3.12b) but Poisson yielded the best trade-off between smoothness, mesh weight, completeness, absence of artifacts, and computational cost that are es-

sential for reliable collision modelling. During the reconstruction, low-density polygons are filtered using a vertex neighbourhood density threshold (e.g., 10 neighbours within *5cm*), removing outliers and mesh artifacts improving the planning precision.



(a) *Alpha Shapes* (b) *Greedy Projection Triangulation* [32]

Figure 3.12: Meshes output on a *cylinder* object.

```
pcl::Poisson<pcl::PointNormal> poisson;
pcl::PolygonMesh meshes;

poisson.setDepth(this->depth_);
poisson.setInputCloud(cloud_normals);
poisson.reconstruct(meshes);
```

Code 3.12: Poisson Reconstruction implementation

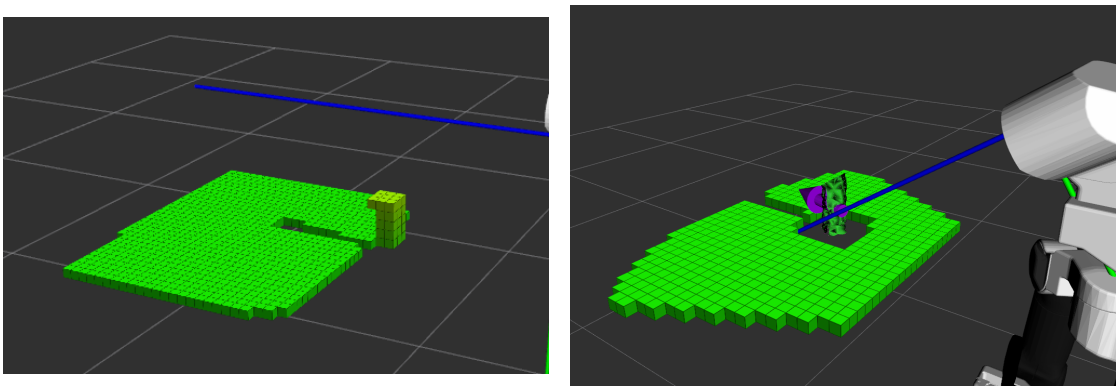
Once the mesh is created, it is converted into a ROS2-compatible `shape_msgs::Mesh` format. The transformation includes translating the mesh vertices using the centroid obtained during *MCC* pre-processing. This ensures that the mesh is accurately positioned in the world frame, coinciding with the robot's current understanding of the object's location. The converted mesh is then published as a `CollisionObject` to the Planning Scene using the `apply_planning_scene` ROS2 service used in Code 3.13.

```
PlanningSceneMsg ps;
CollisionObject co;
co.header.frame_id = this->object_cloud_->header.frame_id.c_str();
co.id = this->object_type_.c_str();
co.meshes.push_back(*mesh);
co.operation = CollisionObject::ADD;

ps.world.collision_objects.push_back(co);
auto request = std::make_shared<PlanningSceneSrv::Request>();
request->scene = ps;
auto result_future = ps_client_->async_send_request(request);
```

Code 3.13: Planning Scene update flow

Before insertion, the node calls a utility routine to remove any previously inserted objects. This prevents contamination from past executions and ensures that only the current object is present in the scene, facilitating cleaner motion planning and grasping attempts. Once the object is added, the system confirms the update's success and publishes the new state to *RViz* for visual confirmation, as shown before and after the update in the Figs. 3.13a-3.13b.



(a) Before

(b) After

Figure 3.13: Planning Scene update.

This implementation encapsulates the GAM pipeline's focus on precision and coherence. It transforms perceptual outputs into planning-ready semantic structures with minimal latency and maximum fidelity. By leveraging service abstraction and standard ROS2 messaging, the handler maintains modularity while aligning with MoveIt2's internal representations. Moreover, by handling mesh

generation and scene population as a dedicated BT node, the pipeline maintains clean boundaries between control logic and world modelling—supporting system reusability and future model upgrades.

3.6 Motion Planning & Execution

The final stage of the GAM pipeline involves the transition from perception-informed reasoning to action: the physical execution of the grasp. This phase is responsible for converting spatial understanding into concrete motor commands, ensuring the robot can approach, grasp, and retreat with the object in a safe and repeatable manner. It directly addresses **RQ3**, which investigates how motion planning systems can be structured to remain reactive and safe despite incomplete sensory feedback and execution uncertainties. Additionally, it continues to contribute to **RQ2** by maintaining robustness in downstream modules operating on learned geometric representations.

To this end, the GAM system leverages the *MoveIt Task Constructor (MTC)* framework, introduced in section 2.5.1. *MTC* enables the composition of high-level, semantically meaningful manipulation behaviours from modular motion planning primitives. This modularity aligns with the architectural principles adopted throughout the GAM design—namely, composability, explicit stage-wise control, and runtime adaptability.

At a conceptual level, the manipulation sequence could be described as a single continuous motion: transition from a "*home*" configuration to a pre-grasp pose, execute the grasp, and return to a "*home*" state with the object secured. However, this unified approach was deliberately decomposed into three independent planning units due to safety and observability constraints.

A key limitation encountered during development was the lack of closed-loop feedback regarding grasp success. Without tactile sensors or binary gripper state validation, the system could not robustly infer whether the object had been successfully grasped. This ambiguity, if unchecked, could lead to unsafe motions—such as the robot lifting an un-grasped object, performing a retreat with an empty hand or re-plan a motion due to an unexpected gripper state. To

mitigate these risks and to isolate potential failure points, the grasping pipeline was split into the following sub-components:

- **Pre-grasp Motion (*MTC* Task):** This task orchestrates the motion from the robot's "*home*" position to a **pre-grasp pose** computed via GPD. It also includes gripper opening and a forward push to enclose the object within the gripper jaws. The goal of this phase is to position the robot optimally for a grasp, without assuming success
- **Gripper Closure (*PlayMotion2* Action):** The grasp itself is executed via a predefined trajectory triggered through the *PlayMotion2* interface. This discrete action simplifies control and introduces a decision boundary: subsequent motion planning is only attempted if the closure completes successfully.
- **Post-grasp Motion (*MTC* Task):** After the assumed grasp, this task plans a safe **retreat path**. It **attaches** the object within the planning scene, lifts it using a Cartesian motion[4], and ultimately returns the arm to a predefined "home" configuration. If the object was not secured, this phase would typically fail at the collision checking or motion execution stage, prompting fallback behaviours.

This staged design not only improved execution reliability but also facilitated more granular debugging and recovery strategies. Moreover, by maintaining clear modular separation, the system remains **extensible**: future additions such as grasp validation, increase of the behaviours complexity or tactile sensing could be seamlessly integrated without disrupting the motion planning logic as suggested in Section 5.2.

Overall, the motion execution component exemplifies GAM's emphasis on operational reliability in real-world scenarios. Through careful architectural decomposition and robust planning with *MTC*, the robot can translate abstract grasp candidates into executable and verifiable action sequences—even in the absence of full sensory observability. The next section, 3.6.1, will dig into the implementation specifics of each task, outlining how planning stages were composed and how runtime safety mechanisms were enforced.

[4]Cartesian motion refers to movement along three perpendicular axes (X, Y, and Z) within a Cartesian coordinate system.

3.6.1 Trajectory Planning & Grasp Execution Implementation

The execution logic for GAM's motion planning is divided across **three** BT nodes, each implementing a distinct motion primitive in the grasp sequence, all included in the `1gam_bt` package. These are developed using the *MTC* framework and coordinated via ROS2 services and action clients to preserve the system's modular and reactive design.

Pre-Grasp Task Execution

The first motion primitive is implemented in the `move_pre_grasp` BT node. Upon activation, the node retrieves the selected grasp pose—previously computed by the *GPD* module—and **constructs** an *MTC* **Task** object composed of several sequential planning **Stages**:

- **Current State Initialization:** The Task begins by capturing the robot's joint state to anchor planning in the present configuration (`Current State` in Fig.3.14).
- **Open Gripper:** A *PlayMotion2* Stage is used to command the end-effector to open. This ensures the gripper is fully ready before proceeding (`Open hand` in Fig.3.14).
- **Move to Pick Position:** Using a sampling-based planner (*OMPL*, Section 2.5.2), the robot transitions from its current pose to a predefined "up" pose, rising its arm 'up' (`Unfold Arm` in Fig.3.14). This serves as an intermediate configuration to improve IK feasibility during approach. **Kinematic feasibility** is enforced during motion planning via MoveIt2's inverse kinematics solver, which discards grasps exceeding the Tiago's joint limits or workspace (`Compute IK` in Fig.3.14).
- **Allow Collisions:** Collisions between hand-object are allowed to let the planner validate plans that let the end-effector touch the object (`Allow collision hand-object` in Fig.3.14). This is required since *GPD* would reject all the plans that are in *collision*. `Allow collision hand-octomap` (in Fig. 3.14) is implemented to mitigate the drop of plans where gripper would hit the table. Adding this Stage once the IK is found, thus when the gripper is already in close to the object, would prevent safety issues or *collision plans*.
- **Approach to Grasp Pose:** The selected grasp pose is translated into a Cartesian trajectory using the *MTC* Connect stage, with constraints on positional and angular tolerances. The motion is planned relative to the robot's base frame (`Approach bowl` in Fig.3.14).

```
[INFO] [1744367451.640619916] [mtc_base_node]: Plan succeded!
  1  - ←   4 →   -  4 / Grasp task
    1  - ←   1 →   -  0 / Current State
    -  0 →   1 →   -  0 / Unfold arm
    -  0 →   1 →   -  1 / Open hand
    -  1 →   4 ←   4  - / Move to Pick
    4  - ←   4 →   -  4 / Grasp container
      9  - ←   9 →   -  0 / Compute IK
       30  - ←  30 →   - 30 / Generate Grasp pose
      -  0 →   9 →   -  0 / Allow collision hand-object
      -  0 →   9 →   -  0 / Allow collision hand-octomap
      -  0 →   4 →   -  4 / Approach bowl
```

Figure 3.14: *MTC*'s output of **pre-grasp** Task in case of grasping a *bowl*.

Figure 3.14 illustrates a representative **output** of the *MTC* for a single planning sequence. The overall task, labeled `Grasp task`, comprises a sequential pipeline of motion planning stages. The central indicator displays the number of full solutions identified—in this instance, **four valid motion plans**—as reflected by the four in the terminal stage, `Approach bowl`.The initial stages—`Current State`, `Unfold arm`, and `Open Hand`—denote deterministic operations that succeed in all feasible paths, each contributing a single outcome to the overall task flow. The `Grasp container` is a composite wrapper stage (described in Section 2.5.1) that encapsulates a nested grasp planning process: it receives as input a set of grasp hypotheses—derived from the *GPD* module—and attempts to compute n inverse kinematics (IK) solutions for each. In this example, from an initial pool of 30 candidate grasps, the system identified 9 valid IK solutions. These candidates were subsequently filtered based on feasibility constraints imposed by the final approach motion: `Approach bowl`. The last stage of the container evaluates each grasp's reachability and motion planning viability, ultimately retaining 4 complete plans that satisfy all geometric and kinematic constraints.

Each stage inherits **global planning parameters** (velocity/acceleration scaling, planners, and IK resolution strategies) set via a shared node handle. The resulting plan is validated, then executed through the standard MoveIt2 execution pipeline. A success at this stage indicates the robot is correctly positioned to close the gripper.

Gripper Closure via PlayMotion2

Gripper actuation is handled independently through the `MoveGraspObject` BT node. This design choice was informed by the absence of **grasp success feedback**. The node interfaces with the *PlayMotion2* action server, sending a named **trajectory goal** (e.g., "`close_left`") depending on the selected (by the planner) grasping side, in case Tiago++ is performing the grasp. Since the action is stateless and deterministic, it does not rely on intermediate planning.

This separation offers two primary benefits

1. It **minimizes** the risk of unsafe planning downstream by ensuring grasp closure is complete before planning any post-grasp motion.
2. It allows **simple future extensions**, such as increasing the motion complexity or implementing feedback for grasp validation (discussed in Section 5.2).

Post-Grasp Task Execution

The final segment of the pipeline is implemented in the `MovePostGrasp` BT node. This stage is responsible for **retreating with the object** and returning to the "home" pose, while maintaining **grasp integrity**. The `createTask()` 3.14 method constructs a full *MTC* pipeline with the following components:

- **Planning Scene Cleanup:**Prior to planning, any previously attached collision objects are removed to avoid interference.
- **Attach Object to End-Effector:** A `Attach bowl` Stage, shown in Fig.3.15, marks the object as attached using the `object_type` identifier. This ensures accurate collision checking throughout execution.
- **Lift Motion:** A Cartesian upward motion lifts the object vertically, computed in the base frame (`Lift bowl` in Fig.3.15). This reduces the risk of unintended collisions near the table.
- **Collision Filtering:** Collision constraints are toggled during motion. First, hand-object and object-octomap[5] collisions are temporarily allowed (`Allow collision object-octomap` in Fig.3.15), then disabled once the object is in a safe zone (`Dis-allow collision` Stages in Fig.3.15).

[5]An **Octomap** is a 3D occupancy grid mapping framework that uses octrees to represent the environment, thus, the obstacles in the Planning Scene.

- **Return to Home:** (`Go home` in Fig.3.15)A *MoveTo* Stage executes a joint-space trajectory to return the robot to its original configuration.
- **Detach Object:** Upon successful arrival, the object is detached from the planning scene and removed from collision checking (`De-attach bowl` in Fig.3.15).

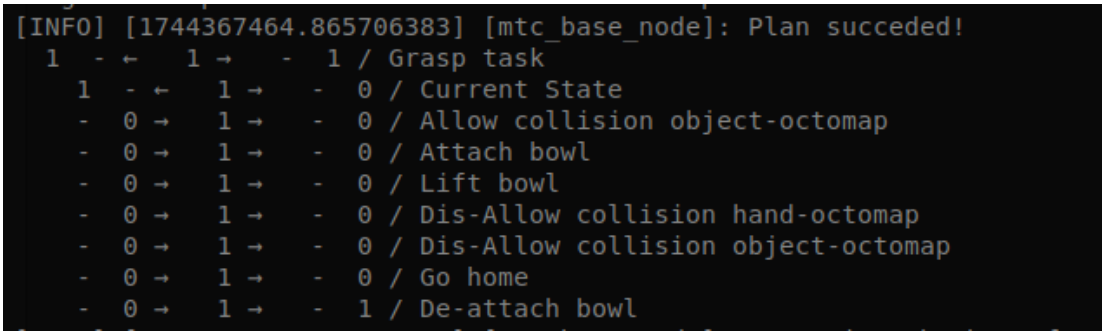


Figure 3.15: *MTC*'s output of **post-grasp** Task in case of grasping a *bowl*.

Each stage is built to tolerate transient planning failures by fallback paths in the Behaviour Tree. The node also validates planner outputs prior to execution and logs the full task status to facilitate debugging and runtime monitoring.

```
task_.clear();
task_ = this->createTask();
std::shared_ptr<const Solution> best_solution = task_.solutions().
    front();
task_.introspection().publishSolution(*best_solution);
task_.printState();
```

Code 3.14: *MTC* creating a Task and publishing a Solution

Architectural Highlights

Both *MTC* nodes share a common inheritance from `MTCBase` 3.14, which abstracts repetitive setup procedures such as planner configuration, node creation, and planning scene interfacing. Planning is performed using a mix of *OMPL* (sampling-based) and Cartesian planners, selected **adaptively per Stage**. Parameters such as velocity scaling and planner timeouts are configured via launch-time `YAML` files, promoting reusability and ease of tuning.

Crucially, this decomposition into atomic BT nodes and dedicated *MTC* tasks ensures:

- **Safety** through stage-wise validation and clear decision boundaries,
- **Responsiveness** by enabling retries or alternative plans at each point of failure,
- **Extensibility** by isolating functionality behind well-defined ROS2 interfaces.

In aggregate, this implementation fulfills the core operational requirements posed by **RQ2** and **RQ3**, delivering robust grasping behaviour in the face of real-world uncertainty.

3.7 Summary and Experimental Outlook

This chapter has outlined the full architectural and implementation spectrum of the Grasp-Anything-Model (GAM) pipeline, describing how modular perception, learning-based inference, and reactive motion planning were tightly integrated into a ROS2-based framework for real-world object grasping. Each stage, from perception to execution, was analysed through both theoretical and practical lenses, summarized in Table 4, linking the design rationale directly to the research questions **RQ1** (generalization under real-world perception constraints), **RQ2** (precision-efficiency trade-offs), and **RQ3** (reactive motion control under incomplete feedback).

The GAM pipeline comprises five core modules, summarized in Table 3.1:

- **Perception (Section 3.2):** A prompt-driven detection module based on GSAM, converting natural language prompts into spatially grounded binary masks using RGB data. This stage handles semantic localization and loads the spatial reasoning pipeline.
- **3D Reconstruction (Section 3.3):** A geometry-aware reconstruction stage using *MCC* to synthesize a complete object representation from the partial ("seen") point cloud. It outputs a centroid-aligned dense cloud, which grounds the physical object's 3D model.
- **Grasp Pose Detection (Section 3.4):** A geometry-centric inference module built around the *GPD* algorithm. It transforms the reconstructed cloud into a set of ranked grasp candidates using contact heuristics and reachability metrics.

- **Planning Scene Handling (Section 3.5):** A ROS2 BT node that converts the reconstructed cloud into a **Poisson mesh** and publishes it to the Planning Scene, ensuring accurate collision representation for downstream planners.
- **Motion Planning and Execution (Section 3.6):** A three-part planning and actuation system, developed using the *MTC* framework and *PlayMotion2* actions. It decomposes the grasp sequence into modular stages: pre-grasp positioning, gripper closure, and post-grasp retreat.

Module	Function	Tool/Model	Output
Perception	Segments object by prompt	GSAM	Binary mask
Reconstruction	Completes object geometry	MCC	Point cloud
Grasp Detection	Finds 6-DoF poses	GPD	Set of grasp poses
Planning Scene	Adds mesh to planning scene	Poisson + MoveIt2	Mesh
Motion Planning	Execute grasp	MTC + PlayMotion2	Action

Table 3.1: Overview of modules used in the grasping pipeline.

The pipeline is executed via BT that orchestrate inference and planning stages with modular fallback strategies and real-time reactivity. The blackboard pattern ensures data consistency, while the use of stateless ROS2 services supports runtime modularity.

Crucially, GAM is realized as a ROS2 workspace structured into seven packages:

- `gam_demo`: CLI-based action client for initializing the GAM pipeline.
- `gam_server`: Action server hosting the BT logic and ticking mechanism.
- `gam_detector`: Service for *GSAM*-based perception.
- `gam_3d_reconstructor`: Service wrapping *MCC* inference for 3D reconstruction.
- `gam_grasp_detector`: *GPD*-based grasp poses detection service server.
- `gam_bt`: Hosts all BT node definitions, utility classes, and planners.
- `gam_msgs`: Definitions for shared messages and actions (e.g., `GAM.action`).

Altogether, this design enables the GAM system to transform a high-level semantic goal (e.g., "*the black mug*") into a sequence of perceptual, inferential, and kinematic operations that are **modular**, **explainable**, and **generalizable**.

With the system architecture and implementation defined, the subsequent chapter turns toward **empirical validatio**n. Chapter 4 evaluates the GAM pipeline in terms of real-world performance, examining:

- Success rates across object types and scene configurations,
- Runtime efficiency and pipeline latency,
- System responsiveness and failure recovery capabilities,
- Limitations encountered during development and deployment.

These evaluations are structured to reflect the research questions defined in Chapter 1, providing quantitative and qualitative analysis of the GAM system's ability to generalize perception, synthesize reliable grasp candidates, and execute manipulation actions robustly in the face of real-world uncertainty.

Through this evaluation, we aim to substantiate the core hypothesis of this thesis: that modular, learning-enabled pipelines can bridge the gap between academic standalone models and reliable real-world robotic behaviour.

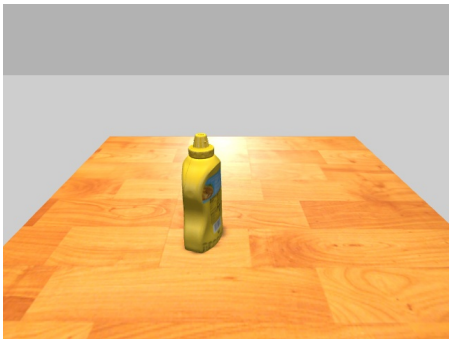
4

Results

This chapter presents a systematic evaluation of the GAM pipeline through both **simulation**-based and **real-world experiments**. Building upon the architectural principles and implementation strategies detailed in Chapter 3, the objective of this evaluation is to empirically assess the system's **grasping performance**, planning reliability, and runtime feasibility. The experiments are structured to validate the design hypotheses introduced earlier, particularly with respect to generalization across novel objects (**RQ1**), responsiveness under perceptual uncertainty (**RQ2**), and motion planning time via structured task planning (**RQ3**).

To this end, the evaluation follows a **quantitative** engineering-driven methodology, focusing on metrics such as grasp success rate, total pipeline latency (in seconds, unless specified otherwise), and task planning time. Trials were executed in two different settings: a controlled **Gazebo-based simulation** with a digital Tiago model, example shown in the figure 4.1a, and a physical deployment using a **real Tiago robot** (Fig. 4.1b). Each module of the pipeline was tested individually prior to full-pipeline execution, ensuring staged integration and validation.

The experiments simulate realistic manipulation tasks in which the robot is required to identify, reconstruct, and grasp objects of varying shape, size, and complexity. The input interface consists of natural language prompts issued by the user, which are grounded through multimodal perception before being con-



(a) Simulation robot's POV.

(b) Real-world robot's POV.

Figure 4.1: Experiments environments.

verted into executable motion plans. Success is defined by the robot's ability to correctly identify and grasp the object—returning to its "home" configuration with the object **securely in hand**.

The results of these experiments offer insight into the viability of GAM as a real-world grasping system, bridging the gap between high-level generalization and low-level hardware execution.

This chapter is structured as follows: Section 4.1 outlines the experimental setup and methodology, including simulation and real-world configurations, and trial design. Section 4.2 reports quantitative results across the defined evaluation metrics, while Section 4.3 discusses performance trends, strengths, and observed failure modes. Finally, Section 4.4 identifies core limitations of the current GAM implementation and outlines opportunities for future refinement. The whole findings are finally summarized in Section 4.5.

4.1 Experimental Setup

Simulation Environment

Experiments in simulation were conducted using a **Tiago model** within a Gazebo world that replicates a tabletop grasping scenario as noticeable in Figures 3.4a-4.1a. The simulated environment included a table and various **3D object meshes** sourced from open-source datasets such as the *YCB-SDF* [31] and *Ycb-tools* [58]. These repositories provide detailed models of common household items, which were randomly placed on the table in varying orientations

and configurations (e.g., upright, lying down, partially occluded, etc.).

Simulation served multiple roles:

- **Module Validation:** Each stage (e.g., Perception, 3D Reconstruction, Grasp Detection) was first verified in isolation to ensure expected output.
- **Pipeline Integration:** After individual validation, the complete pipeline was tested to assess system-level coordination.
- **Iterative Debugging**The simulation enabled safe repetition and parameter tuning, especially useful when real-hardware access was limited.

Real-World Deployment

The physical experiments were performed on a Tiago robot (used both Tiago single arm and Tiago++) equipped with an RGB-D Astra camera [43]. The test area consisted of a table with one or more objects placed at varying distances and angles from the robot, examples see, examples are Fig.4.1b and Fig.1.3. Lighting conditions were kept consistent to ensure image quality and to avoid variability in segmentation and depth estimation.

The real-world deployment aimed to:

- **Validate** full-pipeline performance under actual sensor noise and latency.
- **Observe** and **debug** hardware-specific issues, such as misalignment between RGB and depth points (D).
- **Demonstrate** the applicability of GAM to real, unscripted interactions.

Before executing the full pipeline, a critical debugging phase addressed a **misalignment** issue between the RGB and depth streams of the Astra camera. This misalignment led to incorrect 3D reconstructions and ultimately poor grasp targeting. Once resolved, the pipeline produced satisfying and consistent results in real-world scenarios, though access to the robot was constrained.

Each **grasping trial** followed this procedure:

1. The robot is positioned in front of the table, with objects placed either alone or among distractors.
2. A natural language prompt (e.g., "*the apple*") is issued to initiate the pipeline.
3. The GAM pipeline is triggered via ROS2 action goal.
4. The system performs Perception, 3D reconstruction, grasp pose detection, planning and execution.
5. A trial is considered **successful** if the robot returns to the "*home*" pose with the object securely in its gripper.

Due to hardware access constraints and time limitations, **statistical significance testing was not feasible**. However, trends were observed across multiple diverse trials to qualitatively assess performance across object types and prompt complexities.

As described in Section 4.1, each object was tested over **5-10 trials** with varied prompts and spatial configurations. Although the testing did not focus on a statistically rigorous sampling plan, diversity in object type and pose was emphasized to probe generalization.

Prompts varied in specificity and naturalness, such as (with the goal of grasping a mug):

- "Mug",
- "Blue mug",
- "The object on the right",
- "The pear" (expecting a `FAILURE` at detection level).

Objects used in **simulation** included: shampoo bottle, mug, water bottle, box, pear, drill and bowl. Those used in **real life** included: small and tall cylinders, joystick, ball, cube, and mug. Object orientation and occlusion were not fixed, introducing realistic scene variability.

This experimental framework ensures a **representative**, if not exhaustive, evaluation of GAM's real-world readiness and robustness to perceptual ambiguity and physical variance.

4.2 Evaluation Metrics & Success Criteria

To evaluate the effectiveness and practicality of the GAM pipeline, three key performance metrics were defined, each corresponding to one of the research questions outlined in Chapter 2:

- **Grasp Success Rate (RQ1 – Generalization):** Defined as the percentage of trials in which the robot returned to its "*home*" configuration while securely holding the object. A grasp was considered successful if, at the end of the motion execution, the object was visibly retained in the end-effector without slippage or drop. This metric captures the pipeline's end-to-end effectiveness in generalizing to different object types, orientations, and prompt formulations.
- **Pipeline Latency (RQ2 – Responsiveness and Runtime Feasibility):** Measured as the total time elapsed from the moment the user sends a natural language prompt (action goal) to the completion of the final motion execution. This includes:
  - Perception time (prompt grounding + detection & segmentation),
  - Reconstruction time (via MCC model inference),
  - Grasp Detection time (*GPD*),
  - Planning time (*MTC* task generation and solving),
  - Execution time (arm movement including pre- and post-grasp phases). This metric reflects the system's real-time applicability under computational constraints, especially given the decision to run most (all) modules on CPU.
- **Planning time (RQ3 – Modular Planning Robustness):** Refers specifically to the duration required by *MoveIt Task Constructor (MTC)* to compute feasible motion plans for both the approach and post-grasp tasks. It quantifies the pipeline's ability to produce viable plans under physical and kinematic constraints, serving as a proxy for planning robustness and fallback effectiveness.

Beyond the primary metrics, qualitative observations were made to understand the effects of contextual and environmental variables:

- **Prompt Specificity:** As noted in Section 4.1, prompt variability (e.g., "*blue mug*" vs. "*the object on the right*") was tracked to observe the impact of linguistic specificity on detection accuracy.
- **Object Pose and Scene Complexity:** Trials were designed to include standing, lying, and partially occluded objects. Failures were frequently associated with **low-profile objects** (e.g., small ball, joystick) due to **collision risks** or failed grasp pose detection.

- **Error Propagation Chains:** Failure in early modules—such as poor segmentation or misaligned depth information—was observed to cascade downstream, degrading reconstruction accuracy and resulting in invalid or unreachable grasp candidates. This emphasized the importance of input quality and the value of stateless, retry-capable BT nodes.

Two execution environments were considered:

1. **CPU-only Deployment:** Reflective of real-world robotic deployments. Pipeline execution times ranged from ~**50s to 1 minute per grasp**, dominated by perception and reconstruction stages.
2. **GPU-accelerated Inference (Desktop Testing):** Used to benchmark upper-bound responsiveness. Pipeline execution was consistently under ~**12-17 seconds**, with perceptual modules completing in ~3–5 seconds, sa evidenced in 4.1.

Each object was tested with **5-10 trials**, using varied prompts. Scenes were not scripted, allowing random placement and natural occlusion. Moreover, the trials were considered **binary success/failure**, without intermediary scoring.

While the scope of the experiments does not support formal statistical testing, consistent trends across object classes and scenes provide qualitative insight into the system's performance, reported in Section 4.3. These observations enable qualitative yet structured insights into GAM's functional behaviour across varying task conditions.

4.3 Results

4.3.1 Grasp Success Rate

The GAM pipeline was evaluated across two environments—simulation and real-world deployment—on a set of diverse objects varying in geometry, size, and texture. Each trial was considered a binary success or failure based on whether the robot returned to its home configuration while retaining the object securely in the gripper. The results of the trials are summarized in Figs.4.2-4.3, regarding the real-world and simulation environment, respectively.

Observations

High-performing objects include the tall cylinder, water bottle, and standard geometries (e.g., mug, shampoo bottle). These items had clean shapes and clear

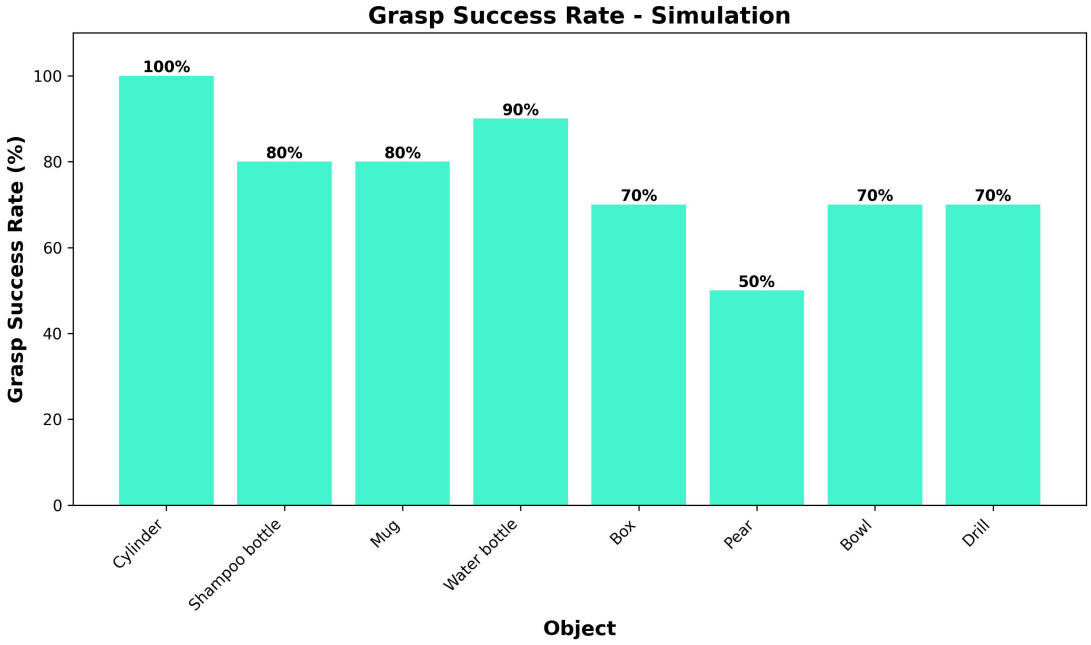


Figure 4.2: Simulation Grasp Success Rates.

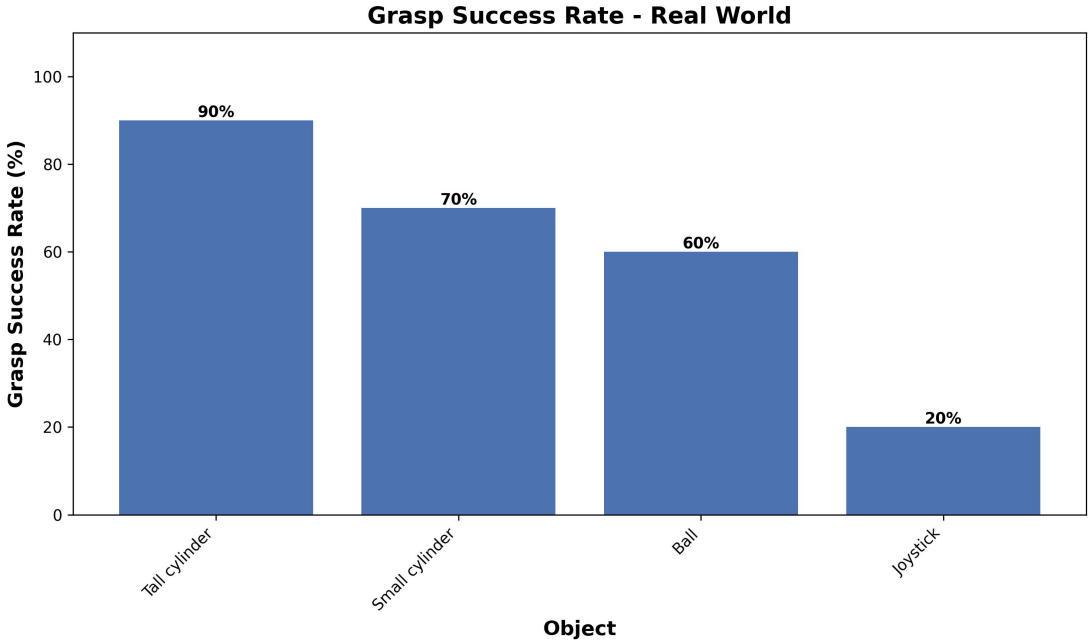


Figure 4.3: Real-world Grasp Success Rates.

point clouds, aligning well with the grasp detection and motion planning logic. The **failures** clustered around three main issues:

1. **Low-profile** shapes (e.g., pear, joystick), which led to collision-prone grasp planning or unstable grasping pose configurations.
2. **Poor segmentation** in ambiguous scenes, especially when objects were occluded and/or visually similar.
3. **Short objects** near the table surface caused the *MTC* to reject planned trajectories due to expected collisions. This was particularly visible in small cylinder, ball and joystick trials, as shown in Fig.4.3.

The system successfully demonstrated **zero-shot generalization** across a wide set of object categories without requiring object-specific tuning or retraining. Prompts such as "*blue mug*" or "*the mug on the right*" were parsed and executed without prior object labels, underscoring the effectiveness of the prompt-grounded segmentation module and its capacity to detect **unseen object classes (RQ1)**.

While the total number of trials per object remains **limited**, the consistency of results across conditions highlights the modular reliability of GAM and its ability to **generalize**. Future work should include statistically rigorous experiments on cluttered scenes and larger trial volumes to formalize these findings.

4.3.2 Pipeline Runtime Breakdown

Although the Grasp-Anything Model (GAM) was designed with an emphasis on modularity and runtime feasibility, empirical testing revealed that real-time performance remains a significant challenge on integrated pipelines. This section addresses **RQ2** (runtime feasibility) and **RQ3** (planning responsiveness) by providing an estimated breakdown of pipeline latency across its key stages, measured during integrated system runs. Runtime estimates, depicted in Table 7, were derived from repeated trials using both CPU-only and GPU-accelerated configurations, with each component invoked as part of a single ROS2 action flow.

Observations

The perception and reconstruction stages, as evidenced in Table 4.1, together account for **over 70% of the total runtime** on CPU. These stages rely on deep

Pipeline Stage	**Estimated Runtime (CPU)**	**Estimated Runtime (GPU)**
Perception	$\sim 27s$	$\sim 4s$
3D Reconstruction	$\sim 12s$	$\sim 3s$
Grasp Detection	$\sim$ 1–2s	$\ll \sim 1s$
Motion Planning	$\sim$ 2–5s	$\sim 1s$
Motion Execution	$\sim$ 5–8s	$\sim$ 5–8s
Total	**∼50–60s**	**∼12–17s**

Table 4.1: Pipeline Runtime Breakdown on CPU and GPU

learning inference without GPU acceleration, highlighting a fundamental trade-off between model expressiveness and deployment feasibility on embedded system. When run locally on a GPU-equipped machine (for benchmarking purposes), the total end-to-end time was **reduced by more than 4×**. With perception running in ∼4 seconds and subsequent stages completing near-instantaneously, this demonstrates that GAM is capable of near real-time responsiveness — but only under hardware conditions not typical of real robotic platforms.

The adoption of ***MoveIt Task Constructor (MTC)*** proved highly effective in reducing planning overhead. Unlike traditional ROS2 action pipelines—which typically require multiple sequential action goals (e.g., open gripper, move arm, compute IK, etc.)—*MTC* handles multi-stage planning as a single structured problem. This results in **more consistent runtimes** and **fewer execution failures**, especially in geometrically constrained scenes, addressing **RQ3**.

One of the core goals of this thesis was to design a grasping system that could operate in real time. The final outcome, however, reflects a common tension in robotics research: integrating high-capacity perception models with resource-limited hardware introduces performance ceilings that cannot be trivially solved without hardware upgrades or algorithmic simplification. While a grasp every 50–60 seconds may seem slow, it is the product of integrating sate-of-the-art **generalist models**—such as *Grounding-DINO*, *SAM*, and *MCC*—into a modular pipeline that remains explainable, interpretable, and reusable. The **latency trade-off** becomes a conscious design decision: by prioritizing generalization and modular validation over brute-force optimization, GAM preserves scientific transparency and architectural flexibility.

The GAM pipeline does not **yet** meet the strict thresholds of real-time grasping (<5s end-to-end), particularly under CPU-only constraints. However, its modular architecture enables fine-grained profiling and targeted optimization of each stage. Future work, addressed in Section 5.2, could explore lightweight model distillation, ONNX implementation, or early-exit mechanisms to reduce latency. Nonetheless, the current results validate GAM's **feasibility on real hardware** and affirm its value as a **foundational** framework for scalable, perception-driven grasping in robotics.

4.4 Limitations

While the Grasp-Anything Model (GAM) demonstrates promising results in both simulated and real-world environments, several limitations emerged during development and evaluation. These limitations reflect both **hardware constraints** and **algorithmic trade-offs** inherent to the system's modular architecture.

A primary limitation concerns the quality and reliability of the depth data provided by the Orbbec Astra RGB-D sensor. During experiments, fluctuations in the returned point clouds were observed across repeated frames, a phenomenon informally described as "***blinking point clouds***", visible in Fig. 1.3 where the small football's depth points are not present in the cloud. These inconsistencies introduced incomplete or noisy geometric reconstructions, particularly in scenes with reflective or texture-less surfaces, to mitigate this, the `cloudFillsMask()` utility function is implemented as highlighted in Code 3.7. Moreover, the sensor struggled to collect depth data when objects were positioned either **too close** to the lens—resulting in segmentation without 3D geometry—or **too far** from the robot, such as outside the robot's workspace (e.g., when placed near the floor with the torso fully extended). In such cases, the reconstruction algorithm produced **coarse approximations** of object shape, leading to degraded grasp detection and motion planning outcomes.

Another critical limitation emerged from the **linguistic interface** used to drive the system. The grounding module, based on Grounded -SAM (introduced in Section 2.2.4), proved **effective** for common prompts like "blue mug" or "the object on the right." However, more **linguistically complex** or **spatially am-**

biguous prompts, such as those involving positional descriptions or unfamiliar object names, often failed to produce reliable detections. Similarly, **very small objects**—typically under two to three centimeters in size—were frequently overlooked by the perception module, highlighting a floor in the detectable object scale. Conversely, **larger** or **more distant** objects often required finer point cloud resolution (granularity) to be accurately reconstructed, which in turn increased inference latency—a clear trade-off between precision and computational cost.

Challenges were also encountered in the grasp planning and execution stages. In particular, **low-profile objects** resting close to the table surface—such as small balls or joysticks—led to two recurring failure modes. First, the *Grasp Pose Detection (GPD)* module occasionally proposed grasp poses from underneath the object, which either resulted in collisions with the table or planning failures. Second, the close **proximity to the tabl**e introduced difficulties for the motion planner in computing feasible, **collision-free** trajectories. Although some mitigation was achieved by including contextual point cloud data from nearby surfaces (e.g., the points within a radius 'r' from object's centroid), this reduced the selectivity of the grasp detection process, occasionally causing the system to confuse overlapping objects.

From a systems perspective, the pipeline's modular structure—while beneficial for transparency and debugging—introduced potential for **error propagation**. A failure in one module, such as inaccurate segmentation, could not be corrected downstream, ultimately **leading to task failure**. While the use of Behavior Trees allowed for basic fallback and retry strategies, recovery from major errors still required restarting the pipeline, leaving margin of improvements as stated in Section 5.2.

Finally, the question of **real-time feasibility** remains a central limitation. Despite being designed for CPU-only deployment, the complete grasping cycle required ~**50-60** seconds to complete under real-world conditions. While GPU-accelerated benchmarks demonstrated that near real-time responsiveness (e.g., ~12-15s) was achievable under **ideal conditions**, these scenarios do not reflect the constraints of typical robotic hardware. Furthermore, the number of real-world trials conducted during this study was constrained by limited hardware access, reducing the **statistical confidence** of the observed results and limiting

generalization.

Together, these limitations highlight the **practical challenges** of integrating high-capacity vision and planning modules into a general-purpose, interpretable pipeline for robotic manipulation. They also emphasize the importance of considering sensor limitations, linguistic generalization, and system latency in the design **of real-world grasping systems**.

4.5 Summary of Findings

The evaluation of the Grasp-Anything Model (GAM) revealed a distinct balance between generalization, modular reliability, and operational feasibility. Across both simulated and real-world deployments, GAM demonstrated the ability to interpret natural language prompts, identify and segment target objects, reconstruct partial 3D shapes, and execute grasping motions with a measurable degree of success. These capabilities validate the system's core architectural principles: **modularity**, **generalizability**, and **explainability**.

With respect to **grasp success rate**, GAM achieved robust performance across a variety of object geometries, including cylindrical, box-shaped, and household items, with success rates reaching up to ~80-90% in favourable conditions. Importantly, the system generalized to novel, unlabelled objects **without explicit training**, confirming the viability of its prompt-driven perception and reconstruction modules (**RQ1**). However, low-profile and occluded objects consistently challenged the grasp planner, often triggering planning failures or unstable grasps. These observations underscore the pipeline's capacity to generalize across task domains, while also identifying geometric and perceptual edge cases that require further refinement.

In terms of **runtime feasibility**, the system's end-to-end latency on CPU platforms (~50–60 seconds) fell short of real-time grasping standards. The majority of this delay stemmed from the perception and reconstruction stages, where deep models such as *Grounded-SAM* and *MCC* were executed without GPU acceleration. Benchmarking on GPU reduced the total pipeline time to under 20 seconds, illustrating the architecture's potential for responsiveness in less resource-constrained deployments (**RQ2**). Additionally, the adoption of the *MoveIt*

Task Constructor (*MTC*) resulted in efficient motion planning even in geometrically constrained scenes, validating the use of structured, task-level planning modules (**RQ3**).

The results also highlighted the importance of system-level integration and the cumulative effect of upstream failures on overall performance. For instance, segmentation inaccuracies or poor point cloud acquisition due to sensor limitations **consistently degraded downstream** reconstruction and grasp prediction. The use of BTs mitigated some of these cascading failures, but recovery remained partial, and the need for more adaptive fallback mechanisms was evident.

In summary, the GAM pipeline succeeded in demonstrating a modular and interpretable grasping system that performs well across a range of unstructured scenarios, despite not **yet** achieving full real-time deployment or high-volume statistical validation. The findings presented in this chapter reinforce the feasibility of GAM as a general-purpose, CPU-deployable grasping framework and provide a concrete basis for future optimization efforts in both algorithmic and hardware domains.

The next and final chapter draws from these empirical insights to critically reflect on the contributions of this work, identify its broader implications, and propose future directions that can elevate GAM from a proof-of-concept into a deployable, scalable grasping system.

5 Conclusions

5.1 Overview

This final chapter concludes the thesis by reflecting on the broader implications of the work and outlining directions for future development. GAM's greatest contribution is its rejection of monolithic design. By decoupling perception (*GSAM*), reconstruction (*MCC*), and planning (*MTC*), it creates a 'plug-and-research' architecture where individual failures can be diagnosed and replaced without systemic collapse. The Grasp-Anything Model was introduced as a **foundational ROS2 grasping pipeline**, designed not to outperform all alternatives, but to provide a modular, interpretable, and deployable framework for vision-based manipulation in realistic robotic settings. Building upon this system-level contribution, the following sections present prospective enhancements that could extend GAM's capabilities, improve runtime feasibility, and increase robustness in real-world applications.

5.2 Future Work

While the GAM successfully integrates several state-of-the-art components into a modular grasping pipeline, its current implementation leaves significant room for **functional**, **architectural**, and **runtime refinement**. The modular nature of GAM is not only a strength in its current form, but also an enabler of **extensibility**—allowing researchers to build upon this foundation without rigid

design constraints. Several avenues for future work are outlined below.

System Optimization and Runtime Efficiency

Reducing the end-to-end execution time remains one of the most critical challenges for GAM. Future work could focus on the integration of lightweight inference engines such as **ONNX Runtime** [10], which enables accelerated model execution on CPU by converting deep learning models (e.g., *MCC*, *Grounded-SAM*) into a highly optimized graph format. Pre- and post-processing steps across modules can also be revisited to eliminate bottlenecks and redundant operations, offering further gains in latency without compromising modularity.

Behaviour Tree Extension and Autonomy

While BTs currently orchestrate module sequencing, their **complexity** can be significantly **increased** to support fallback strategies, dynamic replanning, and conditional recovery. For example, in cases where the object lies outside the manipulator's workspace, navigation behaviour could be integrated to let the robot reposition accordingly before initiating the grasp (e.g., placing itself in front of the table). Similarly, failed detection or planning attempts could trigger context-aware retries rather than full-pipeline resets. These capabilities would enhance GAM's autonomy and its resilience to unpredictable deployment scenarios.

Feedback-Driven Grasp Execution

At present, grasp execution is divided into three independent phases—approach, close, and retreat—without **reactive** grasp feedback between them. Introducing **grasp-success feedback** from the end-effector (e.g., using force sensing or finger-position verification) would enable execution to become more fluid and adaptive, incorporating the whole motion in a **single *MTC* Task**. This would allow the planner to merge these stages into a single closed-loop action, enhancing safety, execution speed, and overall grasp reliability.

Model Flexibility and Module Swapping

One of GAM's key affordances is its agnosticism to specific perception or planning models. Future research could explore **swapping** out or **benchmarking**

alternative architectures, such as replacing *GSAM* or *MCC* with faster perception or reconstruction methods or evaluating transformer-based grasp synthesis models. This not only encourages research reuse but also allows for rapid prototyping without re-engineering the entire pipeline.

Robustness and Generalization

Although GAM performs reliably across many object types, its robustness under adverse conditions (e.g., extreme occlusion, reflective surfaces, or low-texture objects) remains **limited**. Future efforts should include domain adaptation strategies, uncertainty-aware planning, and sensor fusion approaches to improve GAM's resilience. Additional work could also explore **prompt diversification** and **language grounding improvements**, enabling the system to handle more abstract or spatially rich commands.

Extended Functional Scope

Currently, GAM is restricted to single-object grasping in static scenes. Future work could extend its capabilities toward **multi-object reasoning**, sequential manipulation (e.g., **pick-and-place chains**), or interaction-based tasks such as tool use or human-robot handovers. Thanks to its modular foundation, these extensions can be layered incrementally, each becoming a new research opportunity without disrupting the pipeline's existing structure.

5.3 Final Remarks

This thesis set out not to deliver a one-size-fits-all grasping solution, but to define a reusable, interpretable, and **real-world grounded framework for robotic manipulation**. GAM—the Grasp-Anything Model—represents a shift from opaque, monolithic pipelines toward modular, diagnosable systems. Each of its components, from prompt-driven perception to single-view reconstruction and task-aware motion planning, contributes to a broader design philosophy: modularity not as convenience, but as necessity for scalable, transparent, and trustworthy robotic systems.

What began as an attempt to integrate cutting-edge vision models into a ROS2-based pipeline quickly evolved into a deeper exercise in engineering discipline.

The real-world constraints of limited hardware, noisy sensors, and stiff integration logic forced repeated rethinking of what "**feasible**" really means in robotics. GAM taught me the value of robust systems over flashy benchmarks, and that design elegance often lies not in sophistication but in clarity, recoverability, and adaptability.

From debugging a misaligned depth stream to testing zero-shot object detection with vague prompts, each technical challenge uncovered a deeper truth: in robotics, success is rarely about any one model or algorithm. It is about the ability to **fail locally**, **recover modularly**, and **scale systematically**. GAM is not a finished product—it is a **foundation**. A proof-of-concept for what a general-purpose, ROS2-native grasping pipeline can look like when designed with both experimentation and deployment in mind.

In building GAM, I did not just write code—I learned how to think like a systems robotic software engineer. I came to appreciate trade-offs between speed and interpretability, performance and portability, complexity and maintainability. This project gave me the opportunity to work at the edge of current research while staying rooted in the practical realities of deploying on a real robot in an industrial setting. The lessons are not only technical but deeply methodological: validate often, isolate faults, and build systems that invite change rather than resist it.

Looking forward, GAM offers a platform for continued exploration. Its modular structure welcomes new models, improved behaviour logic, and tighter feedback integration. It is an invitation—not a conclusion—for the next generation of roboticists to build systems that are not only smart, but understandable, repairable, and real-world ready.

In that sense, the GAM pipeline is not an end—it is a **beginning**.

References

[1] Aishwarya Agrawal et al. *VQA: Visual Question Answering*. 2016. arXiv: 1505.00468 [cs.CL]. URL: https://arxiv.org/abs/1505.00468.

[2] Matthew Berger et al. "A Survey of Surface Reconstruction from Point Clouds". In: *Computer Graphics Forum* 35.2 (2016), pp. 301–329. DOI: 10.1111/cgf.12802. URL: https://doi.org/10.1111/cgf.12802.

[3] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. 2023. arXiv: 2307.15818 [cs.RO]. URL: https://arxiv.org/abs/2307.15818.

[4] Angel X. Chang et al. *ShapeNet: An Information-Rich 3D Model Repository*. 2015. arXiv: 1512.03012 [cs.GR]. URL: https://arxiv.org/abs/1512.03012.

[5] Xi Chen et al. *PaLI: A Jointly-Scaled Multilingual Language-Image Model*. 2023. arXiv: 2209.06794 [cs.CV]. URL: https://arxiv.org/abs/2209.06794.

[6] Yuhao Chen et al. "MetaGraspNet: a large-scale benchmark dataset for vision-driven robotic grasping via physics-based metaverse synthesis". In: *arXiv preprint arXiv:2112.14663* (2021).

[7] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. "Per-Pixel Classification is Not All You Need for Semantic Segmentation". In: 2021.

[8] Angela Dai et al. *ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes*. 2017. arXiv: 1702.04405 [cs.CV]. URL: https://arxiv.org/abs/1702.04405.

[9] Amaury Depierre, Emmanuel Dellandréa, and Liming Chen. *Jacquard: A Large Scale Dataset for Robotic Grasp Detection*. 2018. arXiv: 1803.11469 [cs.RO]. URL: https://arxiv.org/abs/1803.11469.

[10] ONNX Runtime developers. *ONNX Runtime*. https://onnxruntime.ai/. Version: x.y.z. 2021.

[11] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. "ACRONYM: A Large-Scale Grasp Dataset Based on Simulation". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 9560–9568. DOI: 10.1109/ICRA48506.2021.9560844. URL: https://doi.org/10.1109/ICRA48506.2021.9560844.

[12] Davide Faconti. *BehaviorTree.dev: A platform for Behavior Trees*. Accessed: 2025-05-21. 2024. URL: https://www.behaviortree.dev/.

[13] Hao-Shu Fang et al. "AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains". In: *IEEE Transactions on Robotics (T-RO)* (2023).

[14] Hao-Shu Fang et al. "GraspNet-1Billion: A Large-Scale Benchmark for General Object Grasping". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition(CVPR)*. 2020, pp. 11444–11453.

[15] Hao-Shu Fang et al. "Robust grasping across diverse sensor qualities: The GraspNet-1Billion dataset". In: *The International Journal of Robotics Research* (2023).

[16] Christelle Ferrari and John Canny. "Planning Optimal Grasps". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 1992, pp. 2290–2295.

[17] Brian Flynn et al. *Developing Modular Grasping and Manipulation Pipeline Infrastructure to Streamline Performance Benchmarking*. 2025. arXiv: 2504.06819 [cs.RO]. URL: https://arxiv.org/abs/2504.06819.

[18] Michael* Görner et al. "MoveIt! Task Constructor for Task-Level Motion Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[19] Thibault Groueix et al. *AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation*. 2018. arXiv: 1802.05384 [cs.CV]. URL: https://arxiv.org/abs/1802.05384.

[20] Agrim Gupta, Piotr Dollár, and Ross Girshick. *LVIS: A Dataset for Large Vocabulary Instance Segmentation*. 2019. arXiv: 1908.03195 [cs.CV]. URL: https://arxiv.org/abs/1908.03195.

[21] Zixuan Huang et al. "PointInfinity: Resolution-Invariant Point Diffusion Models". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2024.

[22] Shun Iwase et al. *ZeroGrasp: Zero-Shot Shape Reconstruction Enabled Robotic Grasping*. 2025. arXiv: 2504.10857 [cs.RO]. URL: https://arxiv.org/abs/2504.10857.

[23] Mrinal Kalakrishnan et al. "STOMP: Stochastic trajectory optimization for motion planning". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 4569–4574. DOI: 10.1109/ICRA.2011.5980280.

[24] Dmitry Kalashnikov et al. *QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation*. 2018. arXiv: 1806.10293 [cs.LG]. URL: https://arxiv.org/abs/1806.10293.

[25] David Kappler, Jeannette Bohg, and Stefan Schaal. "Leveraging Big Data for Grasp Planning". In: *Autonomous Robots* 42.1 (2018), pp. 1–20. DOI: 10.1007/s10514-017-9646-9. URL: https://doi.org/10.1007/s10514-017-9646-9.

[26] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[27] Alexander Kirillov et al. "Segment Anything". In: *arXiv:2304.02643* (2023).

[28] Kevin Kleeberger, Reiner Bormann, Wolfgang Kraus, et al. "A Survey on Learning-Based Robotic Grasping". In: *Current Robotics Reports* 1 (2020), pp. 239–249. DOI: 10.1007/s43154-020-00021-6. URL: https://doi.org/10.1007/s43154-020-00021-6.

[29] Anna Konrad, John McDonald, and Rudi Villing. *GP-net: Flexible Viewpoint Grasp Proposal*. 2023. arXiv: 2209.10404 [cs.RO]. URL: https://arxiv.org/abs/2209.10404.

[30] Anna Konrad, John McDonald, and Rudi Villing. "VGQ-CNN: Moving Beyond Fixed Cameras and Top-Grasps for Grasp Quality Prediction". In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2022, pp. 1–8. DOI: 10.1109/ijcnn55064.2022.9892763. URL: http://dx.doi.org/10.1109/IJCNN55064.2022.9892763.

[31] kyouma9s. *ycb_gazebo_sdf: YCB models for Gazebo (.sdf)*. https://github.com/kyouma9s/ycb_gazebo_sdf. Accessed: 2025-05-20.

[32] Zhiqian Lan et al. "SEPT: Towards Efficient Scene Representation Learning for Motion Prediction". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=efeBC1sQj9.

[33] Shilong Liu et al. "Grounding dino: Marrying dino with grounded pre-training for open-set object detection". In: *arXiv preprint arXiv:2303.05499* (2023).

[34] Jeffrey Mahler et al. "Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics". In: *https://berkeleyautomation.gi net/code.html?highlight=cite* (2017).

[35] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*. 2020.

[36] Ben Mildenhall et al. *NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis*. 2020. arXiv: 2003.08934 [cs.CV]. URL: https://arxiv.org/abs/2003.08934.

[37] A.T. Miller and P.K. Allen. "Graspit! A versatile simulator for robotic grasping". In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122. DOI: 10.1109/MRA.2004.1371616.

[38] Chaerin Min et al. *TSDF-Sampling: Efficient Sampling for Neural Surface Field using Truncated Signed Distance Field*. 2023. arXiv: 2311.17878 [cs.CV]. URL: https://arxiv.org/abs/2311.17878.

[39] Douglas Morrison, Peter Corke, and Jürgen Leitner. *EGAD! an Evolved Grasping Analysis Dataset for diversity and reproducibility in robotic manipulation*. 2020. arXiv: 2003.01314 [cs.RO]. URL: https://arxiv.org/abs/2003.01314.

[40] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Learning Robust, Real-Time, Reactive Robotic Grasping". In: *The International Journal of Robotics Research* 39.2-3 (2019), pp. 183–201. DOI: 10.1177/0278364919859066. URL: https://doi.org/10.1177/0278364919859066.

[41] Van Thanh Nguyen. "Constructing Force-Closure Grasps". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 1989, pp. 694–699.

[42] Alex Nichol et al. *Point-E: A System for Generating 3D Point Clouds from Complex Prompts*. 2022. arXiv: 2212.08751 [cs.CV]. URL: https://arxiv.org/abs/2212.08751.

[43] Orbbec. *Astra Series Structured-Light Camera*. https://www.orbbec.com/products/structured-light-camera/astra-series/. Accessed: 2025-05-20.

[44] Alessandro Palleschi et al. "Grasp It Like a Pro 2.0: A Data-Driven Approach Exploiting Basic Shape Decomposition and Human Data for Grasping Unknown Objects". In: *IEEE Transactions on Robotics* 39.5 (2023), pp. 4016–4036. DOI: 10.1109/TRO.2023.3286115.

[45] Jeong Joon Park et al. "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2019.

[46] Andreas ten Pas et al. *Grasp Pose Detection in Point Clouds*. 2017. arXiv: 1706.09911 [cs.RO]. URL: https://arxiv.org/abs/1706.09911.

[47] PickNik Robotics. *MoveIt Task Constructor Tutorial*. https://moveit.picknik.ai/humble/doc/examples/moveit_task_constructor/moveit_task_constructor_tutorial.html. Accessed: 2025-05-20.

[48] Aditya Ramesh et al. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125 [cs.CV]. URL: https://arxiv.org/abs/2204.06125.

[49] Nathan Ratliff et al. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 489–494. DOI: 10.1109/ROBOT.2009.5152817.

[50] Nikhila Ravi et al. "SAM 2: Segment Anything in Images and Videos". In: *arXiv preprint arXiv:2408.00714* (2024). URL: https://arxiv.org/abs/2408.00714.

[51] Jeremy Reizenstein et al. *Common Objects in 3D: Large-Scale Learning and Evaluation of Real-life 3D Category Reconstruction*. 2021. arXiv: 2109.00512 [cs.CV]. URL: https://arxiv.org/abs/2109.00512.

[52] Tianhe Ren et al. *Grounded SAM: Assembling Open-World Models for Diverse Visual Tasks*. 2024. arXiv: 2401.14159 [cs.CV].

[53] Mauricio A. Roa and Ricardo Suárez. "Grasp Quality Measures: Review and Performance". In: *Autonomous Robots* 38 (2015), pp. 65–88. DOI: 10.1007/s10514-014-9402-3. URL: https://doi.org/10.1007/s10514-014-9402-3.

[54] Mike Roberts et al. *Hypersim: A Photorealistic Synthetic Dataset for Holistic Indoor Scene Understanding*. 2021. arXiv: 2011.02523 [cs.CV]. URL: https://arxiv.org/abs/2011.02523.

[55] Ashutosh Saxena, Joel Driemeyer, and Andrew Y. Ng. "Robotic Grasping of Novel Objects Using Vision". In: *The International Journal of Robotics Research* 27.2 (2008), pp. 157–173. DOI: 10.1177/0278364908097119. URL: https://doi.org/10.1177/0278364908097119.

[56] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[57] Johannes Lutz Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*. 2016.

[58] sea-bass. *ycb-tools: Utilities for YCB object models*. https://github.com/sea-bass/ycb-tools. Accessed: 2025-05-20.

[59] Halperin D. Solovey K Salzman O. "Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 35. 2016. DOI: 10.1177/0278364915615688.

[60] Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (2012), pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[61] Martin Sundermeyer et al. "Contact-GraspNet: Efficient 6-DoF Grasp Generation in Cluttered Scenes". In: (2021).

[62] Ao Wang et al. *YOLOE: Real-Time Seeing Anything*. 2025. arXiv: 2503.07465 [cs.CV]. URL: https://arxiv.org/abs/2503.07465.

[63] Chao-Yuan Wu et al. "Multiview Compressive Coding for 3D Reconstruction". In: *arXiv preprint arXiv:2301.08247* (2023).

[64] Qiangeng Xu et al. "Point-nerf: Point-based neural radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 5438–5448.

[65] Daniel Yang et al. *Robotic Grasping through Combined Image-Based Grasp Proposal and 3D Reconstruction*. 2020. arXiv: 2003.01649 [cs.RO]. URL: https://arxiv.org/abs/2003.01649.

[66] Xumin Yu et al. "PoinTr: Diverse Point Cloud Completion with Geometry-Aware Transformers". In: *ICCV*. 2021.

[67] Nir Zabari and Yedid Hoshen. *Semantic Segmentation In-the-Wild Without Seeing Any Segmentation Examples*. 2021. arXiv: 2112.03185 [cs.CV]. URL: https://arxiv.org/abs/2112.03185.

[68] Hanbo Zhang et al. *Robotic Grasping from Classical to Modern: A Survey*. 2022. arXiv: 2202.03631 [cs.RO]. URL: https://arxiv.org/abs/2202.03631.

[69] Hao Zhang et al. "A Simple Framework for Open-Vocabulary Segmentation and Detection". In: *arXiv preprint arXiv:2303.08131* (2023).

[70] Fan Zhu et al. "Failure Handling of Robotic Pick and Place Tasks With Multimodal Cues Under Partial Object Occlusion". In: *Frontiers in Neurorobotics* 15 (Feb. 2021), p. 570507. DOI: 10.3389/fnbot.2021.570507. URL: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2021.570507/full.

[71] Joshua M. Zutell, David C. Conner, and Philipp Schillinger. "ROS 2-Based Flexible Behavior Engine for Flexible Navigation". In: *SoutheastCon 2022*. 2022, pp. 674–681. DOI: 10.1109/SoutheastCon48659.2022.9764047.

Acknowledgments

During the course of this Master's program, I had the opportunity to develop and implement a complete robotic grasping pipeline, from its architectural design to its final experimental validation on a real robotic platform. This journey has been both intellectually demanding and profoundly rewarding, as it required me to navigate the complexities of modular system integration, perception-based inference, and real-time robotic execution within the ROS2 ecosystem. Beyond the technical scope, this experience has significantly shaped my personal and professional development. It taught me the importance of perseverance, attention to detail, and critical thinking when faced with real-world challenges. More importantly, it allowed me to grow not only as an engineer, but also as a person. I am sincerely grateful for the opportunity to meet and collaborate with exceptional individuals, witness cutting-edge research environments first-hand, and immerse myself in a learning process that extended far beyond the boundaries of robotics itself.

I would like to express my deepest gratitude to my family: *Eleonora, Rosario, Antonietta, Elvira, Jolanda, Virginia*, whose unwavering support has been the foundation of my academic and life journeys. Their constant presence, trust, and belief in my decisions gave me the freedom to choose my own path and pursue what I truly believed in. They never imposed expectations, but instead empowered me with the confidence to explore, fail, learn, and grow. Their encouragement and generosity, both emotional and practical, have been instrumental in turning my aspirations into reality. They shape me into who I am and for that I am deeply thankful.

I am grateful to my closest friends, *Circ, Abba, Cast, PeppeG, Manu, and Sandro*, who have always been an essential part of my life, even when distance or time separated us. Our bond has never depended on proximity; each time we reunite, it feels as if not a single day has passed. Their kindness, unwavering presence,

and the purity of their affection continue to remind me of the value of genuine friendship. I cherish every moment we have shared and look forward to all those that are yet to come. Their warmth and willingness to welcome me back, to share time and experiences without hesitation, has been a source of deep comfort and joy throughout this journey.

I also want to thank the *Maronn* group, Francesco, Simone, Alessandro, Andrea, Giulio and Gianluca, along with *Ester, Rosalba and Martina*. You were the people I was lucky to meet during my time in Salerno, and together we managed to overcome the boredom of Fisciano, even for a short time, with affection and some beers that made the journey lighter. Whether it was going out for drinks, sharing the latest gossip, or choosing cards over books when we should have been studying, your presence brought genuine joy and balance to these years. Thank you for honest laughs, unfiltered conversations and for always being there without needing a reason.

A special thanks goes to my childhood friends, *Giuseppe and Federico*. Although distance keeps us from meeting often, every time we reconnect it feels as though no time has passed; we are simply children again. I am grateful for the kindness you have always shown, for the sincerity of our conversations that can touch on any aspect of life, and for the way that you are always ready to listen when it matters most. I appreciate the memories we have built and I deeply value the loyalty and quiet strength of our friendship.

I also would like to express my heartfelt gratitude to three of the kindest people I have had the privilege to meet: *Amin, Annah, and Navid*. Since our first encounter in Padua, they have welcomed me with warm smiles and acts of genuine kindness. I am thankful for the time we spent together studying, sharing meaningful and less meaningful conversations, and enjoying the meals they so generously prepared. Their presence has always been a source of comfort, whether through their support when needed or their willingness to share a spritz while complaining about something. Time may pass, but their rare and unwavering kindness remains a constant that I deeply admire.

A special thanks goes to *Alberto, my lawyer Marianna, and Silver*, my roommates, for a brief but unforgettable chapter. Although we shared only two months together, we managed to create moments that felt like home: from movie nights and concerts to aperitifs, long walks, and even a memorable trip through South of Italy. In a time and place where I could have felt displaced, you made me feel welcomed, supported, and seen. Thank you for always being ready with an

answer, a helping hand, a spare bed, or simply a night out when our paths cross again. I will always be grateful for the laughter, the fights (avvocà...) and the wine we shared.

To *Gabriel*, whose friendship has been a source of joy and lightness since the very first moment we met: thank you. Your smile, warmth, and easygoing presence always made it a pleasure to have you around. You not only welcomed me with your generosity, but also opened your home and, together with your mother, made me feel truly cared for. Your kindness, honesty, and availability, always ready with a thoughtful response or a helping hand, are qualities I deeply value. I am grateful for your friendship and for the moments we shared.

My deepest thanks also go to the extraordinary group of people I met during my Erasmus experience in Graz: *Adit, Aleksy, Aaron, Annalisa, Virginia, Manos, Alfonso, Giorgia, Lucas, Tate, Nandi, Nadine, Adam, Océane, and William*. Each of you became part of a vibrant mosaic of memories: Every night out, every trip, every barbecue, hike, and spontaneous adventure we shared were shaped by the presence of this beautiful and diverse group. We came from different corners of the world, each with their own stories and ways of seeing life, and this incredible mix allowed me to carry home a little piece of each of you. I learned so much simply by being surrounded by your uniqueness. A special mention must go to *Antonio, Lenche, and Kika*, who brought me even closer thanks to our stupid shared mindset, countless laughs and warm Spanish hospitality. Thank you for hosting me, showing me around, and being sources of joy, gossip, and brutally honest but always caring opinions. And finally, to the 'Sterialized' team -*Robin, Ma2, and Leo* - you are a family. We spent an enormous amount of time together, enough to build a connection so deep that we could understand each other without even speaking. Those bonds are rare, and I am grateful to have experienced that with you. The sheer amount of memories we packed into a single Erasmus feels infinite. To all of you, thank you for helping me grow, for letting me enter your cultures and your lives, and for making this period truly unforgettable.

To the friends I met in Spain:*Juan, Carlo, Marco, Mauro-Angel-Maria-Bogotà, Diana and Seb* thank you for being part of this journey. Your presence truly made that time golden. Whether it was crossing the entire city just to meet, heading to the most unexpected places simply to enjoy what awaited us, or sharing sangria and spontaneous trips, your company brought warmth, laughter, and meaning to every moment. A special thanks goes to *Juan*, whose loyalty, kindness,

and good heart never went unnoticed. Our meeting, so random, a month before I even arrived, now feels like it was meant to happen. Thank you for the conversations, the laughter, the countless hours spent together, and even the Call-of- Duty matches. You have been a constant and I am deeply grateful for your honesty, your understanding and the sincere friendship you have shown me throughout.

To the people I met at PAL Robotics: David, Thomas P., Thomas T., Oscar, Ilenia, Viviana, Aina, Daniel, *the Tiago team*, thank you for being with me throughout this journey. What an incredible group of individuals. Brilliant, driven, and kind: Your collective talent and openness made PAL not only a world-class robotics environment, but also a place where I felt constantly supported and inspired. Despite being surrounded by people with such advanced knowledge, I never felt out of place; rather, I felt encouraged to grow. Your willingness to answer questions with patience, care, and honesty helped me do what I did. I greatly appreciated every aspect of my time at PAL: from the work itself to lunches, walks to pick up food, breakfast chats, office parties, apple breaks, gossip sessions, football matches, ski trip, hikes and even basketball games at sunrise. Each experience was enriched by your presence. My sincere thanks also go to Isaac, Sergi, Edgar, the interns Victoria and Aroa, and to Lampros, Harsh, Noel, Andrea C. and Andrea P., each of you contributing meaningfully to this chapter of my life. Special thanks go to David. You are not only a great engineer, but an incredible leader. The strength and cohesion of the Tiago Team are a reflection of your calm, thoughtful guidance, and your quiet professionalism. Your depth of knowledge and the care with which you shared it were fundamental to this work. GAM would not be what it is without you standing behind it. You have a rare ability to make everyone feel at ease, checking on people's well-being, offering support without asking, and lifting the atmosphere both inside and outside PAL. For that and much more, I am truly grateful.

The Ancona crew, *Andrea, Giulia, Niki, Toti, Stefano, Angi and Leti*, thank you for the warmth and spontaneity with which you welcomed me during my time traveling in Ancona city. Your constant curiosity, kindness, and openness to explore made each encounter memorable, from discovering hidden spots to night walks in search of bioluminescence in Grotta. You made me feel like a true part of the pace and way of life in Ancona, as if I were one of you, not just a guest. Thank you for all the experiences, from shared study sessions to spots in random wildlife, the laughter, and for letting me be a marine biologist for the time

I am around.

A special thanks goes to *Giulia*, whom I met in Barcelona and with whom I shared nearly three months of daily life. Your sincerity, joy, and kindness made those days lighter. I truly appreciated every conversation and every laugh we had and every drink we shared, always accompanied by your genuine smile and good heart.

Last but not least, thank you to Eleonora, *Elly*. We spent quite a bit of time togheter, close and far, happy and sad, active and lazy, on vacation and studying, in a city or in another. All these moments with you taught me a lot. I would not have been as good as I am today if it weren't for you. Thank you for being so loving and caring, having this much patience, finding the way to reach me with your purest of the souls even across the ocean. Your support is unique and you maybe don't even know, even if not physically close, your words, suggestions, and spontaneous love are what shaped me. I will manage to make you understand how much I admire you, your way of seeing life, your willingness to explore and know the unknown, your way of being and relating to others. Thank you for being close even when we were apart or when i didn't deserve it, for being always spontaneous, for that amazing heartwarming smile and the care you show me. Can't wait to start a next chapter with you!