

DAVID RAZHIEL CERES ARROYO

TASK ALLOCATION AND SENSOR FUSION
LOCALIZATION FOR AUTONOMOUS SYSTEMS

TASK ALLOCATION AND SENSOR FUSION LOCALIZATION FOR AUTONOMOUS SYSTEMS

DAVID RAZHIEL CERES ARROYO



A Multi-Tool Allocation Approach for Optimized Weed Removal in Autonomous
Agriculture

June 2025 – classicthesis v4.8

David Raziel Ceres Arroyo: *Task Allocation and Sensor Fusion Localization for Autonomous Systems, A Multi-Tool Allocation Approach for Optimized Weed Removal in Autonomous Agriculture*, © June 2025

Ohana means family.
Family means nobody gets left behind, or forgotten.
— Lilo & Stitch

Dedicated to the loving memory of Rudolf Miede.

1939 – 2005

ABSTRACT

Short summary of the contents in English...a great guide by Kent Beck how to write good abstracts can be found here:

<https://plg.uwaterloo.ca/~migod/research/beck00PSLA.html>

ZUSAMMENFASSUNG

Kurze Zusammenfassung des Inhaltes in deutscher Sprache...

PUBLICATIONS

This is just an example.

This might come in handy for PhD theses: some ideas and figures have appeared previously in the following publications:

- [1] Tobias Isenberg, André Miede, and Sheelagh Carpendale. "A Buffer Framework for Supporting Responsive Interaction in Information Visualization Interfaces." In: *Proceedings of the Fourth International Conference on Creating, Connecting, and Collaborating through Computing (C⁵ 2006)*. IEEE, 2006, pp. 262–269. ISBN: 978-0-7695-2563-1.
- [2] Ulrich Lampe, Markus Kieselmann, André Miede, Sebastian Zöller, and Ralf Steinmetz. "A Tale of Millis and Nanos: On the Accuracy of Time Measurements in Virtual Machines." In: *Proceedings of the Second European Conference on Service-Oriented and Cloud Computing (ESOCC 2013)*. Springer, 2013, pp. 172–179. ISBN: 978-3-642-40650-8.
- [3] Ulrich Lampe, Qiong Wu, Ronny Hans, André Miede, and Ralf Steinmetz. "To Frag Or To Be Fragged – An Empirical Assessment of Latency in Cloud Gaming." In: *Proceedings of the Third International Conference on Cloud Computing and Services Science (CLOSER 2013)*. 2013, pp. 5–12. ISBN: 978-898-8565-52-5.
- [4] André Miede. "Theses and other Beautiful Documents with classictthesis." In: *TUGboat – The Communications of the T_EX Users Group* 31.1 (2010), pp. 18–20. ISSN: 0896-3207.
- [5] André Miede, Gökhan Şimşek, Stefan Schulte, Daniel F. Abawi, Julian Eckert, and Ralf Steinmetz. "Revealing Business Relationships – Eavesdropping Cross-organizational Collaboration in the Internet of Services." In: *Proceedings of the Tenth International Conference Wirtschaftsinformatik (WI 2011)*. Vol. 2. 2011, pp. 1083–1092. ISBN: 978-1-4467-9236-0.
- [6] Hsin-Yi Tsai, Melanie Siebenhaar, André Miede, Yu-Lun Huang, and Ralf Steinmetz. "Threat as a Service? Virtualization's Impact on Cloud Security." In: *IEEE IT Professional* 14.1 (2012), pp. 32–37. ISSN: 1520-9202.

Attention: This requires a separate run of `bibtex` for your `refsection`, e. g., `ClassicThesis1-blx` for this file. You might also use `biber` as the backend for `biblatex`. See also <http://tex.stackexchange.com/questions/128196/problem-with-refsection>.

*We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.*

— Donald E. Knuth [6]

ACKNOWLEDGMENTS

Put your acknowledgments here.

Many thanks to everybody who already sent me a postcard!

Regarding the typography and other help, many thanks go to Marco Kuhlmann, Philipp Lehman, Lothar Schlesier, Jim Young, Lorenzo Pantieri and Enrico Gregorio, Jörg Sommer, Joachim Köstler, Daniel Gottschlag, Denis Aydin, Paride Legovini, Steffen Prochnow, Nicolas Repp, Hinrich Harms, Roland Winkler, Jörg Weber, Henri Menke, Claus Lahiri, Clemens Niederberger, Stefano Bragaglia, Jörn Hees, Scott Lowe, Dave Howcroft, José M. Alcaide, David Carlisle, Ulrike Fischer, Hugues de Lassus, Csaba Hajdu, Dave Howcroft, Anonymous, Konrad Höffner, and the whole L^AT_EX-community for support, ideas and some great software.

Regarding LyX: The LyX port was intially done by *Nicholas Mariette* in March 2009 and continued by *Ivo Pletikosić* in 2011. Thank you very much for your work and for the contributions to the original style.

CONTENTS

I BUILDING THE TOOLS

1	INTRODUCTION	3
1.1	Background and Context	3
1.2	Problem Statement	4
1.3	Current Solutions	4
1.4	Proposed Solution	7
2	SIMULATION	9
2.1	The Robot	9
2.2	Simulation	9
2.2.1	URDF	10
2.2.2	SDF	10
2.2.3	Gazebo Plugins	11
2.2.4	Joint Control	12
2.2.5	Localization	13
2.2.6	Weed Detection	14

II THE SHOWCASE

3	TASK ALLOCATION AND OPTIMIZATION	17
3.1	Metrics	17
3.2	Heuristics	19
3.3	Graph Search	21
3.4	Optimization	25
3.5	Market-based	25

III APPENDIX

A	APPENDIX TEST	29
A.1	Configuration Files	29
A.2	Algorithms	31
A.3	Another Appendix Section Test	32

BIBLIOGRAPHY	33
--------------	----

LIST OF FIGURES

Figure 1.1	Multi-robot task allocation classification. Source [4]	4
Figure 2.1	Nuga Platform	9
Figure 2.2	Simulator Components	10
Figure 2.3	Robot definition using URDF	11
Figure 2.4	Weed Infestation Example	12
Figure 2.5	Implement Tool (IT) Workspace Layout	13
Figure 2.6	ROS Interface for back gantry control	13
Figure 3.1	Problem Layout Dimentions	18
Figure 3.2	Mission Dashboard	18
Figure 3.3	Heuristic Algorithm	20
Figure 3.4	Suboptimal solution computed using Heuristic	21
Figure 3.5	Optimal solution	21
Figure 3.6	Node and cost representation	22
Figure 3.7	Node and cost representation	24
Figure 3.8	Mission Dashboard	24

LIST OF TABLES

Table 1.1	Task allocation approaches	5
Table 1.2	Task allocation approaches	6
Table 1.3	A Comparison of different approaches to Task Allocation (TA), Source [4]	7

LISTINGS

Listing A.1	Weed Infestation World config example	29
Listing A.2	ROS2 config example	30
Listing A.3	A floating example (listings manual)	32

ACRONYMS

- TA Task Allocation
SVCA Shapley Value Clustering Algorithm
CBDTA Consensus-Based Distributed Task Allocation Algorithm
VDKM Voronoi Diagram-Based, K-Means Algorithm
CBBA Consensus-Based Bundle Algorithm
CBPAEA Consensus Based Parallel Auction and Execution Algorithm
GRU Gated Recurrent Units
MLP Multi-Layer Perceptron
EDACAM Encoder Decoder Architecture with Cross Attention Mechanism
GNN Graph Neural Network
CNN Convolutional Neural Network
PSO Particle Swarm Optimization Algorithm
GTODT Group Theory and Optimization Duality Theory
MIQP Mixed-Integer Quadratic Program
GA Genetic Algorithm
COQP Constraint Optimization as Quadratic Program
DEMP Diferential Evolution for Multimodal Problems
IT Implement Tool
DOF Degrees of Freedom
URDF Unified Robot Description Format
SDF Simulation Description Format
WS Workspace
DFS Depth-First Search
BFS Breadth-First Search

Part I
BUILDING THE TOOLS

1

INTRODUCTION

1.1 BACKGROUND AND CONTEXT

Autonomous systems are complex agents capable of carrying out operations without human intervention. They have become more capable thanks to technological advancements and increasingly integrated into society with recent remarkable progress in artificial intelligence (AI) techniques. According to Zhang in [14], current trends indicate that the development and adoption of such systems will continue to grow in the coming years.

The agricultural sector is one of the areas where the integration of autonomous technologies has great potential. These systems could significantly benefit farmers by making their work safer and less repetitive. Autonomous systems have already been used in alternative cropping methods such as precision agriculture. Nevertheless, traditional practices are still facing challenges that autonomous systems could perfectly address. Among these challenges, the proliferation of weeds in grass fields raises as a major concern for the livestock well-being for two main reasons. First, weeds compete with grass for resources, reducing the quality of food available for grazing animals. Second, some weed species pose a direct threat to livestock health. In particular, plants like Rumex have been identified as toxic and the cause of livestock poisoning [11].

Removing these plants is a task that farmers must perform manually, as EU regulations restrict the use of pesticides and prevent farmers from combating weed proliferation through chemical means. It is evident that this task, especially in large grass fields, is highly time-consuming and extremely repetitive, making it an ideal candidate for automation. In Germany, companies like Paltech have developed solutions to address this problem using autonomous wheeled robots. Their flagship robot is a differential-drive wheeled system equipped with various sensors for localization and weed detection, as well as an onboard drilling mechanism for weed removal. Currently, if the weed removal process needs to be sped up, the only solution is to deploy a fleet of robots. While this is feasible, developing single units capable of holding more than one drilling mechanism seems like the natural next step in Paltech's solution.

1.2 PROBLEM STATEMENT

The development of systems with more than one drilling unit for weed removal comes with both hardware and software challenges. It is crucial to ensure that tools and system resources are used as efficiently as possible. We want to avoid having more capable units with unused tools, especially since the production and deployment of these improved systems are more costly. Therefore, reducing idle time is a top priority and the focus of this thesis.

Idle time refers to periods when resources, such as drilling equipment, are not actively engaged in productive work. Reducing idle time in this context means minimizing the time tools remain unused and maximizing their productivity in weed removal. To achieve this, allocating detected plants to the correct tools is essential. In the literature, this process is known as [TA](#). Some technical challenges to consider during implementation include computational latency and multi-tool coordination. The [TA](#) algorithm and execution pipeline must be fast enough to process new detections and reassign tools in real time without causing delays, while also ensuring that multiple drilling units operate efficiently without interference or redundancy.

1.3 CURRENT SOLUTIONS

In general, a task allocation system aims to achieve an efficient assignment of tasks to robots (or tools in this case) by considering various characteristics such as the robots' capabilities, task requirements, and system efficiency. This process of task allocation involves three important factors to be consider according to Umashankar in [4]. Robot/tool, environment and coordination as shown in Figure [1.1](#).

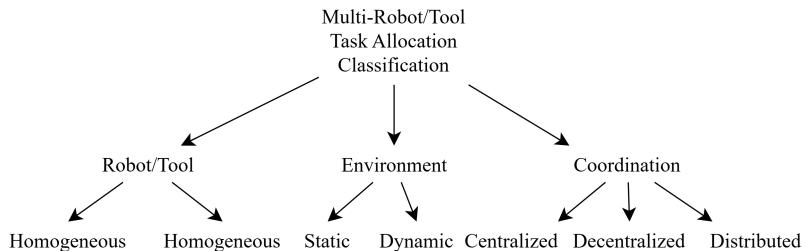


Figure 1.1: Multi-robot task allocation classification. Source [4]

An example of homogeneous tools in this context is a robot equipped with multiple tools of the same type, such as drills. In contrast, heterogeneous tools refer to robots equipped with different types of tools, such as drilling, seeding, and sensing equipment.

The multi-tool task allocation can take place in either a dynamic or static environment. In an environment that is static in nature, tasks are allocated to tools in advance before they begin to execute them. This

method works well in situations when the tasks are predetermined and the environment remains unchanged. In contrast, dynamic task allocation involves the real-time assignment of tasks to tools as they carry out their activities. For the scope of this thesis, we will focus on homogeneous tools operating in a dynamic environment with centralized coordination, where the onboard computer will act as the master, assigning plants to each drilling mechanism.

There are several ways to accomplish multi-tool allocation, including heuristic, cluster-based, market-based, learning-based, and optimization-based techniques. Table 1.1 and 1.2 presents a comprehensive classification of task allocation algorithms found in the literature.

APPROACH	TECHNIQUE / ALGORITHM
Cluster Based	SVCA [10]
	Group Agent Partitioning [3]
	CBDTA [1]
	VDKM [5]
	CBBA [12]
	Cluster First Consensus-Based Strategy
Market Based	K-Means Clustering
	Auction Algorithm [7]
	Improved Auction Algorithm [13]
	Extended Auction Algorithm
	Distributed Auction Algorithm
	Sequential Single-Item Auctions
	Auction-Based Algorithm
	Multihop-Based auction Algorithm
	Based on Sequential Single Item Auctions
	CBPAEA
	Extended Sequential Single Item Auction
	Distributed Auction-Based Algorithm
	The Consensus-Based Bundle Algorithm
	Linear Integer Programming
	Contract Net Protocol

Table 1.1: Comparative overview of cluster-based and market-based approaches to multi-agent TA. [4]

In cluster-based approaches, the goal is to group tasks into a predefined number of clusters. Instead of assigning a single task to each tool, the clustering approach allocates entire groups of tasks to them, reducing the number of individual task assignments and computational

APPROACH	TECHNIQUE / ALGORITHM
Learning Based	GRU , MLP [9]
	Deep Reinforcement Learning [2]
	Heterogeneous Graph Attention Network
	Capsule Attention-Based Mechanism
	EDACAM
	Graph Neural Network (GNN)
	Q-Learning, CNN and a GRU
	Graph Convolutional Neural Networks
	Mixed-Integer Quadratic Program
	Centralized Hungarian Method [8]
Optimization Based	Bin Maximum Item Doubled Packing
	PSO
	GTODT
	Integer Programming
	MIQP
	Genetic Algorithm (GA), A* Algorithm
	COQP
	Heuristic Based
	DEMP
	Fuzzy Optimization

Table 1.2: Comparative overview of learning-based and optimization-based approaches to multi-agent [TA](#). [4]

complexity. Clustering approaches aim to minimize travel distance and maximize task coverage by grouping tasks effectively. However, the optimal clustering of tasks still needs further exploration. Although these approaches simplify task allocation, they struggle to handle dynamic changes in the environment.

An optimization-based strategy aims to select the best solution from a set of available options. These solutions are constrained by specific conditions, and the optimal one is determined based on the objective function. The objective function represents the system's ultimate goal. Some of the optimization algorithms have poor robustness to uncertainties therefore this approach is more suitable for solving well-defined and static problems focusing on theoretically optimal or near-optimal solutions. Additionally, optimization-based approaches require more computational power and are less adaptable to changing environments.

Market-based approaches effectively handle highly combinatorial optimization problems. In this method, an auctioneer informs agents about available tasks and requests bids. Each agent evaluates its capacity to complete the tasks and submits a bid accordingly. The auctioneer then assigns tasks to the agent with the most favorable bid. Generally, task allocation using this approach minimizes travel time. While these methods are flexible and scalable, they may not always achieve a globally optimal solution.

Recent approaches to task allocation incorporate deep learning techniques such as graph neural networks and graph convolutional networks. These types of task allocation methods are commonly referred as learning-based approaches. Most learning-based approaches struggle to generalize to larger-scale problem scenarios beyond those used during training. This characteristic is especially important because real-world task allocation problems frequently require modeling scenarios whose costs increase with the number of tasks and robots. Table 1.3 gives a comparison between all the approaches.

	Clustering	Optimization	Market-Based	Learning-Based
Advantage	Simplifies task allocation and reduces complexity	Provides optimal solutions, well suited for static problems	Flexible, scalable, decentralized	Adaptable, learns, and improves over time
Limitation	May not account for dynamics well	Computationally intensive, less adaptable	May not yield global optima, and needs effective bidding	Requires training time, initially sub-optimal
Best case	Logical tasks	Well-defined, static problems	Dynamic environments with varying tasks	Complex and uncertain environments
Future work	Dynamic clustering, online adaptation	Hybrid models, real-time optimization	Adaptive market mechanisms, incentive models	Transfer learning, meta-learning

Table 1.3: A Comparison of different approaches to TA, Source [4]

1.4 PROPOSED SOLUTION

As Table 1.3 illustrates, algorithm selection must be carefully considered based on the application's nature to achieve optimal performance. In a grass field clearing application, the environment is highly dynamic, especially since weed detections occur while the system is in

motion. Therefore, market-based approaches are well-suited to ensuring the system adapts effectively to such conditions. However, since weeds often spread in clustered areas, a cluster-based approach could help reduce the algorithm's computational load and improve real-time responsiveness.

A hybrid algorithm is proposed to tackle the dynamics of the environment and leverage the benefits of both market-based and cluster-based approaches. By combining real-time adaptability with efficient task grouping, the system can dynamically allocate tasks while minimizing computational overhead. This approach ensures that weed removal remains efficient even as new detections occur, ultimately improving performance in large-scale and rapidly changing field conditions.

2

SIMULATION

2.1 THE ROBOT

Nuga is Paltech's solution for speeding up the weed removal process. Nuga is a mobile platform equipped with two drilling mechanisms, also called **IT**, one main camera at the front for plant detection, two internal cameras for fine adjustment during tools' placement, an IMU, and two GNSS antennas for GPS localization. Each **IT** is mounted on a structure with three Degrees of Freedom (**DOF**) using prismatic joints, allowing movement in X, Y, and Z directions.

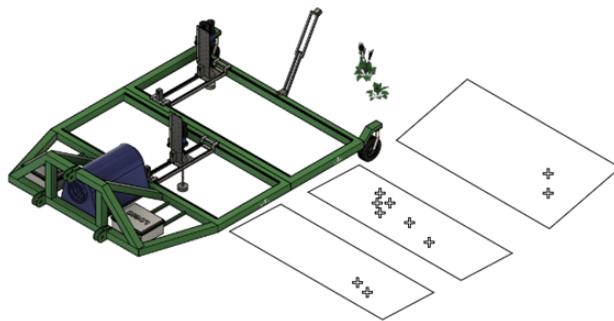


Figure 2.1: Nuga Platform

2.2 SIMULATION

A representative simulation of reality is crucial for developing new algorithms and analyzing robot behavior before real-world implementation. Therefore, building a simulation of the project was a foundational step for this work, ensuring a controlled environment for validation and testing. Gazebo¹ was the selected tool because it provides a physics engine, supports sensor and actuator modeling, and integrates well with ROS², making it ideal for testing robotic systems.

The simulation consists of six key components: URDF files define the robot's structure and properties, SDF files describe the virtual environment, and Gazebo plugins provide additional functionality, such as simulating custom sensors, actuators, or control interfaces.

¹ Gazebo is a physics-based robotics simulation tool that allows testing and validation of robot models before real-world deployment. <https://gazebosim.org/home>

² ROS (Robot Operating System) is an open-source framework that provides tools, libraries, and conventions for developing, managing, and simulating robotic applications. <https://www.ros.org/>

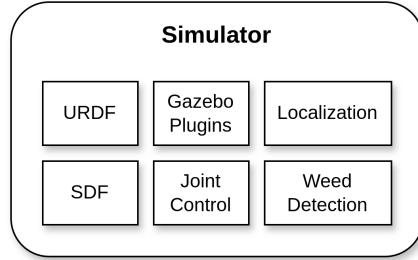


Figure 2.2: Simulator Components

Additionally, core system operations include joint control for managing the movement of the IT, localization for tracking the robot's position, and weed detection, which relies on an AI model for Rumex recognition. Figure 2.2 illustrates these components as building blocks for the simulation.

2.2.1 URDF

Unified Robot Description Format ([URDF](#)) is an XML file used to describe multibody systems for robot simulation. It defines the visual, collision, and inertial properties of rigid body objects, as well as their connections (*joints*). This establishes a spatial relationship between frames, which ROS and Gazebo can later interpret for control and visualization. This file also allows modeling different types of sensors and incorporating Gazebo plugins to link it with ROS control actions. We exploit these capabilities to define camera intrinsics, IMU behavior, GPS properties, and control the IT using `ros2_control`³.

[Figure 2.3a](#) displays the structure of the URDF files, being nuga the highest level entity that joins the robot description, gazebo sensor modeling and plugins, as well as `ros2_control` configuration. Nuga description defines the robot's physical structure, including its links (e.g., chassis, wheels, camera support), joints (fixed, continuous, prismatic connections), sensors (cameras, IMU, GPS), and inertial properties. It organizes these components into a kinematic tree (e.g., `base_link -> chassis_link -> wheels/sensors`) using macros for modularity, and resulting in the model displayed in [Figure 2.3b](#).

2.2.2 SDF

Simulation Description Format ([SDF](#)) also written in XML, describes the properties of the virtual world. Gazebo uses this file to define the terrain, obstacles, lighting conditions, physics parameters, and other

³ `ros2_control` is a ROS 2 framework that provides a standardized interface for managing hardware, enabling modular and reusable control systems for robots.
https://control.ros.org/rolling/doc/getting_started/getting_started.html

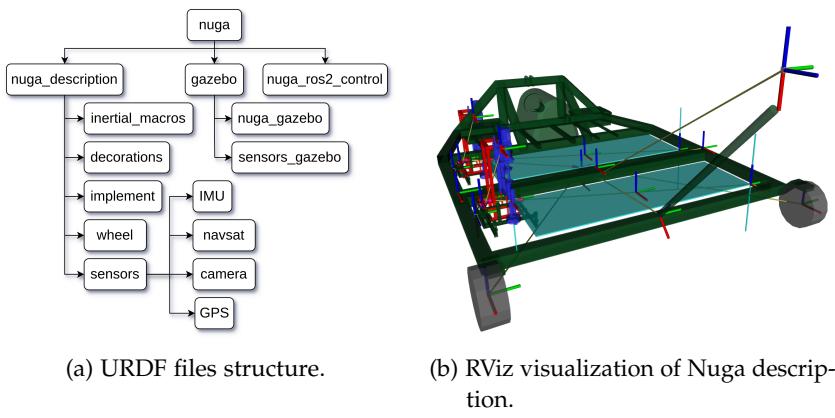


Figure 2.3: Robot definition using URDF

environmental elements that affect the robot's interaction with the simulation. Having repeatability in a simulated world is important for debugging and testing purposes, for this reason a Python script was used to generate easy to configure worlds from a YAML configuration file. An example of the config file is shown in [Listing A.1](#). For reproducibility, a seed value is configured in the simulation settings, the weed infestation pattern is defined within quadrants of specified dimensions (`quadrant_size`) and each quadrant is individually configured with:

- Spatial distribution:
 - *uniform*: Random uniform distribution
 - *clustered*: Random normal distribution with definable standard deviations (σ_x, σ_y)
- Weed density: Weeds per square meter (weeds/m²)
- Direction: Propagation axis for adjacent quadrants ($\pm x, \pm y$)
- Workspace expansion: If `outside_workspace` is true, the infestation area extends 10% beyond the quadrant boundaries.

A visual result of the generated world using such configuration file is shown in [Figure 2.4](#).

2.2.3 Gazebo Plugins

The files `nuga_gazebo` and `sensors_gazebo` from [Figure 2.3a](#) instantiate and configure Gazebo plugins to define sensor behavior, including optical properties for the camera, as well as update rates and noise models for the IMU and GPS. The file `nuga_ros2_control` on the other hand, establishes an interface between the `IT`'s joints and



Figure 2.4: Weed Infestation Example

`ros2_control` framework, specifying the command interface (position), controller type (forward position controller), and movement limits, enabling 3-DOF prismatic motion for each tool. Regarding movement control of the Nuga vehicle, the `ros_planar_move` plugin satisfied all control requirements given the platform’s kinematic constraints, eliminating the need for additional configuration.

2.2.4 Joint Control

The control of both `IT` units was handled using the `ros2_control` framework (configuration example shown in [Listing A.2](#)), as previously described. This framework provides a seamless transition between simulation and real hardware control. In this context, the `forward_command_controller` was used, which is recommended for simulation because it bypasses PID computations by directly sending commands to simulated joints. These joints already track positions perfectly, without the disturbances or error correction needed in real-world scenarios. Simulators like Gazebo inherently handle ideal position tracking, making closed-loop control redundant. However, when switching to real hardware, replacing it with a `position_controller` is needed but straightforward thanks to the flexibility of the framework.

Nuga’s workspace layout and dimensions are shown in [Figure 2.5](#). The gantry carrying the `IT` operates within a zone of 2.09 m in the Y direction, 0.72 m in X , and 0.26 m in Z . An extraction cycle begins with the gantry moving in X and Y to position the tool above the plant, followed by a downward movement in Z to lower the drill and perform the extraction. The gantry has a maximum speed of $1\frac{m}{s}$ in the XY plane, and each extraction can take up to 45 seconds per plant.

The `IT` is controlled using ROS2 actions, which provide a structured way to handle asynchronous tasks with feedback and result reporting. For the XY movement of the gantry, the `AxisPosition` action is used, allowing the specification of target coordinates (x, y) and speed, while providing feedback on the current position and confirming whether

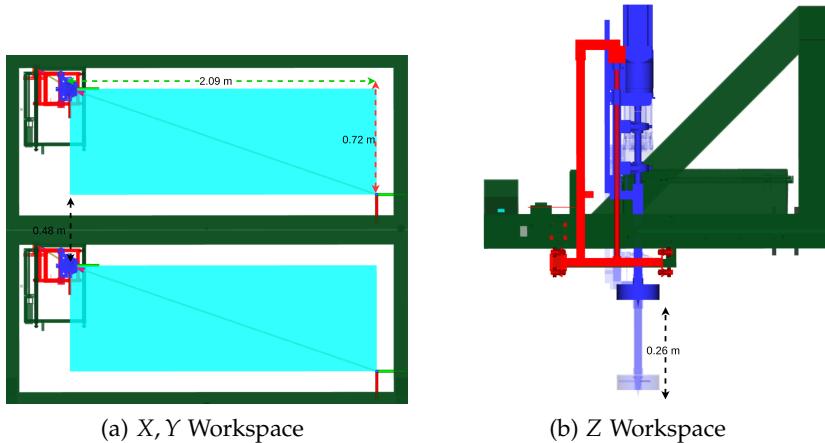


Figure 2.5: IT Workspace Layout

the target was reached. The Extraction action manages the vertical movement of the tool along the Z axis, reporting the depth reached, total time taken, and success status, along with real-time feedback on the current depth. Finally, the ExtractionCycle action coordinates the execution of multiple extractions by accepting an array of target poses and their corresponding IDs, providing feedback on the current status and reporting the results of the extraction process for each pose. These actions enable precise and modular control of the gantry system, ensuring efficient and reliable operation. A diagram summarizing this process is shown in Figure 2.6.

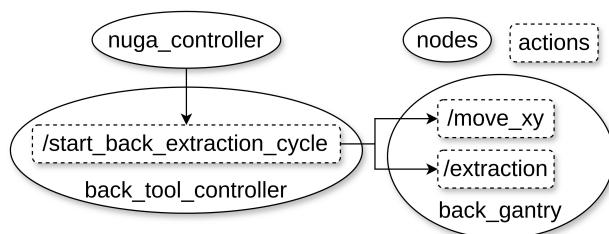


Figure 2.6: ROS Interface for back gantry control

2.2.5 Localization

viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2.2.6 *Weed Detection*

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Part II

THE SHOWCASE

You can put some informational part preamble text here. Illo principalmente su nos. Non message *occidental* anglo-romanic da. Debitas effortio simplificate sia se, auxiliar summarios da que, se avantiate publicationes via. Pan in terra summarios, capital interlingua se que. Al via multo esser specimen, campo responder que da. Le usate medical addresses pro, europa origine sanctificate nos se.

3

TASK ALLOCATION AND OPTIMIZATION

Solving the [TA](#) problem in this context means assigning weed detections to removal tools in the most efficient way. The idea is to find the best allocation of resources and the optimal sequence of stops, that will maximize the number of plants removed, and minimize the tools' idle time. The solution has to consider the constraints of the problem such as:

GEOMETRIC CONSTRAINTS: A stop is considered valid only if the weeds lie within the workspaces of the tools (observe [Figure 3.1](#)).

MOVEMENT CONSTRAINTS: The robot can only move forward in a straight line at a max speed of $0.3 \frac{m}{s}$.

TASK PROCESSING: Each plant removal takes approximately 45 seconds, during which the robot must remain stationary to ensure a successful extraction. Movement during this process is not allowed as it could compromise the tool operation.

DYNAMIC ENVIRONMENT: Weed positions are initially unknown and are discovered dynamically during operation by the camera in front of the vehicle.

PROCESSING TIME: The solution must run online because the robot discovers new weeds during operation, and decisions about stopping and [TA](#) must adapt in real-time.

We define a '*good*' stop as the next robot position that maximizes plant coverage while keeps a balance on the number of tasks assigned to each implement (aiming to minimize idle time). The problem can be approached in two ways: one option is to first determine the optimal stop location, after which task allocation becomes trivial by simply assigning tasks based on whether they fall within the front or back workspace. Alternatively, we can first allocate tasks to the implements and then compute the stop position that satisfies the corresponding geometric constraints. Both approaches are similar, differing mainly in the order of operations, but each perspective opens the door to different methods and algorithms to try.

3.1 METRICS

Defining a good set of metrics is crucial to establish a solid basis for comparison between solutions and to easily identify the flaws

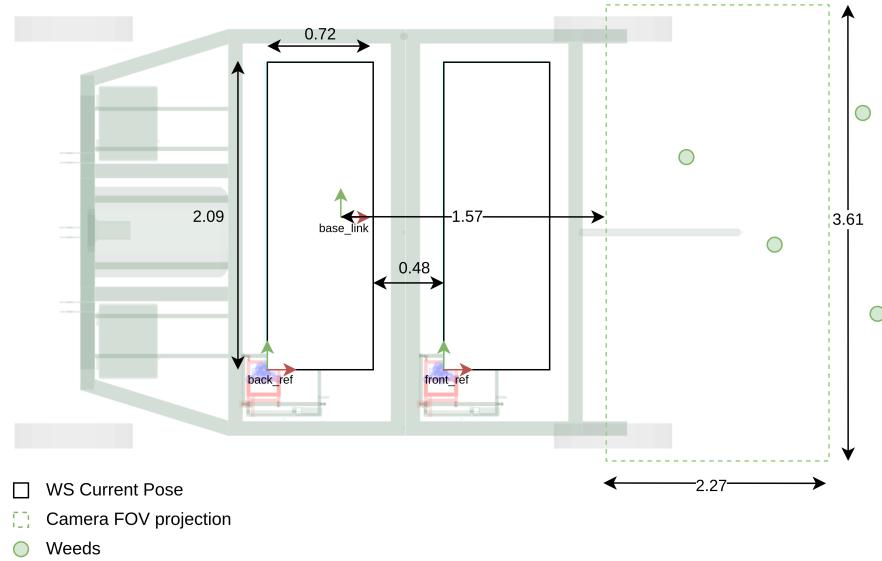


Figure 3.1: Problem Layout Dimensions

of each method, thereby determining the best solution. To address this, we define two main categories. The first summarizes mission-level metrics such as the total idle and productive time of each tool, and the total time the robot spends moving or in a stationary state (defined by equations 3.1 and 3.2). The second category includes task-specific information, for example, task ID, status ('compleated', 'failed', 'out'), number of stops, the tool that processed the task, and the idle and productive time of each tool at that specific stop. These metrics are displayed in a mission dashboard for easy visualization and comparison across missions using different worlds or algorithms (see Figure 3.2a for the first category and 3.2b for the second one).

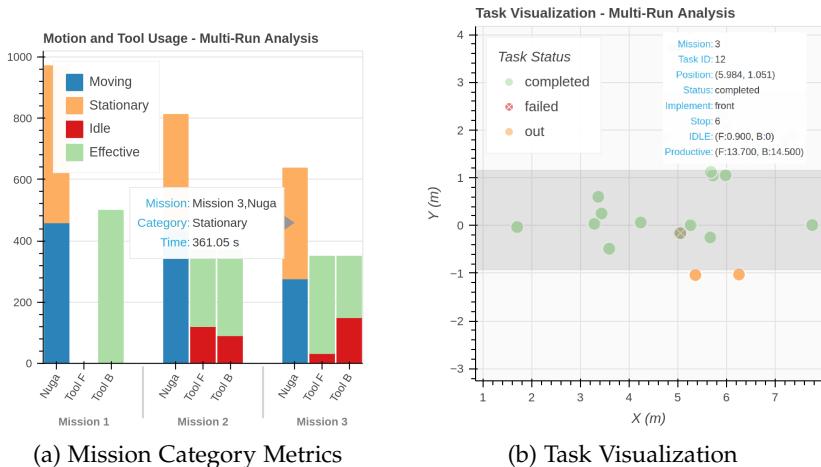


Figure 3.2: Mission Dashboard

FIGURE 3.2A Showcases a bar graph comparing different missions. Each mission includes the robot’s moving and stationary time, as well as the idle and productive time of the onboard implement tools. A hover feature displays the value of each category.

FIGURE 3.2B Displays a 2D grid of the removed tasks’ positions during the mission, along with their corresponding metadata. Hovering over a task reveals its details, and tasks can be filtered by mission.

$$t_{mission} = t_{moving} + t_{stationary} \quad (3.1)$$

$$t_{tool_operation} = t_{idle} + t_{productive} \quad (3.2)$$

3.2 HEURISTICS

Heuristic solutions are commonly employed when the solution space of an optimization problem is too large to explore exhaustively, making an exact optimal solution computationally infeasible. They are also useful in time-sensitive scenarios, such as online implementations, where sacrificing optimality for efficiency is often justified.

In our work, we developed a heuristic algorithm to serve as a baseline solution, this provides a meaningful reference point to assess the performance and potential improvements offered by other algorithms. The algorithm’ description is detailed in 1, with an illustrative example in Figure 3.3.

Algorithm 1 Heuristic

- 1: Get the position of the closest weed from the current robot position.
 - 2: Project the tools’ Workspace (WS) forward (in the future), aligning the trailing edge of the last WS with the closest weed’s position plus a small clearance (e.g., 10 cm).
 - 3: Allocate weeds to each tool if they fall within its projected WS.
 - 4: Move the robot until the tools’ WS are aligned with their projections, then execute the extractions for tools with assigned weeds.
 - 5: Repeat the process until mission has ended.
-

This approach offers a simple implementation and fast computation solution, making it well-suited for online applications. However, its heuristic nature leads to suboptimal solutions, as it does not account for minimizing idle time. In Figure 3.4 we exemplify the algorithm’ suboptimality for a particular case, contrasting with a better solution to support our analysis.

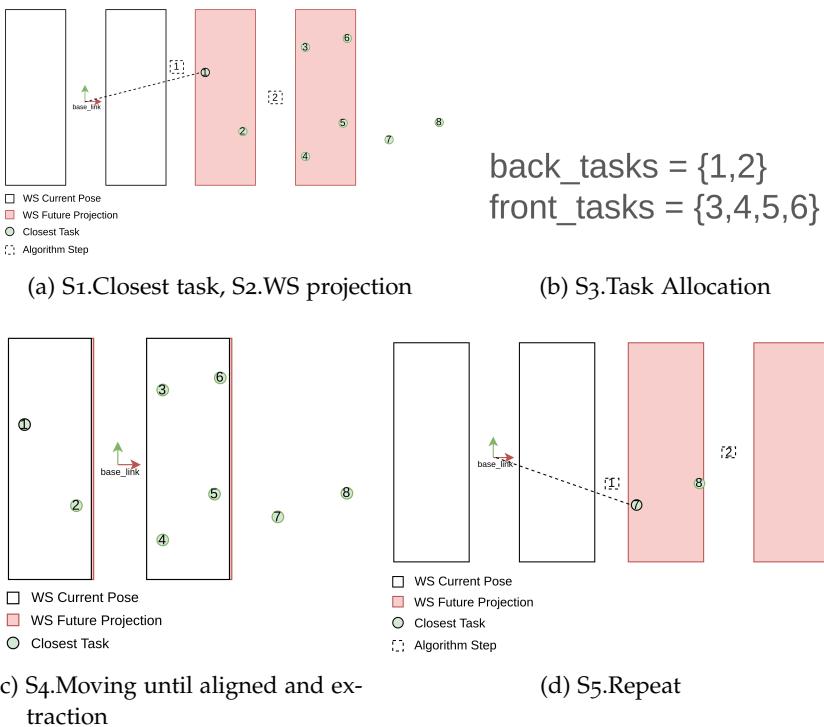


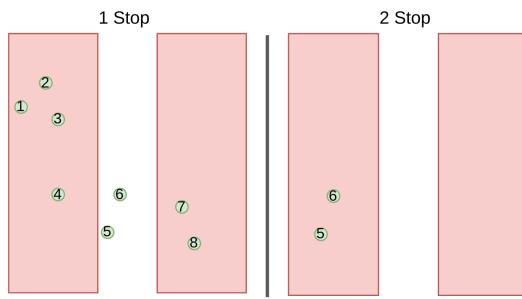
Figure 3.3: Heuristic Algorithm

FIGURE 3.4A Showcases the two stops required to remove eight weed' detections, with the first stop allocating four tasks to the back tool and the second stop allocating two tasks to the front tool.

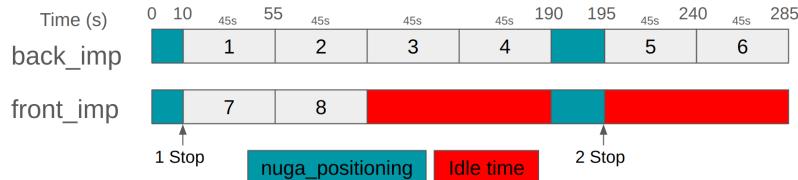
FIGURE 3.4B Illustrates the idle and productive time for the given solution. Blue represents the robot's repositioning time, red indicates the idle time of the respective tool, and white the productive time. The first stop shows a significant idle time for the front tool (90 seconds, equivalent to two tasks), caused by the tool waiting for the back tool to finish before proceeding to the next stop. The second stop shows a similar situation, where idle time arises from the only feasible option at that point: removing tasks 5 and 6 with the back tool.

An alternative solution for the same scenario (this time aimed at reducing idle time) is illustrated in [Figure 3.5](#). In this case, the solution requires three stops: first stop assigns tasks 1 to back tool and task 5 to the front, the second stop processes tasks 2 and 6, and finally the third stop is used to remove tasks 3, 4, 7, and 8. As shown in [Figure 3.5b](#), the balanced task allocation between tools eliminates idle time and reduces the total mission duration compared to the previous solution.

This demonstrates the heuristic's suboptimality, as it fails to minimize idle time and does not consider the possibility of stopping at a location that allows for a more efficient task allocation. The heuristic algorithm is limited to the closest weed detection, which may not always be the best option. In the following sections, we will explore more sofisticated



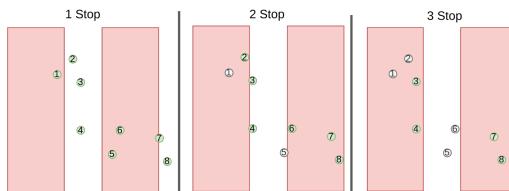
(a) Two stops required



(b) Idle and productive time of each tool

Figure 3.4: Suboptimal solution computed using Heuristic

algorithms that can overcome these limitations and provide better solutions.



(a) Three stops required



(b) Idle and productive time of each tool

Figure 3.5: Optimal solution

3.3 GRAPH SEARCH

Graph Search are a type of algorithms widely used in graph theory to systematically explore or traverse a graph. They are commonly used to find the shortest path between nodes, identify connected components, or solve various optimization problems. Graph search algorithms can

be broadly categorized into two main types: uninformed and informed search algorithms.

Uninformed search algorithms do not have any additional information about the problem domain and explore the graph blindly. Examples include Depth-First Search ([DFS](#)) and Breadth-First Search ([BFS](#)). These algorithms are typically used for tasks like finding connected components or traversing all nodes in a graph. Informed search algorithms, on the other hand, use heuristics or additional information to guide the search process. They are often more efficient than uninformed algorithms for specific problems. Examples include A* search and Dijkstra's algorithm, which are commonly used for finding the shortest path in weighted graphs.

In our context, we exploit the advantages of graph-based algorithms by modeling the problem as such. Nodes represent potential stop locations, where two types of actions are possible: moving to another stop (a *stop node*) or performing weed removal at that location (an *action node*). Edge weights are defined as follows: edges between two *stop nodes* represent *travel cost*, computed using the distance between stops and the robot's velocity or a unitary value as a placeholder. Edges connecting a *stop node* to an *action node* represent *task execution cost*, and it is calculated based on the idle time (if any) of removing the indicated weeds with the assigned tools.

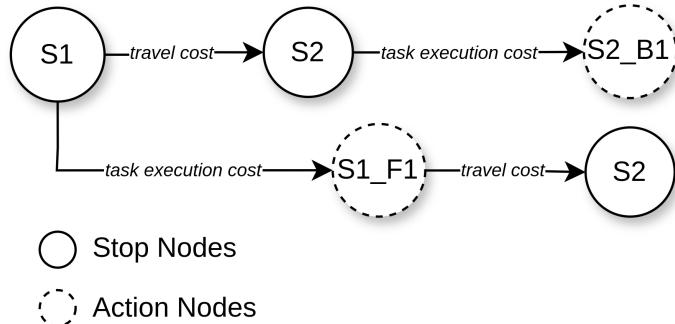


Figure 3.6: Node and cost representation

FIGURE 3.6 Illustrates the convention used to represent the graph. S1 denotes stop number 1, while F or B indicate whether the front or back tool is assigned to remove the weed, followed by the weed ID to be processed. In this example, the task execution cost between S1 and S1_F1 would be 45s, as the back tool must wait for the front tool to process 1 plant. The same logic applies to the cost between S2 and S2_B1 but for the front tool.

The graph search implementation solution follows the next pipeline:

1. Compute **candidate stops** based on the robot's current position and weed detections. Candidate stops are the locations where

an important event occurs, such as a weed entering or exiting the workspace of a tool.

2. **Associate** reachable weeds with candidate stops. This allows the algorithm to determine which weeds can be removed at each stop.
3. Create a **graph representation** of the problem using [DFS](#) algorithm and the convention described in [Figure 3.6](#). The graph is built by connecting *stop nodes* and *action nodes* using appropriate edge costs, taking into account the predefined associations between stops and tasks.
4. Get the **shortest path** in the graph from the root (first stop) to the final node (last stop) using Dijkstra's algorithm.
5. **Decode solution** and return the next optimal stop and task allocation as the algorithm's output.

To build and explore the graph, an implementation of the [DFS](#) algorithm is used (see Algorithm 2). The `get_children_nodes` method expands the graph by generating child nodes from a given parent node. If the parent node is a *stop node*, it creates child nodes for all valid combinations of reachable tasks that can be processed at that stop. Each combination forms a new child node that reflects the state after removing those tasks (*action node*). Additionally a 'trivial' *stop node* is added representing the decision to move to the next stop without removing any weeds. On the other hand, if the parent node is an *action node*, this method creates a single *stop node* representing the next stop. Each created edge is added to the graph with a weight based on the associated *task execution cost* or *travel cost*. [Figure 3.7](#) illustrates the graph expansion process, where the algorithm grows all possible combinations of tasks at each stop and generates the corresponding child nodes.

We observe that from S3 four possible states can be reached (three *action nodes* and one *stop node*). The transition S3_B1_F2 achieves zero idle time by optimally balancing tasks between the front and back tools, whereas the other transitions result in higher idle times due to imbalanced assignments. It's important to note that the same stop can be reached through different paths, depending on the actions taken at previous stops. For instance, if task 1 has been removed earlier, it should no longer be considered in subsequent stops. This consideration is crucial when building the graph, as we must ensure that it is expanded only through valid and realistic transitions that reflect feasible execution scenarios.

Having the complete graph, and given its configuration as a weighted directed graph, we choose Dijkstra's algorithm to find the optimal path (see [Algorithm 3](#)). This algorithm efficiently explores the graph by maintaining a priority queue of nodes to be processed, ensuring that

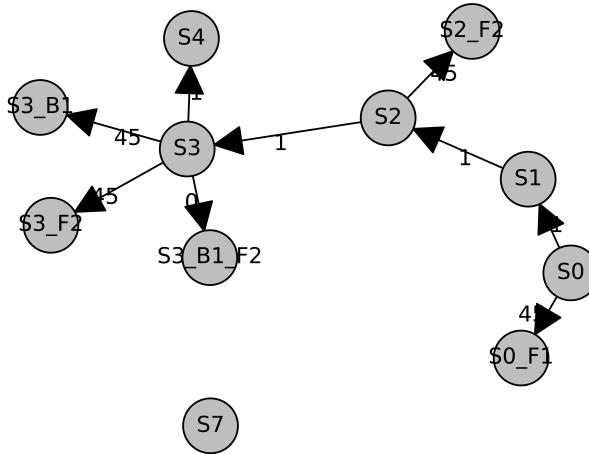


Figure 3.7: Node and cost representation

the node with the lowest cost is always explored first. The algorithm continues until it reaches the final node, which represents the optimal solution. The path is then traced back from the final node to the root, revealing the sequence of stops and actions that lead to the optimal solution.

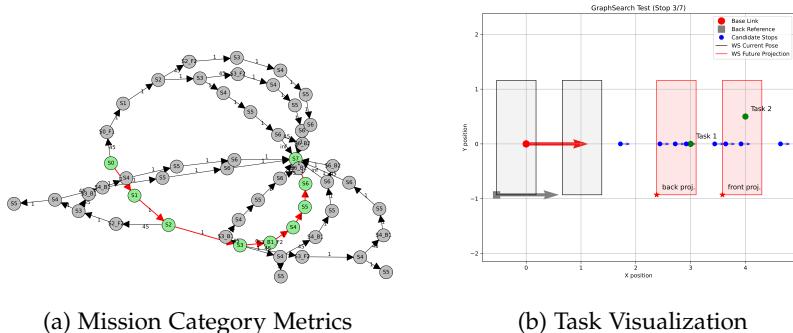


Figure 3.8: Mission Dashboard

An optimal solution obtained by the graph search algorithm in a scenario with two detected weeds is illustrated in Figure 3.8a. The highlighted branch corresponds to the optimal path found by the algorithm. For the given two weeds, the graph contained 54 nodes, and the computation time was 0.001 seconds. A visual representation of the solution is shown in Figure 3.8b, including the current robot pose, projected workspaces in red, candidate stops in blue, and weeds in green.

3.4 OPTIMIZATION

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.5 MARKET-BASED

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Part III
APPENDIX

A

APPENDIX TEST

Lorem ipsum at nusquam appellantur his, ut eos erant homero concludaturque. Albucius appellantur deterruisset id eam, vivendum partiendo dissentiet ei ius. Vis melius facilisis ea, sea id convenire referrentur, takimata adolescens ex duo. Ei harum argumentum per. Eam vidit exerci appetere ad, ut vel zzril intellegam interpretaris.

More dummy text.

A.1 CONFIGURATION FILES

YAML files are the standard method for setting up configuration variables in the Nuga project. A collection of configuration examples is provided in this appendix.

Listing A.1: Weed Infestation World config example

```
seed: 2
quadrant_size: [2.0, 2.0]
quadrants:
  1:
    direction: x
    weed_density: 0.4    # weeds/m2
    spatial_distribution: clustered
    std_dev: [0.5, 0.5]
    outside_workspace: false

  2:
    direction: x
    weed_density: 1.2    # weeds/m2
    spatial_distribution: uniform
    outside_workspace: false
```

Listing A.2: ROS2 config example

```
# ROS2 Control
controller_manager:
  ros_parameters:
    use_sim_time: True
    update_rate: 20 # Hz

  joint_state_broadcaster:
    type: joint_state_broadcaster/JointStateBroadcaster

  forward_position_controller_front:
    type: forward_command_controller/ForwardCommandController

  forward_position_controller_back:
    type: forward_command_controller/ForwardCommandController

  forward_position_controller_front:
  ros_parameters:
    joints:
      - front_x_axis_joint
      - front_y_axis_joint
      - front_implement_tool_joint
    interface_name: position

  forward_position_controller_back:
  ros_parameters:
    joints:
      - back_x_axis_joint
      - back_y_axis_joint
      - back_implement_tool_joint
    interface_name: position
```

A.2 ALGORITHMS

Algorithm 2 Depth-First Search

Require: *root_node, last_node***Ensure:** Explored Graph

```

1: visited ← {last_node} {Initialize with last node}
2: stack ← [root_node] {Start with root node}
3: i ← 0 {Iteration counter}
4: while stack is not empty do
5:   i ← i + 1
6:   node ← stack.pop()
7:   if node  $\notin$  visited then
8:     visited.add(node)
9:     children ← get_children_nodes(node)
10:    if children  $\neq$  None then
11:      stack.extend(children) {Add children to stack}
12:    end if
13:   end if
14:   if i == 5000 then
15:     break {Early termination}
16:   end if
17: end while

```

Algorithm 3 Dijkstra's Algorithm

Require: *graph, start_node, goal_node*

Ensure: Shortest path from *start_node* to *goal_node*

```

1: cost  $\leftarrow$  defaultcost( $\infty$ ) {Distance from start to each node}
2: prev  $\leftarrow$  dict() {To reconstruct shortest path}
3: cost[start_node]  $\leftarrow$  0
4: queue  $\leftarrow$  priority queue initialized with (0, start_node)
5: while queue is not empty do
6:   (current_cost, current_node)  $\leftarrow$  queue.pop()
7:   if current_node == goal_node then
8:     break
9:   end if
10:  for all neighbor of current_node in graph do
11:    new_cost  $\leftarrow$  current_cost + weight(current_node, neighbor)
12:    if new_cost < cost[neighbor] then
13:      cost[neighbor]  $\leftarrow$  new_cost
14:      prev[neighbor]  $\leftarrow$  current_node
15:      queue.push(new_cost, neighbor)
16:    end if
17:  end for
18: end while
19: return shortest path and cost from start_node to goal_node

```

A.3 ANOTHER APPENDIX SECTION TEST

Equidem detraxit cu nam, vix eu delenit periculis. Eos ut vero constituto, no vidit propriae complectitur sea. Diceret nonummy in has, no qui eligendi recteque consetetur. Mel eu dictas suscipiantur, et sed placerat oporteat. At ipsum electram mei, ad aeque atomorum mea. There is also a useless Pascal listing below: [Listing A.3](#).

Listing A.3: A floating example (listings manual)

```

for i:=maxint downto 0 do
begin
{ do nothing }
end;

```

BIBLIOGRAPHY

- [1] Xinye Chen, Ping Zhang, Fang Li, and Guanglong Du. "A cluster first strategy for distributed multi-robot task allocation problem with time constraints." In: *2018 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*. 2018, pp. 102–107. doi: [10.1109/WRC-SARA.2018.8584210](https://doi.org/10.1109/WRC-SARA.2018.8584210).
- [2] Ahmed Elfakharany and Zool Hilmi Ismail. "End-to-End Deep Reinforcement Learning for Decentralized Task Allocation and Navigation for a Multi-Robot System." In: *Applied Sciences* 11.7 (2021). issn: 2076-3417. doi: [10.3390/app11072895](https://doi.org/10.3390/app11072895). url: <https://www.mdpi.com/2076-3417/11/7/2895>.
- [3] Junyan Hu, Parijat Bhowmick, Inmo Jang, Farshad Arvin, and Alexander Lanzon. "A Decentralized Cluster Formation Containment Framework for Multirobot Systems." In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1936–1955. doi: [10.1109/TRO.2021.3071615](https://doi.org/10.1109/TRO.2021.3071615).
- [4] Athira K A, Divya Udayan J, and Umashankar Subramaniam. "A Systematic Literature Review on Multi-Robot Task Allocation." In: *ACM Comput. Surv.* 57.3 (Nov. 2024). issn: 0360-0300. doi: [10.1145/3700591](https://doi.org/10.1145/3700591). url: <https://doi.org/10.1145/3700591>.
- [5] Jeongeon Kim and Hyoung Il Son. "A Voronoi Diagram-Based Workspace Partition for Weak Cooperation of Multi-Robot System in Orchard." In: *IEEE Access* 8 (2020), pp. 20676–20686. doi: [10.1109/ACCESS.2020.2969449](https://doi.org/10.1109/ACCESS.2020.2969449).
- [6] Donald E. Knuth. "Computer Programming as an Art." In: *Communications of the ACM* 17.12 (1974), pp. 667–673.
- [7] Jiamei Lin, Pengcheng Li, Jialong Dai, Shaorui Liu, Wei Tian, Bo Li, and Changrui Wang. "Multi-robot Cooperative Assembly Task Planning for Large-scale Aerospace Structures." In: *2022 37th Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. 2022, pp. 795–800. doi: [10.1109/YAC57282.2022.10023897](https://doi.org/10.1109/YAC57282.2022.10023897).
- [8] Nicholas Lindsay, Robert K. Buehling, and Lihong Sun. "A Sequential Task Addition Distributed Assignment Algorithm for Multi-Robot Systems." In: *Journal of Intelligent & Robotic Systems* 102 (2021), p. 51. doi: [10.1007/s10846-021-01394-7](https://doi.org/10.1007/s10846-021-01394-7).
- [9] Xingjie Liu, Guolei Wang, and Ken Chen. "Option-Based Multi-Agent Reinforcement Learning for Painting With Multiple Large-Sized Robots." In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2022), pp. 15707–15715. doi: [10.1109/TITS.2022.3145375](https://doi.org/10.1109/TITS.2022.3145375).

- [10] Javier G. Martin, Francisco Javier Muros, José María Maestre, and Eduardo F. Camacho. "Multi-robot task allocation clustering based on game theory." In: *Robotics and Autonomous Systems* 161 (2023), p. 104314. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2022.104314>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889022002032>.
- [11] R. J. Panciera, T. Martin, G. E. Burrows, D. S. Taylor, and L. E. Rice. "Acute oxalate poisoning attributable to ingestion of curly dock (*Rumex crispus*) in sheep." In: *Journal of the American Veterinary Medical Association* 196.12 (1990), pp. 1981–1984.
- [12] Darren Smith, Jodie Wetherall, Stephen Woodhead, and Andrew Adekunle. "A Cluster-Based Approach to Consensus Based Distributed Task Allocation." In: *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. 2014, pp. 428–431. DOI: [10.1109/PDP.2014.87](https://doi.org/10.1109/PDP.2014.87).
- [13] Shiguang Wu, Xiaojie Liu, Xingwei Wang, Xiaolin Zhou, and Mingyang Sun. "Multi-robot Dynamic Task Allocation Based on Improved Auction Algorithm." In: *2021 6th International Conference on Automation, Control and Robotics Engineering (CACRE)*. 2021, pp. 57–61. DOI: [10.1109/CACRE52464.2021.9501305](https://doi.org/10.1109/CACRE52464.2021.9501305).
- [14] T. Zhang, Q. Li, Cs. Zhang, et al. "Current trends in the development of intelligent unmanned autonomous systems." In: *Frontiers in Information Technology & Electronic Engineering* 18 (2017), pp. 68–85. DOI: [10.1631/FITEE.1601650](https://doi.org/10.1631/FITEE.1601650).

DECLARATION

Put your declaration here.

Zagreb, Croatia, June 2025

David Raziel Ceres Arroyo

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both L^AT_EX and LyX:

<https://bitbucket.org/amiede/classicthesis/>

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

<http://postcards.miede.de/>

Thank you very much for your feedback and contribution.