Optimized Task Coordination for Heterogenous Multi-Robot Systems

Alfa Budiman

Thesis Submitted to the University of Ottawa
in Partial Fulfillment of the Requirements for the
Master of Applied Science in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Multi-robot systems leverage the numbers and characteristics of different robots to accomplish an overall mission. Efficient task allocation and motion planning of multi-robot teams are essential to ensure each robot’s actions contribute to the overall mission while avoiding conflict with each other.

The original contribution of this thesis is an optimized, efficient, and multi-factor task allocation algorithm to comprise the main component of a task coordination framework (TCF), with motion planning as a secondary component. This algorithm determines which robot performs which tasks and in what order. It presents a novel solution to the multiple robot task allocation problem (MRTA) as an extension of the multiple travelling salesmen (MTSP) problem. This extension to the MTSP considers operational factors representing physical limitations, the suitability of each robot, and inter-task dependencies. The task allocation algorithm calculates an optimized distribution of tasks such that a global objective function is minimized to simultaneously reduce total cost and ensure an even distribution of tasks among the agents. Once an optimized distribution of tasks is calculated, the motion planning component calculates collision-free velocities to drive the robots to their goal poses to facilitate task execution in a shared environment.

The proposed TCF was implemented on teams of unmanned air vehicles (UAVs) and unmanned ground vehicles (UGVs). Test cases considered scenarios where the UAVs executed aerial observation tasks while UGVs executed simulated patrol and delivery tasks. The solutions were tested using real-life robots as a proof of concept and to validate simulations. The robots' kinematic and computer vision models were combined with the task coordination framework to facilitate the implementation. Large-scale simulations involving greater numbers of robots operating in a larger area were also conducted to demonstrate the task coordination framework's versatility and efficacy.

Acknowledgements

I would like to acknowledge my co-supervisors, Dr. Pierre Payeur and Dr. Eric Lanteigne, for accepting me as a graduate student within their research group. Their guidance has contributed to the development of my skills as an engineer and as a researcher.

The Canadian Forces also has my gratitude for sponsoring my education, both at the graduate and undergraduate levels. I am grateful for their investment in my professional development and the opportunities they provided me.

Finally, many thanks to my family as well as friends and colleagues I have met throughout my professional and academic career.

Table of Contents

List of Acronyms and Abbreviations

Acronym / Abbreviation	Definition
CM-PMX	Constrained Multi-agent Partially Mapped Crossover
HT	Homogeneous Transformation
IMU	Inertial Measurement Unit
IR	Infrared
KF / EKF	Kalman Filter / Extended Kalman Filter
MAVLink	Micro Air Vehicle Link
MAVROS	Micro Air Vehicle Robot Operating System
MoCap	Motion Capture
MRTA	Multiple Robot Task Allocation
MTSP	Multiple Travelling Salesmen Problem
OIATA	Order-Independent Agent-Task Assignment
OS	Operating System
PID	Proportional Integral Derivative
POI	Point of Interest
PWM	Pulse Width Modulation
QGC	Q Ground Control
ROS	Robot Operating System
RVO	Reciprocal Velocity Obstacle
TCF	Task Coordination Framework
UAV	Unmanned Air Vehicle
UGV	Unmanned Ground Vehicle
USV	Unmanned Surface Vehicle
UUV	Unmanned Underwater Vehicle
UWV	Unmanned Water Vehicle
VIO	Visual Inertial Odometry
VM	Virtual machine
VO	Velocity Obstacle

List of Tables

List of Figures

List of Algorithms

Chapter 1 – Introduction

1.1 – Motivation

Multi-robot systems can employ their numbers and different characteristics to break down large, complex missions into smaller tasks completed in parallel. At the same time, using multiple robots enables their design to be tailored for a specialized purpose, yielding unique strengths and weaknesses for each robot in the system.

Cooperation within a multi-robot system requires a framework for controlling the actions of each robot. This framework ensures that each robot's actions contribute to the success of the overall mission in an efficient and conflict-free manner. The proposed framework builds on solutions within multi-robot task allocation (MRTA), which entails matching robots to tasks such that the overall cost for all tasks is minimized, subject to applicable constraints. The MRTA problem can be seen as an extension of the multiple travelling salesmen problem (MTSP), in which several cities must be traversed once by a salesman, with multiple salesmen available. Routes must be assigned to each salesman to minimize total travel distance. MRTA solutions determine which robots perform which tasks and in what order.

The challenges within MRTA can be described in terms of formulation and implementation. Within formulation, the MRTA problem, as an extension of MTSP, is NP-hard (nondeterministic polynomial time). The MTSP is a generalization of the TSP (MTSP but with only 1 salesman), which is already considered NP-hard. The TSP is NP-hard because the number of possible solutions grows exponentially as more destinations are added. Furthermore, there is no known algorithm to calculate the theoretical optimal solution without excessive computational demand. This challenge is magnified within the MTSP, as there are multiple salesmen. It is further magnified with the MRTA because the capabilities of each robot and the nature of each task must also be considered. Consequently, MRTA solutions almost always result in approximations of a theoretical global optima.

To implement MRTA solutions, sensing, localization, and motion planning must also be considered. Sensing and localization describe each robot's pose while allowing them to gain information on their environment. It entails using sensors and manipulating the data to gain insight into the robots' state and environment. Motion planning generates goal poses and velocities for the robots, which are then matched with the kinematic constraints of the respective robots to facilitate collision-free movement toward their goals.

Transitioning MRTA solutions from algorithms and simulations into physical robots also introduces several challenges that must be addressed. From a software perspective, compute power and communications limitations must be considered. Regarding hardware, the robots' physical capabilities (kinematics and onboard equipment) must be aligned with the algorithms' task allocation and operating environment. The uncertainty imposed by noisy sensors and the lack of perfect information in reality (as opposed to simulations) must also be considered when implementing a solution on a real-life autonomous robot.

Overall, multi-robot systems have significant potential but present several challenges that must be addressed to ensure each robot contributes meaningfully and efficiently to the

overall mission. While many solutions exist for sensing, localization, motion planning, and MRTA, they are often limited to a specific scenario with specific sets of robots. It is of particular interest to develop a generalized framework for controlling multi-robot teams for various mission profiles and for teams of any number of robots with diversified characteristics.

1.2 – Objectives and contributions

This thesis aims to formulate and implement a generalized, robot-agnostic task coordination framework (TCF) to control a multi-robot system. The primary contribution is a task allocation algorithm that is at least as effective as contemporary solutions but also considers operational factors that contemporary solutions do not. The proposed task allocation algorithm will be:

- Optimized. The task allocation algorithm minimizes (or maximizes) the global objective function to an equal or greater extent than contemporary MRTA and MTSP solutions.

- Computationally efficient. The task allocation algorithm does not require significant computational resources or computation time to generate an optimized solution.

- Multi-factor. The task allocation algorithm generates solutions compliant with constraints imposed by operational factors. These factors are agent-task suitability, inter-task dependencies, robots' work capacities and task actuation cost. In contrast, these factors are not typically considered in contemporary MRTA and MTSP solutions.

The secondary contributions involve the implementation of the TCF through a technical proof-of-concept:

- Implementation of TCF using existing solutions for localization, motion planning and communications to support simultaneous collision-free task execution. Kinematic, motion planning and computer vision models were formulated and tested for implementation on UAVs and UGVs.

- Setup and configuration of software and hardware for robots (UAV and UGV) and motion capture (MoCap) systems for experimentation and future research. This contribution established the foundation of an experimental testbed.

This thesis also contributed to the publication of [1], which discusses using multiple autonomous robots to deploy environmental sensor nodes for pollution monitoring.

1.3 – Organization

The rest of this thesis is organized as follows:

- Chapter 2 presents the literature concerning the relevant topics. It discusses the technical background of existing MRTA solutions and the supporting topics of sensing, localization, and motion planning. The applications and gaps in these topics are provided through this discussion.

- Chapter 3 describes the proposed optimization solution for MRTA. It first discusses the optimization criteria, describing the agent-task dynamics and optimization goals and constraints. It then formulates a combination of deterministic and metaheuristic algorithms to calculate an optimized task allocation. The proposed algorithms are then evaluated against an established benchmark, and the impact of constraints from operational factors is assessed.

- Chapter 4 discusses the mathematical models for the movement and task execution of UAVs and UGVs assigned by the TCF. It describes the kinematic and machine vision models. It proposes aerial observation and ground patrol tasks with mobile robots as a proof of concept for the TCF.

- Chapter 5 presents the technologies and equipment for simulation and experimentation. It describes the robots, hardware and software used to implement the TCF with the models described in Chapter 4 for simulation and experimentation.

- Chapter 6 describes simulations as well as testing with real robots. Simulations validate software components and demonstrate the TCF's efficacy and versatility. Real-life testing serves as a technical proof of concept for the proposed implementation, confirming that the simulations accurately depict reality.

- Chapter 7 concludes the thesis with a summary of the work, major contributions, and possible directions for future research.

Chapter 2 – Literature Review

Multi-robot systems consist of multiple robots working together to complete an overall mission, leveraging their numbers and different characteristics. A mission can be broken down into several smaller tasks which can be completed in parallel. There can also be different types of tasks that require different capabilities.

These advantages, however, pose several challenges, primarily in the task allocation and coordination aspects. A multi-robot system must have a framework for deciding which robot performs which task, and upon receiving that assigned task, the robot must perform it in a manner that contributes to the success of the overall mission while avoiding conflict with other robots. A multi-robot system should be aware of the state of each robot, plan and execute the movement of each robot, as well as communicate this information to the robots and a control station if applicable. Furthermore, supporting task allocation are sensing, localization, motion planning, and communication.

Section 2.1 discusses sensing and localization, enabling the robots to estimate their own pose/state and gather information regarding their environment. Section 2.2 discusses motion planning for individual and multiple robots. Section 2.3 discusses solutions for multiple robot task allocation (MRTA) as an extension of the multiple travelling salesmen problem (MTSP). The overall state of the art (SOTA) in multi-robot systems is then summarized in section 2.4.

2.1 – Sensing and Localization

Sensing is the process by which a robot collects data from its environment. Localization converts that data into a pose estimate for the robot itself or a detected object. These processes can be described in terms of the object being localized, with self-localization determining the robot's own pose in reference to its environment [2] and target localization determining the pose of a detected object. Sensing and localization can also be described in terms of how the object is localized. The localization method can use information from the robot's onboard equipment, such as cameras, light detection and ranging (LiDAR) and inertial measurement unit (IMU). Offboard equipment not carried by the robot can also facilitate localization, with examples including but not limited to global positioning systems (GPS) or artificial landmarks.

Sensing and localization are critical to a multi-robot system as it provides information on the state of each robot. This information would then enable motion planning, task allocation and coordination. For instance, given a multi-robot system and a task location, the multi-robot system must decide which robot to send to that task location. In the simplest case, the closest robot would be deployed. This decision would require knowledge of the position of each robot relative to the target position. Once deployed, that robot must navigate to the task location while avoiding obstacles.

Sensor noise and indirect measurements are two common issues regardless of the object being localized or the localization method. Sensor noise is random variations of sensor output. Indirect measurement refers to calculating position and orientation based on the results of other measurements. Some sensors measure acceleration or velocity, which must be integrated to yield position. This calculation would increase error or drift as the integration would compound the sensor noise from the acceleration or velocity measurement.

A typical solution is data fusion from multiple sensors using a Kalman Filter (KF) or Extended Kalman Filter (EKF) [3]. The KF is a recursive state estimator with two distinct steps: predict and update. In the predict step, the KF estimates the state X (2.1) along with its uncertainty P (2.2). This estimate is calculated from a model of the system (described by F and G) and control input:

$$\hat{X}(k+1|k) = F(k)\hat{X}(k|k) + G(k)u(k) \tag{2.1}$$

$$P(k+1|k) = F(k)P(k|k)F^T(k) + V(k) \tag{2.2}$$

where k is the timestep, $\hat{X}(k+1|k)$ is the estimated state of the system at time $k+1$, $P(k+1|k)$ is the predicted covariance at time $k+1$. $F(k)$ is a matrix describing system dynamics, which is the model of the system. $G(k)$ is a matrix describing how the input $u(k)$ affects the system's dynamics. $V(k)$ is process covariance.

In the update step, the KF observes the estimate and updates using information from a new sensor measurement (2.3). This information also updates the uncertainty (2.4). This recursive algorithm can run in real-time, only requiring present input measurements, previously calculated state, and uncertainty:

$$\hat{X}(k|k) = \hat{X}(k|k-1) + R(k)v(k) \tag{2.3}$$

$$P(k|k) = P(k|k-1) - R(k)H(k)P(k|k-1) \tag{2.4}$$

$$v(k) = y(k) - H(k)\,\hat{X}(k|k-1) \tag{2.5}$$

where $\hat{X}(k|k)$ is the updated state estimate at time k, $\hat{X}(k|k-1)$ is the state estimate of the system at time k using information from timestep $k-1$. $R(k)$ is the kalman gain [3] and $v(k)$ is measurement residual (2.5). Within the measurement residual, $H(k)$ is the matrix that maps states into outputs and $y(k)$ is the system's measured output. $P(k|k)$ is the updated covariance at timestep k. Within (2.3), the Kalman Gain $R(k)$ determines relative value of each component of the overall estimate between $\hat{X}(k|k-1)$ and $v(k)$.

2.1.1 – Self Localization

Self localization is a robot's ability to estimate its own pose $S = [x \quad y \quad z \quad \theta \quad \psi \quad \phi]^T$ where (x, y, z) represents translational position and (θ, ψ, ϕ) represents rotations along the x, y, z axes, respectively. In a multi-robot context, it is critical to localize each agent to ensure they are at the right place and time while preventing collisions. While measurements from onboard and offboard sensors can facilitate this localization, a combination of both types of sensors would be ideal.

Onboard localization is typically facilitated through some form of odometry [4], which is the measurement of distance travelled. This measurement can be a direct distance measurement or an integration of a velocity estimate. Within odometry, there are visual

odometry, LiDAR odometry and wheel encoder odometry, each having their own use cases with distinct advantages and disadvantages.

Visual odometry uses optical flow from imagery to measure changes in the camera's pose [5] and is typically used to facilitate localization in autonomous flight applications [6]. The camera captures a series of consecutive images and extracts features. These features are matched, and changes in matching features' positions in the camera images are measured as optical flow.

LiDAR odometry [7], [8] behaves similarly, having the same process as visual odometry, except it uses point clouds instead of camera imagery. Both visual and LiDAR odometry have their strengths and weaknesses. Visual odometry has a greater range than LiDAR odometry while being cheaper. However, it relies on more complex computer vision algorithms as camera imagery is a 2D representation of the world and does not contain distance information. Abrupt, rapid, or discontinuous changes in the camera imagery are also detrimental to visual odometry accuracy. These changes can occur from rapid vehicle movement or significant changes in the environment. Visual odometry also relies on acceptable lighting conditions for the cameras to detect features.

LiDAR odometry can provide more accurate pose estimates because point clouds are a 3D representation of the world with distance and direction information on the detected point relative to the LiDAR. At the same time, LiDAR can function with poor lighting. However, LiDAR has limited range compared to a camera, limiting its use to close spaces in which the robots' environment has features within LiDAR range.

Based on the strengths and weaknesses of visual and LiDAR odometry, they have their own use cases where one is better. LiDAR odometry is more accurate for enclosed areas, especially if the lighting is poor or the robot is unsteady with significant vibrations [9]. For more open areas, visual odometry is superior, as in these conditions, there may not be sufficient features within LiDAR range.

Wheel encoder odometry measures the rotational speed of wheels $\Omega = (\omega_1, \omega_2, .., \omega_n)$ to estimate the robot's velocity and, subsequently, any changes in its pose [10]. The mobile robot would have a kinematic model describing its position, velocity, and acceleration in terms of the rotational speed of its wheels. The wheels' rotational speeds are measured by the wheel encoders and then inputted into the kinematic model to yield the robot's current pose and velocity, $S = f(\Omega)$ and $\dot{S} = \dot{f}(\Omega)$.

An advantage of wheel encoder odometry is that it does not need external objects or features to be detected. However, it is only limited to unmanned ground vehicles (UGV), and its accuracy can suffer from poor terrain. The kinematic model would only consider Ω and not the type of terrain. If the wheels are spinning, the respective pose and velocity from the kinematic would be calculated, even if the wheels are slipping on ice or mud. A bumpy path in which wheels lose contact with the ground would also result in a noisy estimate.

Lastly, among onboard localization options, robotic agents can also carry an IMU that provides immediate and real-time measurements of acceleration $\ddot{S}$. An IMU cannot be used by itself to estimate pose, but incorporating other sensors that provide odometry can greatly improve pose estimates. and relative orientation (θ, ψ, ϕ). Under theoretical conditions, IMU data can be integrated into velocity and position (dead reckoning) [4]. However, double

integration from acceleration to position would further compound sensor noise. As previously discussed, a KF or EKF can facilitate this sensor fusion.

For offboard localization, robots can use GPS, motion capture or artificial landmarks. These methods require information from an external source, which has its limitations. GPS can provide a global position estimate with minimal drift if there is adequate reception. However, it is not always a continuous measurement and may be inaccurate within several meters. Real-Time Kinematic (RTK) GPS solutions [11] are available but require the robot to operate within range of an RTK station. This solution provides real-time 3D position estimates with cm levels of accuracy to robotic agents within 10 km to 30km of the RTK GPS station, depending on the environment and the system's properties.

Motion capture (MoCap) can provide extremely accurate (within mm), real-time pose estimates but requires the robots to work in the environment under observation with specialized equipment and infrastructure. Artificial landmarks can provide accurate poses if detected but also require prior setup. Examples of artificial landmarks can include visually identifiable markers, radio frequency identification (RFID) tags, and signal-emitting beacons. In addition to requiring prior setup, the robots need a sensor that can detect the artificial landmark.

A complete localization solution should employ a combination of sensors and then fuse them with a KF or EKF. A common localization solution for unmanned aerial vehicles (UAV) is visual inertial odometry (VIO) combined with GPS [12]. This approach uses the EKF twice: the fusion of visual odometry and IMU for VIO and then the fusion of VIO and GPS. This EKF takes VIO pose estimates for the state prediction step and uses GPS position measurements for the state update step, defined in (2.1) to (2.5). This forms a complete localization solution where VIO provides continuous pose estimates while GPS provides intermittent global estimates. A robot could use VIO to localize in between GPS coordinate updates. At the same time, should there be issues with GPS, the EKF could assign less confidence to the GPS update and rely more on the VIO update. While the solution proposed in [12] was considered for a UAV, it can also be applied to other types of robots as it fuses odometry with GPS. For example, a UGV can use wheel encoder odometry and/or visual odometry to fuse with GPS, and [13] also proposes visual odometry for unmanned surface vehicles (USV).

2.1.2 – Target Localization

Target localization is a robot's ability to estimate the pose of another object relative to itself. Common sensors on mobile autonomous robots for target localization are cameras and LiDAR. However, depending on the application, other sensors such as radar, ultrasonic, sonar, and laser range finders are also available.

A common approach to target localization is to use a camera to estimate a target's relative position from an observer and combine it with the observer's position estimate. This approach was discussed in [14] and [15] using a quadcopter with a monocular camera. With the pinhole camera model [16], the pixel coordinates of a detected target can be converted into a unit vector describing the direction from the camera to the target. In [14], this unit vector, P_M^u is denoted in (2.6).

$$P_M^u = [x_m^u \quad y_m^u \quad z_m^u]^T \tag{2.6}$$

The components of this unit vector P_M^u can be combined with the quadcopter's pose estimate to determine the position of the ground target P_N, denoted in (2.7):

$$P_N = \begin{bmatrix} \frac{hx_m^u}{z_m^u} & \frac{hy_m^u}{z_m^u} & h \end{bmatrix}^T + R[x_{b0} \quad y_{b0} \quad z_{b0}]^T \tag{2.7}$$

$$P_N = [\Delta x_N \quad \Delta y_N \quad \Delta y_N]^T + R[x_{b0} \quad y_{b0} \quad z_{b0}]^T$$

where h is the height of the quadcopter, which in [14] was obtained from an onboard ultrasonic sensor, but other options for estimating h can be used, such as barometer or VIO. (x_{b0}, y_{b0}, z_{b0}) describe the position of the camera from robot's pose estimate and physical configuration, and R is a rotation matrix to ensure that both $[\Delta x_N \quad \Delta y_N \quad \Delta y_N]^T$ and $[x_{b0} \quad y_{b0} \quad z_{b0}]^T$ are in the same reference frame.

Vision-based localization can also be supported with the use of target markers. Target markers were used in [17] to enable a UAV to locate, track and land on a UGV using only monocular vision. With this target marker, a specific set of features can be programmed into the image processing algorithms rather than matching the target UGV's general pattern. The imagery from the monocular camera is processed to detect the target marker through changes in pixel intensities. Since the target marker's shape, size and colour are known, it can be digitally represented and searched for on the processed image and then used to estimate the target position. However, some form of image classifier would be required without target markers. A possible option is a machine learning or deep learning model trained on a dataset representative of the expected targets [18] to detect them and estimate their orientation [19].

The same target localization that enables a robot to visually localize an external target can also be used to help a robot localize itself. This self localization uses artificial landmarks, such as in [20] and [21]. Looking back at (2.7), if a robot detects an object whose position is known, we have P_N. If the size of the object and the robot's orientation is known, this information can be used to calculate $[\Delta x_N \quad \Delta y_N \quad \Delta y_N]^T$, which is the vector from the robot to the object. The robot's orientation can also be described as R in (2.7). Therefore, we can solve this equation for (x_{b0}, y_{b0}, z_{b0}) to yield the position of the robot that observed an artificial landmark located at P_N.

Another type of sensor for object detection is LiDAR, which can detect objects around the robot if they are within range. This application was discussed in [22], in which a LiDAR device could detect obstacles of up to 1 meter in size at distances of up to 10 meters. The solution proposed in [22] consisted of data acquisition, data filtering, pre-processing, and clustering. After data was acquired from LiDAR, a median filter was used to remove salt and pepper noise. The data was then pre-processed, identifying the angle and distance to each point. Data points were then clustered based on shape association (circle, line and rectangle) using heuristic rules. An output file listing obstacles, their classifications and positions was then created.

A proposed representation of a detected obstacle was discussed in [23] using 2D LiDAR. This representation converts the distance and angle measurements (d, θ) from the LiDAR and converts them into a point cloud (x_i, y_i), denoted in (2.8):

$$(x_i, y_i) = (d_i * cos\theta_i,\ d_i * sin\theta_i) \tag{2.8}$$

where x_i, y_i correspond to coordinate from the i^{th} LiDAR distance measurement and angle measurement d_i and θ_i, respectively. This is further visualized in Figure 2.1, showing the frontal side of the obstacle detected.

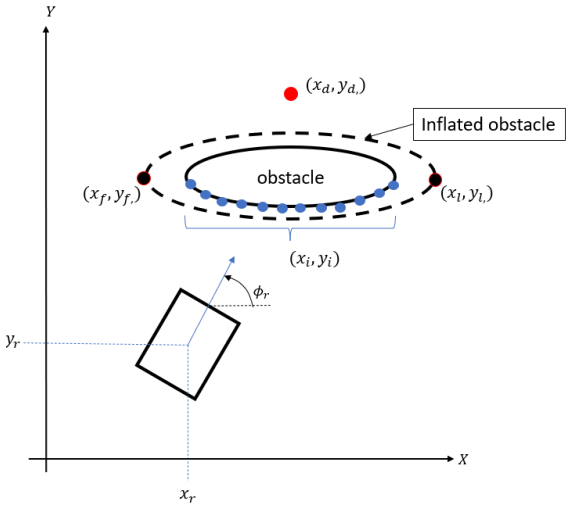


Figure 2.1 – LiDAR Obstacle Detection

After creating that point cloud, the obstacle is inflated, and the edges of that obstacle are extracted. The point on the farthest left edge of the inflated obstacle is (x_f, y_f) and the point on the farthest right edge of the inflated obstacle is (x_l, y_l). (x_r, y_r) is the robot's position, ϕ_r is the robot's heading and (x_d, y_d) is the goal position. If an obstacle is detected, the robot can be re-directed to go to (x_f, y_f) or (x_l, y_l) as an intermediate goal before moving towards (x_d, y_d). This re-direction is why the obstacle must be inflated to provide a margin of safety for the robot to safely navigate around the obstacle and go to the goal.

Both camera and LiDAR have their strengths and weaknesses for target localization [24]. Cameras can distinguish changes in colour and lighting over long distances while being cheaper than LiDAR. These characteristics make cameras suitable for tracking moving objects or multiple objects at a greater distance than LiDAR. LiDAR has a limited range but can provide accurate distance measurements and the physical shape of detected objects. These characteristics make LiDAR suitable for indoor mapping and obstacle avoidance. Both LiDAR and camera can use a variety of classification algorithms on the image or point cloud to determine the type of object being detected. Cameras are more suitable for tasks requiring high-resolution images and colour information. LiDAR is more suitable for tasks that require accurate distance and shape measurements at close range.

2.2 – Motion Planning

Motion planning is the process of generating intermediate poses S, along with their derivatives $\dot{S}$ and $\ddot{S}$ to drive a robot to a desired goal pose. Where pose is defined in (2.9) in terms of (x, y, z) positions, and with (θ, ψ, ϕ) corresponding to rotations along those axes, respectively.

$$S = [x \quad y \quad z \quad \theta \quad \psi \quad \phi]^T \tag{2.9}$$

The generation of intermediate poses, or waypoints, can be facilitated through graph-based algorithms such as A* or randomly exploring random tree (RRT) [25]. Both A* and RRT discretize the known environment into nodes and attempt to find the sequence of intermediate nodes that result in a path from the starting node to the goal node.

The A* algorithm generates a path by selecting intermediate nodes with minimal cost. The cost to go to node i is as follows:

$$F_i(n) = G_i(n) + H_i(n) \tag{2.10}$$

where $G_i(n)$ is the distance from the starting node and $H_i(n)$ is a heuristic function or distance from the end node. The A* algorithm computes $F_i(n)$ of all nodes adjacent to the starting node (node 0) and selects node 1 with a minimal value of $F_i(n)$. This computation is repeated for all nodes adjacent to node 1 to find node 2. It is then repeated until reaching the goal node.

With the RRT algorithm, nodes are placed and connected to form a tree until creating a path that terminates at the goal node or close enough to the goal node:

- An additional node is placed randomly in configuration space at a preset distance from the previous node.
- The additional node is discarded if there is an obstacle between the additional node and the closest node.
- An additional node is connected to the closest existing node or to an existing node to minimize the total distance.
- Iterate until the connected nodes yield a path from the starting node to the goal node or close enough.

With these graph-based algorithms, nodes are connected as a sequence of waypoints for the robot to travel. In addition to generating these waypoints, a motion planner must also generate the required velocities to drive the robot from its current pose S_c, to these desired poses S_d. Within control theory, a similar problem exists of driving a dynamic system from its current state to some desired setpoint state. While many control schemes exist, they often involve measuring the error between the system's measure state and the desired state and then using that error to generate a control signal, such as in PID control [26]. Likewise, this concept is applied to motion planning; a motion planning algorithm would be required to calculate a velocity to drive a robot from its current pose S_c to some desired pose S_d. This velocity will be some function (2.11) whose input is pose error $\Delta S = S_d - S_c$:

$$\dot{S} = f(\Delta S) \tag{2.11}$$

2.2.1 – Single Robot Motion Planning

Motion planning for a single robot must consider the physical capabilities of that robot, along with its intended task and working environment. UGVs, UAVs, unmanned underwater vehicles (UUV) and USVs have different working environments and considerations. UAVs and UUVs are capable of 3D motion, with UAVs moving through the air and UUVs moving underwater. Just as UAVs must respond to changing wind conditions, UUVs and USVs must also respond to the movement of waves and water currents.

Obstacle avoidance can be described in two steps: obstacle detection and trajectory adjustment. Obstacle detection can be facilitated through the methods discussed in Section 2.1. Trajectory adjustment entails modifying the velocity or intermediate goal poses to avoid collision with an obstacle while still driving the robot to its final desired pose. Two options for this trajectory adjustment are bug algorithms [27] and artificial potential fields [28].

In general, bug algorithms entail some form of circumnavigation around an obstacle before continuing toward the next goal pose. This bug algorithm was combined with a Dubins Path [29] to bypass obstacles in [27]. If an obstacle is detected, the robot will attempt to avoid it by moving perpendicular to it until an unobstructed path to the goal is available. The combined Dubins path and bug algorithm solution calculates an intermediate pose $p_{in} = (x_{in}, y_{in}, \phi_{in})$ from the current pose $p_c = (x_c, y_c, \phi_c)$, denoted in (2.12) and (2.13*)* and shown in Figure 2.2. These intermediate poses are calculated such that the curve representing the robot's trajectory is of minimum distance, subject to the robot's turning radius, LiDAR range and safety distance; $r_{turn},\ r,\ r_{safe}$ respectively. The bug algorithm attempts to maintain at least r_{safe} distance away from obstacles detected up to r distance from the robot. These intermediate pose calculations are repeated until a path is available to the final pose $p_f = (x_f, y_f, \phi_f)$.

$$(x_{in}, y_{in}) = (x_c - d_{min} * \sin(\phi_c + \Delta\phi),\ y_c - d_{min} * \cos(\phi_c + \Delta\phi)) \tag{2.12}$$

$$\phi_{in} = \operatorname{atan}\left(\frac{y_f - x_{in}}{x_f - x_{in}}\right) - \operatorname{asin}\left(\frac{r_{turn}}{\sqrt{(y_f - x_{in})^2 + (x_f - x_{in})^2}}\right) \tag{2.13}$$

where:

$$d_{min} * \sin(|\Delta\phi - \phi_n|) \geq r_{safe}, \qquad n = \{1,2\} \tag{2.14}$$

$$\phi_1 = angle\ to\ p_1, \qquad \phi_2 = angle\ to\ p_2$$

It must be noted that d_{min} is the minimal distance to the obstacle, p_1, p_2 are the leftmost and rightmost points detected on the obstacle, as shown in Figure 2.2. The bug algorithm is effective at avoiding collision using locally available information. While it can be sufficient in a simple environment, it is often suboptimal in a more complex environment. A complete motion planning solution should combine local planning, such as the bug algorithm, with some form of

global planning, which can be facilitated by the graph-based algorithms previously discussed [25]. Furthermore, with the bug algorithm, graph-based methods may not require as much granularity on the environment since the robot would have some degree of online autonomous obstacle avoidance and navigation.

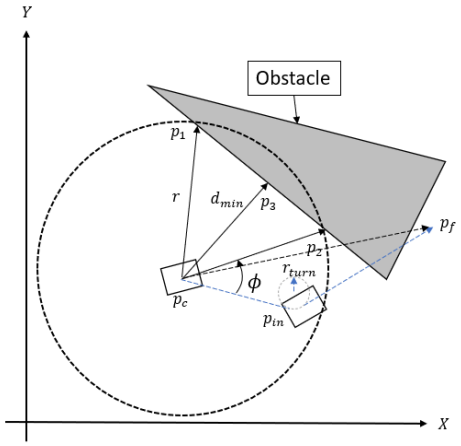


Figure 2.2 – Obstacle Detection and Dubins Path Trajectory

While bug algorithms are typically explained in terms of motion along the XY plane for UGVs (and USVs), they can also be applied to UAVs or UUVs. In the case of robots that can translate in 3 dimensions, a bug algorithm can circumnavigate an obstacle along an arbitrary plane of motion in 3D space (does not have to be parallel to XY plane) such that the travel distance is minimized. Alternatively, the robot can ascend or descend in front of a smaller cross-section of the obstacle and then apply the bug algorithm to bypass it there.

Artificial potential fields [28] can also be used to drive a robot to a destination while avoiding obstacles. This solution involves creating attractive and repulsive vector fields within the robot's working environment. Obstacles can be represented by repulsive vector fields, while the destination can be represented as an attractive vector field. Ideally, the sum of vectors at the robot's position would point toward the goal pose. This vector sum is visualized in Figure 2.3 [28], where $F(X)$ is the overall "net gravitational force", $F_{rep}(X)$ is the net repulsive force and $F_{att}(X)$ is the net attractive force. The robot's motion would be in the same direction as $F(X) = F_{att}(X) + F_{rep}(X)$.

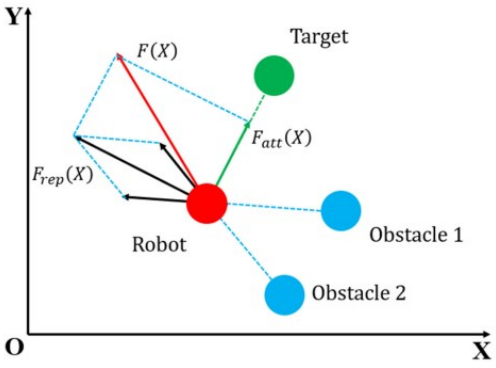


Figure 2.3 – Schematic Diagram of Artificial Potential Field [28]

While Figure 2.3 only depicts a 2D scenario, artificial potential fields can be generalized to 3D motion using 3D vector fields. Artificial potential fields can suffer from issues such as gravity imbalance (attractive force from goal pose behind an obstacle is stronger than repulsive force from that obstacle), local minimum and local oscillation. Additional solutions and modifications are required to address these limitations.

A comparison of bug algorithms, potential fields and A* was discussed in [30]. They each have strengths and weaknesses regarding reliability, optimality, and computational demand. Bug algorithms have low demand and work in real-time but may be suboptimal. A* is the most optimal but requires the most knowledge and computation time. Artificial potential fields are less demanding than A* and can be more optimal than bug algorithms but may get stuck in local optima. Furthermore, while obstacle avoidance is a more prevalent issue with UGVs, it is still an issue with UAVs, USVs and UUVs. The same algorithms that facilitate obstacle avoidance for UGVs can also be applied to other types of mobile robots.

A challenge unique to UGVs is terrain, which must be modelled to avoid suboptimal trajectories. An approach to terrain modelling was proposed in [31], which uses a 3D terrain map to consider the shape of the ground in generating the UGV's trajectory. This proposed motion planner then implements a variation of RRT to navigate along that terrain model.

A cost matrix was applied in [32] to partition a UGV's environment into discrete grid squares, each of them having an associated cost. The motion planner aims to reach a goal pose while minimizing the cost incurred. The cost incurred is the sum of the costs of all traversed grid squares. Within [32], an autonomous UGV generates this cost matrix by fusing data from its sensors and assigning positive numbers to the obstacles they detect. Negative value contributions are added to the cost matrix to areas corresponding to the desired heading. This obstacle detection and cost matrix is shown in Figure 2.4 [32].

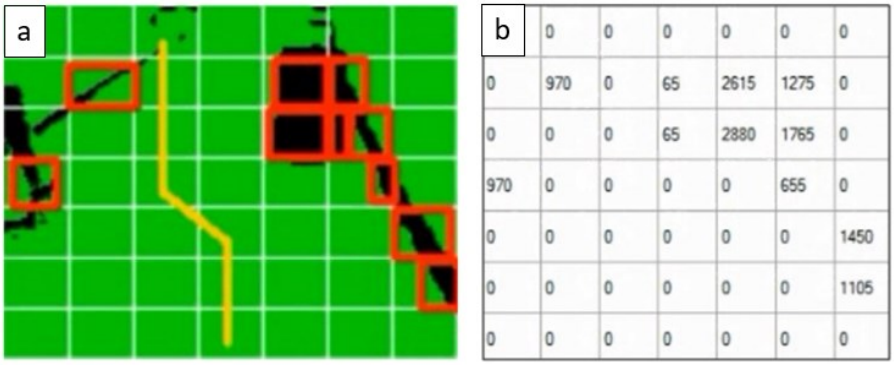


Figure 2.4 – Robot Environment Representation: a) Obstacle Detection, b) Cost Matrix Representation [32]

The UGV in [32] navigates by generating the cost matrix, calculating the cost of all paths, and then moving on to the lowest cost path. This approach, however, is not just limited to UGVs and can be extended to other types of mobile robots. For UAVs and UUVs that can move in 3D, instead of having just a cost matrix, we can have a stack of cost matrices, with one cost matrix for every step of elevation/height. Each entry in this stack of cost matrices would represent the cost of a 3D cube within the environment. This 3D representation can be generated through OctoMaps [33] in a manner similar to [31].

UAVs' flight and ability to move in 3D dimensions offer unparalleled mobility through their ability to change their height at will. Furthermore, multi-rotors can also take off and land vertically, as well as hover in place, making them suitable for flight in constrained environments. Virtual targets were proposed in [34] as a motion planning solution for quadcopter UAVs. This solution entails the quadcopter tracking a virtual target flying in a desired trajectory with the intent that the quadcopter will match the virtual target's trajectory. The quadcopter uses pursuit guidance to follow the virtual target without intercepting it. The geometry of this pursuit guidance is shown in Figure 2.5 [34] where (x_v, y_v, z_v) is quadcopter position and (x_t, y_t, z_t) is target position. ψ_v is heading of quadcopter's direction of motion, ψ_c is commanded azimuth angle and ψ_t is the azimuth angle for the target. θ_t is target elevation angle, θ_v is quadcopter elevation angle and θ_c is the commanded elevation angle.

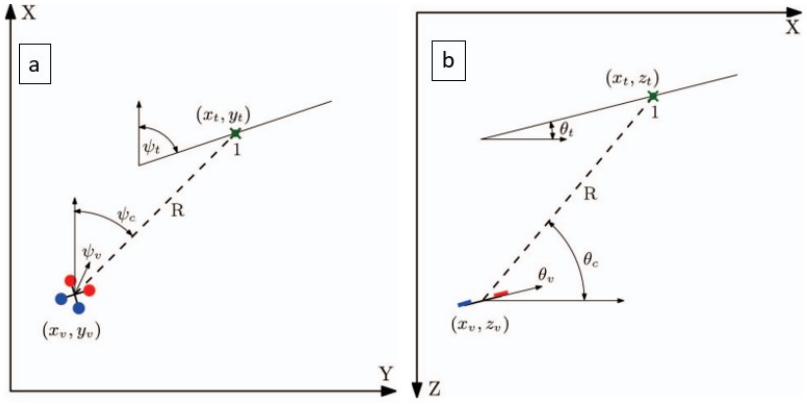


Figure 2.5 – Pursuit Guidance Geometry a) xy Plane, b) zx Plane [34]

Within the geometry of pursuit guidance in Figure 2.5, the commanded azimuth and elevation angles for the quadcopter are denoted on (2.15) and (2.16). While this proposed method is intended for UAVs, it can also be implemented on UUVs that can move in 3D underwater. A limited implementation using only the azimuth angle can provide pursuit guidance for UGVs and USVs.

$$\psi_c = atan\left(\frac{y_t - y_v}{x_t - x_v}\right) \tag{2.15}$$

$$\theta_c = atan\left(\frac{z_t - z_v}{\sqrt{(x_v - x_t)^2 + (y_v - y_t)^2}}\right) \tag{2.16}$$

An approach similar to pursuit guidance with virtual targets is the synthetic waypoint guidance algorithm proposed in [35]. This algorithm implements the guidance law in (2.17):

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}_i = \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}_{i-1} + V_w \begin{bmatrix} cos(\gamma_{ref})\cos(\psi_{ref}) \\ cos(\gamma_{ref})\sin(\psi_{ref}) \\ -\sin(\psi_{ref}) \end{bmatrix} \Delta T \tag{2.17}$$

where $[X_w \quad Y_w \quad Z_w]^T$ is the position of the moving synthetic waypoint, i is the timestep, V_w is the speed of the synthetic position, and ΔT is the desired time horizon to update the virtual waypoint's position. The algorithm in [35] also imposes a minimum virtual range R^*; the UAV

must always maintain this distance away from the virtual waypoint. This distance R^* is determined based on the UAV's speed and a desired time horizon ΔT to initiate a response to flight path changes. Similar to [34], this virtual waypoint can also be implemented on other types of mobile robots with modifications.

While the previously discussed algorithms can also be applied to UUV and USV, the unique operating environment of these types of robots must be considered. In particular, the constantly changing current of the body of water in which the UUV or USV operates was considered in [36] to optimize a global path planning solution. A time variable was used to adjust the cost of moving from one node to another to account for the effect of moving water. This time variable was obtained through a fluid mechanics model of the coastal environment in [36]. With this environment model, both $G_i(n)$ and $H_i(n)$ in (2.10), would change over time.

Motion planning for a single robot entails generating intermediate poses and velocities to drive that robot from its current pose to a goal pose. The underlying algorithms behind each solution can typically be applied to any type of mobile robot with some modification. Notably, while [37] discusses motion planning for UAVs and [38] discusses motion planning for UGVs, there is a significant overlap between the solutions discussed in the literature. For example, bug algorithms are typically used on UGVs. However, there is no reason a UAV cannot mount a LiDAR and circumnavigate an obstacle. There may be situations where circumnavigating an obstacle (instead of flying over it) would be better. Pursuit guidance on UAVs can work with UUVs that can also move in 3D underwater or with UGVs and USVs with modification. Even the aquatic environmental factors in [36] can be modified to use air instead of water and then re-purposed for UAVs.

Motion planning algorithms can also be described in terms of online and offline execution. Typically, an offline algorithm will be more optimal (reach the goal pose with less distance/time) but requires prior knowledge of the environment and can be computationally demanding. Online algorithms are less optimal but can work in an unknown environment. A complete single robot motion planning solution would use both offline and online methods to navigate. For example, a robot can use offline A* for large, well-defined areas of a workspace but then use an online mapping such as in [33] with online A* or bug algorithm for less well-defined areas.

2.2.2 – Multi-Robot Motion Planning

A multi-robot system can employ its robotic agents in the simultaneous execution of different tasks in different locations. However, it imposes the additional requirement of managing multiple positions and velocities and the possibility of collision between two or more robots. The goal of motion planning is still generating intermediate poses and velocities to drive a robot to a goal pose. However, with multiple robots, there are multiple goal poses corresponding to each of the robots, and the motion planner must ensure that the generated intermediate poses and velocities avoid collision.

The fact that there are multiple robots further emphasizes the need for accurate and consistent localization. To account for measurement uncertainty among multiple robots, [39] proposes adaptive Monte-Carlo localization. This solution entails predicting, updating, and resampling the measurements to determine the probability density function for each agent's pose and velocity estimates. With sufficient samples, the localization uncertainty is limited, and

the robot's position can be located within a certain area within an acceptable confidence level. Other solutions for multi-robot localization include heuristically tuned EKF [40], collaborative landmark localization [41], and extensions of the solutions presented in Section 2.1.

A well-established solution for multi-robot motion planning is using the velocity obstacle (VO), the set of velocities that, if unchanged, eventually results in a collision over a finite time period [42]. Each robot calculates a velocity obstacle for the other robots and adjusts its own velocity accordingly. If the velocity is outside the velocity obstacle, then the trajectory is guaranteed to be collision-free. Note that the VO can be applied to any obstacle (moving or stationary), not just other robots.

Two prominent velocity obstacle solutions are the reciprocal velocity obstacle (RVO) [43] and the optimal reciprocal collision avoidance (ORCA) algorithm [44]. Both of these algorithms share the following elements:

- Calculation of ideal velocity for robot i towards its goal, v_i^{pref}. This velocity would yield a straight-line trajectory to the goal in the absence of obstacles.
- Inflation of the robot's physical radius. This inflated radius R_i is the sum of the robot's physical radius and a radius inflation factor to provide a safety margin for velocity adjustments.
- Calculation of velocity obstacles VO and set of collision-free velocities V_{nc} using the geometry of robots' positions, velocities and inflated radii. Collision-free velocities are velocities that do not intersect with the velocity obstacle VO.
- Selection of optimal collision-free velocity v_i such that it is of minimal difference from ideal velocity; $v_i = \underset{(v \in V_{nc})}{argmin} ||v - v_i^{pref}||$.

While similar, RVO and ORCA have two notable differences. Within RVO, if there are no available collision-free velocities, then the selected velocity is the one that maximizes time to collision, with the assumption that dynamic replanning can avoid a collision. Within ORCA, the elements in V_{nc} contain a weighting factor $\lambda_{i,j}$, where $\lambda_{i,j} + \lambda_{j,i} = 1$. This weighting factor defines how much each robot (i or j) is involved in avoiding a collision.

Other types of VO solutions are the acceleration-velocity obstacle (AVO) proposed in [45] as well as the hybrid reciprocal velocity obstacle (HRVO) discussed in [46]. The AVO imposes constraints on acceleration as well as velocity. While the regular VO only defines velocities that would result in a collision, AVO also defines accelerations that would result in a collision. Proportional control was also proposed in [45] to ensure that the acceleration of each robot does not result in a collision. This additional consideration offers improved predictions of other robots' future positions, enabling smoother and more optimal trajectories. However, factoring in acceleration increases computational demand and complexity, and it may yield little benefit in low speed and low acceleration situations where standard VO may suffice.

HRVO differs from ORCA as HRVO builds a "hybrid" velocity obstacle that represents a compromise between reciprocal velocity obstacles (RVO) used in ORCA and a regular VO. With HRVO, responsibility for collision avoidance is shared between two robots but favoured towards the robot with a higher priority [46]. This prioritization can be determined by identifying which side is preferable for passing. Given two robots, robot A and robot B, if robot A tries to pass robot B on the wrong side, then movement priority is assigned to robot B. However, if robot A

chooses the correct side, equal cooperation is assumed. HRVO was found to have more collisions and computation time than ORCA in [46]. However, one advantage of HRVO is that it can be used for a more flexible distribution of responsibility between multiple robots, which may be ideal if certain robots are considered more important than others. For example, suppose a task must be performed with utmost urgency. In that case, priority for movement should be assigned to robots that can perform that task. At the same time, robots unable to perform that task should be considered a lower priority.

In addition to VO based solutions, previously discussed algorithms can be modified to work from a single robot scenario to a multi-robot scenario. Artificial potential fields were given additional rules in [47] to account for multiple robots. The RRT and A* algorithms were also applied in [48] and [49], respectively, to address multi-robot motion planning.

A combined solution involving improved artificial potential fields with priority-based rules was proposed in [47]. These rules were deterministic and were based on the geometry of the robots' positions and velocities relative to each other. Similar to VO, each robot can also be modelled as an obstacle to other robots. For example, given robots A, B, and C, robots B and C can exert a repulsive force on robot A to avoid collision. However, if robot A has to travel to robot B to perform a task involving robot B, the vector field from robot B to robot A can be changed to an attractive force. This option provides flexibility for artificial potential fields in multi-robot motion planning. Furthermore, similar to the single robot case, the static portions of the environment can largely be modelled offline, with robots and detected obstacles being modelled in real-time from sensor data. This modelling is then used to generate attractive and repulsive vector fields offline for the larger environment and online for dynamic features such as other robots moving in the shared workspace.

The RRT* algorithm was extended in [48] by considering motion constraints and sensor ranges of robots to calculate paths in real-time. Note that RRT* is an optimized version of RRT; RRT* seeks to reduce the path length by introducing a "re-wiring" step in which a new node is added to the path and checks if any adjustments are possible to shorten the path length. RRT* typically results in shorter path lengths but requires more computation time. Within [48], the RRT* algorithm was modified to avoid corresponding to robots' possible future positions. This approach of avoiding certain sectors completely relies on estimating future robots' positions. Each estimated sector is effectively an obstacle. Suppose this estimate is of high accuracy and certainty. In that case, the size of the sectors can be minimized, reducing the size of the obstacle and enabling more optimal trajectories. However, a collision may occur if the estimate is wrong, such as if one robot is delayed at a sector at the wrong time.

As for [49], the A* algorithm was applied to multiple robots conducting a search task. Nodes in the grid are tracked as either open or closed. A node is closed if it is an area that has already been evaluated or visited by one of the robots, indicating that the area has been searched and that no other robot needs to search it again. A node is considered open if it has not been visited. This approach is suitable for ensuring area coverage. However, it does not consider shared paths between multiple robots, as it is not designed for that task. It may be possible to consider some nodes as open even after being traversed by a robot so they can be used as shared paths in the environment.

2.3 – Multiple Robot Task Allocation

A combined task allocation and coordination framework dictates the overall behaviour of a multi-robot system. Task allocation in a multi-robot system entails determining which robots execute which tasks and in what order. Coordination describes how the robots interact with each other and the environment in the execution of their tasks. This group behaviour is facilitated by communication between the robots and a control station. The robots must communicate information on their state so that the task allocation and coordination framework can determine the most optimal selection of robots to complete a given set of tasks.

Implementing this task allocation and coordination involves a hierarchical structure describing different levels of control [50], with communications between each level facilitated by technologies in [51]. This hierarchical structure involves a high level, intermediate and low levels of control. Higher Level is primarily concerned with mission planning and data processing, while low level interfaces directly with the sensors, actuators, and overall physical robots. The intermediate level is the interface between higher level and lower level, it processes signals and data between the higher and lower levels. Among those levels, the higher level interacts with a human operator. Another critical function of the high level is acting as a mission coordinator that determines which robots perform which tasks, in what order.

Multi-robot task allocation (MRTA) is an optimization problem that entails distributing tasks to robots such that some objective function is maximized or minimized under a set of constraints [52]. This problem is an extension of the multiple travelling salesmen problem (MTSP) [53]. Within the MTSP, multiple salesmen must travel to multiple cities within a certain sequence and distribution to minimize total travel distance. With the MTSP, the objective function to be minimized is total travel distance; the constraints are that all cities must be visited only once, and the salesmen must return to their starting location [54].

Both the MRTA and MTSP can follow a similar formulation as in [52] and [55], describing some function objective function F to be maximized or minimized (2.18). This objective function is the sum of costs or rewards of all paths taken depending on the formulation of the problem:

$$F = \sum_{i=1}^{n_r} \sum_{j=1}^{n_t} c_{i,j} x_{i,j} \tag{2.18}$$

where n_r, n_t are the number of robots and tasks, respectively, $c_{i,j}$ is a value representing the cost or reward incurred by moving from node i to node j. $x_{i,j}$ is a decision variable, $x_{i,j} = 1$ if the path from node i to node j is selected and $x_{i,j} = 0$ if that path is not selected. This formulation is visualized in Figure 2.6, showing 6 nodes. Nodes 1 and 2 correspond to the start locations of robot 1 and robot 2. Nodes 3, 4, 5, and 6 correspond to tasks 3, 4, 5, and 6, which the two robots must complete in an optimal manner. The overall scenario is shown in Figure 2.6a, and a possible solution is shown in Figure 2.6b. Note that the solution in Figure 2.6b may not be optimal; it is an arbitrary example to visualize a possible solution.

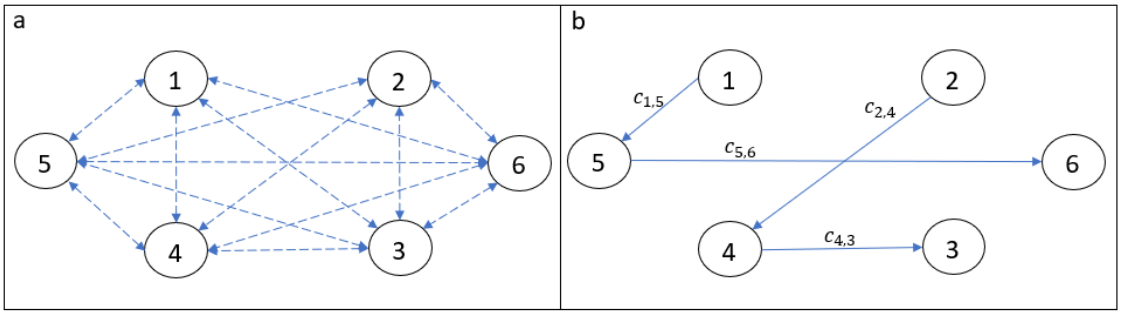


Figure 2.6 – Representation of MTSP / MRTA problem: a) graph representation, b) example task allocation

In Figure 2.6b, the objective function value is $F = c_{1,5} + c_{5,6} + c_{2,4} + c_{4,3}$, calculated from (2.18). If the values of $c_{i,j}$ correspond to distance between nodes, such as in the MTSP, then the optimization goal is to minimize the value of F. However, $c_{i,j}$ can also correspond to a reward function that assigns some quantification of value in completing a given task; in this case, the optimization goal is to maximize F. The desired output of the optimization is the set of $x_{i,j}$ that yield the optimal value of F. In the example provided by Figure 2.6b, $x_{1,5} = x_{5,6} = x_{2,4} = x_{4,3} = 1$, while all other $x_{i,j} = 0$.

Similar to MTSP, within MRTA, the inputs are a set of n_t tasks $T = \{T_1, T_2 \dots T_{n_t}\}$ and a set of n_r robots $R = \{r_1, r_2 \dots r_{n_r}\}$, and the output is some task allocation TA that determines which robots perform which tasks and in which order, such as in Figure 2.6b, with robot 1 performing task 5, then task 6, and robot 2 performing task 4, then task 3 (2.19).

$$TA = \begin{cases} r_1 = (T_5, T_6) \\ r_2 = (T_4, T_3) \end{cases} \tag{2.19}$$

To solve the MRTA / MTSP, several optimization algorithms have been considered in both [52] and [54]. An overview of existing solutions is presented in Table 2.1. The goal of these algorithms in Table 2.1 is to find an optimal TA given T and R.

Table 2.1 – Algorithms for MRTA / MTSP

Algorithm Type	**Description**	**Notes**
Deterministic	Analytical methods to solve a given optimization problem.	Provides an exact solution. Rare and computationally complex or suboptimal
Metaheuristic	Numerical, iterative methods to reach an optimal or close to optimal solution.	Established optimization algorithms. Examples: ant colony, particle swarm, genetic algorithm
Market based	Auction based methods in which tasks are assigned based on a bidding process from each agent	Centralized or decentralized auctions that are based on the best bid for the agent or the best bid for the auctioneer.
Other	Methods not covered in previously mentioned types of algorithms	Fuzzy logic, probabilistic, deep learning, and other methods not covered.

Deterministic algorithms will always produce the same output if given the same input [56]. With a deterministic algorithm, given the same T and same R, the resulting task allocation TA will always be the same. This is a significant advantage because it is predictable and consistent. However, these algorithms do not typically reach the optimal solution; rather, they reach some approximation of an optima, either local or global. On the other hand, deterministic algorithms that do find an optimal solution are complex and/or computationally demanding. Overall, deterministic algorithms can be described in one of two categories:

- Category 1: Optimal but excessive computational demand.
- Category 2: Suboptimal but low computational demand.

Examples of category 1 deterministic algorithms are dynamic programming [57] and mixed integer programming, used in IBM's CPLEX software [58]. Both approaches will yield an exact and highly optimal solution but with significant computational demand. In particular, the IBM CPLEX software was used to generate highly optimal solutions to the single depot minmax MTSP. However, it required several hours of computation [59].

The solution dataset in [59] was either used or cited as a benchmark in research such as [60], [54] and [61]. This solution dataset [59] will also be used to evaluate the optimality of our proposed task allocation algorithm in Chapter 3. Furthermore, the IBM CPLEX software is considered an industry standard [62] for commercially available optimization software. It is widely used to solve combinatorial optimization problems, including MTSP variations [63], [64], [65], but also the vehicle routing problem [66] and multirole assignment problem [67].

In contrast, metaheuristic algorithms do not always produce the same output even if given the same input. Even if the same T and R are provided as inputs, it is probable that the output TA from a metaheuristic solution would differ. Instead of always producing the same result, metaheuristic algorithms try to iteratively reach optimal solutions [56]. Metaheuristic algorithms have the advantage of reaching a highly optimal solution with reasonable computational demand. Their solution is more optimal than category 2 deterministic algorithms, often very close to category 1 deterministic algorithms but using fewer computational resources. A disadvantage, however, is that metaheuristic algorithms have a degree of variability and are typically less optimal than category 1 deterministic solutions. Furthermore, they use more computational resources than category 2 deterministic algorithms.

Market based algorithms model the system as robots bidding for tasks [68]. The best bid can be assessed as the maximum "net gain." Suppose a task T_j has a reward $g_{i,j}$ and cost $c_{i,j}$ when assigned to robot r_i. The net gain is then the difference between reward and cost, $g_{i,j} - c_{i,j}$. Market-based algorithms aim to maximize this net gain for the entire system, which is the sum of net gains for each robot-task assignment. The advantages of market-based algorithms are similar to metaheuristic algorithms in that market algorithms can find an optimal solution with reasonable computational demand. Market algorithms are less prone to local optima than some metaheuristic algorithms but are more computationally complex.

Metaheuristic Algorithms

Metaheuristic algorithms comprise the core of the SOTA regarding MRTA solutions. The most notable, well-established algorithms used to solve MRTA are particle swarm optimization (PSO) [69], genetic algorithm (GA) [70] and ant colony optimization (ACO) [71]. These algorithms are inspired by physical processes. The particle swarm optimization was inspired by the collective behaviour of decentralized, self-organized systems, such as a flock of birds. The genetic algorithm is inspired by the process of reproduction, natural selection, and evolution. Lastly, the ant colony optimization algorithm is inspired by the behaviour of ants following pheromone trails. While these algorithms are the most common within MTSP / MRTA, there are other metaheuristic algorithms, including but not limited to simulated annealing [72] and hill-climbing [73].

Particle Swarm Optimization (PSO)

The PSO algorithm models possible solutions as particles with a position and velocity at a given timestep. The particles move through a search space to find an optimal solution by adjusting their velocities. This velocity adjustment is guided by a particle's personal best position and global best position [74]. This algorithm was applied in [75] using a two-step process to find an optimal task allocation TA for a team of robots. The first step employs PSO to search for the best combination of tasks for robots. This first step determines which robot performs which task, but not the order. The 2nd step in [75] uses a greedy algorithm [76], [77] to determine the order of execution for each robot.

Within each timestep of PSO in [75], the velocity of the i^{th} particle v_i^t is adjusted based on personal best position p_i and global best position p_{gd}:

$$v_i^t = wv_i^{t-1} + c_1\left(p_i - x_i^{t-1}\right) + c_2(p_{gd} - x_i^{t-1}) \tag{2.20}$$

$$x_i^t = x_i^{t-1} + v_i^{t-1}\Delta t \tag{2.21}$$

where c_1 and c_2 are weighting factors between the personal best p_i and the global best p_{gd}, w is a weighting factor to adjust the speed of particle update. x_i^t is the particle's position at time t, and Δt is the time interval. Within (2.20), the particle's velocity has 3 components: a component from its own inertia/update rate, wv_i^{t-1}, a component from personal best, $c_1\left(p_i - x_i^{t-1}\right)$ and a component from global best $c_2(p_{gd} - x_i^{t-1})$. At each iteration the particle reaches a new position x_i^t whose quality is evaluated by some objective function. If the new position is evaluated as better (as evaluated by objective function) than either the personal best p_i or global best p_{gd}, that new position x_i^t becomes the new value for p_i or p_{gd}.

The PSO was also discussed in [69], which uses a similar approach to [70] to solve the cooperative MRTA, which seeks to evenly distribute workload and minimize cost. Within [69], the PSO was extended for a multi-objective approach, which also considers a global pareto front. A pareto front is the set of pareto efficient solutions, which are solutions that cannot make improvements without some sacrifice in the rest of the system. With the MTSP, a pareto solution means that a salesman's travel distance cannot be reduced without increasing another salesman's travel distance. The extended PSO in [69] proposes a probability-based leader

selection strategy for faster convergence. This strategy selects a particle as a leader to be used instead of p_{gd} in (2.20). This leader is selected from the pareto front set based on the relative value of the fitness function. The revised particle velocity in [69] is denoted below in (2.22) with p_{leader} corresponding to the position of the particle selected as leader.

$$v_i^t = wv_i^{t-1} + c_1\left(p_i - x_i^{t-1}\right) + c_2(p_{leader} - x_i^{t-1}) \tag{2.22}$$

Genetic Algorithm (GA)

The GA iteratively generates populations whose members are possible solutions to a given problem. Mutation, crossover, and selection operations are applied at each iteration to generate the next population [78]. Mutation entails changing a member of the population. Crossover entails selecting two population members and having them exchange information to generate two new members (child members) that share traits from both previous members (parent members). Selection entails choosing the fittest members of the population for use in the next iteration while discarding the less fit members. Fitness is typically evaluated using the objective function, which may be a variation of (2.18).

A modified GA was used to solve the MTSP in [70] and MRTA in [79]. The implementation in [70] is executed as follows:

1) Encoding: Possible allocations of routes to salesmen are encoded as a string whose length is equal to $p + q$, where p and q are the number of cities (nodes) and the number of salesmen, respectively. This string is partitioned into q subtours so that it represents the routes of all salesmen.

2) Evaluation: The fitness of each string is evaluated using the objective function.

3) Crossover: The fitness of each string is used to determine if they crossover or not. The crossover operation is the order crossover, in which a randomly chosen “crossover point” splits the parent substrings into left and right substrings. These substrings are re-combined to generate new strings.

4) Mutation: Some of the substrings in the population are randomly chosen and mutated. Mutation operations in [70] entail selecting two points within a string. The mutation reverses the substring between the two points or swaps the two points.

5) Decoding: A new generation has formed, and steps 1 to 4 are repeated until stopping criteria (maximum number of iterations or desired fitness) are reached. The fittest string in the final population is selected.

Within [80], a similar approach was used to solve an MRTA problem but with consideration of task priority. This consideration is reflected in the fitness function in (2.23):

$$F = \sum_{j=1}^{T} a_{i,j} c_{i,j}\, p_j \tag{2.23}$$

where $a_{i,j}$ is the allocation coefficient and determines whether robot i performs task j $(a_{i,j} = 1)$ or does not perform the task $(a_{i,j} = 0)$. $c_{i,j}$ is the cost of robot i performing task j and p_j is the task priority, it is an integer value, smaller p_j means higher priority. A solution is more fit if it has a lower value of F. However, the GA in [80] can generate infeasible solutions, hence the need for a solution repair procedure. A solution is considered unfeasible if tasks are repeated. The repair procedure identifies repeated tasks and replaces them with unallocated tasks. While [80] does provide a novel approach to assessing task priority, it is limited to one-to-one allocations; the number of robots must equal the number of tasks.

A one-to-one comparison of GA and PSO assessed that a modified GA called "Improved Partheno Genetic Algorithm," or IPGA is superior [81]. At a given iteration of GA with the contemporary population, the IPGA creates a temporary population consisting of members not selected from the contemporary population. Then, each member of the temporary population is modified with mutations specifically formulated for the IPGA [81]. The overall IPGA outperformed both GA and PSO and highlighted a possible flaw with both GA and PSO: It is possible that a "super individual" [81] can emerge at earlier iterations, which can induce a local optima trap. Managing mutations and crossover operations within IPGA enables it to avoid these shortcomings. However, the proposed mutation management in [81] can be complex and computationally demanding, as it entails generating a temporary population. It may be possible to avoid local optima trap and premature convergence by using mutations and crossover operations that result in bigger changes to population members.

Ant Colony Optimization (ACO)

The ACO is another well-established metaheuristic algorithm. It is inspired by the foraging behaviour of ants. It represents the problem as a graph and simulation ants traversing the graph to find optimal solutions [82]. The ants visit nodes based on representations of pheromone levels and desirability (objective function value). Pheromones on the graph are updated after each iteration, with better solutions depositing more pheromones. The ACO was proposed in [71] to solve the MRTA problem. Within [71], after initializing parameters and positions, the algorithm iteratively selects task points (2.24). It updates pheromones on each path selected in (2.25) and (2.26). The initial parameters were set to an initial pheromone of $\tau_{ij} = const$ and pheromone increment of $\Delta\tau_{ij} = 0$.

If a robot r_k is capable of performing a task T_i, as determined by its capability vector [71], then its probability of being selected is denoted by $p_{ij}^k(t)$ in (2.24). If it is not capable, then its probability is 0. $allowed_k$ is the set of tasks T_i that satisfy robot capabilities. α and β are weighting coefficients. d_{ij} is distance between task points.

$$p_{ij}^k(t) = \begin{cases} \dfrac{\left(\tau_{ij}(t)\right)^{\alpha}\left(d_{ij}(t)\right)^{-\beta}}{\sum_{s\in(allowed_k)}\left(\tau_{is}(t)\right)^{\alpha}\left(d_{is}(t)\right)^{-\beta}}, & if\ j \in (allowed_k) \\ 0, & if\ j \notin (allowed_k) \end{cases} \tag{2.24}$$

The pheromones are updated in (2.25) and (2.26), where m is the number of robots, ρ is pheromone volatility coefficient which ranges continuously between 0 and 1. $\Delta\tau_{ij}(t)$ is

pheromone increment on path from node i to node j on the graph. $\tau_{ij}^{k}(t)$ is the residual pheromone value of ant k on path ij.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \tag{2.25}$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t) \tag{2.26}$$

This task selection and pheromone update are iterated until the end conditions (maximum iteration count or sufficient objective function value) are satisfied.

Comparison of Metaheuristic Algorithms

The GA, PSO and ACO can find optimal solutions for the MRTA and MTSP problems. However, they have different characteristics regarding centralization, computational demand, global search capability and robustness.

Centralization is the degree to which an algorithm's iterated solutions are modified and selected by global rulesets versus the characteristics of individual solutions. PSO is the most centralized, followed by GA and then ACO. Within PSO, each possible solution is represented by a particle, which all follow the same rules for their position and velocity; the particles act as one unit. Within GA, while the rules for mutation and crossover are shared, the GA generates multiple new solutions in each population, resulting in members in a population having a greater impact on the overall solution. The ACO is the most decentralized as it simulates a population of ants searching independently throughout a graph.

This centralization is why PSO is the least demanding but also the least robust and with the worst global search capability [74]. The GA's population generation, along with mutation and crossover, gives it the best global search ability, enabling new possible solutions to be generated and combined with existing solutions [78]. This population also increases the computational demand of GA. The ACO's highly decentralized nature renders it extremely robust but contributes to its greater computational demand [82]. Based on the literature, a qualitative ranking of these algorithms is proposed in Table 2.2. The algorithms are ranked from 1 to 3 in each criterion (computational demand, global search, robustness). A ranking of 1 corresponds to the best algorithm in that criteria, with 2nd best corresponding to a ranking of 2 and the worst corresponding to a rating of 3.

Table 2.2 – Metaheuristic Optimization Algorithm Comparison

Criteria	Description	Rankings
Computational demand	An algorithm is superior if it is faster, executing in less time while using fewer computational resources.	1 – PSO 2 – GA 3 – ACO
Global Search	An algorithm is superior if it can better explore a wider range of possible solutions.	1 – GA 2 – ACO 3 – PSO
Robustness	An algorithm is superior if it can better adapt to different conditions and avoid being trapped in local optima.	1 – ACO 2 – GA 3 – PSO

The global search capability of the genetic algorithm makes it an ideal candidate for combination with a category 2 deterministic algorithm. These types of deterministic algorithms are suboptimal but have significantly less computational demand than metaheuristic algorithms. It is proposed that incorporating a solution from a deterministic algorithm into the initial population of a genetic algorithm would significantly reduce the computation time required to generate a highly optimal solution. Furthermore, if given appropriate mutation and crossover operations of the genetic algorithm, the global search capability would prevent it from converging onto the solution from the deterministic algorithm. Instead, it would use the deterministic algorithm as a starting point and continue searching for an optimal solution, avoiding a “super individual” [81] problem.

The time saved in calculating an optimal solution can then be applied to enforcing optimization constraints that model the physical limitations of robots. It is proposed that this combination of algorithms would yield a solution that is at least as optimal as the SOTA while also considering factors that the SOTA typically does not consider. These proposals regarding the combination of algorithms, mitigation of the super individual problem and reduction in computational demand will be explored in Chapter 3.

Market Based Algorithms

Market auction algorithm solutions for the MRTA were proposed in [83] and [84]. Within an MRTA context, these algorithms are economic-based solutions that treat tasks as commodities being traded and robots as bidders competing for tasks. The steps within a typical market algorithm are as follows:

1) Initialization: Represent the problem in terms of bidders and commodities.
2) Bidding: Each bidder evaluates the commodity's net gain based on the objective function.
3) Auction: Allocate the commodity to the best bid, evaluated as either best for the overall system or the highest bid available for that commodity, based on the objective function.
4) Price update: Update the price of commodities based on bids received.
5) Iteration: Repeat steps 2 to 4 until end conditions are reached.

The distributed (decentralized) market algorithm in [83] addresses cooperative task allocation with the goal of maximizing total revenue for a heterogenous multi-robot system. Each

possible task allocation has an income e_{ij} representing the gain by robot i performing task j (2.27):

$$e_{ij} = \frac{p_{ij}(\sum_{j \in T} c_i x_{ij})}{d_{ij}} \tag{2.27}$$

where p_{ij} is the degree of matching between robot i and task j, c_i is the capacity factor, x_{ij} is the assignment variable, $x_{ij} = 1$ if task j assigned to robot i and $x_{ij} = 0$ if that assignment isn't made. d_{ij} is the distance between robot i and task j. T is the set of all tasks. The goal is to maximize the objective function. This objective function is the same as (2.18), except $c_{i,j}$ in (2.18) is replaced by e_{ij} from (2.27).

The centralized market algorithm in [84] behaves similarly. However, it approaches bidding from the perspective of a centralized auctioneer rather than each robot evaluating its own income as in (2.27). This is facilitated via strategic pricing of commodities such that the inventory is cleared, subject to bidders' budgets. Strategic pricing involves increasing the price until a market equilibrium is established. This price increase can be applied to an individual or group of tasks. The central auctioneer raises prices on tasks considered preferable for multiple robots until it is no longer profitable for some of them. This encourages the bidders to consider other commodities, as each bidder still has the goal of maximizing their income.

In comparison with metaheuristic algorithms, market based algorithms can offer superior matching capabilities at the cost of greater computational demand/complexity [85]. Each type of robot can have different suitability levels for each type of task, with some task types being impossible for some types of robots. With market algorithms, a more capable/suitable robot would present a better bid for a given task. Within [85], a market algorithm was shown to yield a more optimal allocation in a capability-matching scenario. In contrast, metaheuristic algorithms converged faster in that same scenario.

Other Algorithms

Solutions not classified in the previously mentioned categories were also explored. They include fuzzy logic, neural networks, and probabilistic solutions. A proposed probabilistic solution that considers agent-task suitability was discussed in [86], which uses a specialty fitting confidence level $\hat{\varphi}_{R_i}$ to match agents R_i and task T_k. The speciality fitting confidence level is a product of the specialization vector $\boldsymbol{S}_i$ and probability transition vector $\widehat{\boldsymbol{P}}_T$ (2.28):

$$\begin{gathered} \hat{\varphi}_{R_i} = \boldsymbol{S}_i \widehat{\boldsymbol{P}}_T \\ \boldsymbol{S}_i = [s_1 \quad \dots \quad s_k \quad .. \quad s_T] \\ \widehat{\boldsymbol{P}}_T = \left[\sum_{k=1}^{T} P(C_1|\boldsymbol{X}_k) \quad \sum_{k=1}^{T} P(C_2|\boldsymbol{X}_k) \quad \dots \quad \sum_{k=1}^{T} P(C_T|\boldsymbol{X}_k) \right]^T \end{gathered} \tag{2.28}$$

where each element in $\widehat{\boldsymbol{P}}_T$ represents the probability of observed features for each class of task C_l, estimated by a Bayesian rule [87] and formulated as uncertainty measurements from target object recognition. $\boldsymbol{X}_k$ is a group of features associated with a particular task class. T is the number of tasks associated with class C_l. The speciality confidence levels are then used to

form the suitability matching scheme containing the task allocation probability concerning the detected task for the available agents.

For task detection, [88] proposes using convolutional neural networks (CNN) to classify detected target objects to support MRTA. Once an object is classified, it can be matched to several tasks involving that object, with those tasks already matched to a team of robots. Localization of that detected object can be facilitated through the solutions discussed in Section 2.1 to estimate the position of that task. With the type of task and its location known, the cost to perform that task and the most suitable agents can be determined.

The proposed agent-task suitability matching within [86] enables the selection of the most suitable agent for a given task while considering its capabilities. Consideration for capabilities is typically not an extensive aspect of contemporary solutions for MRTA. This suitability matching can be combined with a metaheuristic optimization algorithm to ensure that tasks are performed by the most capable robots in an optimized sequence.

A variant of the MTSP is the coloured travelling salesmen problem, or CTSP [89]. It is related to agent-task suitability; within the CTSP are “public” and “private” cities. Any salesman can visit any public city, which is analogous to a task that any robot can complete in a multi-robot team. Private cities can only be visited by one specific salesman, and each salesman has their own private cities. This is akin to tasks that only one robot in a multi-robot team can perform. For example, if one robot is a UAV while the other robots are UGVs, only that one UAV can perform an aerial observation task. While the CTSP does have direct comparisons to agent-task suitability, it is limited in that there are no “semi-private” cities that can be visited by some but not all salesmen. In MRTA terms, this “semi-private” city would be analogous to a task that some robots can complete, but not all of them. Suppose a multi-robot team has 3 UAVs and 5 UGVs. In that case, the 3 UAVs can execute aerial observation tasks while 5 UGVs cannot.

2.4 – Summary

This chapter has reviewed the state-of-the-art of the MRTA problem and its supporting topics. The solutions presented in this literature review provide many options for creating a complete task coordination framework. At all levels of execution, the robots’ actions are governed by their current states, the environment, and their goal states. There is a requirement for sensing and localization to gather information on the states of the robots and the environment for comparison with the goal. There is a requirement for motion planning to drive the robots to goal states as dictated by the task coordination framework.

The state-of-the-art (SOTA) within sensing and localization involves using KF / EKF for sensor fusion in both self localization and target localization. The SOTA within motion planning consists of various algorithms that model distance, velocity, and risk of collision to generate safe, optimal trajectories. These algorithms are graph-based (RRT, A*), geometric (bug algorithm, RVO), and artificial potential fields. They can also be executed both online and offline. Each solution has its strengths and weaknesses with desired use cases. The challenge with motion planning is matching the strengths and weaknesses of each solution with the problem while considering environmental factors. Within a multi-robot context, the most dominant strategy consists of VO based solutions that model robots as moving obstacles.

The SOTA within MRTA is dominated by metaheuristic optimization algorithms that try to iteratively reach an optimal solution. Often, the solutions from these established algorithms are very close to the theoretical global optima of the problem set. However, they fall short in terms of modelling the MRTA system. Typically, contemporary solutions model the MRTA problem similarly or identically to the MTSP. This system model does not account for operational factors: robots' individual capabilities, robots' working capacity or inter-task dependencies. Furthermore, while metaheuristic algorithms can efficiently calculate highly optimal solutions, their computational demand cannot be ignored.

To address these gaps, this thesis proposes a task coordination framework that models operational factors while combining the strengths and weaknesses of different optimization algorithms. This approach will yield a task allocation that is:

- Optimized: Objective function is minimized or maximized to an equal or greater extent than contemporary solutions.
- Computationally efficient: Requires equal or less computation time than contemporary solutions.
- Multi-factor: Considers operational factors (agent-task suitability, inter-task dependencies and robots' work capacities) unrepresented in contemporary solutions.

Chapter 3 – Task Coordination Framework

The task coordination framework (TCF) governs the behaviour of the multi-robot team. It consists of a task allocation algorithm and the robots' individual motion planners. The task allocation algorithm determines which robots perform which tasks and in what order. Once the tasks are assigned, the motion planners employ the reciprocal velocity obstacle (RVO) algorithm for multi-robot collision-free motion planning.

The task allocation algorithm combines the greedy [77] and genetic [90] algorithms for optimization. This algorithm represents the primary and original contributions of this thesis and includes the following:

- Extension of contemporary greedy algorithm and genetic algorithm solutions from TSP/MTSP towards MRTA. These algorithms and their components were modified and extended to address MRTA problems of greater complexity than TSP/MTSP.

- Operational factors consideration. The greedy and genetic task allocation algorithms consider the characteristics of robots and tasks as optimization constraints. In particular, the agent suitability of robot/task assignment, inter-task dependencies and task actuation cost with limitations on robots' work capacity.

- Combination of greedy and genetic task allocation algorithms. The greedy algorithm solution is incorporated as a member of the initial population within the genetic algorithm. This combination yields an optimized task allocation with minimal computational demand.

Section 3.1 establishes the structure of the optimization problem. Section 3.2 describes the optimization algorithms. Section 3.3 evaluates the proposed optimization algorithms through comparison with existing benchmarks as well as scenarios and parameters described in Appendix A1, with the optimality evaluation data in Appendix A2. After the tasks are allocated, the robots' motion planners, through RVO, determine collision-free velocities to drive the robots to their goal poses in the execution of their assigned tasks. This motion planning solution is described in Appendix A3.

3.1 – Optimization Criteria

This section describes the structure of the task allocation algorithms. Section 3.1.1 describes the agent-task dynamics. Section 3.1.2 describes the cost functions of individual tasks and the overall system. Section 3.1.3 describes the overall optimization goal and the optimization constraints.

The goal of MRTA is to find an optimal distribution of tasks TA for a team of robotic agents r_i while satisfying constraints, where optimality is the degree to which the global objective function is minimized. As an extension of contemporary MRTA solutions, our proposed solution will also consider task actuation cost, inter-task dependencies, agent-task suitability,

and robots' maximum work capacity as optimization constraints. These considerations are in addition to the distance between tasks, which is considered within contemporary solutions.

3.1.1 – Agent-Task Dynamics

The agent-task dynamics are modelled in a manner inspired by [91] with a list of tasks and a list of robots. The list T of N tasks T_j is described in (3.1) and (3.2):

$$T = \{T_j | j = 1,2, \dots N\} \tag{3.1}$$

where each task T_j (3.2) is described by its task index j, task location (x_j, y_j), task actuation cost c_j, task type t_j and prerequisites $P_j = \{j | j = 1,2,3..N\}$. The actuation cost c_j represents the travel distance required to complete a task. It is a characteristic of the task, independent of any robot and is a component of the total cost $c_{i,j}$ for robot r_i to complete task T_j as described in Section 3.1.2. Task type t_j is a classification used to determine the suitability score for a robot through comparison via a lookup table with the robot's agent type. This suitability score will be further discussed in Section 3.1.3. Prerequisites P_j is a list of task indices corresponding to tasks that must be assigned before T_j itself can be assigned. Conversely, a task T_n is a dependant of T_j if T_j is a prerequisite of T_n; $j \in P_n$.

$$T_j = \{j, x_j, y_j, c_j, t_j, P_j\} \tag{3.2}$$

The list R represents the team of M robotic agents r_i as described in (3.3) and (3.4).

$$R = \{r_i | i = 1,2, \dots M\} \tag{3.3}$$

Where each robot r_i is described by its robot index i and robot position (x_i, y_i). Further details include its usage u_i, work capacity u_{i_max}, and agent type t_i. Usage is the sum of cost functions of tasks assigned to the robot. It is analogous to the fuel requirement to travel a given distance. u_{i_max} is a property of the robot and is analogous to its fuel tank capacity. As the robot executes movements and tasks, its usage u_i increases. This interaction between u_i and u_{i_max} establishes the constraint $u_i \leq u_{i\_max}$, further discussed in Section 3.1.3. Agent type t_i is a variable that establishes heterogeneity among the agents and is combined with task type t_j to determine the suitability score of a task. This heterogeneity is established through different capabilities as determined by the task's suitability scores (discussed in Section 3.1.3); some tasks can only be performed by some (but not all) robotic agents.

$$r_i = \{i, x_i, y_i, u_i, u_{i\_max}, t_i\} \tag{3.4}$$

The task allocation TA is defined in (3.5) as the set containing the sequences of tasks for each robot. The sequence of tasks for robot r_i is defined as TS_i (3.6) where k is the order in which the task is performed by robot r_i and N_i is the number of tasks assigned to robot r_i.

$$TA = \{TS_i | i = 1,2 \dots M\} \tag{3.5}$$

$$TS_i = \{T_j^{i,k} | j = 1,2,3..N \mid k = 1,2,3 \dots N_i\} \tag{3.6}$$
$$T_j^{i,k} = task\ T_j\ assigned\ to\ robot\ r_i\ to\ be\ performed\ at\ k^{th}\ order$$

An example task allocation is shown in (3.7). In this example, robot r_0 performs T_5 first, followed by T_3 and finally T_2; robot r_1 performs T_1 as its first task and T_4 as its second task. Similarly, robot r_2 performs task T_7 and then T_6.

$$TA = \begin{cases} TS_0 = \{T_5^{0,1}, T_3^{0,2}, T_2^{0,3}\} \\ TS_1 = \{T_1^{1,1}, T_4^{1,2}\} \\ TS_2 = \{T_7^{2,1}, T_6^{2,2}\} \end{cases} \tag{3.7}$$

3.1.2 – Cost Functions

The task cost function $c_{i,j}$, is the total cost for robot r_i to complete task T_j as described in (3.8). The $\Delta s_{i,j}$ component represents the distance the robot travels from its current position to the task. The c_j component represents the distance travelled by the robot in the performance of its task.

$$c_{i,j} = \Delta s_{i,j} + c_j \tag{3.8}$$

$$\Delta s_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{3.9}$$

For example, if given a patrol task, the robot travels $\Delta s_{i,j}$ from its current location to the start position of the patrol route. It then travels c_j distance from the patrol start location to the patrol end location as shown in Figure 3.1.

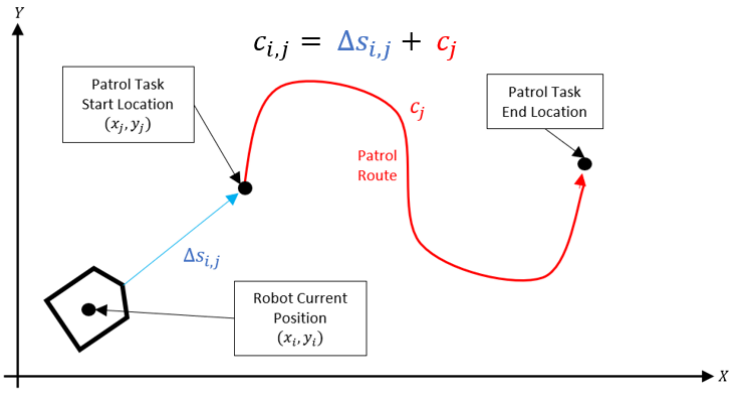


Figure 3.1 – Patrol Task Example

The usage u_i incurred by a robot r_i in the execution of all its assigned tasks is the sum of the tasks' cost functions $c_{i,j}$ for all tasks in the task sequence TS_i assigned to r_i. This usage u_i is described in (3.10), where $T_j^{i,k}$, TS_i are described in (3.6), (3.5) and N is number of tasks.

$$u_i = \sum_{j=1}^{N} c_{i,j} \tag{3.10}$$
$$c_{i,j} = \Delta s_{i,j} + c_j;\ if\ T_j^{i,k} \in TS_i$$
$$c_{i,j} = 0;\ if\ T_j^{i,k} \notin TS_i$$

The total cost C incurred by the overall system is the sum of all agents' usages u_i to complete all their assigned tasks, denoted in (3.11), where M is the number of robots.

$$C = \sum_{i=1}^{M} u_i \tag{3.11}$$

3.1.3 – Optimization Goal and Constraints

The proposed global objective function (3.12) employs a MinMax formulation, which considers both the total cost C and greatest usage, $\max(u_i)$ among the robots. It is proposed that minimizing this global objective function F would result in task allocations that are of minimal cost but also yield an even distribution of work for the multi-robot team.

$$F = C + \max(u_i) \tag{3.12}$$

The optimization goal (3.13) is to find the task allocation TA_{opt} which is admissible and and minimizes the global objective function (3.12) for a given set of robots and known tasks. Admissible is defined as compliant with the constraints described in (3.14) to (3.16).

$$TA_{opt} = \underset{F}{argmin}(C + \max(u_i)) \tag{3.13}$$

This global objective function (3.12) and optimization goal (3.13) were selected to reduce the total cost C while ensuring a relatively even distribution of work among the robots. The even distribution stems from the $\max(u_i)$ component. Incorporating $\max(u_i)$ in the global objective function to be minimized discourages task allocations that disproportionately assign more work to an individual robot. Alternatively, an objective function of only $F = C$ would result in a significantly unbalanced work distribution, as later demonstrated in Section 3.3.2, test case 4. The proposed objective function in (3.12) attempts to satisfy both requirements.

Constraints Imposed on Optimization

The optimization must comply with the following constraints:

1) Each robot's usage must not exceed its work capacity u_{i_max}. This work capacity constraint is described in (3.14).

$$u_i \leq u_{i\_max}, for\ all\ agents\ r_i \tag{3.14}$$

2) The suitability score $s_{i,j}$ of an agent-task assignment must meet or exceed a minimum suitability requirement s_{min}. This suitability score $s_{i,j}$ is a measurement of the capability of robot $r_i$to perform task T_j. This suitability constraint is described in (3.15).

$$s_{i,j} \geq s_{min}, for\ all\ agents\ r_i\ and\ task\ allocations\ T_{i,j} \tag{3.15}$$

The suitability score $s_{i,j}$ for a particular agent-task assignment is determined by a predefined lookup table. An example is provided in Table 3.1. In this particular example, the

assignment of a type α robot to a type B task has a suitability score of $s_{\alpha,B}$. If $s_{\alpha,A} > s_{min}$ but $s_{\beta,A} < s_{min}$, then robots of type α can perform tasks of type A, while robots of type β cannot.

Table 3.1 – Arbitrary Example of Agent/Task Suitabilities

	Task Type (t_j)				
Agent Type (t_i)	A	B	C	D	E
α	$s_{\alpha,A}$	$s_{\alpha,B}$	$s_{\alpha,C}$	$s_{\alpha,D}$	$s_{\alpha,E}$
β	$s_{\beta,A}$	$s_{\beta,B}$	$s_{\beta,C}$	$s_{\beta,D}$	$s_{\beta,E}$
ε	$s_{\varepsilon,A}$	$s_{\varepsilon,B}$	$s_{\varepsilon,C}$	$s_{\varepsilon,D}$	$s_{\varepsilon,E}$

3) Dependants must have their prerequisites already assigned before they can be assigned. For a given task, P_j must be a subset of already allocated tasks $T_{allocated}$. This inter-task dependency constraint is described in (3.16).

$$P_j \subseteq T_{allocated}, for\ all\ agents\ r_i\ and\ task\ allocations\ T_{i,j} \tag{3.16}$$

$$T_{allocated} = list\ of\ all\ allocated\ tasks$$

It must be noted that constraint (3.14) can limit the number of tasks completed/assigned, leaving some of them incomplete/unassigned. This occurs if usage u_i (calculated from costs $c_{i,j}$) exceed the robot's work capacities u_{i_max}. Therefore, two criteria are used to determine the optimality of a task allocation, assuming it is admissible (complies with all constraints).

1) A task allocation TA is more optimal if it results in a greater number of tasks completed, regardless of objective function value F. This criteria only applies when limited work capacity prevents certain tasks from being completed/assigned.
2) If given an equal number of tasks completed, a task allocation TA is more optimal if its global objective function (3.12) is lower.

3.2 – Task Allocation Algorithms

This section describes the algorithms involved in our proposed MRTA solution. Section 3.2.1 describes a deterministic greedy algorithm. Section 3.2.2 describes metaheuristic MRTA solutions through hill climbing and genetic algorithms. Section 3.2.3 describes the combined solution that uses the algorithms described in Sections 3.2.1 and 3.2.2. Unless otherwise stated, all algorithms share the same optimization criteria described in Section 3.1.

3.2.1 – Efficient Deterministic Optimization

As stated in Section 2.3, deterministic algorithms are typically fast but suboptimal or extremely optimal with unreasonable computational demand. We propose using a fast but suboptimal deterministic algorithm to efficiently generate a "usable" task allocation. This algorithm attempts to generate as optimal as possible of a task allocation with only one iteration, minimizing computation time. In this case, "usable" is defined as having a global objective function value much lower than a completely random task allocation, closer to an optimal value, but less optimal than metaheuristic solutions.

This suboptimal task allocation can be used immediately or combined with metaheuristic algorithms discussed in Section 3.2.2. This combination expedites the computation of a globally optimal solution. Furthermore, while this deterministic optimization is suboptimal, its optimality greatly exceeds that of metaheuristic solutions if provided the same amount of (minimal) computation time. In other words, the optimality is disproportionate to its computational demand, as demonstrated in Section 3.3.

The deterministic optimization is facilitated through the greedy task allocation ($GrdTA$) algorithm inspired by [77], which selects locally optimal tasks based on distance for the travelling salesman problem (TSP). The $GrdTA$ extends the solution from [77] to an MRTA scenario while also considering actuation cost and the constraints discussed in Section 3.1.3.

The $GrdTA$ algorithm models a multi-robot team that consecutively selects locally optimal tasks until all tasks are assigned or constraints can no longer be satisfied. The resulting task allocation from the $GrdTA$ algorithm stems from the robots' task sequences in this model. A locally optimal selection minimizes the local objective function (3.17). Where $c_{i,j}$ and u_i were defined in (3.8) and (3.10), respectively.

$$f = c_{i,j} + u_i \tag{3.17}$$

The local objective function is similar to the global objective function (3.12). Both functions have a cost component (C in F and $c_{i,j}$ in f) as well as a usage component ($\max(u_i)$ in F and u_i in f). This similarity stems from the goals of reducing cost and generating an even distribution of tasks. Repeated selections of the same robot r_i will increase its usage u_i and consequently, subsequent task assignments to r_i will result in greater values of f, which are less optimal. This discourages subsequent assignments to r_i until the usages of other robots increase, resulting in a more even distribution of work.

To facilitate these locally optimal selections, an intermediate step involves calculating agent-task matchings $T_{i,j}$, described in (3.18) where i and j are the index of robot and task. $s_{i,j}$ and $c_{i,j}$ are the agent-task suitability and task cost function, respectively.

$$T_{i,j} = \{i, j, s_{i,j}, c_{i,j}, {c_{i,j}}'\} \tag{3.18}$$

The agent-task matching proposes a task debit ${c_{i,j}}'$ to represent preferences in task allocation related to globally available information from all known tasks. This task debit is described in (3.19). Where $\Delta s_{i,j}$ and c_j have the same meaning as in (3.8).

$${c_{i,j}}' = K_p * (\Delta s_{i,j} + c_j) \tag{3.19}$$

This task debit ${c_{i,j}}'$ is used to filter tasks before making a locally optimal selection based on minimizing (3.17). Tasks with lower task debit ${c_{i,j}}'$ are preferred over tasks with greater task debit. The task preference considered is the prioritization of tasks that are prerequisites so that the $GrdTA$ algorithm can allocate their dependants. This prioritization provides more options for the $GrdTA$ algorithm, which can generate a more optimal solution. This preference is quantified in the dependency coefficient K_p in the task debit ${c_{i,j}}'$ which is calculated using (3.20):

$$K_p = 1 - \frac{N_d}{N_T} * \left(1 - \frac{|\overline{\Delta s_d}|}{|\overline{\Delta s_d}| + |\overline{\Delta s_n}|}\right);\ \ if\ N_d > 0, N_d \neq N_T \tag{3.20}$$

where:

$$K_p = 0;\ if\ \ N_d = N_T - 1$$

$$K_p = 1;\ if\ N_d = 0$$

$$0 \leq N_d < N_T, \qquad 0 \leq K_p \leq 1$$

N_d and N_T are the number of dependants of T_j, and the number of all tasks, respectively. $|\overline{\Delta s_d}|$ is the average distance between T_j and its dependants. $|\overline{\Delta s_n}|$ is the average distance between T_j and all other tasks. Assuming all other variables are held constant, and $0 \leq N_d < N_T$:

- K_p decreases as N_d increases. If T_j has more dependants, then T_j would be of lower task debit $c_{i,j}{}'$. It would be preferred as completing it would increase the number of viable allocations earlier. This effect is illustrated in Figure 3.2a, where the robot r_0 selects T_2 even though T_1 is closer. T_2 is selected because it has 3 dependants, while T_1 has no dependants.
- K_p decreases as $|\overline{\Delta s_d}|$ decreases, which means that if T_j has several dependants closer to it, T_j would have a lower task debit $c_{i,j}{}'$. It would be preferred because those dependants could then be subsequent selections after T_j. This effect is illustrated in Figure 3.2b. T_1 and T_2 are of equal distance from r_o and both have 3 dependants. However, the dependants of T_2 are closer, so it is selected over T_1.

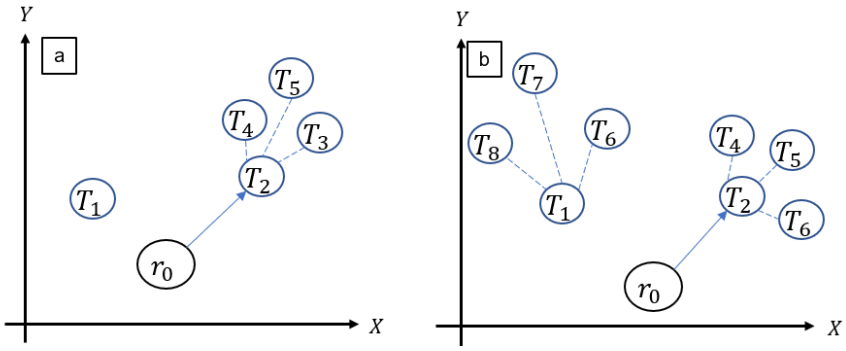


Figure 3.2 – Effect of Dependency Coefficient on Task Allocation a) Task with more Dependants is Preferred, and b) Task with Closer Dependants is Preferred

If $N_d = N_T - 1$, all other tasks are dependent on T_j so it must be performed before all other tasks, making it a priority because $K_p = 0 = c_{i,j}{}'$. If $N_d = 0$, this implies that T_j has no dependants, so it should not receive any additional prioritization, and therefore, its task debit should not be reduced. N_d must always be less than N_T. A value of $N_d = N_T$ would mean all tasks are dependent on T_j, including T_j itself. This means that T_j must be allocated before it can be allocated, but all tasks have that requirement as well. This means that no tasks can be allocated.

The $GrdTA$ takes in a list of tasks (3.1) and a list of robot agents (3.3). It models a multi-robot team that consecutively selects locally optimal tasks, executing local choice iterations until all tasks are assigned or constraints can no longer be satisfied. At each local choice iteration, admissible agent-task matchings $T_{i,j}$ are calculated which generates a list T^{min} consisting of each robot's preferred agent-task matching. This list contains one agent-task matching $T_{i,j}$ from each robot, such that the task debit ${c_{i,j}}'$ is minimized for its respective robot. From these preferred agent-task matchings T^{min}, the locally optimal agent-task matching $T_{i,j}^{opt}$ is the one that minimizes the local objective function (3.17). $T_{i,j}^{opt}$ is selected for task allocation and its execution is simulated. This simulation entails adding the task cost $c_{i,j}$ to the selected robot's usage u_i and setting the robot's position as the task's position. These local choice iterations occur until all tasks are assigned or until the constraints in (3.14) to (3.16) cannot be satisfied. The task sequences from this simulation are the output as the task allocation from the $GrdTA.$

Algorithm 1 – Greedy Task Allocation Algorithm *($GrdTA$)*

1) A team of robots $R = \{r_i | i = 1,2, \dots M\}$ receives a list of tasks $T = \{T_j | j = 0,1, \dots N\}$. Initialize an empty list of allocated tasks $T_{allocated}$ as well as allocation $TA = \{TS_i | i = 0,1,2,3 \dots M\}$ containing empty task sequences TS_i. As described in (3.5) to (3.7), TS_i contains the tasks T_j assigned to robot r_i and orders the sequence they are performed by robot r_i.

2) Calculate the set A, which contains lists A_i of agent-task matchings $T_{i,j}$ for all agents r_i for all unallocated tasks.

$$A = \{A_i | i = 0,1,2 \dots M\} \tag{3.21}$$

$$A_i = \{T_{i,j} | j = 0,1,2,3 \dots N | T_j \notin T_{allocated}\}$$

3) Remove agent-task matchings $T_{i,j}$ that violate constraints described in (3.14) to (3.16).

4) For each agent, select an admissible $T_{i,j}$ such that the task debit ${c_{i,j}}'$ (3.19) is minimized. The list containing each agent's admissible minimal cost matchings is T^{min}.

$$T^{min} = \left\{ T_{i,j} | i = 0,1,2 \dots M | j = \underset{{c_{i,j}}'}{argmin} T_{i,j} \right\}, T_{i,j} \in A_i \tag{3.22}$$

5) Select the locally optimal agent-task matching $T_{i,j}^{opt}$ from the list T^{min}. This optimal selection (3.23) is the one that minimizes the local objective function, f (3.17).

$$T_{i,j}^{opt} = \underset{(f)}{argmin} T_{i,j}, \qquad T_{i,j} \in T^{min} \tag{3.23}$$

6) Assign T_j to r_i based on the selection in step 5. Within TA append T_j to TS_i as $T_j^{i,k}$

7) Update the list of tasks already assigned. Remove T_j from T and add T_j to $T_{allocated}$

8) Simulate movement and task execution of robot r_i. Update the usage u_i of r_i by adding the cost $c_{i,j}$ of moving to and performing task T_j. Update the position of r_i by setting its position to the location of the assigned task T_j.

9) Repeat steps 2 to 8 until all tasks are assigned or constraints can no longer be satisfied. Once all tasks are assigned or constraints can no longer be satisfied, the $GrdTA$ terminates, and the task allocation TA is assigned to the robots.

Overall, the locally optimal selections of the $GrdTA$ yield a usable task allocation with minimal computational demand due to the single iteration and deterministic nature of the algorithm. Furthermore, metaheuristic algorithms alone require much more computation time to meet or exceed the optimality of the $GrdTA$. However, metaheuristic solutions will find more optimal solutions if given sufficient computation time.

The following elements of the $GrdTA$ represent original contributions:

- Extension of computationally efficient greedy algorithm solutions from the TSP for MRTA scenarios.
- The use of the task debit $c_{i,j}{}'$ with dependency coefficient K_p to maximize the optimality of the single iteration.
- The consideration of prerequisites, suitability, actuation cost and usage. Contemporary MRTA solutions do not typically consider these factors.

3.2.2 – Metaheuristic Optimization

Metaheuristic algorithms iteratively generate progressively more optimal solutions. These types of algorithms can achieve greater optimality than deterministic solutions while approaching global optimality and having reasonable computational demand. The considered algorithms for metaheuristic optimization are the hill climbing [73] and genetic [90] algorithms. The hill climbing algorithm was selected due to its simplicity and low computational demand. The genetic algorithm was selected for its superior global search capability (discussed in Section 2.3). It is proposed that combining these metaheuristic algorithms with the $GrdTA$ would significantly expedite convergence to an optimal solution. However, the iterative nature of metaheuristic algorithms imposes variability and greater computational demand than suboptimal deterministic algorithms. In the case of our considered algorithms, this variability stems from random decisions involved in the population generation, mutation operation and crossover operation.

The mutation operation considered for both hill climbing and genetic algorithms is a task re-assignment in which a task from one robot is given to another based on the cross-route mutation discussed in [93]. The hill climbing task allocation ($HCTA$) algorithm initializes a random task allocation TA and applies a mutation operation to generate a new task allocation TA. If the new task allocation TA is of lower global objective function value, it is retained for the next iteration. If not, it is discarded, and the previous TA is retained.

The metaheuristic algorithms iterate until reaching the desired number of iterations n_{iter}. A systemic approach to determining n_{iter} was considered out of scope as, in reality, this value would be determined by time constraints, available computational resources and the

complexity of the scenario. For this thesis, n_{iter} was set to balance improvements in the task allocation while avoiding excessive computation time.

Algorithm 2 – Hill Climbing Task Allocation Algorithm ($HCTA$)

1) Generate an initial random task allocation TA based on the multi-robot team R and list T of tasks.

2) Apply the mutation operation to generate new task allocation TA.

3) Evaluate global objective function F (3.12) of new task allocation. If F in new task allocation is less than F in previous task allocation, then the new task allocation is retained.

4) Repeat steps 2 and 3 for the desired number of iterations n_{iter}.

The genetic task allocation ($GenTA$) algorithm initializes a population of several task allocations TA. It applies mutation and crossover operations while selecting the most optimal task allocations to generate the next population at each iteration. The crossover operation entails two task allocations in a population exchanging information, creating two new task allocations TA with components from its parents. The most optimal solution at each iteration is chosen through tournament selection [94] in which the fittest (lowest global objective function value) solutions are retained for the next iteration.

Mutations and crossovers are also not guaranteed within the $GenTA$; there is a probability of mutation and crossover, p_m and p_c. A higher probability of mutation and crossover would result in greater diversity. However, the increased variability can disrupt the most optimal solutions. A greater population size n_p would result in greater diversity but greater computational demand. The proposed parameters for this thesis are in Table A.1 in appendix A1. High probabilities of mutation and crossover ($p_m = p_c = 0.7$) were combined with a modest population size of $n_p = 10$ to provide sufficient diversity with reasonable computational demand. This diversity enables the genetic algorithm to effectively execute global search without being stuck in local optima, as discussed in Section 2.3.

Algorithm 3 – Genetic Task Allocation Algorithm ($GenT$)

1) Define mutation and crossover probabilities $0 \leq p_m \leq 1$ and $0 \leq p_c \leq 1$ as well as population size $n_p > 1$ and number of iterations n_{iter}.

2) Generate an initial population P (3.24) consisting of n_p randomly generated allocations TA_n based on the multi-robot team R and list T of tasks.

$$P = \{TA_n | n = 1,2, \dots n_p\} \tag{3.24}$$

3) Evaluate global objective function F (3.12) of each task allocation TA in the population.

4) Remove the worst 40% of task allocations within the population through tournament selection. These are the allocations with the greatest objective function values. Replace these with randomly generated admissible task allocations.

5) Identify population members to mutate based on p_m. A random number $0 \leq N_m^n \leq 1$ is generated for each task allocation TA_n. If $N_m^n \leq p_m$, then the mutation operation is applied to TA_n. If the mutated task allocation is admissible and has a smaller objective function F than the original task allocation, then the mutation is retained, or the mutation is discarded.

6) Identify population members for crossover operation. A random number $0 \leq N_c^n \leq 1$ is generated for each task allocation TA_n. If $N_c^n \leq p_c$, then the crossover operation is applied to TA_n. If TA_n is selected for crossover operation, then a second task allocation TA_k is randomly selected from the population members to crossover with TA_n to generate two new task allocations, ${TA_n}'$ and ${TA_k}'$. The two most fit admissable task allocations between TA_n, TA_k, ${TA_n}'$ and ${TA_k}'$ are retained.

7) Repeat steps 2 to 6 for n_{iter} iterations.

8) The admissible task allocation with the lowest global objective function value F at the final iteration is selected as the most optimal solution.

Furthermore, all task allocations within hill climbing and genetic algorithms are admissible; randomly generated task allocations are generated to comply with constraints (3.14) to (3.16). The mutation and crossover operations are designed to comply with these constraints and avoid generating inadmissible task allocations.

Mutation Operation

The mutation operation is a task re-assignment inspired by the cross-route mutation discussed in [93]. It randomly selects a task and then assigns it to another robot in a random order while satisfying suitability and prerequisite requirements. This mutation operation enables the genetic algorithms to explore the search space and escape local optima because it involves two robots' task sequences. The involvement of multiple task sequences increases the diversity of mutated solutions. At the same time, because only one task is re-assigned, it is unlikely to disrupt optimal solutions. While cross-route mutation operations have been used to generate solutions for MTSP, they typically do not consider optimization constraints imposed by robots' and tasks' physical characteristics. The task re-assignment mutation operation has been modified to enforce the optimization constraints as described in steps 3 to 5 below.

Algorithm 4 – Task Re-assignment Mutation Operation

1) Within a given task allocation TA, randomly select a robot's r_i task sequence TS_i
2) From TS_i, randomly select a task $T_j^{i,k}$.
3) Within that same task allocation TA, randomly select another robot's r_h task sequence TS_h, such that the suitability between r_h and $T_j^{i,k}$ satisfies minimum suitability constraints $s_{h,j} \geq s_{min}$.
4) Let k_1 be equal to the k value corresponding to the task that is the latest occurrence of all of $T_j^{i,k}$'s prerequisites. Let k_2 be equal to the k value corresponding to the task that is the earliest occurrence of all of $T_j^{i,k}$'s dependants.

5) Insert $T_j^{i,k}$ into TS_h at a random order between k_1 and k_2. $T_j^{i,k}$ is now $T_j^{h,q}$, $k_1 \leq q \leq k_2$. This ordering ensures that $T_j^{i,k}$ is executed after its prerequisites and before its dependants.

Crossover Operation

The crossover operation is the Constrained Multi-Agent Partially Mapped Crossover (CM-PMX) operation. It is based on the partially mapped crossover (PMX) [95] operation typically used in the TSP. Crossover operations select two solutions within a population and exchange information between them. The information exchanged is the order in which a given task is performed (or, in the case of MTSP, the order in which a city is visited). The PMX operation has been heavily modified into CM-PMX operation to generate MRTA solutions:

- Generalization for a multiple-agent scenario. The original PMX was formulated for the TSP, only considering one tour. The CM-PMX is purpose-built for MTSP/MRTA scenarios and considers a set of multiple tours/task sequences.
- Enforcement of prerequisite constraints. The original PMX freely exchanges the order in which cities are visited / tasks are performed. Due to the possibility of multiple changes in task execution order, a systemic approach was developed to ensure prerequisite constraints were enforced. This approach entails comparing the order of both prerequisites and dependants of the swapped tasks. It is described in steps 3 to 6 of the CM-PMX algorithm.

The CM-PMX operation compares task sequences between two task allocations and swaps the order of identical Order-Independent Agent-Task Assignments (OIATA), as shown in Figure 3.3. An agent-task matching is considered identical if the same robot r_i is assigned the same task T_j, regardless of its execution order k. For example, in Figure 3.3, $T_1^{q,g,1}$ from TA_q and $T_1^{p,g,2}$ from TA_p are considered identical OIATA. They both correspond to task T_1 assigned to robot r_g, even though it is performed first $(k = 1)$ within TA_q and second $(k = 2)$ within TA_p. Furthermore, the CM-PMX operation does not need to consider suitability requirements because no tasks are being re-assigned from one robot to another.

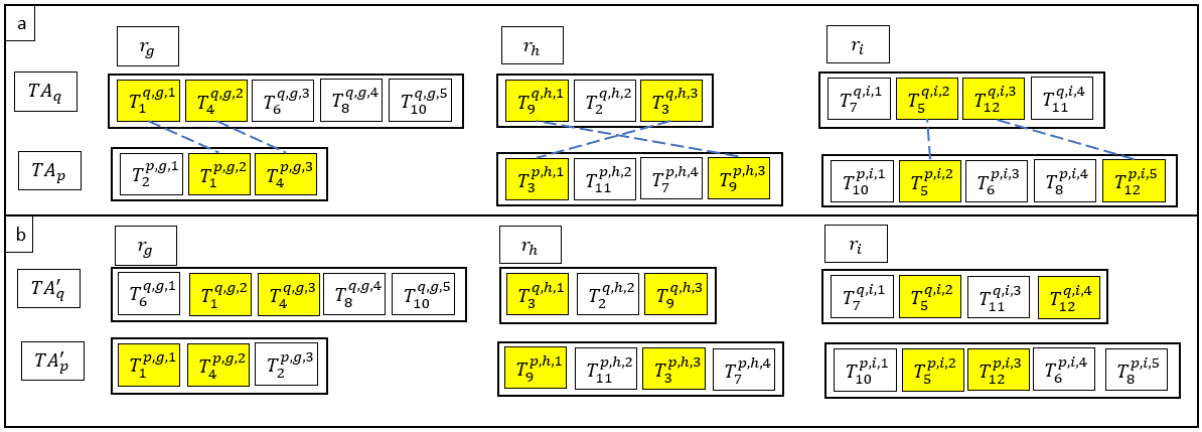


Figure 3.3 – Crossover Operation Between Two Task Allocations: (a) Before Crossover and (b) After Crossover

Algorithm 5 – Constrained Multi-Agent Partially Mapped (CM-PMX) Crossover Operation

1) Select two task allocations, TA_p and TA_q, within a population using step 6 of Algorithm 3.
2) Compare each robot's task sequences within TA_p and TA_q. Let $TS_{p,i}$ be a task sequence of robot r_i within TA_p and $TS_{q,i}$ be a task sequence of robot r_i within TA_q.
3) Identify identical OIATA within the two task allocations. Within each task sequence $TS_{q,i}$ and $TS_{p,i}$, identify matching tasks. Let $T_{j_q}^{q,i,k_q}$ and $T_{j_p}^{p,i,k_p}$ be $T_j^{i,k}$ from TA_q and from TA_p, respectively. $T_{j_q}^{q,i,k_q}$ is the task T_j performed at the ${k_q}^{th}$ order by robot r_i within the task allocation TA_q. Because $j_q = j_p = j$, both $T_{j_q}^{q,i,k_q}$ and $T_{j_p}^{p,i,k_p}$ are the same task T_j and are OIATA. Therefore $T_{j_q}^{q,i,k_q} = T_j^{q,i,k_q}$ and $T_{j_p}^{p,i,k_p} = T_j^{p,i,k_p}$. This step is shown in Figure 3.3a.
4) Identify prerequisites and dependants of T_j within TA_p. Since T_j^{q,i,k_q} and T_j^{p,i,k_p} are the same task T_j, let k_1 be equal to the k value corresponding to the task that is the latest occurrence of all of T_j's prerequisites within TA_p. Let k_2 be equal to the k value corresponding to the task that is the earliest occurrence of all of T_j's dependants within TA_p.
5) Verify compliance with the prerequisite constraint. If both criteria $k_1 \leq k_q \leq k_2$ and $k_1 \leq k_p \leq k_2$ are true, the resulting swap is compliant. This ordering ensures that post-crossover, both T_j^{q,i,k_q} and T_j^{p,i,k_p} would be executed after their prerequisites and before their dependants while complying with work capacity constraints.
6) If a swap is admissible, then swap task sequence order between T_j^{q,i,k_q} and T_j^{p,i,k_p}. T_j^{p,i,k_p} becomes T_j^{p,i,k_q} within TA_p. T_j^{q,i,k_q} becomes T_j^{q,i,k_p} within TA_q. The swapped tasks are highlighted in Figure 3.3b.
7) Repeat steps 4 to 6 for all identical OIATA (as determined from step 3) within TA_p and TA_q.

Combination of Greedy Task Allocation with Hill Climbing or Genetic Algorithms

The speed of the greedy task allocation ($GrdTA$) algorithm can be combined with the iterative improvements from the hill climbing ($HCTA$) algorithm and genetic ($GenTA$) algorithms. These combinations entail using the $GrdTA$ algorithm to generate an intermediate task allocation TA_0 and incorporating it into the initial conditions of either algorithm. By using the $GrdTA$ solution instead of random initial conditions as described in the original $HCTA$ and $GenTA$, the computation time required to achieve an optimal solution would be greatly reduced.

Algorithm 6 – Greedy + Hill Climbing Task Allocation Algorithm ($GrdHCTA$)

The proposed Greedy + Hill Climbing Task Allocation Algorithm ($GrdHCTA$) uses the $GrdTA$ to generate an initial task allocation TA_0 that is used in the first iteration of the $HCTA$. This process is shown in Figure 3.4.

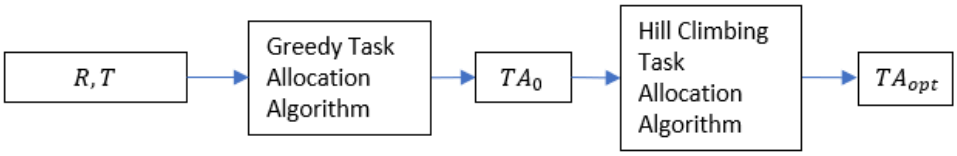


Figure 3.4 – Greedy + Hill Climbing Task Allocation Algorithm Pipeline

1) Execute $GrdTA$ (Algorithm 1) to generate intermediate task allocation TA_0.

2) Execute $HCTA$ (Algorithm 2) using TA_0 instead of a randomly generated solution for the initial task allocation.

Algorithm 7 – Greedy + Genetic Task Allocation Algorithm ($GrdGenTA$)

The proposed Greedy + Genetic Task Allocation Algorithm ($GrdGenTA$) uses the $GrdTA$ to generate an initial task allocation TA_0 which is incorporated into the initial population of the $GenTA$. This incorporation is shown in Figure 3.5.

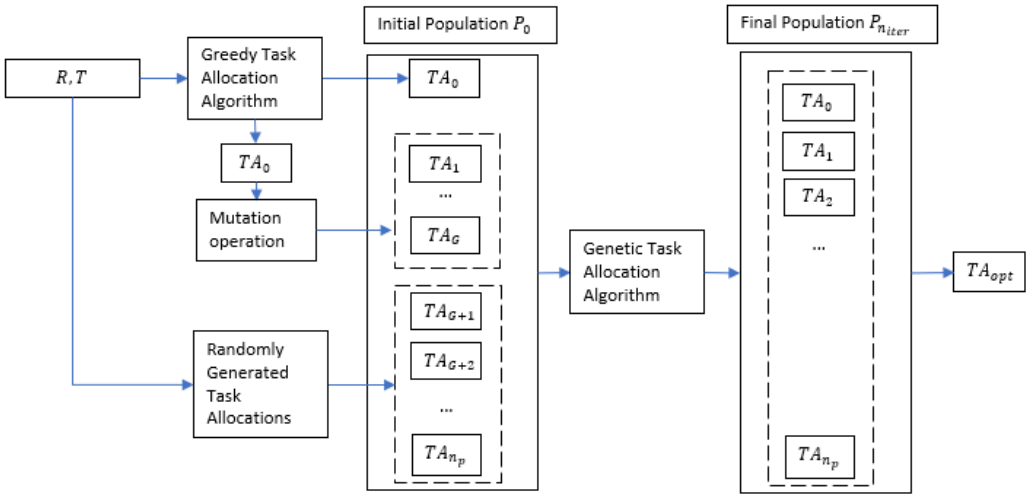


Figure 3.5 – Greedy + Genetic Task Allocation Algorithm Pipeline

1) Execute $GrdTA$ (Algorithm 1) to generate intermediate task allocation TA_0.

2) Generate an initial population P_0 consisting of n_p possible task allocations TA_n. Within this initial population, TA_0 is generated from $GrdTA$, task allocations TA_1 to TA_G are versions of TA_0 that have been modified with mutation operation. Task allocations TA_{G+1} to TA_{n_p} are randomly generated admissible task allocations.

3) Execute $GenTA$ (Algorithm 3), starting with P_0 instead of a completely randomly generated initial population.

Analysis

It must be noted that combining these algorithms in the matter proposed in algorithms 6 and 7 greatly elevates the risk of a “super individual” problem. This “super individual” is a solution that dominates the population and results in premature convergence / local optima trap [96]. This is because the $GrdTA$ is essentially creating a “super individual” which is TA_0. This risk is compounded due to the tournament selection criteria in the $GrdGenTA$, and $GenTA$ which typically results in less diversity than a more computationally demanding roulette wheel selection [97] approach. It is further compounded in the $GrdHCTA$ which does not have the population-based approach of the solutions involving genetic algorithm.

This risk is accepted as a trade-off in exchange for reducing computational demand, which is required to optimally allocate tasks in time-constrained scenarios. The following aspects of the task allocation algorithms are proposed to mitigate the risk of local optima from a super individual:

- The task re-assignment crossover operation, through exchanging tasks between robots and the CM-PMX operation, through the exchange of task execution order, can generate sufficient diversity to explore the search space and avoid local optima traps.
- The optimization constraints already reduce the diversity of possible solutions such that further reductions in diversity from tournament selection and super individual problems do not adversely impact the optimization.

The following aspects of the considered metaheuristic algorithms are original proposals:

- Combination of an efficient deterministic algorithm with a metaheuristic algorithm that iteratively produces more optimal solutions. This combination significantly reduces the computation time required to generate an optimized solution.
- Enforcement of optimization constraints in crossover and mutation operations. In particular, there was a requirement for a systematic approach to maintaining compliance with prerequisites and dependencies in PMX crossover due to the possibility of multiple tasks changing their execution order.

3.3 – Evaluation of Task Allocation Algorithms

This section assesses the performance of the algorithms presented in Section 3.2. Section 3.3.1 directly compares against an existing benchmark, measuring our algorithms' optimality and computational demand. Section 3.3.2 assesses our algorithms’ ability to respond to the optimization constraints in (3.14) to (3.16). All algorithms considered for evaluation used the parameters in Table A.1 within Appendix A1, which, as discussed in Section 3.2.2, were selected to balance diversity (explore a wide range of different solutions) and computational demand requirements.

3.3.1 – Evaluation of Optimality Against Established Benchmark

The algorithms proposed in Section 3.2 were evaluated against an existing benchmark [59] for the MinMax single depot MTSP. This problem involves multiple nodes that multiple salesmen have to traverse optimally and then return to their starting location. As discussed in Section 2.3, the dataset in [59] and the CPLEX software used to generate it are considered established standards within industry and optimization research.

The criteria that will be used to evaluate the proposed algorithms are the global objective function F value (3.12) and computation time in seconds. Computation time is representative of the relative computational demand of each solution. It must be noted that if these evaluations were repeated, computation times would differ depending on the computer's specifications and the implementation of the algorithms. However, the overall trends relative to each algorithm should be similar. All evaluations in this section used the Python programming language on an MSI Leopard GP65 computer on its native operating system.

In the eil51 single depot MTSP, there are 51 cities and 5 salesmen who start at city 1. They visit the 50 other cities and return to city 1. This scenario had to be replicated to facilitate a direct one-to-one comparison, which entails all robots starting at the same and returning to the same location.

- The 51 cities were converted into 50 "go-to-goal tasks" and 1 start location.
- All 5 robots start at the same location and have infinite work capacity.
- 5 "return tasks" were created, and their locations are identical to the robots' start locations. These return tasks had the 50 go-to-goal tasks as their prerequisites.
- Each return task had agent-task suitabilities of 0 for all except its corresponding robot.

These modifications ensured that the return tasks would only be allocated after all 50 go-to-goal tasks were finished, and each robot would execute its return task. The modifications are required because the proposed solution for MRTA does not inherently "hard code" return tasks. A "go-to-goal" task entails moving to a setpoint position. This task has an actuation cost of $c_j = 0$; its task cost $c_{i,j}$ (and task debit $c_{i,j}{}'$) is only based on the distance $\Delta s_{i,j}$ between the robot and its goal position.

The global objective function (3.12) value of each algorithm was plotted over computation time in Figure 3.6. The global objective function value of the greedy algorithm $(GrdTA)$ alone and the benchmark CPLEX solution were also plotted. The global objective function value from the benchmark was the sum of "Optimal MinMax" $(\max(u_i))$ and "Optimal Cost" (C) from the dataset in [59]. Since a range was provided for "Optimal MinMax," a range for the objective function value was also provided for the benchmark. This range was a global objective function value between 712.1 to 739.15 for the benchmark.

The $GenTA,\ HCTA,\ GrdGenTA,$ and $GrdHCTA$ algorithms were each trialled ten times. Each trial had 120 seconds of computation time in which the algorithms continuously iterated. The average computation time and global objective function value of each algorithm's iterations were recorded and plotted in Figure 3.6, with each algorithm's trials in Appendix A2.

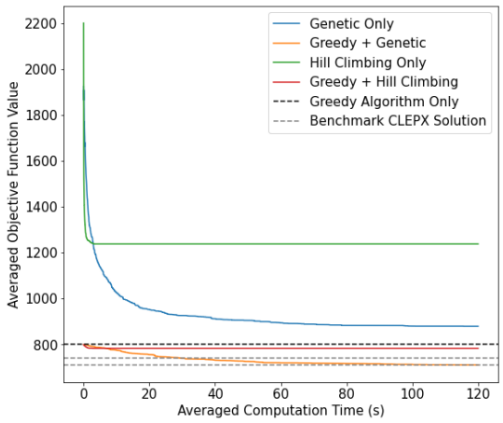


Figure 3.6 – Proposed Task Allocation Algorithms' Averaged Global Objective Function Value over Computation Time for eil51 with 5 Salesmen

On average, a single iteration of the genetic algorithm required 0.0123 seconds, and a single iteration of hill climbing required 0.000572 seconds. On average, the trials resulted in 9792 iterations of $GenTA$, 209735 iterations of $HCTA$, 9508 iterations of $GrdGenTA$ and 207336 iterations of $GrdHCTA$. The $GrdTA$ algorithm was executed once, with an objective function value of $F = 799.22$ with a computation time of 1.71 seconds. It was counted as part of the execution of the $GrdGenTA$ and $GrdHCTA$ algorithms, which resulted in slightly fewer iterations for them when compared to $GenTA$ and $HCTA$, respectively.

Figure 3.6 shows the relative performance of each algorithm in finding an optimal task allocation. The $GrdGenTA$ algorithm is the most optimal because, as per the criteria in Section 3.1, it achieves the lowest objective function value, which was also maintained throughout the entire duration of code execution. Furthermore, it appears that additional iterations of $GenTA$ and $GrdGenTA$ would likely result in an even lower objective function value for their respective trial runs.

It can also be noted that the $GrdTA$ algorithm alone is more optimal than the $GenTA$ algorithm alone for this particular MTSP problem with the 120-second time limit for computation. The $GrdTA$ algorithm only required 1.71 seconds of computation time to generate a usable task allocation, while the $GenTA$ algorithm alone, even after 120 seconds, did not reach the same optimality as the $GrdTA$ algorithm. Overall, the $GrdTA$ is extremely efficient, providing a disproportionate level of optimality compared to its computation time.

The solutions involving hill climbing appear to be trapped within local optima. They rapidly converge onto and remain constant at suboptimal global objective function values. The objective function value remains constant after around 2 seconds of computation instead of continuing to decrease like in the $GenTA$ and $GrdGenTA$ solutions. Due to the methodology of averaging ten different trials and the implementation of hill climbing, which discards worse solutions, this observation regarding hill climbing is only possible because each of the ten trials gets trapped in its own distinct local optima.

Statistics of the final iteration from the ten trials of each algorithm are listed in Table 3.2. The final iteration count and objective function values for each trial of each algorithm are in Appendix A2. The benchmark and greedy algorithm only solutions are also listed. The average global objective function is the value plotted in Figure 3.6. Note that at the final iteration at 120

seconds, the average objective function value of the $GenTA$ was still greater than the $GrdTA$, which only required 1.71 seconds.

Table 3.2 – Task Allocation Algorithms' Objective Function Statistics at Final Iteration For eil51, 5 Salesmen Problem

Solution	Worst Global Objective Function (F)	Best Global Objective Function (F)	Average Global Objective Function (F)	Worst – Best Global Objective Function	Standard Deviation	Average Iteration Count
Benchmark	712.1 to 739.15			N/A	N/A	N/A
$GrdTA$	799.22			N/A	N/A	1
$GenTA$	1218.70	724.78	878.71	493.92	132.33	9792
$GrdGenTA$	730.07	698.19	708.86	31.87	10.18	9508
$HCTA$	2206.19	778.47	1238.21	1427.72	501.32	209735
$GrdHCTA$	787.23	774.59	782.80	12.65	5.95	207336

According to Table 3.2, the solutions that employ an initial greedy algorithm have significantly less standard deviation than those that do not. This difference in standard deviations is because of the starting conditions of each algorithm. The metaheuristic-only solutions start with random conditions that differ from each trial. In contrast, the combination solutions start with the same greedy algorithm solution in each trial. A greater standard deviation can be interpreted as a reduced consistency, with a greater standard deviation possibly generating wildly different task allocations.

The standard deviations over computation time are also plotted in Figure 3.7. Note the high constant standard deviation combined with the constant average objective function value from Figure 3.6 for hill climbing only. These observations are indicative of hill climbing being stuck in local optima. However, the high standard deviation indicates that each trial's local optima differs vastly. This observation was also confirmed in Table 3.2, with hill climbing having the greatest difference between the worst and best objective function values.

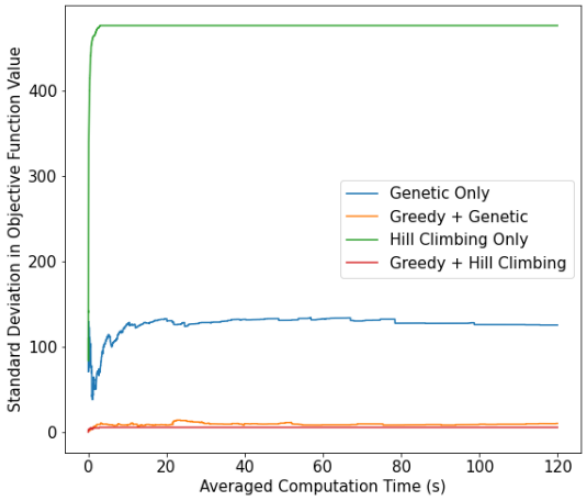


Figure 3.7 – Standard Deviation in Objective Function Value for Proposed Task Allocation Algorithms over Time

Under real-life conditions, an algorithm would be required to produce an optimal task allocation within a reasonable timeframe. It is also possible that the algorithm only has one opportunity to generate this task allocation. Therefore, consistency is critical to prevent wildly different and unpredictable results. From Table 3.2 and Figure 3.7, the $GrdGenTA$ algorithm exihibits low standard deviation compared to the other algorithms. The $GrdHCTA$ algorithm is the most consistent with the least standard deviation; however, each trial ends up stuck in local optima close to the "super individual" $GrdTA$ solution. Overall, the $GrdGenTA$ solution is the most optimal solution while also having a high level of consistency.

Selected solutions from the proposed algorithms are shown in Figure 3.8b to Figure 3.12, with the benchmark shown in Figure 3.8a. In these plots, each coloured path corresponds to the trajectory of one individual robot or salesman. The different colours represent different robots/salesmen. The overall performance of these solutions is listed in Table 3.3, describing total cost (3.11), global objective function (3.12) value and computation time. Note that the computation time was not available from the benchmark but was ascertained to range from 70 hours to 100 hours based on the other test cases in the benchmark [59].

Figure 3.8 to Figure 3.12 visualizes the effect of greater optimality (lower global objective function value) on the robots' trajectories. In particular, as the task allocation's global objective function decreases, the robots' trajectories become cleaner and more efficient. More optimal task sequences have fewer intersecting / overlapping paths, which result in shorter paths and, consequently, lower usage u_i, depicted by a lower total cost which sums up the overall usage of all robots. The opposite is also true; more overlapping paths result in greater distance and a less optimal solution. This is because overlapping trajectories often mean that one or more robots travel the same or similar paths multiple times. An optimal solution would ensure that each path segment is covered only as needed and no more. This is evident from the observation of Figure 3.8 to Figure 3.12 and comparing it with the global objective function F values in Table 3.3; as the global objective function value decreases, the robots' paths have fewer intersections.

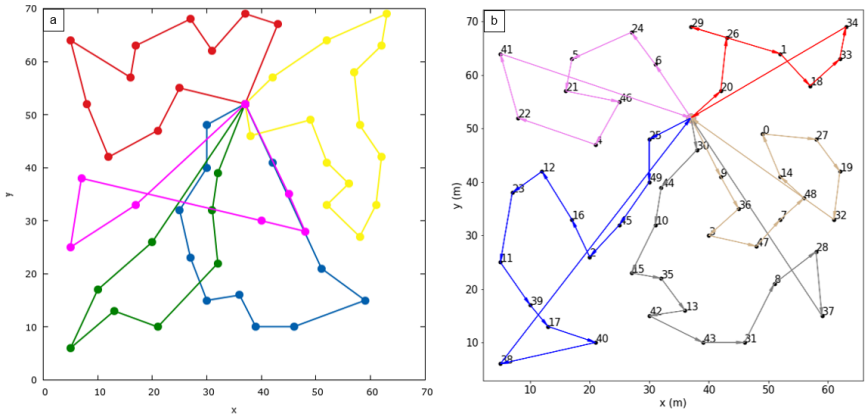


Figure 3.8 – eil51, 5 Salesmen Solutions: a) Benchmark [98], b) Greedy Task Allocation Algorithm

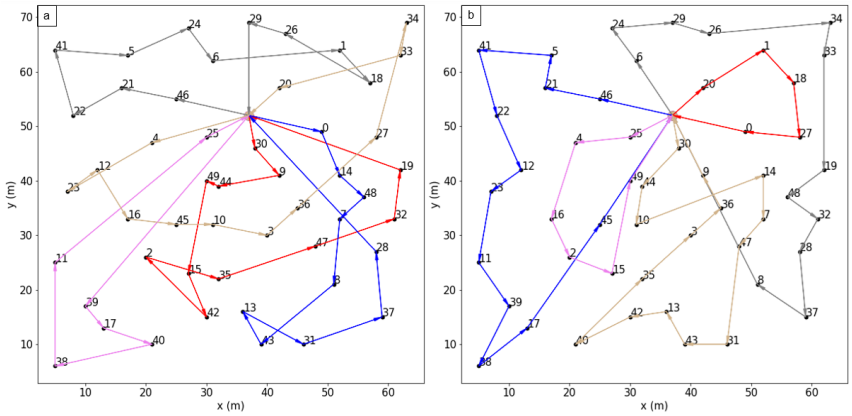


Figure 3.9 – eil51, 5 Salesmen Solutions from Genetic Algorithm Only: (a) 1500 iterations and (b) 7500 Iterations

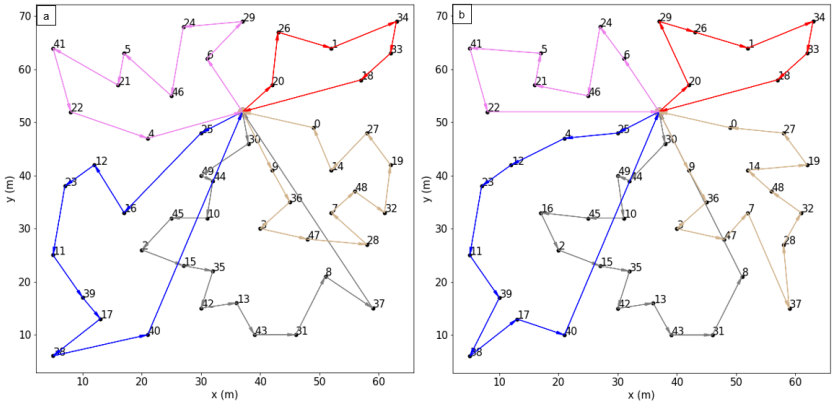


Figure 3.10 – eil51, 5 Salesmen Solutions from Combined Greedy + Genetic Algorithm with (a) 1500 Iterations and (b) 6000 Iterations

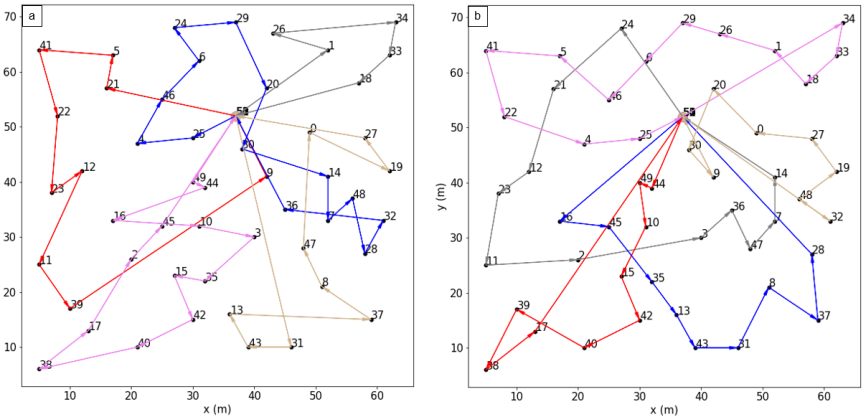


Figure 3.11 – eil51, 5 Salesmen Solutions from Hill Climbing only with (a) 30000 Iterations, and (b) 150000 Iterations

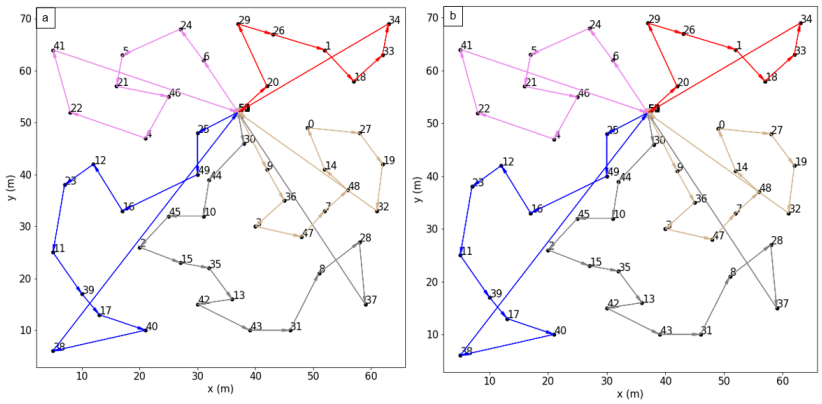


Figure 3.12 – eil51, 5 Salesmen Solutions from Greedy + Hill Climbing Algorithm with (a) 30000 iterations, and (b) 120000 Iterations

Table 3.3 – Comparison of Solutions for eil51, 5 Salesmen Problem

Solution	**Figure**	**Total Cost** (C)	**Objective Function** (F)	**Iterations** (n_{iter})	**Computation Time** (s)
Benchmark	Figure 3.8a	615.19	712.1 to 739.15	N/A	Several hours
$GrdTA$	Figure 3.8b	632.94	799.22	1	1.71
$GenTA$	Figure 3.9a	715.21	869.69	1500	16.85
$GenTA$	Figure 3.9b	617.66	778.94	7500	80.74
$GrdGenTA$	Figure 3.10a	589.79	734.05	1500	18.22
$GrdGenTA$	Figure 3.10b	577.83	710.99	6000	67.27
$HCTA$	Figure 3.11a	718.33	882.14	30000	17.12
$HCTA$	Figure 3.11b	677.59	830.64	150000	97.29
$GrdHCTA$	Figure 3.12a	631.05	787.24	30000	19.44
$GrdHCTA$	Figure 3.12b	631.05	787.24	120000	76.80

The performance of the $GrdGenTA$ algorithm was also compared to four other solutions from the same benchmark dataset [59] as the eil51, 5 salesmen solution. These additional benchmark solutions involve different numbers of salesmen and "go-to-goal tasks," which are plotted in [99], [100], [101] and [102] with solutions from the proposed $GrdGenTA$ algorithm plotted in Figure 3.13. The performance of the $GrdGenTA$ algorithm in these additional test cases was similar to the eil51 with 5 salesman test case.

Table 3.4 – Benchmark vs. Greedy + Genetic Algorithm Solutions

Test Case	**Algorithm**	**Total Cost** (C)	**Objective Function** (F)	**Iterations** (n_{iter})	**Computation Time** (s)
eil51, 3 salesmen	Benchmark [99]	477.15	627.85 to 636.72	N/A	N/A
	$GrdGenTA$	566.98	772.13	3000	33.27
eil51, 7 salesmen	Benchmark [100]	762.83	838.74 to 874.90	N/A	N/A
	$GrdGenTA$	659.88	785.06	3000	42.29
berlin52, 5 salesmen	Benchmark [101]	12084.90	13756.50 to 14526.29	N/A	N/A
	$GrdGenTA$	10978.11	14377.49	3000	38.66
eil76, 7 salesmen	Benchmark [102]	964.69	1053.04 to 1104.31	N/A	N/A
	$GrdGenTA$	973.78	1133.42	3000	60.22

According to Table 3.4, the proposed $GrdGenTA$ algorithm outperformed the benchmark, with a global objective function of 6.5% to 10.2% lower. On the other hand, the benchmark was superior in the eil51, 3 salesmen test case, with its global objective function being 17.53% to 18.86% less than the $GrdGenTA$ algorithm. Differences between the other 2 test cases were small. As for computation time, the vast majority of it comprised of genetic algorithm iterations, with the greedy algorithm component completing in under 2 seconds for each test case. The $GrdGenTA$ performed similarly in these test cases as in the eil51, 5 salesman test case.

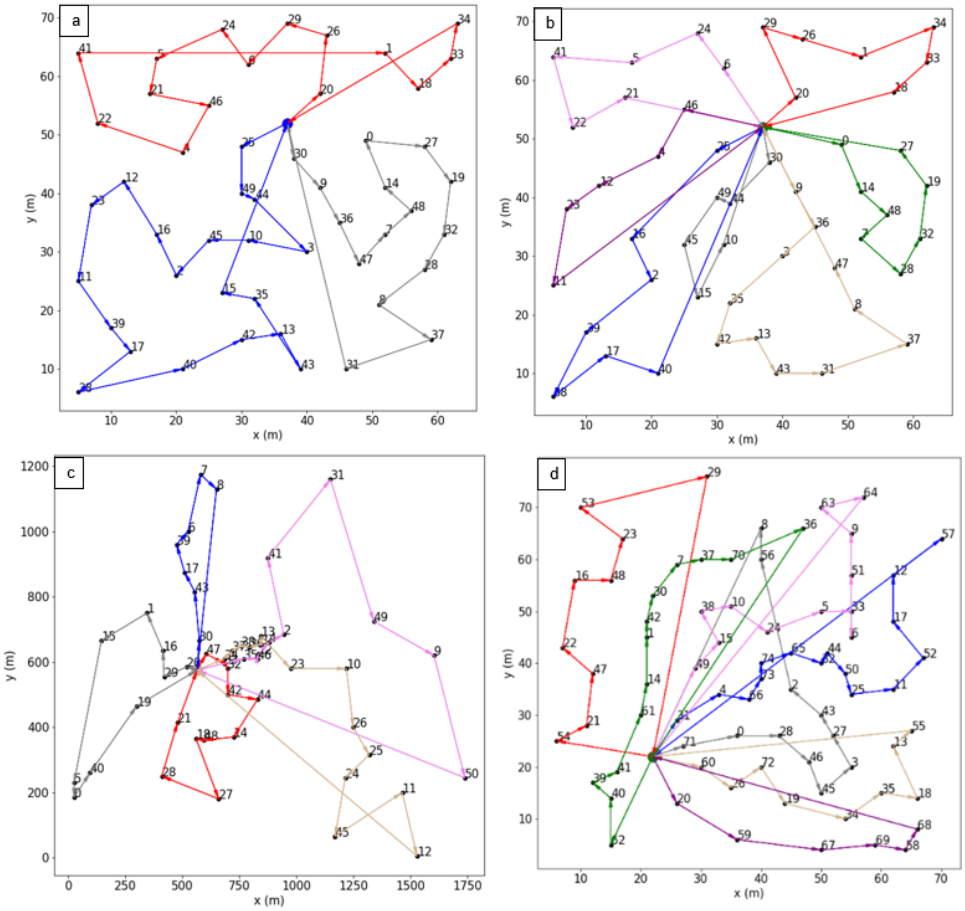


Figure 3.13 – Greedy + Genetic Algorithm Solutions for (a) eil51, 3 Salesmen, (b) eil51, 7 Salesmen, (c) berlin52, 7 Salesmen, and (d) eil76, 5 Salesmen

Overall, the test cases in Section 3.3.1 have demonstrated that the $GrdGenTA$ is capable of achieving the same or greater levels of optimality as contemporary solutions but with reduced computational demand. Except for the eil51, 3 salesmen test case, it calculated task allocations with greater or equal optimality as measured by the objective function (3.12). These solutions were also obtained within a reasonable timeframe measured in seconds, not minutes or hours. This rapid task allocation enables it to be used effectively with real-life and simulated robots in time-sensitive conditions.

The efficiency of the $GrdTA$ algorithm was also demonstrated. It generated "usable" task allocations with only one iteration, resulting in minimal computation time. While the $GrdTA$ task allocations do not approach global optimality, they are still much more optimal than randomly generated task allocations.

3.3.2 – Evaluation of Constrained Optimization

This section evaluates the ability of the $GrdTA$ and $GrdGenTA$ algorithms to respond to the optimization constraints in (3.14) to (3.16), which are work capacity, suitability and prerequisites. The impact of changes in the objective function related to total cost and task distribution is also evaluated. The evaluations were conducted with the use of the 31T-6R, 16T-3R and 24T-5R scenarios described in Appendix A1, Table A.3 to Table A.8 with agent-task

suitabilities described in Table A.9. These scenarios involve a variable number of tasks (index T) and robots (index R). Unless otherwise stated, the tasks, robots and their properties are identical to these test cases. The genetic algorithm component used an iteration limit of $n_{iter} = 1000$. For reference, the solutions for unconstrained test cases are shown in Figure 3.14 for the $GrdTA$ algorithm and Figure 3.15 for the $GrdGenTA$ algorithm. The unconstrained test cases ignore the work capacity, suitability and prerequisite constraints. The performance of the individual robots is listed in Table 3.5 for the $GrdTA$ algorithm and Table 3.6 for the $GrdGenTA$ algorithm. The performance of the overall system for both algorithms is listed in Table 3.7. The unconstrained test cases will be used as a reference point to assess the impact of applying the optimization constraints in (3.14) to (3.16).

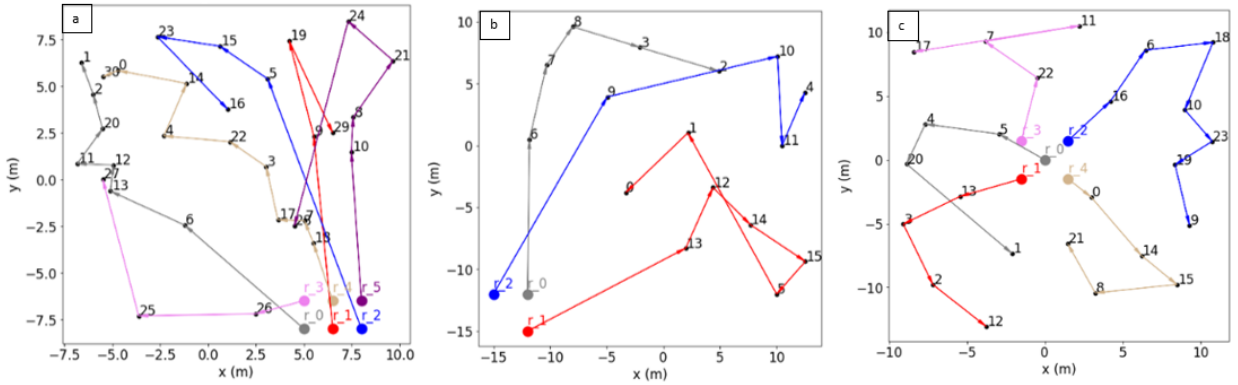


Figure 3.14 – Unconstrained Greedy Task Allocation Solutions for (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

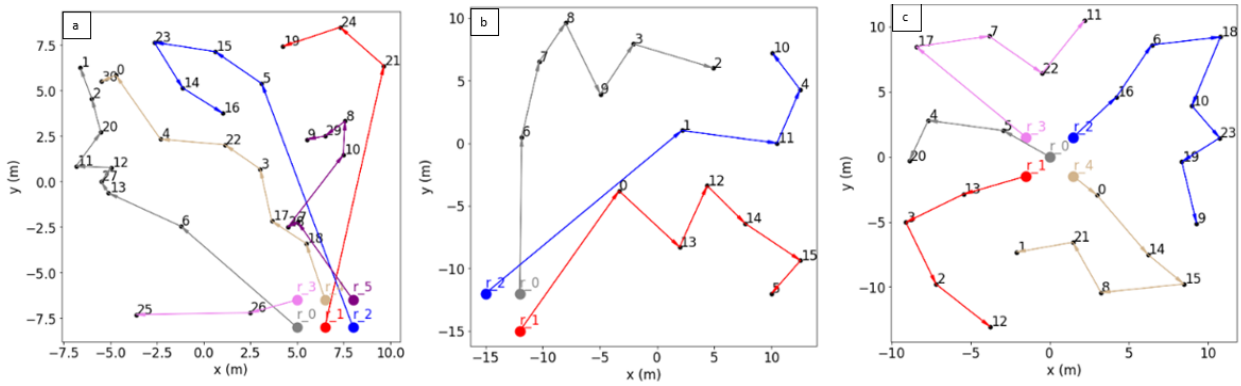


Figure 3.15 – Unconstrained Greedy + Genetic Task Allocation Solutions for (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

Table 3.5 – Robots' Performance with Greedy Task Allocation in Unconstrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	6, 13, 12, 11, 20, 2, 1	21.93	6, 7, 8, 3, 2	83.94	5, 4, 20, 1	33.452
r_1	9, 19, 29	33.06	13, 12, 14, 15, 5, 1, 0	105.30	13, 3, 2, 12	24.27
r_2	5, 15, 23, 16	41.96	9, 10, 11, 4	70.17	16, 6, 18, 10, 23, 19, 9	35.541
r_3	26, 25, 27	28.27	N/A	N/A	22, 7, 11, 17	30.49
r_4	18, 7, 17, 3, 22, 4, 14, 0, 30	34.19	N/A	N/A	0, 14, 15, 8, 21	28.41
r_5	10, 8, 21, 24, 28	40.03	N/A	N/A	N/A	N/A

Table 3.6 – Robots' Performance with Greedy + Genetic Task Allocation in Unconstrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	6, 13, 27, 12, 11, 20, 2, 1	34.22	6, 7, 8, 9, 3, 2	89.14	5, 4, 20	19.71
r_1	21, 24, 19	21.12	0, 13, 12, 14, 15, 5	64.39	13, 3, 2, 12	24.27
r_2	5, 15, 23, 14, 16	42.10	1, 11, 4, 10	86.46	16, 6, 18 10, 23, 19, 9	35.54
r_3	26, 25	8.72	N/A	N/A	17, 7, 22, 11	27.80
r_4	18,17, 3, 22, 4, 0, 30	31.29	N/A	N/A	0, 14, 15, 8, 21, 1	36.07
r_5	7, 28, 10, 8, 29, 9	39.03	N/A	N/A	N/A	N/A

Table 3.7 – System Performance in Unconstrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$
Total Cost (C)	199.43	176.46	259.41	239.99	152.163	143.39
Global Objective Function (F)	241.39	218.56	364.71	329.13	187.704	179.46
Computation Time (s)	0.372	8.04	0.035	5.27	0.157	7.144

Test Case 1 – Performance with Prerequisite Constraint

This test case explores the effect of prerequisite constraints on task allocation. It involves the 3 test scenarios (31T-6R, 16T-3R and 24T-5R) in which prerequisite constraints (3.16) are applied as defined in Tables A.3, A.5 and A.7 of Appendix A1, but suitability (3.15) and work capacity (3.14) constraints are ignored. The aforementioned tables describe each tasks' location, task type, actuation cost and prerequisites.

The solutions for the prerequisite-constrained test cases are shown in Figure 3.16 for $GrdTA$ and Figure 3.17 for $GrdGenTA$. The performance of individual robots is listed in Table 3.8 for $GrdTA$ and Table 3.9 for $GrdGenTA$. The performance of the overall system for both algorithms is listed on Table 3.10.

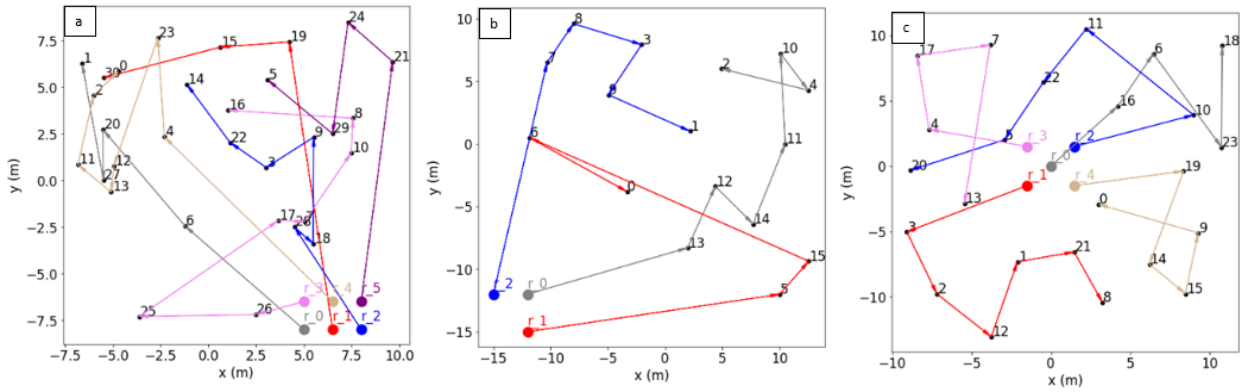


Figure 3.16 – Prerequisite Constrained Greedy Task Allocation Solutions for (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

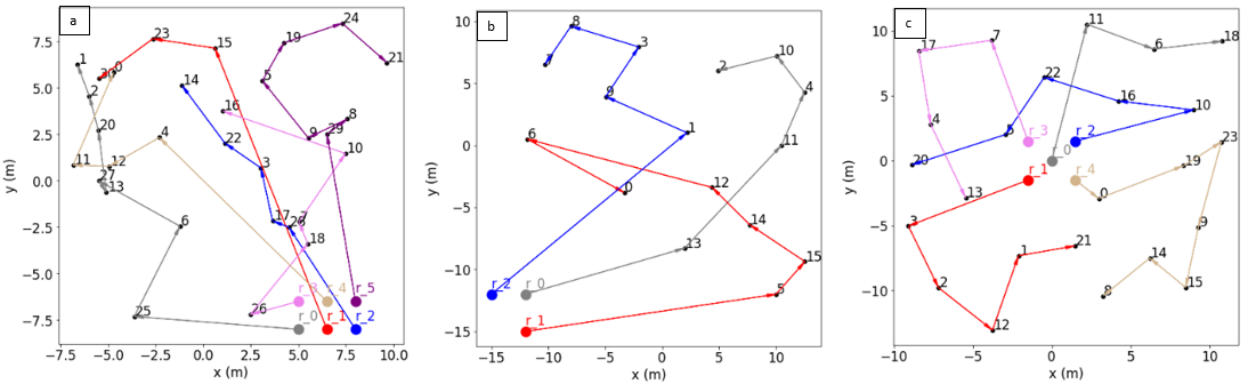


Figure 3.17 – Prerequisite Constrained Greedy + Genetic Task Allocation Solutions for (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

Table 3.8 – Robots' Performance with Greedy Task Allocation in Prerequisite Constrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	6, 20, 27, 1	36.12	13, 12, 14, 11, 10, 4, 2	98.18	16, 6, 23, 18	28.91
r_1	19, 15, 30	37.55	5, 15, 6, 0	85.73	3, 2, 12, 1, 21, 8	44.10
r_2	28, 18, 9, 3, 22, 14	34.70	7, 8, 3, 9, 1	89.78	10, 11, 22, 5, 20	39.68
r_3	26, 25, 17, 7, 10, 8, 16	47.87	N/A	N/A	4, 17, 7, 13	38.94
r_4	4, 23, 12, 13, 11, 2, 0	34.38	N/A	N/A	19, 14, 15, 9, 0	35.02
r_5	21, 24, 29, 5	38.62	N/A	N/A	N/A	N/A

Table 3.9 – Robots' Performance with Greedy + Genetic Task Allocation in Prerequisite Constrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	25, 6, 27, 13, 20, 2, 1	38.82	13, 11, 4, 10, 2	88.21	11, 6, 18	19.77
r_1	15, 23, 30	35.12	5, 15, 14, 12, 6, 0	86.26	3, 2, 12, 1, 21	37.85
r_2	28, 17, 3, 22, 14	28.53	1, 9, 3, 8, 7	92.24	10, 16, 22, 5, 20	37.18
r_3	26, 18, 7, 10, 16	36.01	N/A	N/A	7, 17, 4, 13	34.58
r_4	4, 12, 11, 0	22.86	N/A	N/A	0, 19, 23, 9, 15, 14, 8	37.85
r_5	29, 8, 9, 5, 19, 24, 21	37.41	N/A	N/A	N/A	N/A

Table 3.10 – System Performance in Prerequisite Constrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$
Total Cost (C)	229.25	198.76	273.68	266.71	186.66	167.23
Global Objective Function (F)	277.12	237.58	371.87	358.95	230.76	205.08
Computation Time (s)	0.40	9.37	0.037	6.15	0.16	8.23

The prerequisite constraint (3.16) reduces the number of admissible selections, forcing the task allocation algorithm to satisfy sequential requirements and reducing the solution's optimality. With this constraint, tasks with no dependants tend to be performed later, and tasks with dependants are performed earlier, even if this selection results in greater usages and, consequently, global objective function value. This impact on total cost and the global objective function is observed in Table 3.10. Compared to Table 3.7, the test cases with prerequisite constraints all have a greater total cost and global objective function value.

Suppose we have the task sequence $TS_i = \{T_J^{i,k_J} | J \in T \big| k_J = 1,2,3 \dots N_i\}$ and identify a task within that sequence $T_j^{i,k_j} \in TS_i$, where task T_J^{i,k_J} is a prerequisite of T_j^{i,k_j}, $J \in P_j$ which also makes T_j^{i,k_j} a dependant of T_J^{i,k_J}. The application of the prerequisite constraint ensures that within each task sequence, the execution order k_J of prerequisites is earlier than the execution order k_j of dependants as described in (3.25). If (3.25) is true for all task sequences TS_i within a task allocation TA, then the prerequisite constraint (3.16) is satisfied, as dependants are allocated after their prerequisites.

$$\begin{gathered} k_J < k_j \\ J \in P_j \\ for\ all\ T_j^{i,k_j} \in TS_i\ and\ T_J^{i,k_J} \in TS_i \end{gathered} \tag{3.25}$$

The condition in (3.25) was satisfied by the $GrdTA$ and $GrdGenTA$ which means that the task allocations in Figure 3.16 and Figure 3.17 are admissible. Some examples of compliance within the prerequisite-constrained test case are listed in Table 3.11**.** This table provides several examples of “task chains” describing an admissible chronological order (first to last) leading up to the completion of a task(s). Note that tasks without prerequisites, even though they may be completed as part of robots’ task sequences, are not listed in these task chains. For example, the task chain in Table 3.11 for Figure 3.17c describes the order of tasks leading up to the completion of T_0 in the 24T-5R scenario. First, T_{10} is performed by r_2 satisfying P_2, which is the list of prerequisites for T_2. This enables r_1 to perform T_2 and T_1. With T_1, completed, T_0 can then be performed by r_4.

Table 3.11 – Compliance with Prerequisite Constraints

Scenario	Algorithm	Figure	Task Chain
31T-6R	$GrdTA$	Figure 3.16a	1) T_{29} performed by r_5. P_8, P_{10}, P_9 satisfied. 2) T_{28} performed by r_2. P_7, P_{17}, P_{18} satisfied. 3) T_{17}, T_7, T_{10}, T_8 performed by r_3. 4) T_{16} performed by r_3. P_{22}, P_{14} satisfied. 5) $T_{18}, T_9, T_{22}, T_{14}$ performed by r_2.
31T-6R	$GrdGenTA$	Figure 3.17a	1) T_{29} performed by r_5. P_8, P_{10}, P_9 satisfied. 2) T_{28} performed by r_2. P_7, P_{18} satisfied. 3) T_8, T_9 performed by r_5. 4) T_{18}, T_7, T_{10} performed by r_3.
16T-3R	$GrdTA$	Figure 3.16b	1) T_5 performed by r_1. P_{14} satisfied. 2) T_{14}, T_4 performed by r_0. P_6 satisfied. 3) T_6, T_0 performed by r_1. P_1, P_9 satisfied. 4) T_9, T_1 performed by r_2.
16T-3R	$GrdGenTA$	Figure 3.17b	1) T_5 performed by r_1. P_{14}, P_{15} satisfied. 2) T_{14}, T_{15} performed by r_1. 3) T_4 performed by r_0. P_6 satisfied. 4) T_6 performed by r_1.
24T-5R	$GrdTA$	Figure 3.16c	1) T_{10} performed by r_2. P_2, P_{11} satisfied. 2) T_{11} performed by r_2. 3) T_2 performed by r_1. 4) T_1 performed by r_1. P_{22} satisfied. 5) T_{22} performed by r_2.
24T-5R	$GrdGenTA$	Figure 3.17c	1) T_{10} performed by r_2. P_2 satisfied. 2) T_2, T_1 performed by r_1. P_0 satisfied. 3) T_0 performed by r_4.

It must be noted that the optimization only considers robots' travel distances and does not consider task completion time. While the sequences are admissible, they may not be optimized for time. In reality, a robot may have to wait for other robots to complete their tasks. For example, in the task chain in Table 3.11 for Figure 3.17c, the first task assigned to r_4 is T_0. However, as described earlier, T_0 cannot be performed until after T_{10}, T_2, T_1. Furthermore, there are tasks assigned to T_0 that have no prerequisites, such as T_9 and T_{14}. From a completion time perspective, it could make sense to perform these prerequisite-free tasks instead of doing nothing before T_0 is performed.

It is also observed that some of those agent-task assignments in the unconstrained test cases would have rendered the task allocation inadmissible if the prerequisite constraint had been considered. Some examples of these would-be violations are listed in Table 3.12**.** These violations are instances in which a prerequisite and its dependant are assigned to the same robot but with the prerequisite to be performed after the dependant. In other words, (3.25) is not satisfied. These violations do not occur when the prerequisite constraint is enforced, as in Figure 3.16 and Figure 3.17.

Table 3.12 – Violations of Prerequisite Constraints

Scenario	Algorithm	Figure	Agent-Task Assignments	Notes
31T-6R	$GrdTA$	Figure 3.14a	T_9 and T_{29} assigned to r_1. T_9 before T_{29}.	T_{29} is a prerequisite of T_9.
31T-6R	$GrdGenTA$	Figure 3.15a	T_{10} and T_{29} assigned to r_5. T_{10} before T_{29}.	T_{29} is a prerequisite of T_{10}.
16T-3R	$GrdTA$	Figure 3.14b	T_1 and T_0 assigned to r_2. T_1 before T_0	T_0 is a prerequisite of T_1.
16T-3R	$GrdGenTA$	Figure 3.15b	T_{14} and T_5 assigned to r_1. T_{14} before T_5	T_5 is a prerequisite of T_{14}
24T-5R	$GrdTA$	Figure 3.14c	T_7 and T_{17} assigned to r_3. T_7 before T_{17}	T_{17} is a prerequisite of T_7
24T-5R	$GrdGenTA$	Figure 3.15c	T_0 and T_1 assigned to r_4. T_0 before T_1	T_1 is a prerequisite of T_0.

Overall, the $GrdTA$ and $GrdGenTA$ algorithms were able to adapt to the prerequisite constraint, generating task allocations such that prerequisites are completed before dependants in an optimized manner. However, this optimality is based on travel distance but may be suboptimal when considering time. This time suboptimality represents a limitation and an area for future work discussed in Sections 6.3 and 7.3.

Test Case 2 – Performance with Suitability Constraint

This test case explores the effect of suitability constraints on task allocation. It involves the 3 test scenarios (31T-6R, 16T-3R and 24T-5R) in which suitability constraints (defined in Table A.9 of Appendix 1) are applied, but prerequisite (3.16) and work capacity (3.14) constraints are ignored. For the suitability constraint (3.15), the minimum suitability requirement was set to $s_{min} = 0.65$.

The solutions for the suitability-constrained test cases are shown in Figure 3.18 for $GrdTA$ and Figure 3.19 for $GrdGenTA$. The performance of individual robots is listed in Table 3.13 for $GrdTA$ and Table 3.14 for $GrdGenTA$. The performance of the overall system for both algorithms is listed in Table 3.15.

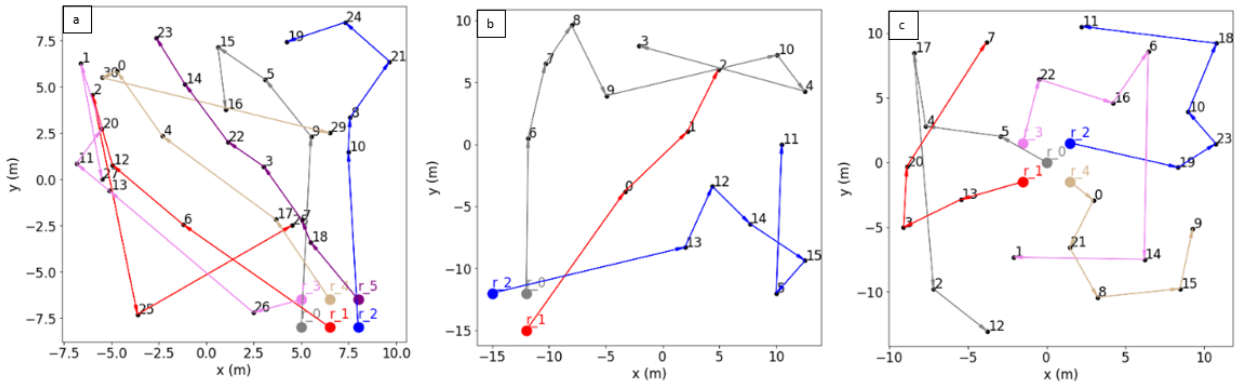


Figure 3.18 – Suitability Constrained Greedy Task Allocation Solutions for (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

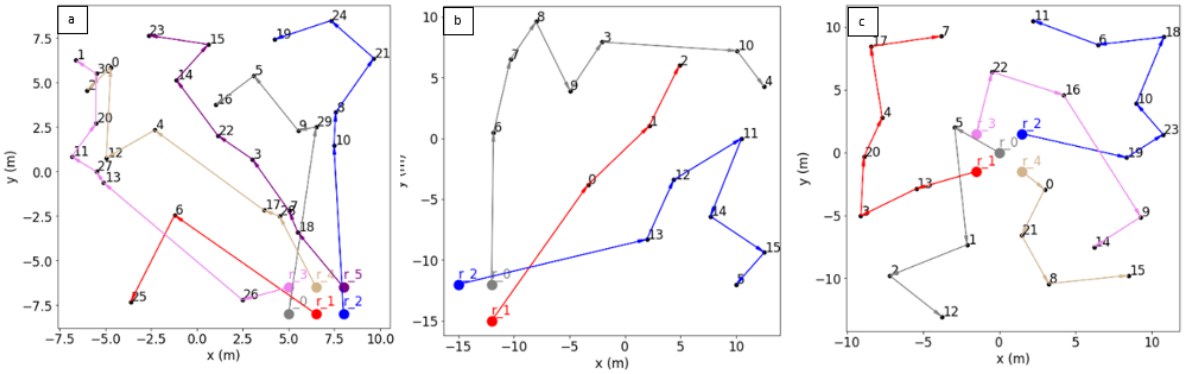


Figure 3.19 – Suitability Constrained Greedy + Genetic Task Allocation Solutions for (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

Table 3.13 – Robots' Performance with Greedy Task Allocation in Suitability-Constrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	9, 5, 15, 16	36.69	6, 7, 8, 9, 10, 4, 3	111.19	5, 4, 17, 2, 12	49.09
r_1	6, 12, 2, 25, 28	51.94	0, 1, 2	99.16	13, 3, 20, 7	29.97
r_2	10, 8, 21, 24, 19	21.43	13, 12, 14, 15, 5, 11	48.68	19, 23, 10, 18, 11	27.42
r_3	26, 13, 11, 20, 1, 27	39.29	N/A	N/A	22, 16, 6, 14, 1	45.08
r_4	17, 4, 0, 30, 29	54.13	N/A	N/A	0, 21, 8, 15, 9	32.26
r_5	18, 7, 3, 22, 14, 23	17.89	N/A	N/A	N/A	N/A

Table 3.14 – Robots' Performance with Greedy + Genetic Task Allocation in Suitability Constrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	29, 9, 5, 16	46.16	6, 7, 8, 9, 3, 10, 4	97.88	5, 1, 2, 12	33.29
r_1	6, 25	14.94	0, 1, 2	99.16	13, 3, 20, 4, 17, 7	38.68
r_2	10, 8, 21, 24, 19	21.44	13, 12, 11, 14, 15, 5	46.18	19, 23, 10, 18, 6, 11	27.78
r_3	26, 13, 27, 11, 20, 30, 1	45.41	N/A	N/A	22, 16, 9, 14	30.91
r_4	28, 17, 4, 12, 0, 2	34.79	N/A	N/A	0, 21, 8, 15	23.54
r_5	18, 7, 22, 14, 15, 23	20.91	N/A	N/A	N/A	N/A

Table 3.15 – System Performance in Suitability Constrained Test Cases

	31T-6R		**16T-3R**		**24T-5R**	
	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$
Total Cost (C)	221.39	183.64	259.03	243.23	183.83	154.38
Global Objective Function (F)	275.53	229.73	370.22	342.39	232.91	193.24
Computation Time (s)	0.39	8.13	0.03	4.98	0.16	6.79

The suitability constraint (3.15) prevents certain tasks from being allocated to certain robots, reducing the number of available choices for the task allocation algorithm and consequently reducing its optimality. With this constraint, tasks of lower cost may be avoided due to their lack of suitability. In contrast, tasks of greater cost may be selected because they are suitable. This selection results in a greater overall cost and global objective function value. This suboptimality is observed in all cases, as listed in Table 3.13, similar to the prerequisite-constrained scenarios. Compared to Table 3.7, the test cases with suitability constraints all have a greater global objective function F value.

This reduction in optimality stems from the impact on the robots' task sequences. Similar to the prerequisite constraint, the suitability constraint can force robots to take farther paths by preventing them from selecting closer (unsuitable) tasks. Compared to unconstrained scenarios, the suitability constraint forces some tasks to be assigned to different robots due to suitability requirements. Examples include T_6 in 31T-6R, T_9 in 16T-3R and T_7 in 24T-5R. These tasks were assigned to different robots with the suitability requirement. These tasks' suitability scores in the unconstrained test case were less than the minimum requirement of $s_{min} = 0.65$, which necessitates their re-assignment to comply with (3.15). This change resulted in significantly

different task sequences, which overall increased the usage u_i of each robot, as listed in Table 3.13 and Table 3.14, compared to Table 3.5 and Table 3.6.

Test Case 3 – Impact of Work Capacity Variation with Prerequisite and Suitability Requirements

This test case explores the effect of varying the work capacities of each robot. It tests the algorithms' ability to respond to constraint (3.14). It involves three variations of the 31T-6R scenario, which originally had infinite work capacity for each robot. The variations of the 31T-6R were developed to limit the robots' work capacities. These work capacities for each scenario are listed in Table 3.16. All other properties, such as prerequisites and suitability, are applied and are identical to the original 31T-6R scenario defined in Table A.3 and Table A.4 of Appendix A1.

The solutions for these 31T-6R variants with different work capacities are shown in Figure 3.20 for the $GrdTA$ and Figure 3.21 for the $GrdGenTA$. The performance of individual robots is listed in Table 3.17 for the $GrdTA$ and Table 3.18 for the $GrdGenTA$. The performance of the overall system for both algorithms is listed in Table 3.19.

Table 3.16 – Robots' Work Capacities for 31T-6R Scenario Variants

Robot (r_i)	**31T-6R50 Work Capacities** (u_{i_max})	**31T-3R44-3R26 Work Capacities** (u_{i_max})	**31T-2R65-4R25 Work Capacities** (u_{i_max})
r_0	50	44	25
r_1	50	44	65
r_2	50	44	25
r_3	50	26	25
r_4	50	26	65
r_5	50	26	25

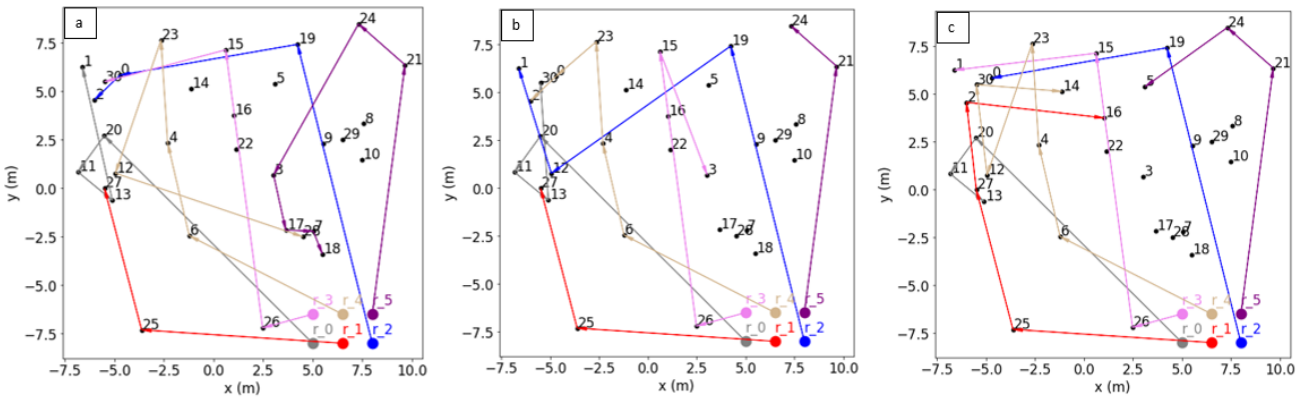


Figure 3.20 – Greedy Task Allocation Solutions for 31T-6R Variants (a) 31T-6R50, (b) 31T-3R44-3R26, (c) 31T-2R65-4R25

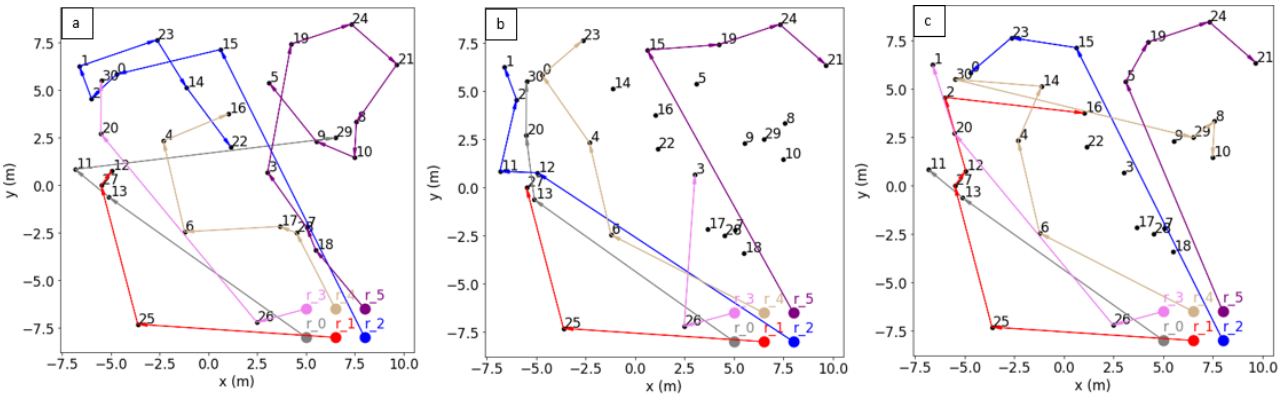


Figure 3.21 – Greedy + Genetic Task Allocation Solutions for 31T-6R Variants (a) 31T-6R50, (b) 31T-3R44-3R26, (c) 31T-2R65-4R25

Table 3.17 – Robots' Performance with Greedy Task Allocation in 31T-6R Variants

	31T-6R50		**31T-3R44-3R26**		**31T-2R65-4R25**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	20, 11, 13, 1	26.62	20, 11, 13, 30	37.11	20, 11, 13	19.57
r_1	25, 27	29.69	25, 27	29.69	25, 27, 2, 16	57.37
r_2	19, 0, 2	26.77	19, 12, 1	32.95	19, 0	24.96
r_3	26, 15, 30	35.42	26, 15, 3	23.98	26, 15, 1	24.37
r_4	6, 4, 23, 12, 28	48.21	6, 4, 23, 0, 2	23.51	6, 4, 23, 12, 30, 14	47.35
r_5	21, 24, 3, 17, 7, 18	30.59	21, 24	16.13	21, 24, 5	21.35

Table 3.18 – Robots' Performance with Greedy + Genetic Task Allocation in in 31T-6R Variants

	31T-6R50		**31T-3R44-3R26**		**31T-2R65-4R25**	
Robot (r_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)	**Task Sequence** (TS_i)	**Usage** (u_i)
r_0	13, 11, 29	40.22	13, 20, 30	30.66	13, 11	14.78
r_1	25, 27, 12	30.65	25, 27	29.69	25, 27, 12, 2, 16	57.66
r_2	15, 0, 2, 1, 23, 14, 22	36.92	12, 11, 2, 1	23.15	15, 23, 0	22.89
r_3	26, 20, 30	30.14	26, 3	10.53	26, 20, 1	19.07
r_4	28, 17, 6, 4, 16	46.80	6, 4, 0, 23	20.62	6, 4, 14, 30, 29, 8, 10	60.66
r_5	18, 7, 3, 19, 24, 21, 8, 10, 8, 5	33.63	15, 19, 24, 21	25.55	5, 19, 24, 21	21.59

Table 3.19 – System Performance of 31T-6R Variants

	31T-6R50		**31T-3R44-3R26**		**31T-2R65-4R25**	
	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$	$GrdTA$	$GrdGenTA$
Total Cost (C)	197.32	218.36	163.98	140.21	194.98	196.66
Global Objective Function (F)	245.53	265.16	201.69	170.87	252.35	257.32
Computation Time (s)	0.41	10.65	0.42	10.97	0.41	10.57
Number of Tasks Completed	23/31	31/31	19/31	19/31	21/31	24/31

Both the $GrdTA$ and the $GrdGenTA$ were able to adapt to the limited work capacities of the robots while still complying with both the prerequisite and suitability constraints. However, the (sub)optimality of the $GrdTA$ is evident. In the 31T-6R50 and 31T-2R65-4R25 scenarios, the $GrdGenTA$ resulted in solutions with more tasks allocated while still complying with constraints. As discussed in Section 3.1, an admissible task allocation is more optimal if more tasks are completed, even if its objective function is greater. In the 31T-3R44-3R26 scenario, the number of completed tasks is identical. However, the global objective function value was reduced from 201.69 to 170.87, a 15.28 % reduction with the application of the genetic algorithm component of the $GrdGenTA$.

Only the $GrdGenTA$ solution for 31T-6R50 can be compared with the previous test cases for the original 31T-6R scenario; it is the only one that has all 31 of 31 tasks completed. This comparison is in Table 3.20. As more constraints are added, the solution becomes less optimal than the unconstrained scenario, evident by the increase in objective function value. However, the $GrdGenTA$ algorithm is able to adapt to those constraints to yield an optimized admissible task allocation.

Table 3.20 – Comparison of 31T-6R solutions with Greedy + Genetic Task Allocation

Scenario	**31T-6R50 (add. Work Capacity Constraint)**	**31T-6R (Suitability Constraint Only)**	**31T-6R (Prerequisite Constraint Only)**	**31T-6R (no constraints)**
Reference Table	Table 3.19	Table 3.15	Table 3.10	Table 3.7
Total Cost (C)	218.36	183.64	198.76	176.46
Global Objective Function (F)	265.16	229.73	237.58	218.56

Test Case 4 – Effect of MinMax Optimization

The proposed objective functions in (3.12) and (3.17) were formulated as a MinMax problem. This formulation simultaneously attempts to reduce overall cost and evenly distribute tasks. This distribution was facilitated through the $\max(u_i)$ component so that additional tasks were less likely to be allocated to the robot with the most usage. However, if this $\max(u_i)$ component was removed by using the global objective function in (3.26), the task allocation would become significantly imbalanced.

$$F = C \tag{3.26}$$

The corresponding change in the greedy algorithm is made by adjusting the local optimization function criteria in (3.17) to instead use (3.27). This alternate function is the local analogue of the global objective function in (3.26), which emphasizes total cost. The alternative objective functions in (3.26) and (3.27) will be considered the "MinCost" formulation.

$$f = c_{i,j} \tag{3.27}$$

The 31T-6R, 16T-3R and 24T-5R scenarios, without constraints but with the alternative MinCost objective functions (3.26), (3.27), are plotted in Figure 3.22 for the $GrdTA$ and in Figure 3.23 for the $GrdGenTA$. These scenarios were plotted earlier in Figure 3.14 and Figure 3.15 with the MinMax objective functions.

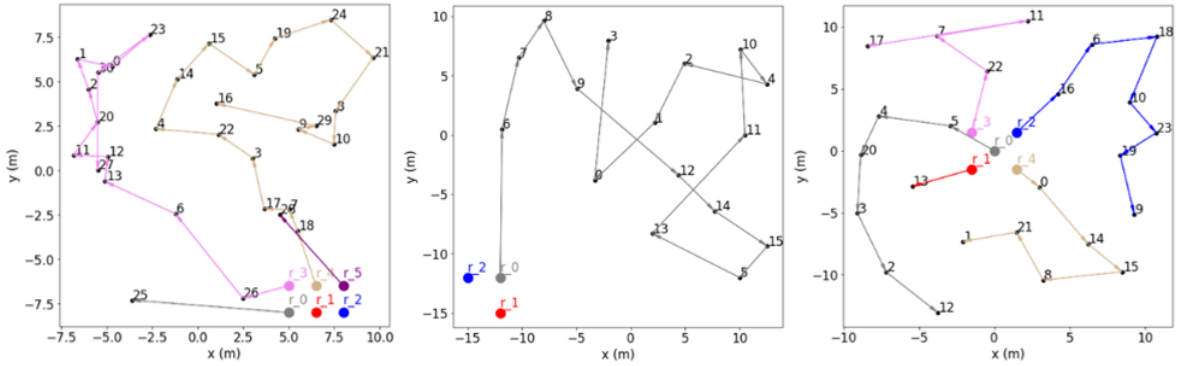


Figure 3.22 – Unconstrained Greedy Task Allocation Solutions with Alternative Global Objective Function: (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

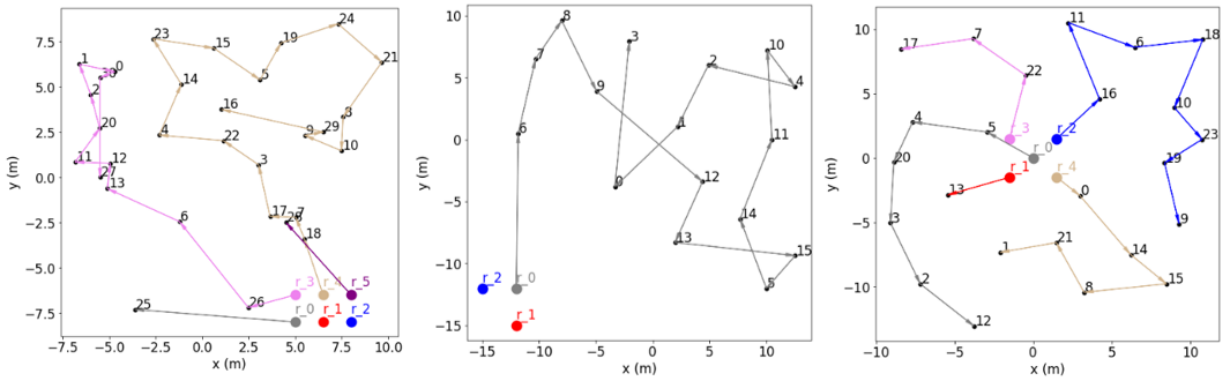


Figure 3.23 – Unconstrained Greedy + Genetic Task Allocation Solutions with Alternative Global Objective Functions: (a) 31T-6R, (b) 16T-3R, and (c) 24T-5R

Based on the observation of Figure 3.22 and Figure 3.23, compared to the same scenarios with the original MinMax objective functions in Figure 3.14 and Figure 3.15, the alternative objective function results in extremely unbalanced task allocations. An exception is the 24T-5R scenario due to its geometry with the robots in a relatively central location to the tasks. An equality index E based on the gini coefficient GC can measure the imbalance in the distributions [103]. An equality index of $E = 1$ corresponds to all robots having the same usage, and an equality index of $E = \frac{1}{M}$ corresponds to one robot (within a team of M robots) having all the usage, while the remaining robots have zero usage.

$$E = 1 - GC, \qquad \frac{1}{M} \leq E \leq 1 \tag{3.28}$$

The gini coefficient GC is a measurement of the inequality of a distribution and is described in (3.29).

$$GC = \frac{2}{M} * \frac{\sum_{i'=1}^{M} i' * u_{i'}}{\sum_{i'=1}^{M} u_{i'}} - \frac{M+1}{M}, \qquad 0 \leq GC \leq 1 \tag{3.29}$$

$$M = number\ of\ robots$$

$$u_{i'} \subseteq \mathrm{U}$$

$$\mathrm{U} = \mathrm{list}\ of\ all\ robots' usages, sorted\ from\ smallest\ to\ largest$$

The usages u_i of all robots from R are sorted from smallest to largest in the list U. The list U contains the sorted usages $u_{i'}$, with its index i' corresponding to its ranking from smallest to largest usage and are not the same as the indices i in R. For example, $u_{i'} = u_1$ would refer to the smallest usage, while $u_{i'} = u_M$ would refer to the largest usage. The weighted sum of sorted values is normalized and adjusted so that a perfectly equal distribution would result in $GC = 0$.

The performance of the task allocation with the MinMax objective functions in (3.12) and (3.17) are compared to the task allocation with the MinCost objective functions in (3.26) and (3.27). This comparison is listed in Table 3.21 for the $GrdTA$ and in Table 3.22 for the $GrdGenTA$. These tables also compare the equality index, E for both sets of task allocations. The values for the original objective function were obtained from Table 3.7. Note that "MinMax Cost" in Table 3.21 and Table 3.22 is the same equation as the original global objective function described in (3.12).

Table 3.21 – Comparison of Greedy Task Allocation Performance with Different Objective Functions

	31T-6R		**16T-3R**		**24T-5R**	
Objective Functions	MinMax	MinCost	MinMax	MinCost	MinMax	MinCost
Total Cost (C)	199.43	160.40	259.41	239.04	152.16	146.57
Maximum usage $(\max(u_i))$	41.96	74.42	105.30	239.04	35.54	36.28
MinMax Cost $(C + \max(u_i))$	241.39	234.82	364.71	478.08	187.7	182.85
Equality Index (E)	0.88	0.42	0.91	0.33	0.93	0.83

In Table 3.21, with the $GrdTA$, the MinMax cost was slightly lower in the 31T-6R and 24T-5R scenarios using the MinCost formulation, so technically, those solutions from the alternate objective function are more optimal (based on the goal of minimizing MinMax cost). However, the difference in MinMax cost was small in those scenarios. With the 16T-3R scenario, the MinCost formulation resulted in a significantly higher MinMax cost. Overall, while it can be argued that the MinCost formulation can result in a more optimal distribution to minimize MinMax cost (for the $GrdTA$ algorithm), it is extremely inconsistent and generally results in less equal distributions. The original MinMax objective functions for the $GrdTA$ and $GrdGenTA$ in (3.12) and (3.17) are preferred for minimizing total cost while equally distributing tasks. The superior equality of the original cost functions in the $GrdTA$ is evident in Table 3.21 from their greater equality indices.

Table 3.22 – Comparison of Greedy + Genetic Task Allocation Performance with Different Objective Functions

	31T-6R		**16T-3R**		**24T-5R**	
Objective Functions	MinMax	MinCost	MinMax	MinCost	MinMax	MinCost
Total Cost (C)	176.46	158.53	239.99	237.29	143.39	140.47
Maximum usage $(\max(u_i))$	42.10	78.01	89.14	237.29	36.07	36.28
MinMax Cost $(C + \max(u_i))$	218.56	236.54	329.13	474.57	179.46	182.31
Equality Index (E)	0.79	0.41	0.93	0.33	0.88	0.76

In Table 3.22, with the $GrdGenTA$, the original objective functions resulted in a lower MinMax cost and greater equality index for all three scenarios. Overall, the addition of the $\max(u_i)$ and u_i components in the global and local objective functions are critical to ensure the even distribution of work in the task allocation algorithms.

3.4 – Summary

This chapter has developed and evaluated a novel solution for the MRTA problem. The proposed solution is the greedy + genetic task allocation ($GrdGenTA$) algorithm. The $GrdGenTA$ uses a greedy algorithm ($GrdTA$) to rapidly generate a usable solution, which is incorporated into the initial population of a genetic algorithm ($GenTA$). This incorporation significantly expedites convergence to a highly optimal solution.

Furthermore, the super individual/local optima problems were avoided through diverse task re-assignment mutation and CM-PMX crossover operations with high probabilities. These operations enable the $GrdGenTA$ to exploit the global search capabilities of the genetic algorithm. This global search was evident throughout Section 3.3, in which the solutions from the $GrdGenTA$ were often significantly different than the initial $GrdTA$ solution while having a lower global objective function value.

Overall, this $GrdGenTA$ algorithm efficiently generates constraint-compliant and highly optimal task allocations. It possesses the following characteristics:

- Optimized: minimizes a global objective function while avoiding local optima.
- Computationally efficient: generates an optimized solution with minimal or reasonable computation time.
- Multi-factor: adapts the task allocation to comply with constraints imposed by operational factors of agent-task suitability, inter-task prerequisites and work capacity.

Original contributions from this chapter are as follows:

- Consideration of operational factors in task allocation (inter-task dependencies, suitability, task actuation cost and work capacity) as optimization constraints.
- Formulation of task debit and local objective function for a deterministic task allocation whose optimality is disproportionate relative to its computation time.
- Generalization of cross-route mutation and partially mapped crossover (PMX) operations from TSP/MTSP scenarios to solve MRTA problems. This generalization entailed modifying the original algorithms to consider a set of multiple tours (task sequences) and optimization constraints.
- Combination of greedy and genetic algorithms for optimized task allocation with significantly reduced computational demand.

Once the tasks are allocated, the robots must move to and execute their assigned tasks in the shared workspace while avoiding collision with each other. Task execution entails moving to the task location ($\Delta s_{i,j}$ component) and executing the required movements (c_j component) at that location. The mathematical models regarding task execution for a proposed implementation involving UAVs and UGVs are described in Chapter 4.

Additionally, multiple robots will execute these tasks at the same time in a shared workspace. This thesis implements reciprocal velocity obstacles (Appendix A3) to generate optimal collision-free velocities v_i^{opt} to drive robots to goal poses in the execution of their tasks while avoiding collision with each other. The conversion from optimal velocity v_i^{opt} to control input v_i^{in} to drive the robots is also described in Chapter 4.

Chapter 4 – Action Model for Mobile Robots

This chapter discusses the mathematical models that govern a multi-robot team's execution of tasks assigned by the task coordination framework (TCF). The considered implementation consists of a team of quadcopters and a team of differential drive robots (DDR). The quadcopters are equipped with a downwards-facing camera and perform aerial observation tasks. The differential drive robots perform ground patrol and simulated delivery/sensing tasks.

Section 4.1 discusses the kinematic models that convert optimal collision-free velocity v_i^{opt} from reciprocal velocity obstacles (RVO detailed in Appendix A3) into control inputs v_i^{in} to drive robots to their goal poses. Section 4.2 discusses the path generation required for the execution of the proposed tasks. Section 4.3 discusses the localization of visually detected points of interest from aerial observation tasks.

4.1 – Kinematic Model

Pose and motion for the multi-robot system are described in terms of a global frame, robots' initialization frames and robots' body frames. These reference frames and their homogeneous transformations [104] are shown in Figure 4.1.

- Global frame, RF_G: The global frame is the reference frame corresponding to the workspace shared by all robots.

- Robot initialization frame, RF_L: Each robot establishes its own initialization frame and uses onboard sensors for its pose estimate relative to RF_L. The origin of a robot's initialization frame is the pose at which the robot was powered up. The x, y and z directions of a robot's initialization frame are the forwards, left, and up directions, respectively, at the moment it was powered up.

- Robot body frame, RF_r: Each robot has a body frame whose origin is the robot's current pose. The x, y and z directions of the body frame correspond to the robot's current forwards, left, up directions, respectively. Control inputs are in reference to RF_r.

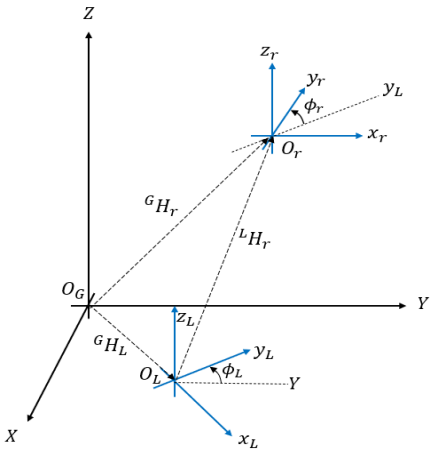


Figure 4.1 – Reference Frames and Coordinate Transformations

Figure 4.1 has a robot that was powered up at O_L and is hovering at O_r. The translation and rotation of the robot from the global frame origin O_G to the initialization frame origin O_L is the homogeneous transformation ${}^{G}H_{L}$. The translation and rotation of the robot from the initialization frame to the body frame is the homogeneous transformation ${}^{L}H_{R}$ which is calculated from the robot's pose estimate measured from its onboard sensors.

Given a pose $S = [x \quad y \quad z \quad \theta \quad \varphi \quad \phi]^T$, we define a homogeneous transformation (4.1) as the translation and rotation to that pose in where θ, φ, ϕ are roll, pitch, yaw, or rotations along the x, y, z axes, respectively.

$$\boldsymbol{HT}(S) = \begin{bmatrix} cos\varphi\ cos\phi & sin\theta\ sin\varphi\ cos\phi - cos\theta\ sin\phi & cos\theta\ sin\varphi\ cos\phi + sin\theta\ sin\phi & x \\ cos\varphi\ sin\phi & sin\theta\ sin\varphi\ sin\phi + cos\theta\ cos\phi & cos\theta\ sin\varphi\ sin\phi - sin\theta\ cos\phi & y \\ -sin\varphi & sin\theta\ cos\varphi & cos\theta\ cos\varphi & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$\boldsymbol{P}(\boldsymbol{HT}(S)) = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\boldsymbol{R}(\boldsymbol{HT}(S)) = \begin{bmatrix} cos\varphi\ cos\phi & sin\theta\ sin\varphi\ cos\phi - cos\theta\ sin\phi & cos\theta\ sin\varphi\ cos\phi + sin\theta\ sin\phi \\ cos\varphi\ sin\phi & sin\theta\ sin\varphi\ sin\phi + cos\theta\ cos\phi & cos\theta\ sin\varphi\ sin\phi - sin\theta\ cos\phi \\ -sin\varphi & sin\theta\ cos\varphi & cos\theta\ cos\varphi \end{bmatrix}$$

Given a robotic agent, S_0^G is the robot's pose in the global frame when it was powered up. Therefore, the homogeneous transformation from the global frame to its initialization frame of that robot is ${}^{G}H_{L}$:

$$S_0^G = [x_0^G \quad y_0^G \quad z_0^G \quad \theta_0^G \quad \varphi_0^G \quad \phi_0^G]^T \tag{4.2}$$
$${}^{G}H_{L} = \boldsymbol{HT}\left(S_0^G\right)$$

similarly, we define ${}^{L}H_{R}$ as follows in (4.3) where S^r is the pose estimate from robot's onboard sensors in reference to its initialization frame, RF_L.

$$S^r = [x^r \quad y^r \quad z^r \quad \theta^r \quad \varphi^r \quad \phi^r]^T \tag{4.3}$$
$${}^{L}H_{r} = \boldsymbol{HT}(S^r)$$

Therefore, the homogeneous transformation representing the robot's current pose in reference to the global frame is ${}^{G}H_{r}$ (4.4).

$${}^{G}H_{r} = {}^{G}H_{L}\ {}^{L}H_{r} \tag{4.4}$$

The conversion from optimal collision-free velocity v_i^{opt} to control input v_i^{in} is facilitated through the kinematic models of each robot combined with closed-loop PID-like control structures. These models describe the controllable states and the inputs to change those states.

4.1.1 – Quadcopter Kinematic Model

The relevant states S (4.5) for the quadcopter are its motion in the x, y, z directions and yaw ϕ, as well as those velocities $\dot{S}$ (4.6). For the purposes of aerial observation, we are not concerned with roll $(\theta = 0)$ and pitch $(\varphi = 0)$. The quadcopter's controller can drive S and $\dot{S}$ to desired values while maintaining values of θ, $\dot{\theta}$, φ and $\dot{\varphi}$ that are not detrimental to the aerial observation tasks. This controller prevents abrupt changes in roll and pitch, which may disrupt the imagery recorded from the drone's embedded camera.

$$S = [x \quad y \quad z \quad \phi]^T \tag{4.5}$$

$$\dot{S} = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\phi}]^T \tag{4.6}$$

For flying the drone, the control input v_i^{in} consists of forward speed v_x, lateral speed v_y, vertical speed v_z and rotation rate ω about the vertical axis (4.7). The quadcopter's velocity can also be described in terms of this control input (4.8). Its relevant states and control inputs are shown in Figure 4.2.

$$v_i^{in} = [v_x \quad v_y \quad v_z \quad \omega]^T \tag{4.7}$$

$$\dot{S} = \begin{bmatrix} v_x cos\phi + v_y sin\phi \\ v_x sin\phi + v_y cos\phi \\ v_z \\ \omega \end{bmatrix} \tag{4.8}$$

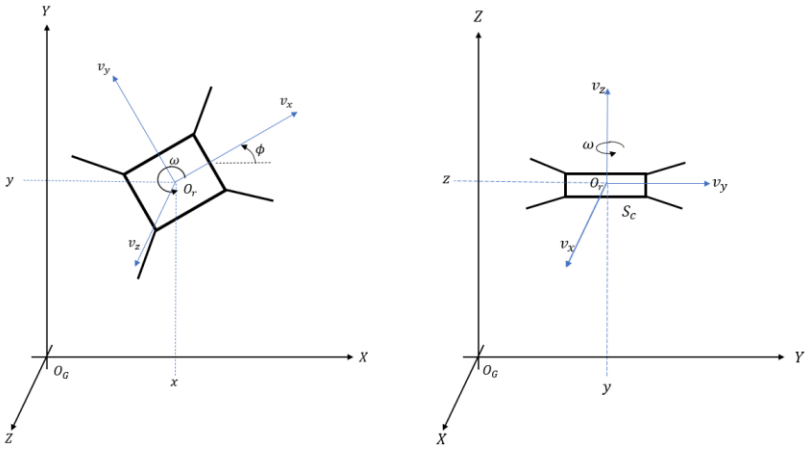


Figure 4.2 – Quadcopter Kinematic Model (Illustrated from Above and Lateral Views)

The kinematic model described in (4.5) to (4.8) and Figure 4.2 is a simplified kinematic model of a quadcopter. A quadcopter can freely translate along x, y, z in any orientation ϕ but in reality a quadcopter's horizontal translation is coupled with roll and pitch angles; the complete kinematic model considering these orientations is nonholonomic [104]. However, the simplified model is holonomic, as roll and pitch are ignored.

We use the kinematic model along with the optimal velocity v_i^{opt} from the motion planner to calculate v_x and v_y for the quadcopter. While the RVO motion planner can be used for 3D motion, this thesis will only use it for 2D motion. Separate kinematic controllers are formulated to calculate v_z and ω to drive the quadcopter from its current pose $S_c = [x_c \quad y_c \quad z_c \quad \phi_c]^T$ to its goal pose $S_d = [x_d \quad y_d \quad z_d \quad \phi_d]^T$. These poses and optimal velocity are shown in Figure 4.3.

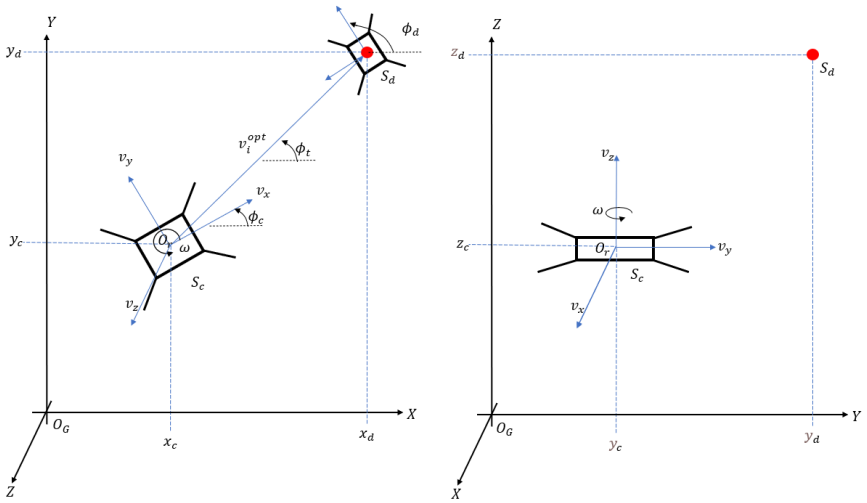


Figure 4.3 – Current and Goal Pose for Quadcopter (Illustrated from Above and Lateral Views)

We define the pose error and yaw error in (4.9) and (4.10), where (x_d, y_d, z_d) is desired position, (x_c, y_c, z_c) is current position, ϕ_t is direction of travel, ϕ_c is current yaw and ϕ_d is desired yaw. These variables are shown on Figure 4.3. This figure also assumes that the direction of travel from v_i^{opt} is a straight-line path towards the goal pose, $v_i^{opt} = v_i^{id}$. The presence of obstacles would change the direction of v_i^{opt}, but the conversion from v_i^{opt} to v_i^{in} would remain identical.

$$\Delta S = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \phi \end{bmatrix} = \begin{bmatrix} x_d - x_c \\ y_d - y_c \\ z_d - z_d \\ \phi_t - \phi_c \end{bmatrix} \tag{4.9}$$

$$\Delta \phi_h = \phi_d - \phi_c \tag{4.10}$$

$$\phi_t = atan2(\dot{y}_{opt}, \dot{x}_{opt}) \tag{4.11}$$

$$if\ v_i^{opt} = v_i^{id}, then\ \ \phi_t = atan2(y_d - y_c, x_d - x_c)$$

Kinematic control for v_z and ω is facilitated via proportional control where v_v and ω_{max} are maximum allowable vertical speed and yaw rate, respectively. For v_z (4.12), as the distance $|\Delta z|$ between the goal height and current height increases, the vertical speed will increase until it is equal to v_v. v_v is multiplied by $\Delta z/|\Delta z|$ to ensure proper sign and therefore correct direction (up or down) of travel. If the vertical distance is less than the threshold distance $v_v * \Delta t_v$, then v_z is set to $\Delta z\ /\Delta t_v$ so that the vertical speed decreases as the error decreases, limiting

overshoot. Δt_v is a constant parameter representing a time interval. If the vertical distance exceeds the threshold, the v_z is set to the maximum allowable vertical speed v_v.

$$v_z = \begin{cases} v_v * \Delta z/|\Delta z|, & |\Delta z| > v_v * \Delta t_v \\ \Delta z\ /\Delta t_v & |\Delta z| \leq v_v * \Delta t_v \end{cases} \tag{4.12}$$

For ω, we use the sine of $\Delta\phi_h$ to scale the rotation rate from 0 to ω_{max} and determine whether the rotation is clockwise or counter-clockwise (4.13). If $\Delta\phi_h = 0$, then $\omega = 0$; the rotation rate decreases smoothly as $\Delta\phi_h$ decreases.

$$\omega = \omega_{max} * \sin(\Delta\phi_h) \tag{4.13}$$

The quadcopter can freely adjust z and ϕ independently of x and y. The preferred motion is for the quadcopter to orient itself towards ϕ_d and translate towards x_d, y_d, z_d. As detailed in Appendix A3, the motion planner calculates $v_i^{opt} = [\dot{x}_{opt} \quad \dot{y}_{opt}]^T$ to drive (x_c, y_c) to (x_d, y_d) in a collision-free manner.

Calculate desired horizontal speed.

$$v_{xy} = \sqrt{\dot{y}_{opt}{}^2 + \dot{x}_{opt}{}^2} \tag{4.14}$$

Convert to control inputs.

$$v_x = v_{xy} * \cos(\Delta\phi) \tag{4.15}$$

$$v_y = v_{xy} * sin(\Delta\phi) \tag{4.16}$$

Therefore, we have the control inputs for the quadcopter v_x and v_y expressed in (4.15) and (4.16), as well as v_z and ω expressed in (4.12) and (4.13). For v_x and v_y, the quadcopter is horizontally translated as per v_i^{opt}. The sine and cosine multiplying v_{xy} adjusts the forwards and lateral speed components of overall velocity such that quadcopter's yaw angle ϕ_c does not affect the direction or translation speed of travel. The direction of travel is maintained as ϕ_t regardless of ϕ_c.

4.1.2 – Ground-based Differential Drive Robot Kinematic Model

The UGVs considered for this thesis use the DDR [105] kinematic model. Its relevant states S are its motion in the x, y directions and yaw ϕ, as well as those velocities $\dot{S}$, which are denoted in (4.17) and (4.18).

$$S = [x \quad y \quad \phi]^T \tag{4.17}$$

$$\dot{S} = [\dot{x} \quad \dot{y} \quad \dot{\phi}]^T \tag{4.18}$$

The control inputs are its forward speed v and yaw rate ω, as described in (4.19).

$$v_i^{in} = [v \quad \omega]^T \tag{4.19}$$

Therefore, the robot's velocity expressed in terms of control inputs is expressed in (4.20). The relevant states and control inputs are shown in Figure 4.4.

$$\dot{S} = \begin{bmatrix} v\,cos\phi \\ v\,sin\phi \\ \omega \end{bmatrix} \tag{4.20}$$

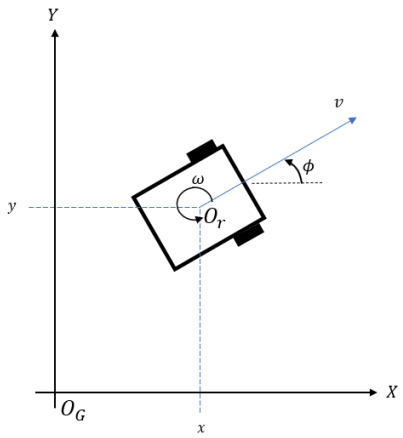


Figure 4.4 – Differential Drive Robot Kinematic Model

Conversion from v_i^{opt} to v_i^{in} follows a similar process as the quadcopter but is simpler due to the more limited nonholonomic kinematic model [104]. This robot can only translate in its present orientation's forward or backward directions. We define the yaw error in (4.22), where ϕ_t is desired direction of translation and ϕ_c is current yaw. We define the magnitude of desired forwards speed as the magnitude of optimal velocity v_i^{opt}. These poses and optimal velocity are shown on Figure 4.5.

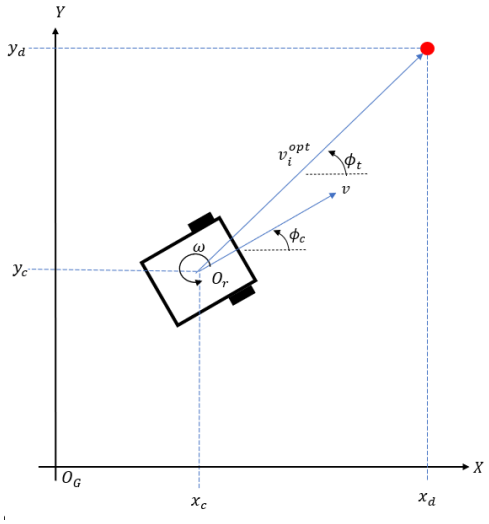


Figure 4.5 – Current and Goal Pose for Differential Drive Robot

Calculate travel heading ϕ_t (direction of v_i^{opt}) and heading error $\Delta\phi$.

$$\phi_t = atan2(\dot{y}_{opt}, \dot{x}_{opt}) \tag{4.21}$$

$$\Delta\phi = \phi_t - \phi_c \tag{4.22}$$

Calculate desired speed.

$$v_d = \sqrt{\dot{y}_{opt}{}^2 + \dot{x}_{opt}{}^2} \tag{4.23}$$

For ω, we use the sine of $\Delta\phi$ to scale the rotation rate from 0 to ω_{max} and determine whether the rotation is clockwise or counter-clockwise. If $\Delta\phi = 0$, then $\omega = 0$; the rotation rate decreases smoothly as $\Delta\phi$ decreases. The desired speed v is multiplied by a factor of $\cos(\Delta\phi)$ to adjust it as the heading changes. If heading error $\Delta\phi$ increases, the rotation rate increases while the forward speed decreases, enabling the DDR to correct its trajectory. If $\Delta\phi$ decreases, the DDR will move faster in that direction while having a lower ω to maintain that direction. Therefore, the control inputs are described in (4.24) and (4.25).

$$\omega = \omega_{max}\sin(\Delta\phi) \tag{4.24}$$

$$v = v_d\cos(\Delta\phi) \tag{4.25}$$

4.2 – Path Generation for Task Execution

The four tasks proposed for proof of concept of the task coordination framework are listed in Table 4.1 and formulated as follows: 1) The stationary observation task commands the UAV to hover at a goal pose S_d and height of h to observe an area centred at (x_s, y_s) of width w_c facing a heading of ϕ_d as per Figure 4.6a. 2) The raster search observation task commands the UAV to fly in a raster pattern to provide coverage over a box centred at (x_s, y_s) as per Figure 4.7a. This box has a side length L and the raster pattern is flown at a height of h. 3) The ground patrol task commands a UGV to drive a square pattern of length L centred at (x_s, y_s). 4) The go-to-goal task commands the robot to move to the target position, with $z_d = 0$ if the robot is a UGV.

Table 4.1 – Task Representations

Task	**Inputs**
Stationary observation	• Observation area center (x_s, y_s) • Search height (h) • Observation area width (w_c) • Viewing angle (ϕ_d)
Raster Search Observation	• Observation area center (x_s, y_s) • Search height (h) • Search area length (L)
Ground Patrol	• Patrol area center (x_s, y_s) • Patrol area side Length (L)
Go-to-Goal	• Target Position (x_d, y_d, z_d)

Recall from (3.1) where each task has a location (x_j, y_j) and actuation cost c_j. For both observation tasks and the ground patrol task, the task location (x_j, y_j) corresponds to the observation area or patrol area center (x_s, y_s). For the go-to-goal task, the task location (x_j, y_j) corresponds to the target position (x_d, y_d). The task actuation cost c_j is modelled to represent the distance travelled in the execution of each task in Table 4.1.

The geometry of the stationary observation task is shown in Figure 4.6, depicting the drone's pose as well as relevant angles relative to the observation area center (x_s, y_s).

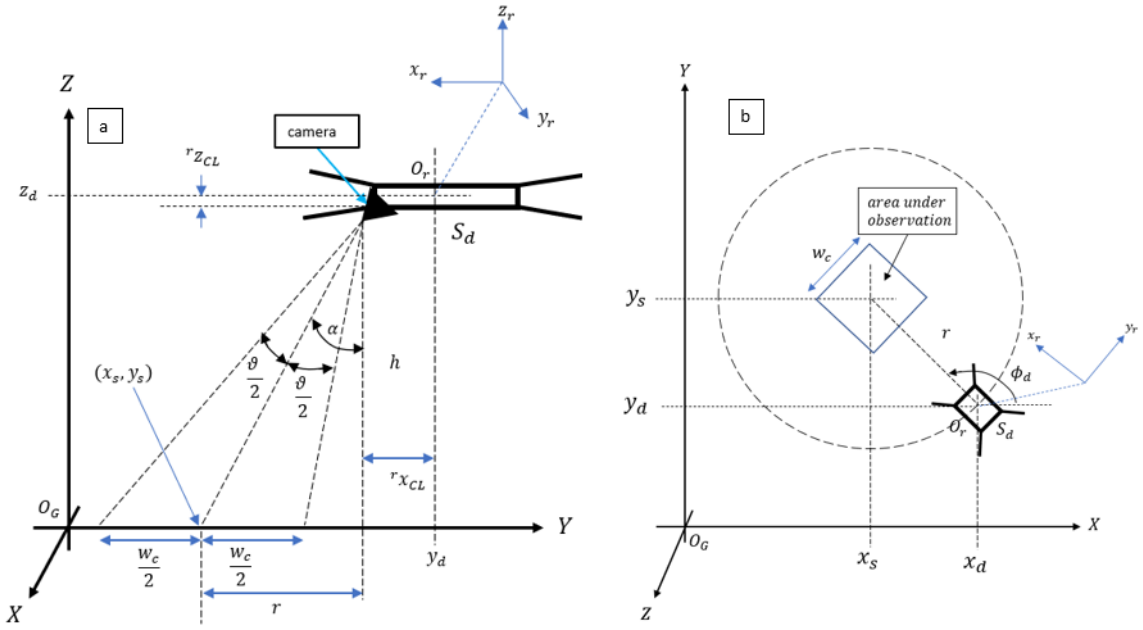


Figure 4.6 – Stationary Observation Task

From Figure 4.6a, the camera's angular field of view is ϑ and the rotation of the camera along the drone's y_r axis is $\frac{\pi}{2} - \alpha$. These parameters, combined with the user inputs, are used to calculate the drone's goal pose to observe the target area. The observation distance, r, and observation height, h, are calculated using equations (4.26) and (4.28), which are derived from the right-angle triangle relations from the geometry of the stationary observation task.

The tangent of an angle is equal to the opposite over the adjacent side. For the angle α, associated with the camera's orientation, the opposite side is the observation distance r and the adjacent side is the observation height h. For the angle $\alpha - \frac{\vartheta}{2}$, corresponding to the lower bound of the camera angle observation, the opposite side is the observation distance subtracted by half the observation width. In contrast, the adjacent side remains as the observation height h.

$$\tan(\alpha) = \frac{r}{h} \tag{4.26}$$

$$\tan\left(\alpha - \frac{\vartheta}{2}\right) = \frac{r - w_c/2}{h} \tag{4.27}$$

Combining (4.26) and (4.27) yields the observation height h in (4.28).

$$h = \frac{w_c}{2(\tan(\alpha) - \tan\left(\alpha - \frac{\vartheta}{2}\right))} \tag{4.28}$$

These values of h and r are then used to calculate the goal pose S_d for the UAV to enable observation of the target area (4.29), with h calculated from (4.28), r calculated from (4.26). ${}^{r}x_{CL}$, and ${}^{r}z_{CL}$ are the longitudinal, and vertical offsets, respectively, between the position of the camera and the UAV's position. The UAVs' considered in this thesis have no lateral offset for their embedded camera. These quantities originate from the camera's extrinsic properties (discussed in section 4.3.1). They are the physical dimensions of the UAV, which, for this thesis, are described in Appendix A4.

$$S_d = \begin{bmatrix} x_d \\ y_d \\ z_d \\ \phi_d \end{bmatrix} = \begin{bmatrix} x_s + (r + {}^{r}x_{CL}) * \cos(\phi_d) \\ y_s + r * \sin(\phi_d) \\ h + {}^{r}z_{CL} \\ \phi_d \end{bmatrix} \tag{4.29}$$

Therefore, (4.29) describes the goal pose S_d required to observe the target location (x_s, y_s), with an observation width of w_c from a direction of ϕ_d. The values of x_d and y_d are derived from the target location, adjusted along a circle of radius r, as shown on Figure 4.6b, calculated based on camera observation geometry and desired direction of observation ϕ_d.

The desired direction of observation ϕ_d could be set to any desired angle to achieve specific points of view. However, by default, ϕ_d would be set as the heading from the drone's current pose to the observation area center (4.30), where (x_s, y_s) and (x_c, y_c) are the observation area center and drone's x, y position, respectively.

$$\phi_d = atan2(y_s - y_c, x_s - x_c) \tag{4.30}$$

The actuation cost c_j of the stationary aerial observation and go-to-goal tasks is considered to be 0 as no additional movement is required to execute these tasks.

The raster search observation task defines a raster flight pattern for the UAV to observe a square-shaped area. The drone flies from one end of the square to the other until the entire search area is scanned, as shown in Figure 4.7a. The search area is centred on (x_s, y_s) and has side length of L. The width between passes is w_s and is calculated based on sensor's field of view width and input search height $z_d = h$. In Figure 4.7b, ϑ and α have the same meaning as in Figure 4.6a, corresponding to angular camera field of view and downwards camera orientation, respectively. w_c is the camera field of view, which is greater than w_s, to ensure complete observation over the search area.

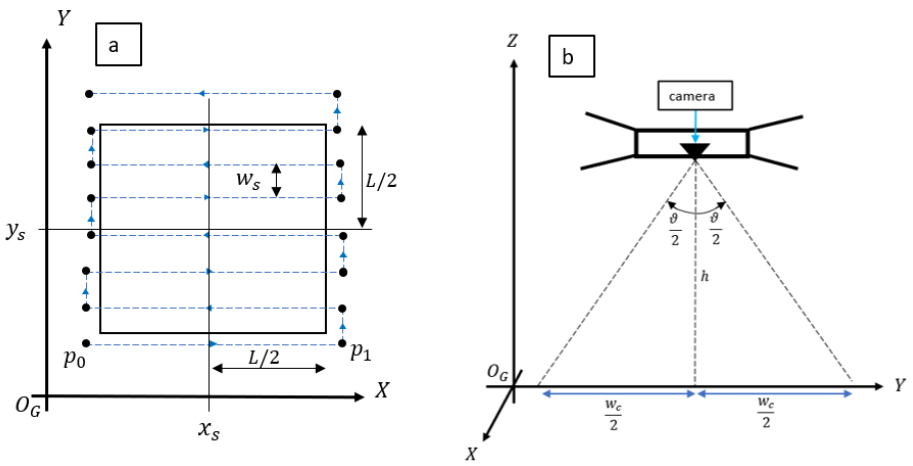


Figure 4.7 – Raster Search Observation Task

Calculation of scan width w_s is defined in (4.31) where k is a scaling factor to ensure the scan width w_s is less than the camera's field of view width w_c obtained from (4.28). A scaling factor of $k < 1$ ensures that there are no gaps in observation while flying the search pattern.

$$w_s = k * w_c \tag{4.31}$$

A raster pattern generation algorithm creates a list of N waypoints $W = \{p_i | i = 0,1,2, \dots N\}$ for the UAV to travel for the raster search observation task. It first initializes the first two points as $p_0 = (x_s - L/2, y_s - L/2, h)$, $p_1 = (x_s + L/2, y_s - L/2, h)$ and appends them to the list W. These two initial points correspond to the bottom left and bottom right waypoints on the search pattern in Figure 4.7a. Subsequent waypoints p_{N+1} are calculated based on the last two points in the list W [106].

If the last two y coordinates are the same, but x coordinates differ, it means those two positions correspond to a lateral line. The next waypoint needs to start a new lateral line, therefore p_{N+1} has the same x coordinate as p_N, but the y coordinate needs to be increased.

- If $x_N \neq x_{N-1}, y_N = y_{N-1}$:

$$p_{N+1} = (x_N, y_N + w_s, h) \tag{4.32}$$

If the last two x coordinates are the same, but y coordinates differ, it means those two positions correspond to a longitudinal line. The next waypoint needs to complete a vertical line; therefore, p_{+1} will have the same y coordinate as p_N, but the x coordinate will change based on whether $x_N > x_s$ or $x_N < x_s$.

- If $x_N = x_{N-1}, y_N \neq y_{N-1}, x_N > x_s$:

$$p_{N+1} = (x_s - L/2, y_N, h) \tag{4.33}$$

- If $x_N = x_{N-1}, y_N \neq y_{N-1}, x_N < x_s$:

$$p_{N+1} = (x_s + L/2, y_N, h) \tag{4.34}$$

These calculations are repeated with p_{N+1} being appended to W until the entire length and width of the square are covered: $x_N \geq x_s + L/2$ and $y_N \geq y_s + L/2$. The motion planner generates optimal velocities v_i^{opt} to drive the quadcopter through these waypoints, which are converted into control inputs v_i^{in} using the kinematic model in Section 4.1.1.

The actuation cost of the raster search task is described in (4.35). The distance travelled by each lateral pass is equal to L. The number of lateral passes conducted is approximately equal to L/w_s, where w_s is the distance between lateral passes (4.31). The longitudinal distance accrued by all transitions between lateral passes is approximately equal to L.

$$c_j = L * \left(\frac{L}{w_s}\right) + L \tag{4.35}$$

Finally, the ground patrol task entails driving in a square pattern centred on (x_s, y_s) with a side length of L as follows:

1) $(x_s - \frac{L}{2},\ y_s - \frac{L}{2})$ to $(x_s - \frac{L}{2},\ y_s + \frac{L}{2})$,
2) $(x_s - \frac{L}{2},\ y_s + \frac{L}{2})$ to $(x_s + \frac{L}{2},\ y_s + \frac{L}{2})$,
3) $\left(x_s + \frac{L}{2},\ y_s + \frac{L}{2}\right)$ to $(x_s + \frac{L}{2},\ y_s - \frac{L}{2})$, and
4) $(x_s + \frac{L}{2},\ y_s - \frac{L}{2})$ to $(x_s - \frac{L}{2},\ y_s - \frac{L}{2})$.

The actuation cost of this task is $c_j = 4L$, which is the square's perimeter.

4.3 – Points of Interest Localization from Aerial Observation

In order to perform aerial observation, the UAV executes movements per the goal poses of its assigned tasks. During these maneuvers, the UAV's downwards-facing camera visually detects points of interest (POIs). At each camera image, the pixel coordinates (p_x, p_y) of the detected POI are extracted using image classification and the camera's intrinsic properties. These pixel coordinates are converted into a raw position estimate $(\hat{x}_t, \hat{y}_t)$ using the camera's extrinsic properties. Therefore, each camera image that detects a POI yields a raw position estimate $(\hat{x}_t, \hat{y}_t)$; which are aggregated into the list $\hat{P}$. The list $\hat{P}$ is then filtered through Kmeans clustering to obtain $\hat{T}^*$, the list of cluster centers, which are the estimated positions of the POIs. This process of localizing POIs is described in Figure 4.8.

For this thesis, it will be assumed that the number of clusters, n and consequently, the number of POIs is known. Autonomously calculating the number of POIs is beyond the scope of this research, as the proposed aerial observation tasks serve as a proof of concept for tasks that the TCF can allocate.

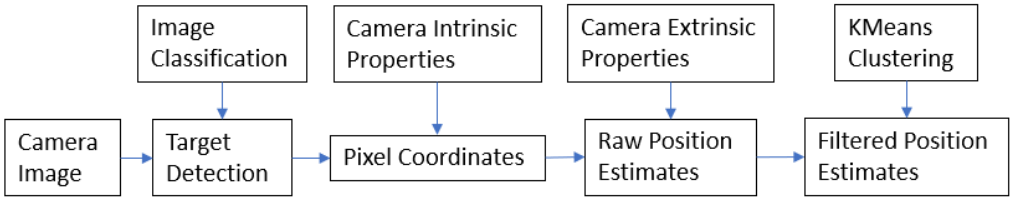


Figure 4.8 – Point of Interest Localization Process

This action model facilitates collaborative aerial search operations by describing aerial observation tasks that can be assigned to multiple UAVs. Furthermore, the aerial observation tasks estimate the positions of POIs, which can be the subject of tasks performed by another team of autonomous robots. An example would be a multi-UAV team finding locations for delivering supplies or sensor nodes by a multi-UGV team.

4.3.1 – Camera Extrinsic and Intrinsic Parameters

The extrinsic parameters describe the pose of the camera image frame relative to the robot body frame. In contrast, the intrinsic parameters describe the internal properties of the camera. These properties are shown in Figure 4.9.

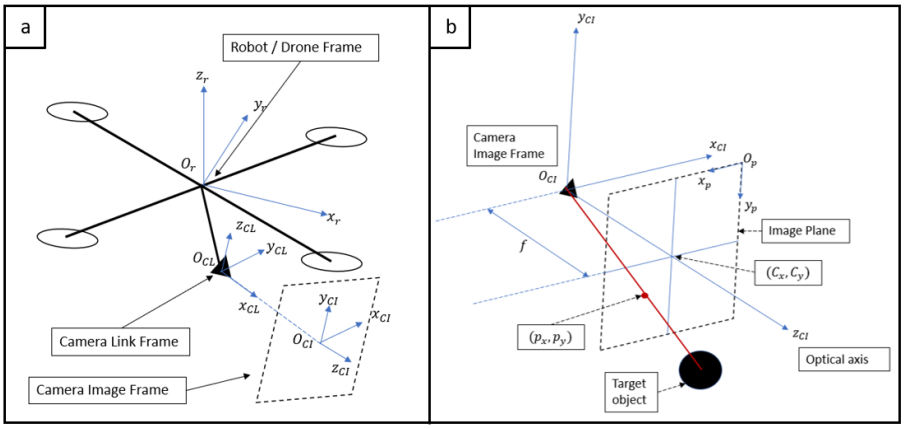


Figure 4.9 – Camera Properties: (a) Extrinsics, (b) Intrinsics

The robot body frame RF_r, corresponds to the robot's pose. RF_r in Figure 4.9a is the same as RF_r in Figure 4.1. The transformation from the global frame RF_G to RF_r was described in (4.4). The camera link frame RF_{CL} corresponds to the pose of the physical structure housing the camera relative to the drone. The camera image frame RF_{CI} describes the rotation relative to the camera link frame to match the coordinate system with the image plane of the camera. On the image plane, x and y correspond to horizontal and vertical axes, with the z axis corresponding to movement towards or away from the image plane.

Therefore, the rotation ${}^{G}R_{CI}$ and homogeneous transformation ${}^{G}H_{CI}$ from the global frame RF_G to the camera image frame RF_{CI} is described in equations (4.36) to (4.38).

$$
\begin{gathered}
{}^{G}R_{CI} = {}^{G}R_{r} * {}^{r}R_{CL} * {}^{CL}R_{CI} \\
{}^{G}R_{r} = rotation\ from\ world\ to\ drone \\
{}^{r}R_{CL} = rotation\ from\ drone\ to\ camera\ link \\
{}^{CL}R_{CI} = rotation\ from\ camera\ link\ to\ camera\ image
\end{gathered}
\tag{4.36}
$$

$$
\begin{gathered}
{}^{G}H_{CI} = {}^{G}H_{r} * {}^{r}H_{CL} * {}^{CL}H_{CI} \\
{}^{G}H_{r} = HT\ from\ global\ frame\ to\ drone \\
{}^{r}H_{CL} = HT\ from\ drone\ to\ camera\ link \\
{}^{CL}H_{CI} = HT\ from\ camera\ link\ to\ camera\ image
\end{gathered}
\tag{4.37}
$$

$$ {}^{G}H_{CI} = \begin{bmatrix} {}^{G}R_{CI} & & & {}^{G}x_{CI} \\ & & & {}^{G}y_{CI} \\ & & & {}^{G}z_{CI} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.38} $$

Using the pinhole camera model, the properties of interest are focal length f and location of the principal point $(C_{x,}\, C_y)$. These properties are shown in Figure 4.9b. The optical axis is the same line as z_{CI}; it is the direction of where the camera is facing and is pointed towards the principal point. The pixel coordinate system RF_p has its origin O_p at the top left corner of the camera image, with positive x and y directions pointed towards the right and down directions, respectively. The location of the principal point is expressed in terms of the pixel coordinate system. Within Figure 4.9b, there is also a detected target object whose center is projected onto the image plane. The location of the center of the target object is projected onto the image plane as (p_x, p_y), which is also expressed in terms of the pixel coordinate system.

4.3.2 – Target Detection and Localization Scheme

For this thesis, the observation tasks rely on Apriltags [107] and existing image classifier [108] to detect them. These Apriltags act as artificial landmarks representing points of interest (POIs). Detecting general POIs corresponding to specific objects would require an advanced image classification system, which is beyond the scope of this research. An example of an Apriltag is shown in Figure 4.10.

Figure 4.10 – Apriltag Example [109]

The image classifier draws bounding boxes on each detected Apriltag and extracts the pixel coordinates of the center of these bounding boxes. The detection algorithm relies on complex computer vision algorithms to process camera images. These algorithms would match the pattern of pixel intensities from the image with a digital representation of an Apriltag and then estimate the center of the corresponding pixel intensities. These Apriltags will be used as the target objects / POIs depicted in Figure 4.9b.

A simulated camera image is shown in Figure 4.11 with the pixel coordinate system RF_p, camera principal point (C_x, C_y) and pixel coordinates (p_x, p_y) of a detected Apriltag (corresponding to the center of its bounding box). These variables in Figure 4.11 have the same meaning as in Figure 4.9.

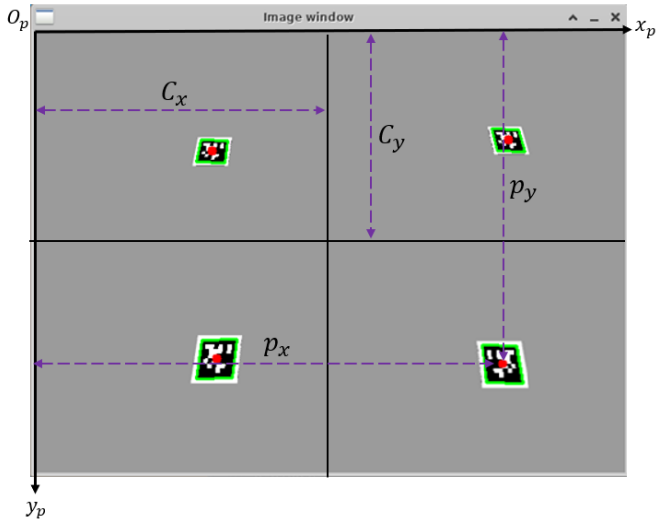


Figure 4.11 – Simulated Camera Image with Apriltags and Bounding Boxes

In the case of multiple detections, each pair of pixel coordinates is extracted individually and processed sequentially through individual camera images. Note that this detection requires the complete Apriltag to be visible; otherwise, it would not be detected. The number of passes in the raster search discussed in Section 4.2 can be increased to avoid missed detections from partial visibility. This increase in the number of passes is facilitated by selecting a scaling factor k to reduce the scan width w_s in (4.31).

The pixel coordinates of the center of the detection (p_x, p_y) are combined with intrinsic properties to obtain a unit vector, p_{poi}^{cam} (4.39). This unit vector represents the direction from the camera to the target object, expressed in terms of RF_{CI}. Converting to a unit vector is desirable as we only require info on the direction to the detection at this stage in the localization. As stated earlier, it is assumed that the detection size is known (or can be accurately estimated) or that the detected object is on the ground with a position of $z = 0$.

$$p_{poi}^{cam} = \begin{bmatrix} p_x - C_x \\ p_y - C_y \\ f \end{bmatrix} * \frac{1}{\sqrt{(p_x - C_x)^2 + (p_y - C_y)^2 + f^2}} \tag{4.39}$$

In order to estimate the position of the target in the global frame, RF_G, the unit vector p_{poi}^{cam} must be converted in terms of the global frame using ${}^{G}R_{CI}$ (4.36) from the extrinsic properties. The direction from the camera to the target p_{poi}^{G} is described in (4.40).

$$p_{poi}^{G} = {}^{G}R_{CI} * p_{poi}^{cam} = \begin{bmatrix} \Delta x_{poi}^{G} \\ \Delta y_{poi}^{G} \\ \Delta z_{poi}^{G} \end{bmatrix} \tag{4.40}$$

Equation (4.40) is then combined with (4.38) to yield the equation of a line intersecting with both the origin of the camera image frame O_{CI} and the target at $(x_j, y_j, 0)$ (4.41). This line is shown on Figure 4.12 where we see the components ${}^{G}P_{CI}$ and $k * p^{G}_{poi}$ of that line, l.

$$l = {}^{G}P_{CI} + k * p^{G}_{poi} \tag{4.41}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} {}^{G}x_{CI} \\ {}^{G}y_{CI} \\ {}^{G}z_{CI} \end{bmatrix} + k \begin{bmatrix} \Delta x^{G}_{poi} \\ \Delta y^{G}_{poi} \\ \Delta z^{G}_{poi} \end{bmatrix}$$

Where ${}^{G}P_{CI}$ is the position of the origin of the camera image frame in terms of global frame calculated from (4.38), p^{G}_{poi} is from (4.40) and k is a scaling factor equal to physical distance from the camera to the target. Therefore, $k * p^{G}_{poi}$ can be considered as the complete vector (magnitude and direction) from the camera to the target, expressed in the global frame. Finding the target's location in 3D space entails finding the value of k such that the coordinates in (4.41) match the target's location.

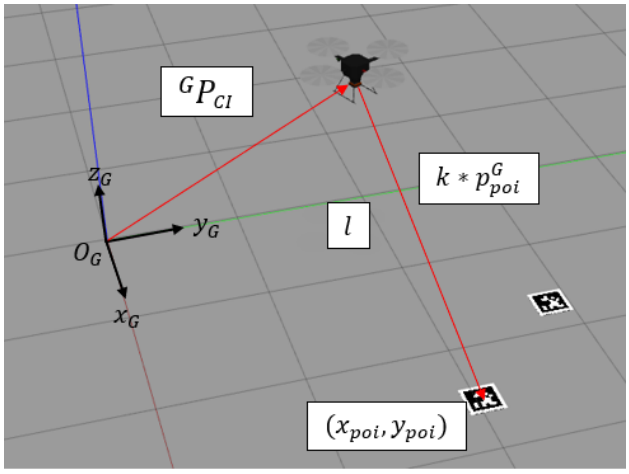


Figure 4.12 – UAV and POI Transformations in the Global Frame

Each t^{th} Apriltag detection yields a pair of pixel coordinates that is combined with both intrinsic and extrinsic properties to yield a raw estimate of the POI's position $(\hat{x}_t, \hat{y}_t)$, denoted in (4.43). For experimentation purposes, it can be assumed that the target is on the ground and that the ground is a flat surface. Therefore, on (4.41), we can set $z = 0$ and solve for k that satisfies this condition, yielding $k = k_{z0}$ (4.42).

$$k_{z0} = \frac{-{}^{G}z_{CI}}{\Delta z^{G}_{poi}} \tag{4.42}$$

$$\begin{bmatrix} \hat{x}_t \\ \hat{y}_t \\ 0 \end{bmatrix} = \begin{bmatrix} {}^{G}x_{CI} \\ {}^{G}y_{CI} \\ {}^{G}z_{CI} \end{bmatrix} + \frac{-{}^{G}z_{CI}}{\Delta z^{G}_{poi}} \begin{bmatrix} \Delta x^{G}_{poi} \\ \Delta y^{G}_{poi} \\ \Delta z^{G}_{poi} \end{bmatrix} \tag{4.43}$$

This approach only works if the surface is perfectly flat, or at the very least if both the surface from which the UAV took off (associated with RF_L in Section 4.1), and the surfaces at the POIs are at the same height and treated as $z = 0$. If this is not the case, we must use the estimated size and pose of the Apriltag to calculate k.

If the size of a detection is known, then the distance between it and the camera can be calculated, like in [110], using (4.44). L_R is real life length of the detected object, f is camera focal length and L_p is length of detection in number of pixels on image.

$$k = \frac{L_r * f}{L_p} \tag{4.44}$$

When this value of k is substituted into (4.41), we have the estimated 3D coordinates of the POI in the global frame without assuming $z = 0$, shown on (4.45).

$$\begin{bmatrix} \hat{x}_t \\ \hat{y}_t \\ \hat{z}_t \end{bmatrix} = \begin{bmatrix} {}^{G}x_{CI} \\ {}^{G}y_{CI} \\ {}^{G}z_{CI} \end{bmatrix} + \frac{L_r * f}{L_p} \begin{bmatrix} \Delta x^{G}_{poi} \\ \Delta y^{G}_{poi} \\ \Delta z^{G}_{poi} \end{bmatrix} \tag{4.45}$$

The challenge with this method is matching the features of the target such that the L_R and L_p correspond to the same line segment on the target. A proposed feature-matching approach is shown in Figure 4.13.

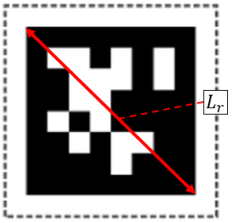


Figure 4.13 – Apriltag with Diagonal Cross-section

The Apriltag is a square whose dimensions are known, so L_r is also known. The camera image in Figure 4.11 shows each Apriltag detected with a bounding box. The pixel coordinates of each corner of the bounding box can be used to calculate L_p.

4.3.3 – Data Filtering

The observation task combined with the position estimate described in Sections 4.3.1 and 4.3.2 generates a list $\hat{P}$ with K observations $(\hat{x}_t, \hat{y}_t)$, where each observation is a raw position estimate. This results in many noisy raw estimates that must be grouped into a set $\hat{T}$ containing n cluster centers $(\bar{x}_m, \bar{y}_m)$ representing the filtered estimates of the POIs' positions. Ideally, these cluster centers would be identical to the ground truth such that $(\bar{x}_m, \bar{y}_m) = (x_{poi}, y_{poi})$, as depicted in Figure 4.14.

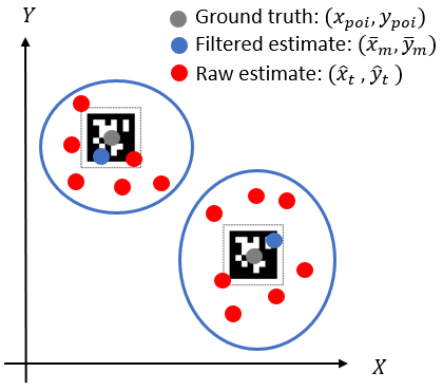


Figure 4.14 – POI Detection Clustering

Kmeans clustering [111] is the considered solution to group the observations $(\hat{x}_t, \hat{y}_t)$ into n clusters, each centered on $(\bar{x}_m, \bar{y}_m)$. The objective is to determine the list $\hat{T}^*$ of cluster centers such that the inertia, I, which is the cluster sum of squared error, is minimized. The objective function of Kmeans clustering is denoted in (4.46).

$$\hat{T}^* = \underset{\hat{T}}{argmin}\, I$$
$$I = \sum_{m=1}^{n} \sum_{t=1}^{K} (\hat{x}_t - \bar{x}_m)^2 + (\hat{y}_t - \bar{y}_m)^2 \tag{4.46}$$

4.4 – Summary

This chapter has formulated the mathematical models describing the actions to be taken by a UAV and UGV for a proposed implementation of the Task Coordination Framework (TCF). It proposed a kinematic model for quadcopters and differential drive robots and a description of the tasks that the TCF can assign to these robots. These models will be used to control the robots in simulations and experiments using the technologies described in Chapter 5.

Chapter 5 – Experimental Testbed Technical Description

This chapter describes the technologies used for the simulation and experimentation of the task coordination framework (TCF) and its implementation on mobile robots. The system consists of a high-level controller and a multi-robot team, shown in Figure 5.1.

- The High-Level Controller consists of a user interface, mission planner, and global localization. It receives information on the pose S_i, and usage u_i of each robot, as well as user input, which is the list T of tasks to be performed. The high-level controller then implements the task allocation algorithm from Chapter 3 to assign task sequences TS_i to the robots.
- The robots would interpret each task $T_j^{i,k}$ from its assigned sequence TS_i, matching it with its capabilities and kinematic model to perform collision-free movements.

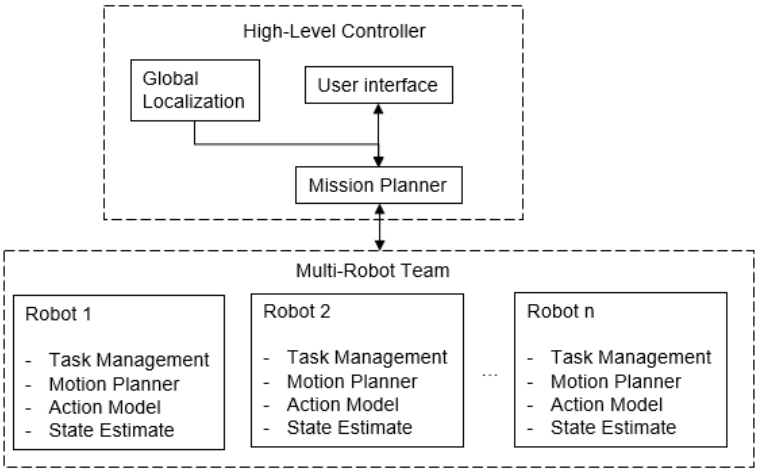


Figure 5.1 – System Architecture for Implementation

This thesis considered two multi-robot teams: a team of UAVs and a team of UGVs. The UAVs performed aerial observation tasks. The UGVs performed ground patrol and simulated delivery/sensing tasks. Each multi-robot team concurrently performed their assigned tasks within a shared working environment. The solutions presented in Chapters 3 and 4 were tested on both simulated and real robots.

5.1 – Test Environment

The test environment for experimentation on real robots is a motion capture environment that measures 10 m by 6 m by 4 m. The motion capture system consists of 21 cameras, infrared (IR) reflectors on robots, a router and supporting software from the manufacturer [112]. The cameras are distributed along the edges of a cage with a mesh net that forms the motion capture environment. The cameras are Prime X 13 cameras [113]. In this motion capture (MoCap) environment, a robot placed inside is shown in Figure 5.2, with some of the cameras visible and indicated with yellow boxes. The global frame RF_G as well as robot frame RF_r are also shown. Note that in Figure 5.2, $RF_r = RF_L$.

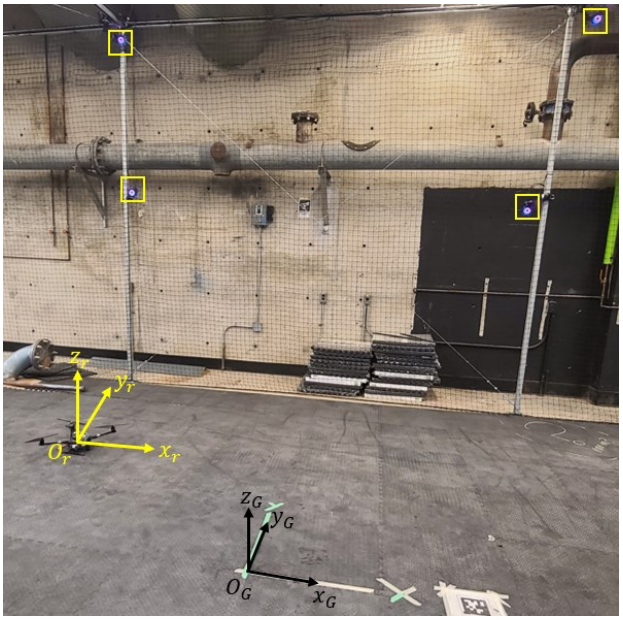


Figure 5.2 – Motion Capture Environment

The user interface for this motion capture environment is depicted in Figure 5.3, which shows the positions and orientations of all cameras within the environment. The yellow lines are pointing toward the IR reflectors on the RB5 drone. It can be observed that several cameras can see the IR reflectors on the RB5 drone when it is placed at $(x, y) = (-1.49, 1.08)$ relative to the origin of the global frame, O_G.

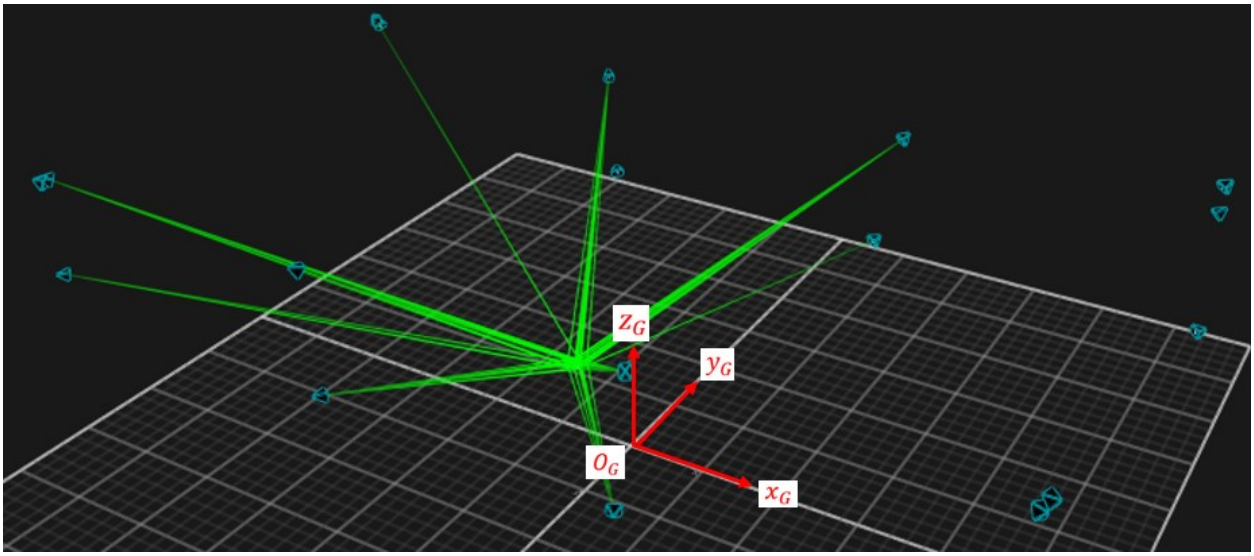


Figure 5.3 – Motion Capture User Interface

Each camera has an IR emitter as well as an IR receiver. The IR emissions from the cameras are reflected by IR reflectors on the robots. These reflections are then detected by the cameras' IR receivers and processed to estimate the positions of those IR reflectors within the motion capture environment. The cameras were placed along the perimeter and calibrated as per the manufacturer's documentation [114]. This calibration establishes the cameras' relative

positions and orientations as well as the position and orientation of the global frame's origin, O_G. With the pose of each camera known and the IR reflector detected, each detection of an IR marker can be converted into a position. The method for this conversion involves aggregating the data from multiple cameras with proprietary computer vision solutions developed by the manufacturer.

The arrangement of IR markers on the robots is shown in Figure 5.4, with Figure 5.4a corresponding to the RB5 drone and Figure 5.4b corresponding to the Wafflebot ground robot. The IR markers are grouped as a single rigid body, and the change in positions of IR markers is aggregated to calculate the change in pose of the overall rigid body. A minimum of 3 markers in an asymmetric plane are required to track the pose of a rigid body. However, each camera's line of sight is limited, and the robot's physical structure can obstruct some markers. Therefore, additional markers at different elevations are essential for continuous tracking.

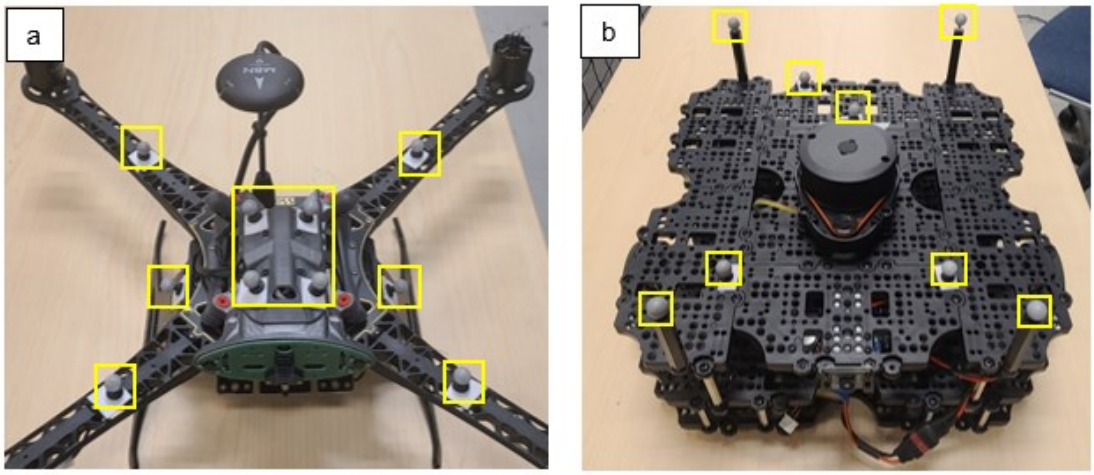


Figure 5.4 – Robots with Reflective IR Markers: a) RB5 Drone, and b) Wafflebot

The rigid body representation from IR markers on the RB5 drone is shown in Figure 5.5. This rigid body corresponds to the RB5 drone with its IR reflectors as per Figure 5.4a and is placed at O_r at $(x, y) = (-1.49,\ 1.08)$ in reference to RF_G.

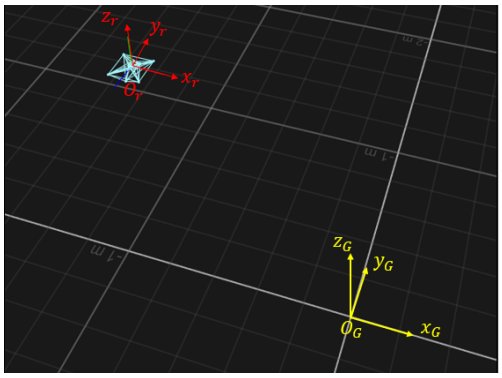


Figure 5.5 – Rigid Body Representation of IR Markers on RB5 Drone

5.2 – Control Station and Common Software

The High-Level Controller consists of a control station computer and motion capture system for global localization.The control station computer runs Ubuntu 20.04 with ROS Noetic [115] and Q Ground Control [116]. For experimentation, communication between the control station computer and robots was facilitated using a Wi-Fi hotspot from a cell phone for constant IP addresses.

The high-level controller runs the *multi_agent_coord* ROS package developed for this research. This package implements the task allocation algorithm (presented in Chapter 3), user interface and monitoring functions. The ROS nodes in *multi_agent_coord* are described in Table 5.1.

Table 5.1 – Control Station Scripts

Serial	Script	Description
1	*command_interface*	User interface. Allows the user to send commands to control a multi-robot system.
2	*system_monitor*	Reports and displays pose S_i, usage u_i, and availability a_i of each robot for the user.
3	*mission_planner*	Implementation of task allocation algorithm. Calculates TA_{opt} and publishes task sequences TS_i to the robots.

For simulation and experimentation, the list of tasks was saved as a .csv file, and the user input to the *command_interface* was the name of that file. The .csv files were encoded as per the lists of tasks in Appendix A1, such as Table A.3. The *mission_planner* generates the list T of tasks by parsing through the .csv file and generates the list R of robot descriptions through information (pose, usage, availability, status) received from robots. It then inputs R and T into the task allocation algorithm to generate an optimized task allocation TA using the $GrdTA$ or $GrdGenTA$ described in Chapter 3.

The implementation of the task allocation algorithms used matrix representations from graph theory. The scenario (robots and tasks) was represented by a cost matrix whose elements are the costs for the robot to perform tasks, analogous to the distance matrix containing distances between destinations in TSP / MTSP. Each robots' task sequences were represented as adjacency matrices. Each robot's cost was calculated as the sum of elements of the resultant matrix from elementwise multiplication between the robot's adjacency matrix [117] and the cost matrix.

Computer communications among different robots were facilitated through extensive use of namespace remapping in a similar manner to [118] and [119]. A hierarchical system for ROS topics compartmentalizes the information associated with specific robots, as each robot has several shared topics. This hierarchical structure with examples of shared topics is shown in Figure 5.6, where each robot has a namespace *r_i/,* from *i = 0* to *i = n.* Each robot in Figure 5.6 also has topics corresponding to control input (*cmd_vel),* goal pose *(WP)* and other ROS topics. Note that these namespaces were not "hard coded" into the software of the ROS packages.

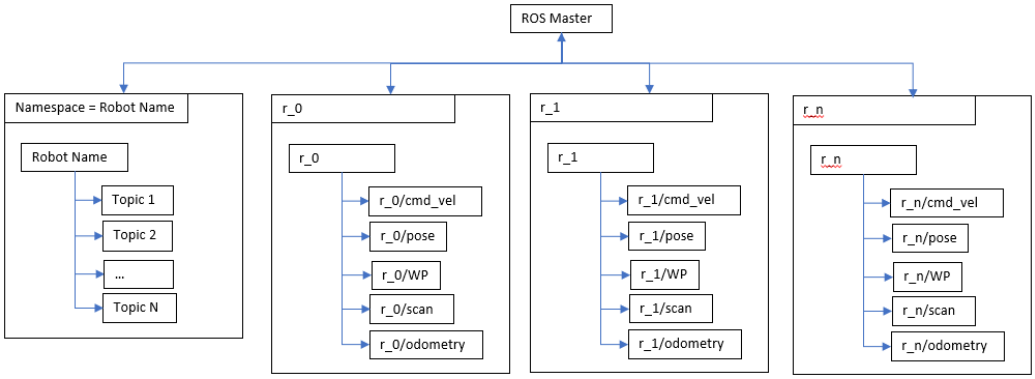


Figure 5.6 – ROS Topic Management Structure

The respective publishers and subscribers were dynamically instantiated. The ROS packages developed for this thesis were programmed to iterate through all the ROS topics, identify namespaces, and match those namespaces with the respective topics. The process of dynamic instantiation is described below and represented in Figure 5.7.

1) Initialize dictionaries for each shared ROS topic. Each dictionary's keys are the full ROS Topic name, and the entries are the ROS topics' publisher or subscriber, depending on the use case.
2) Enumerate all ROS Topics and save them as a list, which we will name *ALL_TOPICS_LIST.*
3) From *ALL_TOPICS_LIST*, identify all robot names from namespaces and create another list containing namespaces. This will be called *NAMESPACE_LIST*.
4) Match namespace with ROS Topic and publisher / subscribe. Iterate through *NAMESPACE_LIST.* For each entry in *NAMESPACE_LIST,* iterate through all shared topics and use string addition to generate the full ROS Topic and pass it as an argument for a ROS publisher or subscriber.
5) As a continuation of step 4, a dictionary is created whose key-value pairs consist of the ROS topic as the key and the publisher or subscriber of that topic as the value.

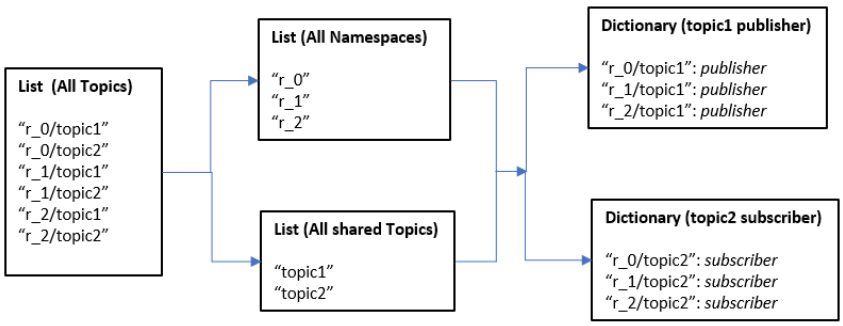


Figure 5.7 – Dynamic Instantiation of Publishers and Subscribers

Dynamic instantiation and namespace/topic matching enabled the High-Level Controller to command multi-robot teams of any size and type. The only requirement was prior knowledge of each robot's ROS topics. If each robot has the same ROS topics, as shown in Figure 5.7, then the above steps do not require adjustment. However, if different robots have different ROS

topics, additional logic must be implemented to differentiate them. This logic would be implemented within steps 3 and 4.

This dynamic instantiation and namespace/topic matching were applied to the UGV and UAV's ROS packages. The roslaunch file for the UGV and UAV would dictate the namespace(s). The underlying logic for those robots would automatically detect it, record it, and prepend it to its ROS topics. This automatic detection and prepending ensured that the robot's ROS topics matched the expectations of the high-level controller.

5.3 – Robots Description

The robots considered for simulation and experimentation are as follows:

- Hector Quadcopter (Gazebo simulation only)
- Turtlebot3 Waffle (Wafflebot) Differential Drive Robot (real-life experimentation and Gazebo simulation)
- RB5 Drone Quadcopter (real-life experimentation)

Information regarding the physical properties of the Wafflebot, Hector Quadcopter and RB5 Drone is available in [120], [121] and [122], respectively. The physical properties of the Wafflebot and RB5 Drone are also listed in Appendix A4.

5.3.1 – Hector Quadcopter and Wafflebot Software

The Wafflebot and Hector Quadcopter ran the *ugv_nav_ops* and *uav_nav* ROS packages, respectively. These ROS packages were developed for this thesis to implement the robots' action models described in Chapter 4 and facilitate their interaction with the TCF. The core scripts in both of these ROS packages are listed in Table 5.2. A single centralized ROS master on the control station computer was used for simulation and experimentation involving the Hector Quadcopter and Wafflebot. This centralization was selected for ease of use; these tests aimed to assess the TCF and action model, not to assess computer communication.

Table 5.2 – Scripts for Action Model of Robot r_i

Serial	ROS Node	Description
1	*task_list_subscriber*	Receives a task sequence TS_i from the *mission_planner* in the High-Level Controller. Publishes task $T_j^{i,k}$ if availability $a_i = 1$, and there are incomplete tasks remaining.
2	*status_task_manager*	Converts tasks into goal poses for the robot while updating the status of task execution.
3	*pose_reporter*	Estimates the robot's pose S_i in reference to the global frame.
4	*RVO_planner*	Implementation of RVO motion planning (Appendix A3). Calculates optimal velocity v_i^{opt}.
5	*velocity_publisher*	Converts optimal velocity v_i^{opt} from *RVO_planner* into control inputs v_i^{in} using the robot's kinematic model
6	*robot_usage*	Calculates distance travelled by the robot to measure usage u_i.

The *ugv_nav_ops* and *uav_nav* ROS packages have different versions of the scripts described in Table 5.2. The differences concern the nature of their kinematic models (reflected in *velocity_publisher* and *RVO_planner*) and the nature of their tasks (reflected in *status_task_manager*). The *uav_nav_ops* package also has scripts for implementing the proposed target localization system in Section 4.3.

The operation of the action model software is as follows:

- ROS package *multi_agent_coord* generates a task allocation TA and assigns a task sequence TS_i to robot r_i.
- Node *task_list_subscriber* receives that task sequence and monitors the robot's status (busy or available). A robot is busy if it is moving to or performing a task and is available otherwise. This node *task_list_subscriber* waits until the robot is available before publishing the next task $T_j^{i,k}$ from TS_i, starting at order $k = 1$ and ending at $k = N_i$.
- Node *status_task_manager* receives task $T_j^{i,k}$ and converts it into goal poses $S_{i,d}$.
- Node *RVO_planner* receives goal pose $S_{i,d}$While also tracking the positions and velocities of other robots. *RVO_planner* calculates optimal collision-free velocity v_i^{opt}.
- Node *velocity_publisher* converts optimal velocity v_i^{opt} to control input v_i^{in} to drive the robot in the execution of its current task.
- Node *status_task_manager* updates the availability of the robot to $a_i = 0$ while it is moving to and performing the task. This node updates the availability to $a_i = 1$ after the task is complete.

5.3.2 – RB5 Drone Software

The RB5 drone was a real-life proof of concept for the UAV's POI localization. Its flight control was managed through PX4 [123] and MAVROS [124]. PX4 uses the MAVLink [125] protocol to establish communications for commands, telemetry, and flight data. MAVROS is the ROS interface for MAVLink; its core function is the translation between MAVLink messages and ROS topics. Overall, MAVROS facilitates programmatic drone control using the MAVLink protocol. This programmatic flight control requires significant integration and immediate communication between the ROS master and PX4. Therefore, the ROS master for the RB5 drone is running on its onboard flight computer and not on the control station computer. The procedure for setup and configuration of the RB5 Drone for autonomous flight with MAVROS is described in Appendix A5.

PX4 / MAVROS interprets goal poses in terms of its initialization frame, which also uses the East, North, Up (ENU) convention. However, our kinematic model, described in Section 4.1, uses the Forwards, Left, Up (FLU) convention. The conversion from a pose in the global frame to a pose in the initialization frame is facilitated through transformations described in Section 4.1 and described below. We define our desired pose in the global frame as S_d^G and desired pose in the initialization frame (ENU) as $S_d^{L'}$ in (5.1).

$$S^G = [x_d^G \quad y_d^G \quad z_d^G \quad 0 \quad 0 \quad \phi_d^G]^T \tag{5.1}$$

$$S_d^{L'} = \left[x_d^{L'} \quad y_d^{L'} \quad z_d^{L'} \quad 0 \quad 0 \quad \phi_d^{L'}\right]^T$$

The transformation from S_d^G to $S_d^{L'}$ requires the homogeneous transformation from the initialization frame to the global frame, ${}^{L}H_G$, as well as rotation from ENU to FLU convention. The transformation from the global frame to the initialization frame, ${}^{G}H_L$ is the drone's pose in the global frame the instant it was powered up, S_0^G. Therefore, ${}^{L}H_G$ can be calculated as the inverse of ${}^{G}H_L$ (5.2).

$$S_0^G = [x_0^G \quad y_0^G \quad z_0^G \quad \theta_0^G \quad \varphi_0^G \quad \phi_0^G]^T \tag{5.2}$$

$${}^{G}H_L = \boldsymbol{HT}(S_0^G)$$

$${}^{L}H_G = ({}^{G}H_L)^{-1}$$

The transformation from ENU to FLU convention, ${}^{ENU}H_{FLU}$ is a rotation of 90 degrees, or $\pi/2$ radians, along the z axis:

$${}^{ENU}S_{FLU} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \pi/2\,]^T \tag{5.3}$$

$${}^{ENU}H_{FLU} = \boldsymbol{HT}({}^{ENU}S_{FLU})$$

Combining (5.2) and (5.3) yields the transformation ${}^{L'}H_G$ from the RB5 drone's initialization frame to the global frame:

$${}^{L'}H_G = {}^{ENU}H_{FLU} * {}^{L}H_G \tag{5.4}$$

Therefore, the goal pose, $S_d^{L'}$ can be obtained with (5.5) calculating the position in terms of the initialization (ENU) frame and with (5.6) calculating yaw in terms of the initialization (ENU) frame. With (5.6), ϕ_d^G is subtracted by ϕ_0^G to account for rotation in ${}^{L}H_G$ and is subtracted by $\pi/2$ to account for ${}^{ENU}H_{FLU}$ rotation.

$$[x_d^{L'} \quad y_d^{L'} \quad z_d^{L'} \quad 1]^T = {}^{L'}H_G * [x_d^G \quad y_d^G \quad z_d^G \quad 1]^T \tag{5.5}$$

$$\phi_d^{L'} = \phi_d^G - \phi_0^G - \pi/2 \tag{5.6}$$

This conversion enables the RB5 drone to understand goal poses expressed in the global frame. Without this conversion, any goal pose would be interpreted as relative to where the drone was initially powered up and would have its x and y coordinates swapped.

For autonomous flight, the RB5 drone runs PX4 and the following ROS packages:

- *MAVROS*: interface between PX4 and MAVLink. It facilitates the translation between PX4's MAVLink messages and ROS topics.
- *voxl_ros_to_mpa*: streams camera imagery as ROS topics.
- *uav_nav_ops*: autonomous flight and computer vision functions to validate the proposed aerial observation solution.

The *MAVROS* and *voxl_ros_to_mpa packages* were pre-installed as part of the drone's operating system image [122]. The *uav_nav_ops* package was developed as part of this thesis and runs the scripts in Table 5.3.

Table 5.3 – RB5 Drone Scripts

Serial	Script	Description
1	*mocap_to_drone*	Receives a pose estimate from the motion capture system to calculate homogeneous transformation from MoCap frame to drone initialization frame.
2	*uav_camera*	Detects Apriltags using the RB5 Drone's downwards-facing camera and extracts its pixel coordinates.
3	*target_localization*	Estimates the position of Apriltag in reference to the global frame. Records these estimated positions and generates a list of these estimated positions.
4	*user_interface*	User interface node. User inputs parameters to calculate the desired flight pattern.
5	*goal_pose*	Calculates goal poses for search patterns based on user input.

The operation of the RB5 drone is as follows:

- Launch the *MAVROS*, *voxl_ros_to_mpa* and *uav_nav_ops* ROS packages.
- Execute *mocap_to_drone* to capture transformation from global frame to initialization frame.
- Node *user_interface* allows the user to input a desired pose in the global frame, S_d^G. Node *goal_pose* converts this goal pose so that it is in reference to the initialization frame, $S_d^{L'}$.
- *MAVROS* and PX4 generate the required velocity to fly the drone from its current pose to its goal pose.
- Nodes *uav_camera* and *target_localization* detect Apriltags within the camera's field of view and estimate their positions.

5.4 – Summary

This chapter has described the key technologies required for the action models that implement the task coordination framework. In particular, the following ROS packages were developed for this thesis:

- *multi_agent_coord* – implementation of task allocation algorithm from the TCF.
- *ugv_nav_ops* – implementation of action model for a team of Wafflebots.
- *uav_nav* – implementation of action model for a team of Hector Quadcopters.
- *uav_nav_ops* – autonomous flight and computer vision for real-life testing of proposed aerial observation with RB5 drone.

The dynamic instantiation of ROS topics also represents a secondary contribution. It enables a single ROS master to control multi-robot teams without prior knowledge regarding quantity but with prior knowledge of types of robots. The technologies discussed in this chapter will be used to facilitate simulation and experimentation in Chapter 6.

Chapter 6 – Simulation and Experimentation

This chapter describes testing using simulations of robots on Gazebo [126] and experiments with real-life robots. The tests consist of a multi-robot team performing tasks allocated from the $GrdGenTA$ and $GrdTA$ algorithms described in Chapter 3 and executing their assigned tasks using the solutions described in Chapters 4 and 5 as well as Appendix A3.

The tasks considered for all tests were described in Section 4.2 and are listed in Table 6.1. Unless otherwise stated, the $GrdGenTA$ used the parameters described in Table A.1. The genetic algorithm component of $GrdGenTA$ used an iteration count of $n_{iter} = 300$.

Table 6.1 – Task Descriptions

Task Type (t_j)	**Task Description**	**Actuation Cost** (c_j)	**Notes**
B	Stationary Observation	0	Observes a POI as described in Section 4.2.
C	Raster Search Observation	$\frac{L^2}{w_s} + L$	Executes raster search flight pattern as described in Section 4.2.
D	Ground Patrol	$4L$	Drive in a square pattern centred around a target location as described in Section 4.2
E	Sensor Node Delivery	0	Movement to goal pose to drop off payload. Modelled as a go-to-goal task
F	Sensor Activation	0	Move to goal pose and activate onboard sensors. Modelled as a go-to-goal task

Unless otherwise stated, all robots used the constant parameters listed in Table A.2, where Hector Quadcopters are considered robots of type α and Wafflebots are considered robots of type β, ε, τ. The V_{max}, ω_{max} parameters for the Hector Quadcopter were based on the maximum speed set for the RB5 drone's flight tests such that its motion was not disruptive to aerial observation tasks. The R_i parameter for the Hector Quadcopter was decided based on a visual inspection of the gazebo simulation. The V_{max}, ω_{max}, R_i parameters for Wafflebots were based on their real-life properties. There was also an element of experimental validation to determine the value of R_i that provided enough time to avoid collision, considering the robots' kinematics.

The simulation environment was Ubuntu 20.04, running on Windows Subsystem for Linux with graphical user interface. Its setup instructions are in [127].

6.1 – Collaborative Aerial Observation

An application of the Task Coordination Framework (TCF) for collaborative aerial observation was explored using a team of Hector Quadcopters. These UAVs simultaneously execute stationary observation and raster search observation tasks assigned by the TCF. As they execute these tasks, their embedded cameras detect and estimate the positions of Apriltag markers, representing points of interest (POIs). Section 6.1.1 describes the performance of the

TCF and the UAVs as they execute assigned tasks. Section 6.1.2 describes the quality of the POIs' localization from the UAVs' task execution.

Real-life testing was also performed using an RB5 Drone to confirm the real-life validity of the computer vision model described in Section 4.3. The RB5 Drone executed a series of stationary observation tasks for this test. As the RB5 Drone flies in accordance with these tasks, its embedded camera detects Apriltag markers as POIs. This real-life test was also replicated with a Gazebo simulation. This confirmation testing is described in Section 6.1.3.

6.1.1 – Multiple UAV TCF Simulation

The multiple UAV TCF simulation used the 16T-3R scenario with the tasks and robots described in Table A.5 and Table A.6, respectively. Task types B and C are stationary observation and raster search tasks, as per Table 6.1. The prerequisite, suitability and work capacity constraints were not considered, as the aim was to demonstrate an application of the TCF for collaborative aerial observation. The scenario is shown in Figure 6.1, where each Apriltag marker corresponds to a POI whose position will be estimated by the Hector Quadcopters performing aerial observation tasks. The task locations (x_j, y_j) are shown as the red circles. The Hector Quadcopters in their start positions are labelled as r_0, r_1 and r_2.

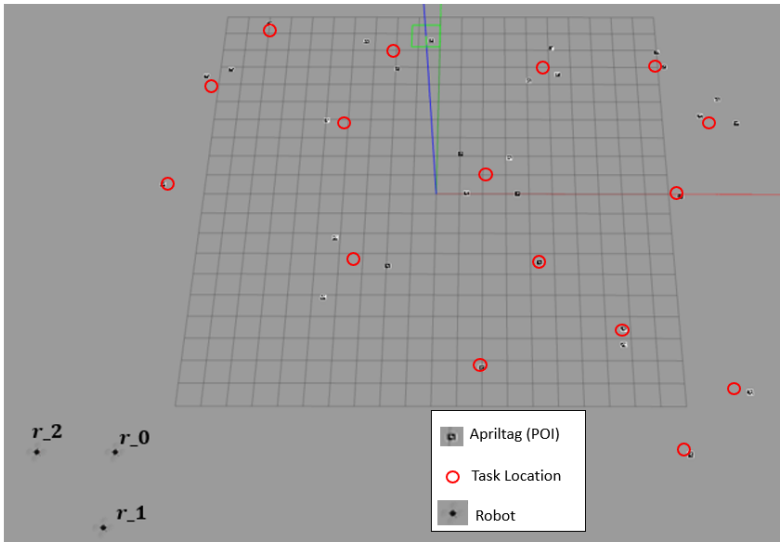


Figure 6.1 – Gazebo Simulation of 16T-3R Scenario

Figure 6.2 shows the calculated task allocation from the $GrdTA$ (Figure 6.2a) and from the $GrdGenTA$ algorithm (Figure 6.2b). The performance of the robots, indicating their assigned task sequences and usages, is documented in Table 6.2. The total cost C and global objective function F value of the entire system is documented in Table 6.3. The type and index of each task (corresponding with Table A.5) is also shown in Figure 6.2.

Note that Figure 6.2 does not show the robots' trajectories. It shows the locations of the robots' starting positions, as well as the task locations $(x_j,\ y_j)$ and the sequences in which their assigned tasks are performed. The robots' physical trajectories from the gazebo simulation using the $GrdGenTA$ algorithm are shown in Figure 6.3. The robots' performance and overall system performance of the gazebo simulation are also described in Table 6.2 and Table 6.3. These values were measured from the robots' simulated odometry. The values of usage u_i and

subsequently total cost C and global objective function F from the gazebo simulation correspond to the measured distance travelled by each robot along the xy plane.

Similar to the test cases in Section 3.3, the $GrdGenTA$ re-distributed the tasks assigned by the $GrdTA$. This effect is observed in Figure 6.2 and Table 6.2, in which robots' task sequences become more efficient with the $GrdGenTA$. This improved efficiency reduces the overall usages of the robots and, consequently, the total cost and global objective function value of the system, shown in Table 6.3.

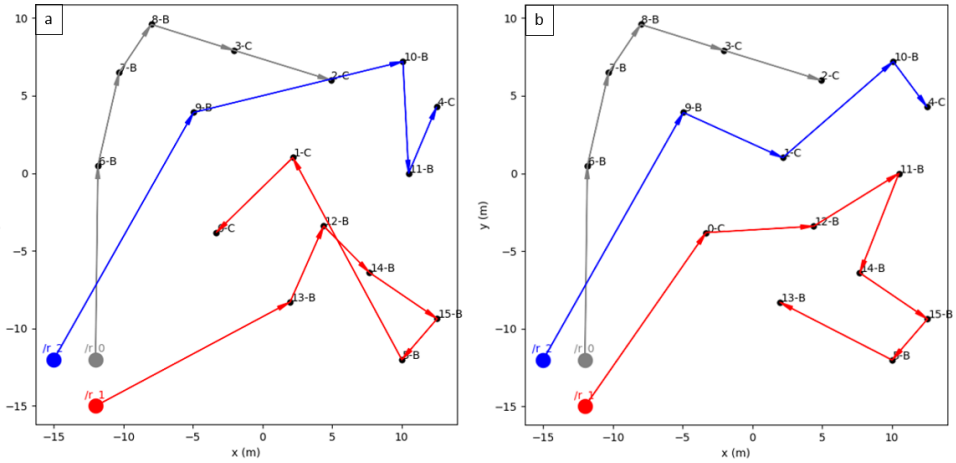


Figure 6.2 – Task Allocation for Multiple UAV Scenario with (a) Greedy Algorithm and (b) Greedy + Genetic Algorithm

Figure 6.3 shows the physical trajectories of each in the Gazebo Simulation and the location of each task. As described in Section 4.2, the quadcopters fly toward the location of stationary observation tasks (type B), stopping a distance r away to observe them, as well as flying raster patterns for the raster search observation task (type C). For the stationary observation task, the viewing angle ϕ_d is the same as the heading from the UAV's current pose to the task location, resulting in a straight-line path directly to the task location. For the raster search observation task, the side length L of the square is calculated from (4.35).

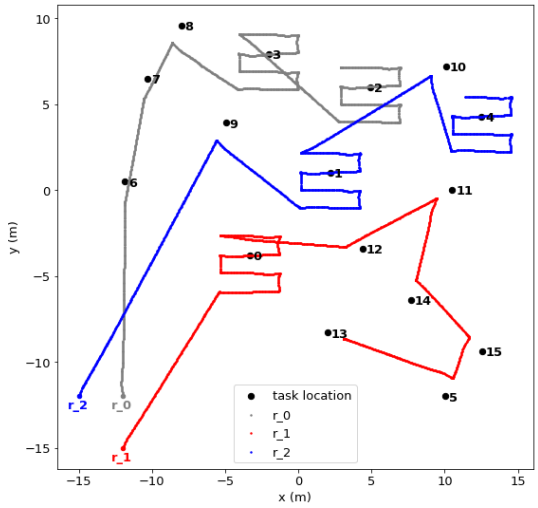


Figure 6.3 – Robot Trajectories from Gazebo Simulation of Multiple UAV Scenario and Greedy + Genetic Algorithm

Within Table 6.2 and Table 6.3, it is also observed that the overall usages u_i, cost C and global objective function value F are less in the $GrdGenTA$ Gazebo simulation than in the $GrdGenTA$ calculation. This discrepancy stems from the imperfect modelling of task actuation cost. The calculation of F uses the cost function (3.10), which assumes the robot moves exactly to the task location. However, in the Gazebo simulation u_i, C and F were obtained through robots' odometry. In reality, the robots do not move to their exact task locations (x_j, y_j). In the case of the stationary observation task, the robot moves over some distance $(\Delta s_{i,j} - r)$ while the calculation assumes it moves a distance of $\Delta s_{i,j}$, where $\Delta s_{i,j}$ is from (3.9) and r is from (4.26). In the case of the raster search, the robot flies to the first waypoint of the raster search pattern (described in Section 4.2), while the $GrdGenTA$ assumes it flies to the task location (x_j, y_j). These discrepancies result in some level of inaccuracy regarding the values of u_i, C and F; the values obtained from the task allocation algorithms should be considered as estimates/approximations.

Table 6.2 – Robot Performance for Multiple UAV Scenario

	Figure 6.2a – $GrdTA$ Calculation		Figure 6.2b – $GrdGenTA$ Calculation		Figure 6.3 – $GrdGenTA$ Gazebo Simulation	
Robot	**Task Sequence**	**Usage** (u_i)	**Task Sequence**	**Usage** (u_i)	**Task Sequence**	**Usage** (u_i)
r_0	6, 7, 8, 3, 2	83.95	6, 7, 8, 3, 2	83.94	6, 7, 8, 3, 2	73.88
r_1	9, 10, 11, 4	105.30	9, 1, 10, 4	78.09	9, 1, 10, 4	66.33
r_2	13, 12, 14, 5, 1, 0	70.17	0, 12, 11, 14, 15, 5, 13	88.37	0, 12, 11, 14, 15, 5, 13	77.83

Table 6.3 – System Performance for Multiple UAV Scenario

Test Case	**Total Cost** (C)	**Global Objective Function Value** (F)	**Computation Time** (s)
Figure 6.2a – $GrdTA$ calculation	259.41	364.71	0.05
Figure 6.2b – $GrdGenTA$ calculation	250.41	338.79	5.04
Figure 6.3 – $GrdGenTA$ Gazebo simulation	218.04	295.87	N/A (measured from simulated odometry)

The computation time of 5.04 s for the $GrdGenTA$ with 300 iterations in the Gazebo simulation is also slower than 5.27 s for the $GrdGenTA$ with 1000 iterations in Table 3.7 for the unconstrained 16T-3R scenario. It is also noted that the greater number of iterations resulted in a more optimal task allocation, with a lower global objective function value of 329.13 in Table 3.7. In contrast, 300 iterations yielded a global objective function value of 338.79 in Table 6.3

The reason for this slower computation is due to the simulation environment. In the evaluation of task allocation algorithms (Section 3.3), the computer is only executing the $GrdGenTA$ algorithm. In the simulation, the computer is running ROS nodes that execute the $GrdGenTA$ algorithm (described in Section 5.2) and ROS nodes for the 3 robots' action model (motion planning and computer vision described in Section 5.3). The computation rate of a real-life implementation would be closer to the rate in Section 3.3, as the control station computer

would not need to run the ROS nodes for the robots' action model. Instead, it would only need to implement the functionality in Section 5.2 with intermittent communication with the robots.

Overall, the $GrdGenTA$ algorithm was effective in generating an optimized allocation of aerial observation tasks for a multi-UAV team. The $GrdTA$ solution was optimized with the $GrdGenTA$ algorithm, resulting in more efficient task sequences with less overlap, reducing the overall usages of the robots and consequently reducing the value of the global objective function. This task allocation was also generated with reasonable computational demand; the slower computation rate was attributed to the simulation setup, not the algorithm itself.

6.1.2 – Collaborative Aerial Observation

The UAVs observe the ground as they maneuver in accordance with their assigned tasks from the $GrdGenTA$. They detect the Apriltags as POIs, obtaining raw position estimates $(\hat{x}_t, \hat{y}_t)$ which are converted into filtered estimates $(\bar{x}_m, \bar{y}_m)$ of the POIs' positions using the solutions described in Section 4.3.

The results of the collaborative aerial observation are shown in Figure 6.4 with raw POI position estimates $(\hat{x}_t, \hat{y}_t)$, ground truth (x_{poi}, y_{poi}) POI positions and filtered estimates $(\bar{x}_m, \bar{y}_m)$ of the Apriltags' positions. The filtered estimates are from the list $\hat{T}^*$ of cluster centers from KMeans clustering. The ground truth POI positions were obtained by subscribing to the ROS topic *gazebo/model_states,* which contains ground truth information of the pose of all objects in the simulation.

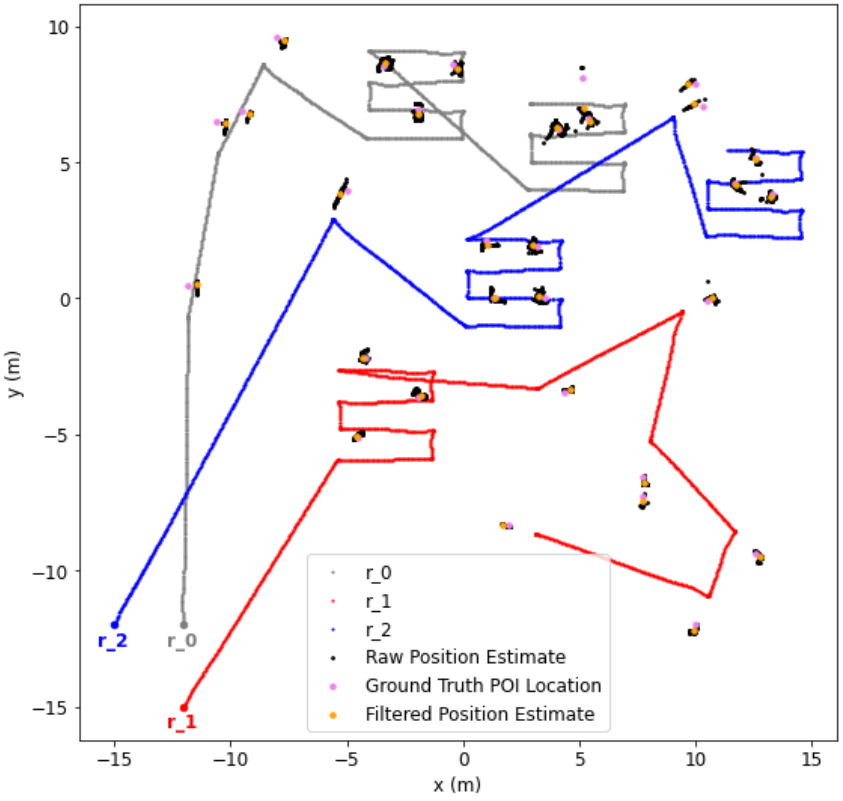


Figure 6.4 – Results of Collaborative Aerial Localization from Multiple UAV Scenario with Greedy + Genetic Algorithm

The filtered estimates $(\bar{x}_m, \bar{y}_m)$ and ground truth positions (x_{poi}, y_{poi}) are listed in Table 6.4, which includes the error between $(\bar{x}_m, \bar{y}_m)$ and (x_{poi}, y_{poi}) . The error between those values is the Euclidean 2D distance, as all Apriltags are on the ground with $z = 0$. Clusters were assigned to ground truths using linear sum assignment from the SciPy package [128]. Linear sum assignment finds one-to-one matchings between each ground truth and each filtered estimate such that the sum of all errors is minimized. The clusters are listed in order of decreasing error in Table 6.4, with the greatest error listed first. Apriltag 1 has a relatively large error of $1.142\ m$ compared with the rest of the Apriltags, whose errors range from $0.384\ m$ to $0.045\ m$. For reference, the simulated environment was a square of $30\ m$ by $30\ m$.

Table 6.4 – Ground Truth and Estimated POIs with Error for Multiple UAV Simulation

Apriltag # $(poi\ /\ m)$	Ground Truth Positions (x_{poi}, y_{poi})		Cluster Center / Filtered Estimate Positions $(\bar{x}_m, \bar{y}_m)$		Error (m)
	x (m)	y (m)	x (m)	y (m)	
1	5.150	8.107	5.182	6.965	1.142
2	-4.956	3.953	-5.315	3.816	0.384
3	10.287	7.040	9.913	7.128	0.384
4	-11.835	0.486	-11.467	0.514	0.369
5	-10.611	6.472	-10.244	6.442	0.368
6	-8.020	9.599	-7.681	9.457	0.368
7	-9.529	6.877	-9.200	6.742	0.356
8	10.007	7.886	9.659	7.893	0.349
9	1.983	-8.319	1.712	-8.312	0.271
10	4.386	-3.442	4.603	-3.363	0.231
11	-0.411	8.602	-0.240	8.448	0.229
12	10.512	-0.098	10.692	0.044	0.229
13	10.000	-12.000	9.946	-12.221	0.227
14	7.733	-6.557	7.777	-6.774	0.221
15	3.496	0.018	3.288	0.079	0.217
16	12.562	-9.375	12.750	-9.474	0.213
17	7.696	-7.267	7.718	-7.461	0.195
18	1.019	2.128	1.067	1.951	0.183
19	3.164	1.916	3.012	1.929	0.153
20	-3.422	8.525	-3.357	8.658	0.148
21	-1.936	6.925	-1.951	6.783	0.143
22	-1.946	-3.613	-1.825	-3.549	0.137
23	13.294	3.813	13.208	3.719	0.128
24	5.397	6.601	5.469	6.498	0.126
25	-4.213	-2.219	-4.324	-2.186	0.116
26	11.708	4.217	11.769	4.142	0.097
27	1.303	0.044	1.375	0.013	0.078
28	4.075	6.223	4.037	6.273	0.063
29	12.561	5.161	12.585	5.115	0.052
30	-4.527	-5.099	-4.561	-5.069	0.045

Based on the observation of Figure 6.4 and Table 6.4, there was a detection error involving Apriltags 1, 24, and 28. Furthermore, several Apriltags' raw estimates were observed to form an elongated pattern. These observations are annotated in Figure 6.5a. A close-up of the detection error is shown in Figure 6.5b.

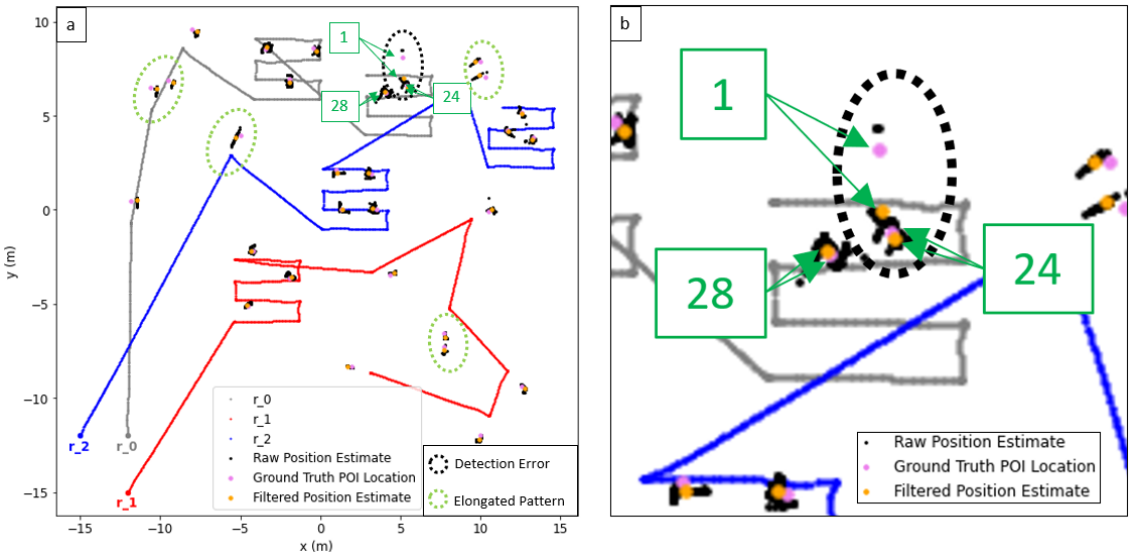


Figure 6.5 – Estimated / Ground Truth POIs: a) Overview with Detection Error and Elongated Detection Patterns, b) Close-Up View of Detection Error

The detection error refers to the abnormally large error/distance $(1.142\ m)$ between the filtered estimate $(\bar{x}_1,\ \bar{y}_1)$ and ground truth position (x_1, y_1) of Apriltag 1. In contrast, the distance between (x_{24}, y_{24}) and $(\bar{x}_1,\ \bar{y}_1)$ is $0.423\ m$. The closest ground truth to $(\bar{x}_1,\ \bar{y}_1)$ should be (x_1, y_1), not (x_{24}, y_{24}). This error can be attributed to the Kmeans clustering and the flight pattern that generated the list of raw position estimates $\hat{P}$.

Apriltags 1, 24 and 28 are close to each other. The flight pattern collected many raw estimates for Apriltags 24 and 28 but very few for Apriltag 1. This distribution resulted in the Kmeans clustering generating a cluster center for Apriltag 1 closer to Apriltags 24 and 28. The error would be reduced if the flight pattern's trajectory had more observation over Apriltag 1, generating more raw estimates closer to the ground truth location of Apriltag 1. Alternatively, different clustering algorithms can reduce the relative weight of raw estimates closer to each other and increase the relative weight of raw estimates farther from each other. This weighting would make the clustering algorithm more effective at finding discrete points from the list of raw position estimates $\hat{P}$.

The raw estimates also form elongated patterns in the same direction as the UAV. This elongation can have a detrimental effect on the POIs' position estimates. However, it can be mitigated by viewing the markers from different directions, as observed in the markers detected by the raster search observation task. With the raster search, which views markers from multiple directions, the raw estimates do not exhibit these elongated patterns.

Within this scenario, the mean average distance $\bar{d}_{ip}$ between each ground truth POI position was $12.29\ m$. The average distance between a given POI and all other POIs was calculated. This average distance calculation was repeated for all POIs, and the mean of those

averages is $\bar{d}_{ip}$. The average error of all filtered estimates, with the Apriltag 1 entry included, is $0.241\ m$, and the average error without Apriltag 1 is $0.210\ m$, or $1.96\ \%$ and $1.71\ \%$, respectively, when compared against $\bar{d}_{ip}$. If extrapolated over a larger area, with an average distance between POIs as $1000\ m$, the result would be an average error of approximately $20\ m$. If the estimated POI positions are used as waypoints for other robots, they would have to rely on some level of autonomy to navigate the gap between the POI's ground truth location and its estimated position. However, the autonomy required to navigate that gap is well within the current state-of-the-art motion planning solutions. Therefore, this average error of $1.96\ \%$ and 1.71% is within acceptable limits for guiding other autonomous robots.

The proposed task coordination framework (TCF) and aerial observation with POI localization effectively employed a team of 3 UAVs to survey an area, providing accurate estimates of the positions of 29/30 POIs. The POI corresponding to Apriltag 1 had greater error due to the flight patterns not generating enough data points for that Apriltag. The error was also within acceptable levels and was attributed to a combination of flight pattern, camera projection and clustering algorithms. Overall, these issues are related to computer vision and not directly related to the task allocation algorithms. This simulation confirms that the TCF can effectively coordinate the movement and task execution of UAVs for a collaborative aerial observation mission. However, more work is required to refine the computer vision and localization aspects.

6.1.3 – POI Localization Proof of Concept with RB5 Drone

The proposed POI localization solution in Section 4.3 was tested on a single RB5 Drone in real life. This test was also replicated on Gazebo using the Hector Quadcopter. In these tests, the UAV (either the physical RB5 drone or simulated Hector Quadcopter) performs the stationary aerial observation tasks (described in Section 4.2) as per Table A.10. These tasks were structured to assign a flight path for the UAV such that its embedded camera will detect the Apriltag markers and apply the localization solution. The real-life test scenario is shown in Figure 6.6a. A simulation replicating the real-life test is shown in Figure 6.6b. In both the real-life test and the simulation, the same tasks were performed in the same sequence and from the same start location. The stationary observation tasks also sought to view the apriltag markers from different angles to obtain more accurate position estimates, as the goal is to validate that the Gazebo simulation accurately depicts reality.

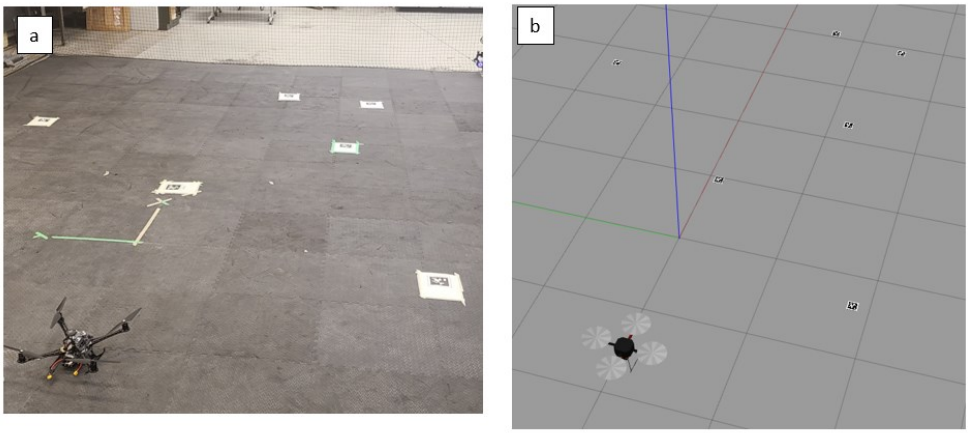


Figure 6.6 – Apriltags Localization Test a) Real-Life Test with RB5 Drone, b) Gazebo Simulation with Hector Quadcopter

In order to facilitate a direct comparison, the homogeneous transformation ${}^{r}H_{CI}$ from the robot body frame to the camera image frame on the Hector Quadcopter was set to match that of the RB5 Drone. This configuration ensured identical camera geometry. The value of ${}^{r}H_{CI}$ is described in Appendix A4.1. Other differences, such as the lower-level control loops and the size of the drones, did not impact the test. The experiment was also limited to the small test environment described in Section 5.1.

The UAVs' trajectories, along with raw estimates, filtered estimates and ground truths of Apriltags' positions are shown in Figure 6.7, where both the RB5 Drone and Hector Quadcopter perform the same tasks in the same order from the same start location. The sequence of aerial observation tasks is also annotated as the numbers from 1 to 18 in Figure 6.7, corresponding to the task order in Table A.10. Note that the position in Table A.10 corresponds to the task location, not the UAV's position. However, in Figure 6.7, the annotations are placed at the UAV's position when executing a stationary observation task. This annotation was selected because several observation tasks have the same task location but different viewing angles. The order in which the Apriltags were observed is also annotated as the blue numbers in Figure 6.7. These numbers also correspond to the "Apriltag #" column in Table 6.5.

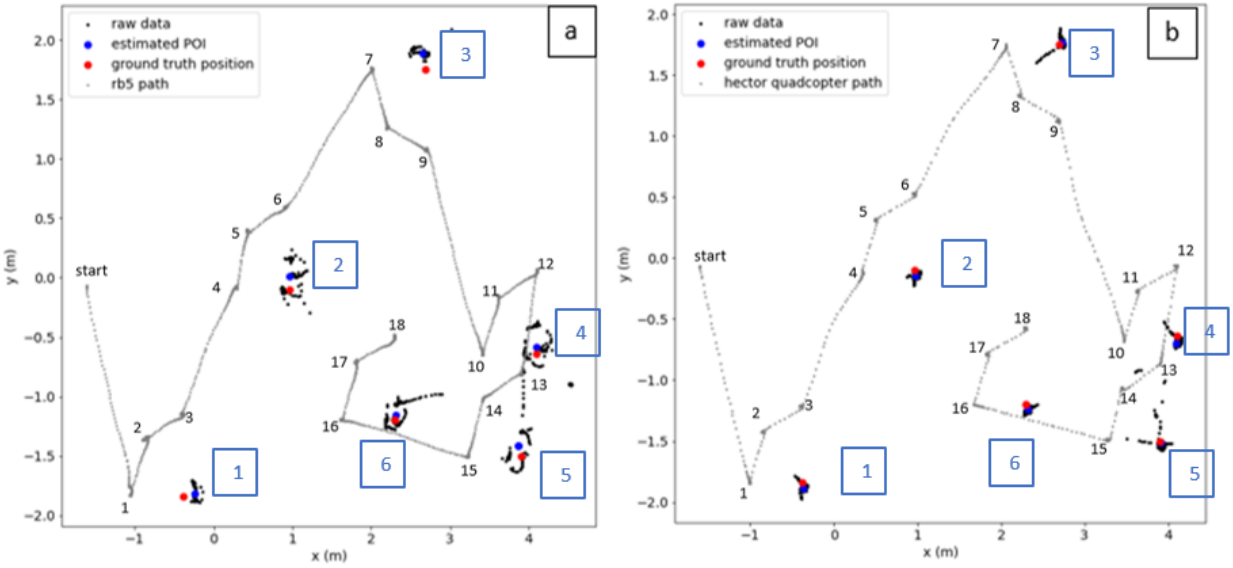


Figure 6.7 – POI Position Estimates with UAV Trajectory for (a) RB5 Drone and (b) Hector Quadcopter

The Ground truth positions for Apriltags' real-life positions were measured by the motion capture (MoCap) system as described in Section 5.1. For that matter, the RB5 drone with reflective IR markers was successively placed on top of those Apriltags and its (x, y) position measurement from MoCap was recorded. To replicate that scenario, these coordinates were inputted as the Apriltags' ground truth positions on Gazebo. The results of the aerial observation tasks are listed in Table 6.5, showing that both the RB5 Drone and Hector Quadcopter were able to obtain close estimates of the Apriltags' positions. Similar to the collaborative aerial observation in Section 6.1.1, linear sum assignment was used to match filtered estimates of POI positions with ground truth POI positions.

Table 6.5 – Position Estimate, Ground Truth, and Relative Error between Estimate and Ground Truth

		RB5 Drone Experiment		Hector Quadcopter Simulation	
Apriltag #	Ground Truth$(x, y)\ m$	Estimated POI Position $(x, y)\ m$	Error (m)	Estimated POI Position $(x, y)\ m$	Error (m)
1	(0.97, -0.10)	(0.964, 0.008)	0.108	(0.981 -0.146)	0.047
2	(-0.38, -1.84)	(-0.241, -1.819)	0.141	(-0.366, -1.883)	0.045
3	(2.69, 1.75)	(2.649, 1.894)	0.149	(2.728, 1.769)	0.042
4	(3.90, -1.50)	(3.869, -1.412)	0.093	(3.907, -1.508)	0.011
5	(4.10, -0.64)	(4.094, -0.583)	0.057	(4.097, -0.699)	0.059
6	(2.30, -1.20)	(2.307, -1.154)	0.047	(2.321, -1.240)	0.045

From the observation of Table 6.5 and the comparison of average error, the position estimate error for both tests is similar, with the physical RB5 drone having more error than the simulated Hector Quadcopter. The average error for the RB5 Drone was $0.099\ \mathrm{m}$, while the average error for the Hector Quadcopter simulation was $0.042\ \mathrm{m}$. With a mean average distance $\bar{d}_{ip}$ between the Apriltags of 2.38 m, this is an average error of 4.15 % and 1.76 % for the real-life test and simulation, respectively.

This error difference can be attributed to experimental factors:

- Lighting conditions and camera imperfections on the RB5 Drone may have resulted in fewer and noisier detections, evident from the greater dispersion in Figure 6.7a.
- Vibrations from flight could have resulted in an unsteady camera, which would reduce the accuracy of the Apriltag detection, visual odometry and IMU measurements. This would result in reduced pose estimate accuracy for the drone and reduced detection accuracy of Apriltag markers, which would then degrade the position estimates of the Apriltag markers.
- The ground truth position of Apriltags in real life may differ from the recorded value. Placing the RB5 drone on the tag and reading the MoCap pose may not be the most accurate approach, as it depends on the user's placement and interpretation of the "center" of the tag by visual inspection.
- Pose estimate noise/error may have resulted in a less accurate estimate of the POI position. The proposed target localization solution in Section 4.3 relies on an accurate pose estimate of the camera that is observing the detections. The pose estimate of the RB5 drone, and consequently, the pose of its camera, was not perfect and had some error. This error is described in Appendix A6.1, which reports on the experimental evaluation of the RB5 Drone's visual inertial odometry.

Both the RB5 drone and Hector Quadcopter effectively estimated the positions of the POIs represented by the Apriltags. The RB5 drone's performance was similar to that of the Hector Quadcopter, with differences explained by the experimental setup. If these experimental factors were replicated on Gazebo, or if the experimental factors were better compensated, the difference between real life and simulation would be smaller. Overall, the Gazebo simulation accurately depicts reality for the target localization solution described in Section 4.3

6.2 – TCF Validation with UGVs

Implementations of the TCF were tested using both simulated and real-life wafflebots performing assigned tasks. The simulation served as a large-scale stress test to evaluate the performance of the TCF for general-purpose multi-robot task execution. This simulation is described in Section 6.2.1.

Real-life testing was also performed at a smaller scale to confirm that the simulations accurately depict reality. The real-life tests were then replicated on Gazebo, and their results were compared. This real-life testing is described in Section 6.2.2. The robot used for both the simulations and real-life testing is the wafflebot.

6.2.1 Multiple UGV Simulated Testing

This simulation consisted of the 31T-6R50 scenario (tasks described in Table A.3, robots start positions described in Table A.4, work capacity described in Table 3.16). Task type D is a ground patrol task, and types E and F are go-to-goal tasks as per Table 6.1.

The 31T-6R50 scenario was selected primarily for its quantity and diversity of robots and tasks. It was also selected because its work capacity constraints (3.14) limited the multi-robot team but still allowed all tasks to be completed, as in Section 3.3.2, test case 3. This simulation was a stress test that demonstrated generalized task allocation with the prerequisite, suitability, and usage constraints.

The 31T-6R50 scenario simulation is shown in Figure 6.8, with the tasks' locations represented by the red squares and the robots (labelled from r_0 to r_5) in their starting positions. Note that the red squares in the scenario have no interaction with the robots; they only represent the tasks' locations as described by the (x_j, y_j) values in Table A.3.

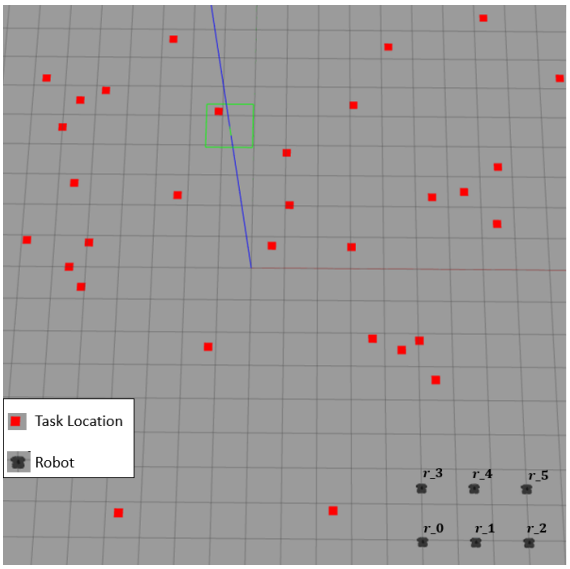


Figure 6.8 – Gazebo Simulation of 31T-6R50 Scenario

Figure 6.9 shows the calculated task allocation from the $GrdTA$ (Figure 6.9a) and from the $GrdGenTA$ (Figure 6.9b). The performance of the robots, indicating their assigned task

sequences and usages, is documented in Table 6.6. The total cost C and global objective function F value of the entire system is documented in Table 6.7. The index and type of each task (corresponding with Table A.3) are also shown in Figure 6.9.

Figure 6.9 only shows the locations of robots' starting positions and task locations, as well as the sequence in which they perform their assigned tasks. The robots' physical trajectories from the gazebo simulation using the $GrdGenTA$ algorithm task allocation (Figure 6.9b) are shown in Figure 6.10. The performance of individual robots and the overall system from the gazebo simulation are also described in Table 6.6 and Table 6.7. The values for usage u_i, total cost C and global objective function F were obtained from the robots' simulated odometry.

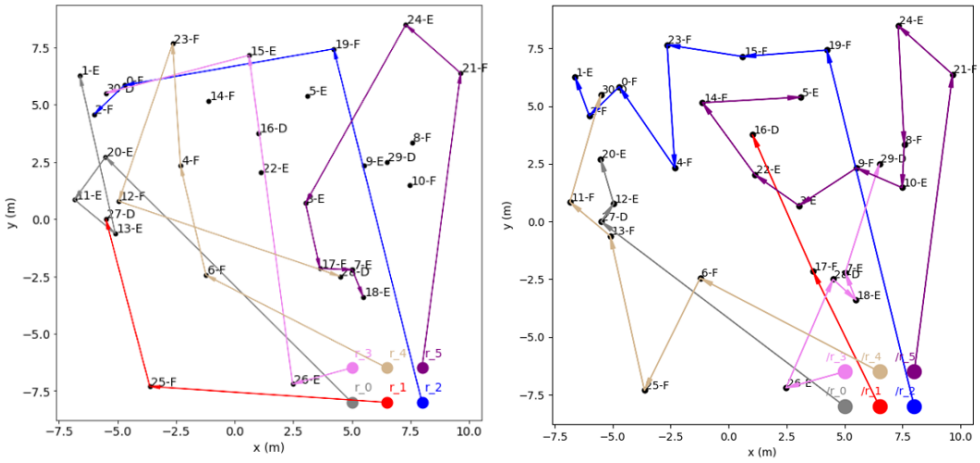


Figure 6.9 – Task Allocation for 31T-6R50 Gazebo Simulation with (a) Greedy Algorithm, and (b) Greedy + Genetic Algorithms

Within the UGV simulation of the 31T-6R50 scenario, the $GrdTA$ and $GrdGenTA$ performed similarly to the corresponding test case in Section 3.3. Both algorithms satisfied all constraints (suitability, prerequisites, work capacity). Note that Figure 6.9a is identical to Figure 3.20a, with 23/31 tasks completed, demonstrating the deterministic nature of the $GrdTA$. However, despite using the same 31T-6R50 scenario, Figure 6.9b differs from Figure 3.21b, demonstrating the metaheuristic nature of the genetic algorithm component of the $GrdGenTA$. Overall, the result of the task allocation algorithms in the Gazebo simulation is identical to the evaluation, with differences attributed to the metaheuristic nature of the genetic algorithm component.

The work capacity constraint (3.14) was satisfied, as shown in Table 6.6; none of the robots' usages u_i exceeded the constraint of $u_{i\_max} = 50$. Furthermore, the $GrdGenTA$ optimized the robots' task sequences such that all 31/31 tasks can be assigned within the work capacity constraints instead of just 23/31 tasks with the $GrdTA$.

The suitability constraint (3.15) was satisfied. The task allocation complied with the agent-task matching of the 31T-6R50 scenario (Table A.4 and Table A.9). Robots of type β (r_0 and r_3) were not assigned type F tasks, robots of type ε (r_1 and r_4) were not assigned type E tasks, and robots of type τ (r_2 and r_5) were not assigned type D tasks. This allocation occured because of the minimum suitability requirement of $s_{min} = 0.65$. The agent-task matchings with a suitability score below s_{min} were not assigned.

The prerequisite constraint (3.16) was satisfied. There were no instances in which a prerequisite was assigned after its dependant within each task sequence, as described in (3.25) in section 3.3, test case 1. For example, in Figure 6.9b with the $GrdGenTA$, there are several instances where it is evident the prerequisite constraint is enforced including:

- T_{27} assigned as the first task for r_0, satisfying prerequisites for T_{11}, T_{12}, T_{13}. T_{12} is the second task for r_0 while T_{13} and T_{11} are the 3rd and 4th tasks for r_4.
- T_{16} assigned as the second task for r_0, satisfying prerequisites for T_{22}, T_{14}, T_5. These tasks are the 7th, 8th and 9th tasks for r_5.

However, similar to section 3.3 test case 1, the task allocation with the $GrdGenTA$ in Figure 6.9b is time suboptimal. There are several instances in which a robot may have to wait for certain tasks (assigned to other robots) to be completed before it can execute its current task:

- r_1 cannot perform T_7 (1st task for r_1) until r_3 finishes T_{28} (2nd task for T_{28}).
- r_5 cannot perform T_8 (3rd task for r_5) until r_3 finishes T_{29} (5th task for r_3).

Figure 6.10 shows the physical trajectories of each robot in the Gazebo simulation. As described in Section 4.2, the robots perform go-to-goal tasks (task types E and F) and ground patrol tasks (task type D). The ground patrol tasks were prerequisites for the go-to-goal tasks within their patrol area, as described in Table A.3. This patrol area can be observed in Figure 6.10, where the patrol tasks' square-shaped areas can be seen due to the robots' trajectories where those squares are the patrol routes of type D tasks.

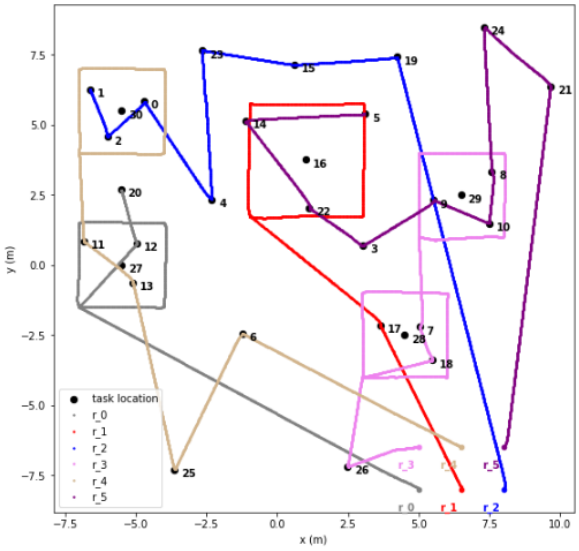


Figure 6.10 – Robot Trajectories from Gazebo Simulation of 31T-6R50 Scenario with Greedy + Genetic Task Allocation Algorithm with Task Locations

There were also several instances where the robots' trajectories were deflected, indicative of the RVO algorithm adjusting velocity to avoid a collision. This deflection is notable in the vicinity of tasks T_{26} and T_{29} in Figure 6.10. While this simulation is not an exhaustive test of the RVO algorithm, when combined with the supplementary experiments in Appendix A6.2 (which use the same robot but in real life), it is evident that the RVO is effective at adjusting trajectories for collision avoidance of the differential drive robots. This figure also does not show

robots reducing or increasing their speed (but maintaining the same direction) to avoid a collision.

Similar to the test cases in Section 3.3, the $GrdGenTA$ re-distributed the tasks assigned by the $GrdTA$. This effect is observed in Figure 6.9 and Table 6.6, in which robots' task sequences become more efficient with the $GrdGenTA$. This improved efficiency enables more tasks to be completed while still complying with the work capacity constraints.

Table 6.6 – Robot Performance for Multiple UGV Task Allocation

	Figure 6.9a – $GrdTA$ calculation		Figure 6.9b – $GrdGenTA$ calculation		Figure 6.10 – $GrdGenTA$ Gazebo Simulation	
Robot	**Task Sequence**	**Usage** (u_i)	**Task Sequence**	**Usage** (u_i)	**Task Sequence**	**Usage** (u_i)
r_0	20, 11, 13, 1	26.62	27, 12, 20	28.17	27, 12, 20	30.12
r_1	25, 27	29.69	17, 16	28.97	17, 16	28.01
r_2	19, 0, 2	26.77	19, 15, 23, 4, 0, 2, 1	35.99	19, 15, 23, 4, 0, 2, 1	35.65
r_3	26, 15, 30	35.42	26, 28, 18, 7, 29	39.28	26, 28, 18, 7, 29	35.30
r_4	6, 4, 23, 12, 28	48.21	6, 25, 13, 11, 30	40.11	6, 25, 13, 11, 30	37.93
r_5	21, 24, 3, 17, 7, 18	30.59	21, 24, 8, 10, 9, 3, 22, 14, 5	38.68	21, 24, 8, 10, 9, 3, 22, 14, 5	38.31

Table 6.7 – System Performance for Multiple UGV Task Allocation

Scenario	**Total Cost** (C)	**Global Objective Function Value** (F)	**Computation Time** (s)
Figure 6.9a – $GrdTA$ Calculation (23/31 tasks)	197.32	245.51	1.07
Figure 6.9b – $GrdGenTA$ Calculation (31/31 tasks)	211.20	251.31	41.35
Figure 6.10– $GrdGenTA$ Gazebo Simulation (31/31 tasks)	205.32	243.63	N/A (measured from simulated odometry)

Within both Table 6.6 and Table 6.7, there is a discrepancy between the $GrdGenTA$ calculation and the Gazebo simulation using the $GrdGenTA$ algorithm. The values of usage, and consequently total cost and global objective function, differ. This discrepancy is similar to the one observed in Section 6.1 between Table 6.2 and Table 6.3. It also stems from the imperfect task actuation cost model, particularly with the ground patrol task. With regards to the ground patrol task, the task allocation algorithm considers the robot as moving a distance $\Delta s_{i,j}$ toward the task location (x_j, y_j) and then executing a patrol route of distance $4L$. In reality, the robot does not move to the exact task location; it moves to the first goal pose in the patrol task, resulting in a travel distance to the task that is less than $\Delta s_{i,j}$. As for the go-to-go tasks, the robots do not move exactly to the task location in the gazebo simulation. The robot is considered "on location" when it is some threshold distance Δs away from it. The task allocation

considers the robot as moving a distance $\Delta s_{i,j}$ when in reality, it moves a distance $(\Delta s_{i,j} - \Delta s)$. In addition to task modelling discrepancies, the task allocation algorithms do not consider collision avoidance. The application of RVO to dynamically adjust trajectories would result in travel distances that differ from the calculation of the task allocation algorithms.

Similar to Section 6.1, the calculated usages, total cost, and global objective function values can be considered close estimates instead of exact quantities. Also similar to Section 6.1, the slower computation rate of $41.35\ s$ for 300 iterations (compared to approximately $11\ s$ for 1000 iterations in fully constrained 31T-6R50 in Section 3.3) can be attributed to the simulation environment. The computer had to run ROS nodes for the action models of the 6 robots in addition to the task allocation algorithm, which consumed computational resources. In reality, each individual robots' computer would manage its own action model software, significantly reducing the computational demand imposed on the control station computer.

Overall, the $GrdGenTA$ algorithm was effective in generating an optimized and admissible task allocation for execution by a multi-robot team. The $GrdTA$ solution was optimized by the $GrdGenTA$ algorithm, resulting in more efficient task sequences, reducing the value of the global objective function. Furthermore, both the $GrdTA$ and $GrdGenTA$ were successful in enforcing the prerequisite, suitability and work capacity constraints while reducing total cost and evenly distributing tasks. The computational demand was also reasonable, as the slower computation rate was attributed to the simulation setup, not the algorithm itself.

6.2.2 – Multiple UGV Real-Life Testing

Two small-scale tests were conducted using the 6T-2R (tasks in Table A.11 and robots in Table A.12) and 11T-2R scenarios (tasks in Table A.13 and robots in Table A.14). The purpose of these tests was to verify that the Gazebo simulations accurately depict reality.

Each small-scale test consisted of a real-life experiment and a matching simulation on Gazebo. The experimental and simulation setup is shown in Figure 6.11, showing the start locations of each robot. The robots' start positions in Figure 6.11 represent both the 6T-2R and 11T-2R scenarios, as their start locations are nearly identical. The white landmarks on the ground in Figure 6.11a correspond to the location of the different tasks for the 6T-2R scenario.

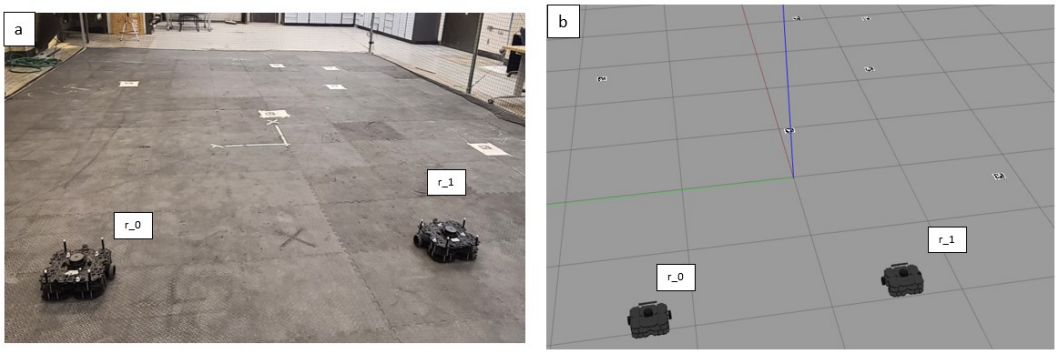


Figure 6.11 – Small Scale TCF Test: a) Real-life Wafflebots, b) Gazebo Simulation

To facilitate a direct comparison with identical task allocations, only the $GrdTA$, with its deterministic nature, was used. The task allocations calculated from the $GrdTA$ are shown in Figure 6.12. The 6T-2R scenario did not consider suitability or prerequisites. However, these

constraints were considered in the 11T-2R scenario, using a minimum suitability requirement of $s_{min} = 0.65$ and the agent-task suitabilities in Table A.9. In both test cases, the work capacity constraint was not considered; u_{i_max} was set to infinity for all robots, as it was not required to verify the gazebo simulations' accuracy.

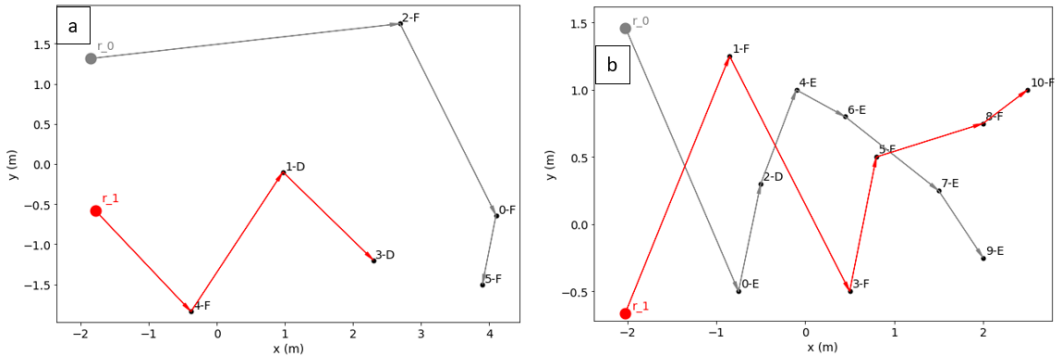


Figure 6.12 – Greedy Algorithm Task Allocation: a) 6T-2R Scenario and b) 11T-2R Scenario

The application of constraints resulted in the alternating pattern in the task allocation of the 11T-2R scenario in Figure 6.12b. Each robot can only fulfill tasks for which it is suitable and in a specific order. At the same time, the robots cannot select tasks that are just closer to their current positions. In the 6T-2R and 11T-2R scenarios, the $GrdTA$ efficiently generated optimized and constraint-compliant task allocations.

The robots' trajectories in the 6T-2R scenario, both real-life test and Gazebo simulation, are shown in Figure 6.13, and likewise with the 11T-2R scenario in Figure 6.14. In both scenarios, the robots' trajectories in real life are similar to their simulation; the robots travel from their start positions to their goal poses in the execution of their assigned tasks. The functionality of the task allocation is also completely identical between the real-life experiments and the gazebo simulations. In both real life and the gazebo simulation, the robots perform the same tasks in the same order, which validates the TCF as a solution that can be used with real robots.

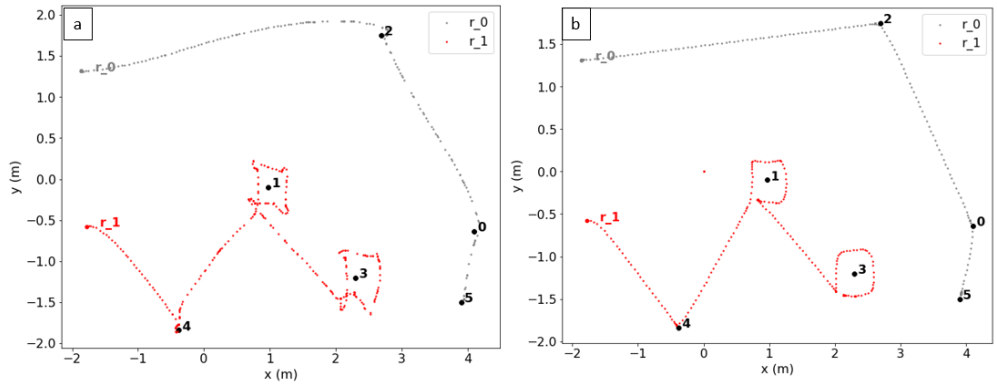


Figure 6.13 – Wafflebot Trajectories for 6T-2R Scenario: a) Real-life test, and b) Gazebo Simulation

The motion planner was also observed adjusting the robots' trajectories to avoid collision in the 11T-2R scenario due to the forced crisscrossing of trajectories imposed by the task allocation. These trajectory adjustments are exemplified in the movement of r_0 toward T_0 and r_1 toward T_1. This particular trajectory adjustment, combined with the additional tests of the RVO algorithm in Appendix A6.2, demonstrates the motion planner's efficacy.

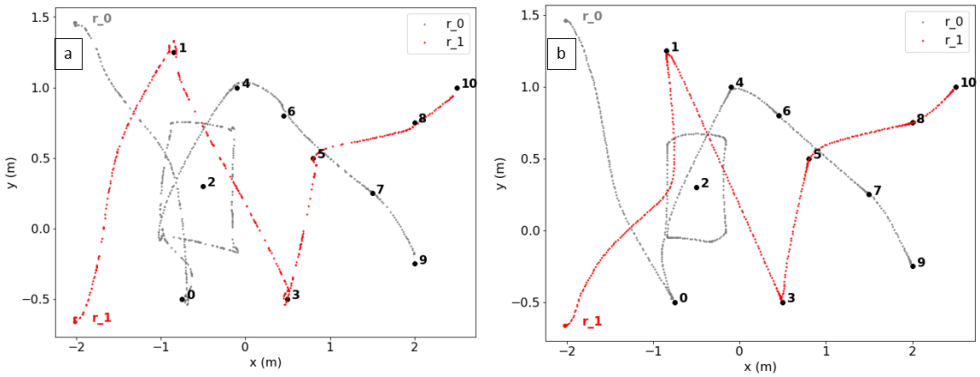


Figure 6.14 – Wafflebot Trajectories for 11T-2R Scenario: a) Real-life test, and b) Gazebo Simulation

It must also be noted that the trajectories in real life exhibited some distortion. The trajectories in the simulation were more efficient and direct. This discrepancy was attributed to differences in the pose estimate between real-life execution and gazebo simulation due to motion capture (MoCap) misalignment. This misalignment is the translational and angular difference between the rigid body representation of the Wafflebot and the real-life Wafflebot's pose. For real-life tests, MoCap was used to provide a pose estimate as the Wafflebot's onboard odometry only uses dead reckoning, which, as discussed in Section 2.1, is susceptible to drift.

The rigid body representations are generated from the placement of the IR reflectors, as discussed in Section 5.1. This discrepancy is visualized in Figure 6.15. RF_G is the global frame, RF_r is the robot body frame in reality and $RF_{r'}$ is the robot body frame according to MoCap. $RF_{r'}$ is also the robot body frame generated from the rigid body representation whose pose is measured with MoCap.

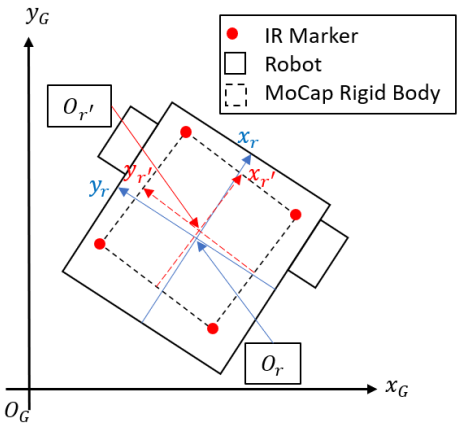


Figure 6.15 – MoCap Rigid Body Mis-alignment

It is apparent in Figure 6.15 that the MoCap pose (corresponding to $RF_{r'}$) is different than the robot's real pose (corresponding to RF_r). This difference would stay constant and would yield consistent deviations in movement as the motion planner attempts to drive $O_{r'}$ to goal poses, with the belief that $RF_{r'} = RF_r$ when in reality $RF_{r'} \neq RF_r$.

The orientation difference, in particular, is problematic as it is directly related to the direction of forward motion from a control input v, described by the kinematic model in Section 4.1. Due to the misalignment, a forwards velocity input v would drive the robot in the direction of x_r, while the kinematic model believes the robot is moving in the direction of $x_{r'}$. This discrepancy results in a drift that is eventually corrected, and the real-life robot does reach its goal pose. This drift is visible in Figure 6.13a with r_0 moving to T_2 where its path in real-life exhibits some undulation, while its path in the simulation shown in Figure 6.13b is a direct straight line.

These discrepancies in pose estimates generally result in longer trajectories and, therefore, greater usages in real-life experiments than simulations. The performance of individual robots is listed in Table 6.8 and Table 6.9 for the 6T-2R and 11T-2R scenarios, respectively. The performance of the overall system is listed in Table 6.10 and Table 6.11 for the 6T-2R and 11T-2R scenarios, respectively.

Both scenarios' robots exhibit greater usage in real-life tests than in gazebo simulations. Consequently, both scenarios have a greater total cost and global objective function value in real life than in their gazebo simulations. The imperfect task actuation cost model (discussed in Sections 6.1.1 and 6.2.1) also contributes to discrepancies between usages, cost and global objective function between real-life tests and gazebo simulation.

Table 6.8 – Robot Performance for 6T-2R Scenario

Robot	**Task Sequence**	**Usage ($GrdTA$ calculation)**	**Usage (real life)**	**Usage (gazebo simulation)**
Figure	All	Figure 6.12a	Figure 6.13a	Figure 6.13b
r_0	2,0,5	8.23	8.38	8.03
r_1	4,1,3	9.81	11.11	8.29

Table 6.9 – Robot Performance for 11T-2R Scenario

Robot	**Task Sequence**	**Usage ($GrdTA$ calculation)**	**Usage (real life)**	**Usage (gazebo simulation)**
Figure	All	Figure 6.12b	Figure 6.14a	Figure 6.14b
r_0	0, 4, 6, 7, 9	9.26	10.79	8.78
r_1	1, 3, 5, 8, 10	9.82	7.40	7.39

Table 6.10 – System Performance for 6T-2R Scenario

Scenario	**Figure**	**Total Cost (C)**	**Global Objective Function Value (F)**
$GrdTA$ Calculation	Figure 6.12a	18.04	27.85
Real Life	Figure 6.13a	19.49	30.59
Gazebo Simulation	Figure 6.13b	16.32	24.61

Table 6.11 – System Performance for 11T-2R Scenario

Scenario	**Figure**	**Total Cost** (C)	**Global Objective Function Value** (F)
$GrdTA$ Calculation	Figure 6.12b	19.08	28.90
Real Life	Figure 6.14a	18.19	28.98
Gazebo Simulation	Figure 6.14b	16.17	24.95

Overall, the real-life test wafflebots behaved similarly to the gazebo simulation wafflebots in the 6T-2R and 11T-2R scenarios. They were able to reach their goal poses in the execution of their assigned tasks in a collision-free manner. The tasks were also performed in an identical sequence between the real-life tests and the gazebo simulations. These results validate the task allocation algorithm and the robots' action models. The discrepancies between the real-life and simulation performance were primarily due to pose estimate misalignment and not because of the action model or task allocation algorithms.

6.3 – Discussion

The simulations in Sections 6.1 and 6.2 validate the $GrdGenTA$ as an effective algorithm for distributing tasks for teams of UAVs and UGVs in a highly optimal manner. These simulations also validate the proposed action model and RVO motion planning described in Chapter 4 and Appendix A3. The real-life experiments validate the gazebo simulations as accurate depictions of reality; the real robots' performance, with the wafflebot and the RB5 Drone (representing the Hector Quadcopter), was consistent with the simulation performance. Differences between real life and simulation were attributed to the experimental setup, not the task allocation algorithm or action model.

6.3.1 – Proposed Improvements to Task Allocation Algorithm

Two notable limitations were observed with the task allocation algorithms in Sections 6.1 and 6.2:

1) Usage Discrepancy: Usages and, consequently, total cost and global objective function value differed between the calculated values in the task allocation framework and the measured values from simulation and real-life experiments.

2) Time suboptimality: The task allocation algorithm's cost functions do not consider the time required to complete tasks but only the distance travelled. This can result in idling time while a robot waits for prerequisite tasks to be completed.

The cause of the usage discrepancy was explained as discrepancies between the estimated cost in the task allocation algorithm and the robots' physical trajectories. This issue can be addressed by improving the modelling of the travel costs of each task. For example, the raster search assumes the robot travels to the task location (x_j, y_j) when calculating the travel distance $\Delta s_{i,j}$ of the task cost function $c_{i,j}$. Instead of using the task location, it could use the first goal pose of the raster search. Similar adjustments can be made to the ground patrol and aerial observation tasks using the goal pose rather than task location.

However, a challenge with the travel cost estimate is motion planning considerations such as collision avoidance. This fusion between task allocation and motion planning would entail some pre-calculation of robots' trajectories before their movement to influence task allocation and improve the accuracy of cost estimates. This solution would be computationally complex and demanding while posing a significantly different challenge than the primary contributions of this thesis that focus on optimized, efficient and multi-factor task allocation for multi-robot systems. Therefore, optimizing robots' physical trajectories as part of the task allocation before execution was considered beyond the scope of this thesis but can present an opportunity for future work.

The fusion of task allocation and motion planning would improve the task allocation algorithms' usage estimates. This improvement would be most noticeable in denser environments. These environments are smaller but have more robots, significantly increasing the level of trajectory adjustment by online motion planning such as RVO. At the same time, because of more frequent online trajectory adjustments, the accuracy of offline usage estimates would be significantly degraded, which may necessitate some fusion between task allocation and motion planning. Furthermore, an inaccurate usage estimate can lead to unfeasible task allocations by violating the work capacity constraint (3.14). A robot may be assigned a set of tasks that results in a trajectory whose travel distance (usage) exceeds the robot's maximum range (work capacity) after collision avoidance.

Regarding time suboptimality, the task allocation algorithms attempt to minimize a global objective function that only considers the distance travelled by each robot moving to their tasks and in the execution of their tasks. These algorithms do not consider the time required to complete the task, which may pose issues when working with prerequisites. In particular, the prerequisite constraint is modelled such that it is satisfied if a dependent task is assigned later in the task sequence. Suppose there are two tasks, $T_j^{i,3}$ which is the 3^{th} task of robot r_i and $T_n^{k,6}$ which is the 6^{th} task of robot r_k. While $T_j^{k,6}$ is later in the sequence TS_k, it is still possible that $T_n^{k,6}$ is performed before $T_j^{i,3}$. This can occur if the tasks assigned to r_i before $T_j^{i,3}$ are more time-consuming. If prerequisites are involved, a robot may be idling while waiting to complete them if one task is required before another can be physically started. For example, a door to a building needs to be opened before tasks inside it can be performed.

A possible solution would be implementing a "re-wiring" step within the task allocation algorithm. This step would entail checking for instances of waiting time. Waiting time can occur if a robot's tasks have prerequisites performed by another robot, as discussed in Section 3.3.2, test case 1.

Suppose that usage u_i and task cost $c_{i,j}$ are representations of time, which is not an inaccurate assumption. In general, more time is required to travel a farther distance. Also, suppose a task $T_j^{f,p}$ is the p^{th} task performed by r_f and has a prerequisite of $T_g^{n,q}$, which is the q^{th} task performed by r_n. The re-wiring step would evaluate the usages of robots r_f and r_n leading up to those tasks, denoted in equations (6.1) and (6.2). It then calculates the difference between those usages and uses that information to represent waiting time (6.3).

$$u_f(p-1) = \sum_{j=1}^{N} c_{f,j} \tag{6.1}$$

$$c_{f,j} = \Delta s_{f,j} + c_j;\ if\ T_j^{f,k} \in TS_f;\ k < p$$

$$c_{f,j} = 0;\ if\ T_j^{f,k} \notin TS_f$$

$$u_n(q) = \sum_{j=1}^{N} c_{n,j} \tag{6.2}$$

$$c_{n,j} = \Delta s_{n,j} + c_j;\ if\ T_j^{n,k} \in TS_n;\ k \leq q$$

$$c_{n,j} = 0;\ if\ T_j^{n,k} \notin TS_n$$

$$\Delta u = u_n(q) - u_f(p-1) \tag{6.3}$$

$$T_g^{n,q} \in P_j$$

$$n \neq f$$

$$u_n(q) > u_f(p-1)$$

where $u_f(p-1)$ is the usage of robot r_f to perform its assigned tasks up to but not including $T_j^{f,p}$. Alternatively, $u_f(p-1)$ describes the work performed by r_f before it would perform $T_j^{f,p}$. Likewise, $u_n(q)$ is the usage of robot r_n to perform its assigned tasks up to and including $T_g^{n,q}$, which is a prerequisite for $T_j^{f,p}$. Therefore $u_n(q)$ is the work that must be performed before $T_j^{f,p}$ can be performed. Δu_f is the difference between usages $u_n(q)$ and $u_f(p-1)$. This re-wiring step would entail checking the task sequence of r_f to see if any tasks are re-arranged so they are performed before $T_j^{f,p}$ such that any increase in usage is less than Δu. This step would most likely entail using another optimization algorithm and represents an area for future work.

For example, suppose usages accurately represent time spent moving to or performing tasks. If $u_f(p-1) = 100\ minutes$ and $u_n(q) = 125\ minutes$, then it means:

- r_f would work for 100 minutes to perform all of its tasks leading up to but not including $T_j^{f,p}$
- $T_g^{n,q}$, which is a prerequisite for $T_j^{f,p}$ would not be performed by r_n until 125 minutes.
- $\Delta u = u_n(q) - u_f(p-1) = 25\ minutes$

Consequently, there would be 25 minutes of waiting time for r_f before it can perform its next task of $T_j^{f,p}$. The re-wiring step would check if r_f can perform other tasks within 25 minutes and eventually perform $T_j^{f,p}$ after $T_g^{n,q}$ is completed.

6.3.2 – Summary of Results

The $GrdGenTA$ algorithm was able to effectively distribute tasks to teams of different types of robots in the execution of different types of tasks. These robots performed the tasks as dictated by the task allocation algorithm in simulation and real life. This consistent task execution also validates the action models and RVO motion planner, which govern the robots' movements during task execution.

The $GrdGenTA$ algorithm generated task allocations that minimize the global objective function to a significant degree relative to its computational demand. In particular, the performance of $GrdGenTA$ algorithm within simulation and experimentation was consistent with its performance in evaluation (Section 3.3). As stated in Sections 6.1 and 6.2, the slower computation rate was attributed to the experimental setup, which involved executing the robots' action models and task allocation algorithms simultaneously from the same computer. If the computer did not have to run the robots' action models, the computation time of the task allocation algorithms would be significantly less.

The $GrdGenTA$ algorithm was also able to adapt to the operational factors of usage (3.14), suitability (3.15) and inter-task dependencies (3.16) as optimization constraints. The robots were only assigned suitable tasks in an admissible sequence. The proposed task allocation algorithms were validated as versatile, efficient, and multi-factor solutions.

Chapter 7 – Conclusion

7.1 – Summary

The main objective of this thesis was the development of an optimized, efficient and multi-factor task allocation algorithm. The proposed task allocation algorithm is the $GrdGenTA$ algorithm, which uses a two-step optimization process. The first step is generating a usable but suboptimal task allocation using a fast and deterministic greedy algorithm. The second step entails using a metaheuristic genetic algorithm to refine the greedy algorithm solution, which was incorporated as a member of the initial population of the genetic algorithm. This combination exploited the greedy algorithm's speed and the genetic algorithm's global search capability to efficiently generate a highly optimal task allocation.

The proposed $GrdGenTA$ algorithm considers the operational factors of task actuation cost, robot's work capacity, agent-task suitability and inter-task dependencies. In contrast, contemporary solutions for the multiple robot task allocation (MRTA) and multiple travelling salesmen problem (MTSP) typically only consider the distance between tasks/cities. Within the $GrdGenTA$ algorithm, these operational factors represent the physical conditions of tasks to be performed and the capabilities of individual robots. Overall, the $GrdGenTA$ algorithm has equal or greater optimality than contemporary solutions while also considering factors that contemporary MRTA solutions do not.

The proposed $GrdGenTA$ and $GrdTA$ algorithms were implemented using the action models in Chapter 4 and technologies in Chapter 5. The implementation involved simulations on Gazebo and experiments with real robots. The simulations demonstrated the versatility and generalizability of the $GrdGenTA$ and $GrdTA$ algorithms, as they effectively commanded teams of UAVs (Hector Quadcopters) and UGVs (Wafflebots) with different tasks and kinematic models. Once a task allocation is calculated, the robots move to the required goal poses in the execution of their assigned tasks while avoiding collision with each other in a shared workspace, which also validates their action models and the RVO motion planner.

Experiments with real robots also verified that the simulations accurately depicted reality. The RB5 drone validated the aerial localization of points of interest. In contrast, two Wafflebots were used to validate the task allocation algorithm's behaviour and UGV's real-life action model. The real-life robots behaved similarly to the robots in the Gazebo simulations, with differences attributed to the experimental setup, primarily pose estimate error and sensor noise. However, the task allocation algorithm and action models did not cause discrepancies between real life and simulations. As such, the proposed $GrdGenTA$ was also validated as a solution that can work with real robots.

7.2 – Contributions

This thesis makes the following original contributions within multi-robot systems, focusing on optimized multi-factor task allocation:

- Development of a generalized task coordination framework (TCF) to optimally assign and coordinate the execution of tasks to multi-robot teams of any type and number.
- Formulation, design, and implementation of an optimization algorithm whose performance exceeds contemporary MRTA algorithms through equal or superior optimality and reduced computation time.
- Extensions of contemporary TSP, MTSP and MRTA solutions to consider operational factors of task actuation cost, robot-task suitability matching, inter-task prerequisites, and limitations on robots' work capacity.
- Application of kinematic models, motion planning and pinhole camera model to facilitate technical proof of concept of the TCF with different types of robots and tasks and support the collaborative operation of aerial and ground robots.

This research also contributed to the publication of [1], which investigates methodologies to efficiently distribute environmental sensor nodes with multiple autonomous robots.

7.3 – Future Work

The work presented in this thesis leaves significant room for improvement in task allocation and coordination for multi-robot systems. Areas for improvement can be categorized as improvements in the task allocation and improvements in its implementation.

Improvements in the task allocation algorithm were discussed in Section 6.3.1, in which task execution time and motion planning can also be considered when generating task allocation. The task allocation algorithm only considers the sequence of task execution, not duration. This can result in idling time while a robot waits for prerequisite tasks to be completed. A proposed solution was a "re-wiring step," which attempts to check for idling time and assign tasks that can be completed within that duration. The task allocation algorithms do not consider robots' trajectories when calculating a solution. This can result in discrepancies between the calculated usages and the robots' real usages. To address these discrepancies, motion planning can be merged with task allocation to better define the expected usage and improve the optimality of the task allocation algorithm. Furthermore, additional use of graph theory in the formulation, and not just in the implementation, may yield improvements in scalability.

Online task discovery and re-allocation algorithms represent an area of improvement for the TCF, as it is limited to situations where tasks are known prior to execution. Online task discovery would enable robots to identify new tasks that were previously unknown and then be incorporated into the robots' task sequences. This incorporation can be facilitated in real-time using re-allocation algorithms to extend the proposed "re-wiring step." The online task discovery could also be facilitated using the aerial observation solutions discussed in Sections 4.2 and 4.3. A team of UGVs could be supported by a team of UAVs identifying new tasks in real-time as the UGVs are concurrently patrolling a region of interest.

Finally, the implementation can be improved by building upon the action models and motion planning solutions. In particular, more action models can be incorporated to accommodate additional types of tasks and robots. Motion planning can be improved by combining the RVO algorithm with existing motion planning solutions such as A* or RRT so that the robots can navigate in more complex environments. A more accurate pose estimate would also improve the performance of the robots. Pose estimate error was the root cause of discrepancies between simulations and real-life experiments, where the real-life robots had longer trajectories despite having the same goal poses and starting locations.

References

[1] A. Budiman *et al.*, “Dynamic Sensor Nodes Distribution with Coordinated Autonomous Vehicles for Environment Pollution Monitoring and Modeling,” presented at the 7th International Conference of Recent Trends in Environmental Science and Engineering, Ottawa, Ontario, Canada, Jun. 2023.

[2] S. Huang and G. Dissanayake, “Robot Localization: An Introduction,” in *Wiley Encyclopedia of Electrical and Electronics Engineering*, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016, pp. 1–10. doi: 10.1002/047134608X.W8318.

[3] J. Z. Sasiadek and P. Hartana, “Sensor data fusion using Kalman filter,” in *Proceedings of the Third International Conference on Information Fusion*, Paris, France: IEEE, 2000, p. WED5/19-WED5/25 vol.2. doi: 10.1109/IFIC.2000.859866.

[4] M. Ben-Ari and F. Mondada, “Robotic Motion and Odometry,” in *Elements of Robotics*, Cham: Springer International Publishing, 2018, pp. 63–93. doi: 10.1007/978-3-319-62533-1_5.

[5] D. Nister, O. Naroditsky, and J. Bergen, “Visual odometry,” in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Washington, DC, USA: IEEE, 2004, pp. 652–659. doi: 10.1109/CVPR.2004.1315094.

[6] P. Kim, H. Lee, and H. J. Kim, “Autonomous flight with robust visual odometry under dynamic lighting conditions,” *Auton Robot*, vol. 43, no. 6, pp. 1605–1622, Aug. 2019, doi: 10.1007/s10514-018-9816-4.

[7] J. Zhang and S. Singh, “LOAM: Lidar Odometry and Mapping in Real-time,” in *Robotics: Science and Systems X*, Robotics: Science and Systems Foundation, Jul. 2014. doi: 10.15607/RSS.2014.X.007.

[8] X. Zheng and J. Zhu, “Efficient LiDAR Odometry for Autonomous Driving,” *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 8458–8465, Oct. 2021, doi: 10.1109/LRA.2021.3110372.

[9] M. Sokolov, O. Bulichev, and I. Afanasyev, “Analysis of ROS-based Visual and Lidar Odometry for a Teleoperated Crawler-type Robot in Indoor Environment:,” in *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics*, Madrid, Spain: SCITEPRESS - Science and Technology Publications, 2017, pp. 316–321. doi: 10.5220/0006420603160321.

[10] A. Jha and M. Kumar, “Two wheels differential type odometry for mobile robots,” in *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*, Noida, India: IEEE, Oct. 2014, pp. 1–5. doi: 10.1109/ICRITO.2014.7014709.

[11] J. P. Barboux, “Practical Real Time Kinematic Aplications of GPS,” presented at the Third International Conference on Differential Satellite Navigation Systems, London, UK, 1994.

[12] N. H. K. Tran and V.-H. Nguyen, “An EKF-Based Fusion of Visual-Inertial Odometry and GPS for Global Robot Pose Estimation,” in *2021 6th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, Kedah, Malaysia: IEEE, Dec. 2021, pp. 1–5. doi: 10.1109/ICRAIE52900.2021.9703965.

[13] D. Cortes-Vega, H. Alazki, and J. L. Rullan-Lara, “Visual Odometry-Based Robust Control for an Unmanned Surface Vehicle under Waves and Currents in a Urban Waterway,” *JMSE*, vol. 11, no. 3, p. 515, Feb. 2023, doi: 10.3390/jmse11030515.

[14] W. Zhao, H. Liu, and X. Wang, “Vision-Based Robust Position Control for Ground Target Tracking and Hovering of Quadrotors,” in *2018 37th Chinese Control Conference (CCC)*, Wuhan: IEEE, Jul. 2018, pp. 3751–3755. doi: 10.23919/ChiCC.2018.8482672.

[15] Z. Su, C. Wang, X. Wu, Y. Dong, J. Ni, and B. He, “A Framework of Cooperative UAV-UGV System for Target Tracking,” in *2021 IEEE International Conference on Real-time*

Computing and Robotics (RCAR), Xining, China: IEEE, Jul. 2021, pp. 1260–1265. doi: 10.1109/RCAR52367.2021.9517437.
[16] P. Sturm, “Pinhole Camera Model,” in *Computer Vision*, K. Ikeuchi, Ed., Boston, MA: Springer US, 2014, pp. 610–613. doi: 10.1007/978-0-387-31439-6_472.
[17] C. Hui and C. Yousheng, “Autonomous takeoff, tracking and landing of a UAV on a moving UGV using onboard monocular vision,” in *Proceedings of the 32nd Chinese Control Conference*, 2013, pp. 5895–5901.
[18] M. Manoj krishna, M. Neelima, M. Harshali, and M. Venu Gopala Rao, “Image classification using Deep learning,” *IJET*, vol. 7, no. 2.7, p. 614, Mar. 2018, doi: 10.14419/ijet.v7i2.7.10892.
[19] B. Okorn, M. Xu, M. Hebert, and D. Held, “Learning Orientation Distributions for Object Pose Estimation,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 10580–10587. doi: 10.1109/IROS45743.2020.9340860.
[20] D. Chen, Z. Peng, and X. Ling, “A low-cost localization system based on artificial landmarks with two degree of freedom platform camera,” in *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, Bali, Indonesia: IEEE, Dec. 2014, pp. 625–630. doi: 10.1109/ROBIO.2014.7090400.
[21] A. M. G. Pinto, A. P. Moreira, and P. G. Costa, “Indoor Localization System based on Artificial Landmarks and Monocular Vision,” *TELKOMNIKA*, vol. 10, no. 4, pp. 609–620, Dec. 2012, doi: 10.12928/telkomnika.v10i4.640.
[22] A. N. Catapang and M. Ramos, “Obstacle detection using a 2D LIDAR system for an Autonomous Vehicle,” in *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, Penang, Malaysia: IEEE, 2016, pp. 441–445. doi: 10.1109/ICCSCE.2016.7893614.
[23] D. Ghorpade, A. D. Thakare, and S. Doiphode, “Obstacle Detection and Avoidance Algorithm for Autonomous Mobile Robot using 2D LiDAR,” in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*, PUNE, India: IEEE, Aug. 2017, pp. 1–6. doi: 10.1109/ICCUBEA.2017.8463846.
[24] P. Wang, “Research on Comparison of LiDAR and Camera in Autonomous Driving,” *J. Phys.: Conf. Ser.*, vol. 2093, no. 1, p. 012032, Nov. 2021, doi: 10.1088/1742-6596/2093/1/012032.
[25] M. M. Costa and M. F. Silva, “A Survey on Path Planning Algorithms for Mobile Robots,” in *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, Porto, Portugal: IEEE, Apr. 2019, pp. 1–7. doi: 10.1109/ICARSC.2019.8733623.
[26] N. Sprague, “Controlling Physical Systems,” 2019. [Online]. Available: https://w3.cs.jmu.edu/spragunr/CS354_F22/resources/pid.pdf
[27] Q.-L. Xu, T. Yu, and J. Bai, “The mobile robot path planning with motion constraints based on Bug algorithm,” in *2017 Chinese Automation Congress (CAC)*, Jinan: IEEE, Oct. 2017, pp. 2348–2352. doi: 10.1109/CAC.2017.8243168.
[28] N. He, Y. Su, jilu Guo, X. Fan, Z. Liu, and B. Wang, “Dynamic path planning of mobile robot based on artificial potential field,” in *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*, Sanya, China: IEEE, Dec. 2020, pp. 259–264. doi: 10.1109/ICHCI51889.2020.00063.
[29] S. Manyam, D. Casbeer, A. Von Moll, and F. Zachariah, “Shortest Dubins Path to a Circle.” Apr. 19, 2018. [Online]. Available: https://arxiv.org/pdf/1804.07238.pdf
[30] F. Haro and M. Torres, “A Comparison of Path Planning Algorithms for Omni-Directional Robots in Dynamic Environments,” in *2006 IEEE 3rd Latin American Robotics Symposium*, Santiago, Chile: IEEE, Oct. 2006, pp. 18–25. doi: 10.1109/LARS.2006.334319.

[31] Y. Ji *et al.*, “Adaptive Motion Planning Based on Vehicle Characteristics and Regulations for Off-Road UGVs,” *IEEE Trans. Ind. Inf.*, vol. 15, no. 1, pp. 599–611, Jan. 2019, doi: 10.1109/TII.2018.2870662.
[32] N. A. Rawashdeh and H. T. Jasim, “Multi-sensor input path planning for an autonomous ground vehicle,” in *2013 9th International Symposium on Mechatronics and its Applications (ISMA)*, Amman: IEEE, Apr. 2013, pp. 1–6. doi: 10.1109/ISMA.2013.6547399.
[33] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: an efficient probabilistic 3D mapping framework based on octrees,” *Auton Robot*, vol. 34, no. 3, pp. 189–206, Apr. 2013, doi: 10.1007/s10514-012-9321-0.
[34] A. Manjunath, P. Mehrok, R. Sharma, and A. Ratnoo, “Application of virtual target based guidance laws to path following of a quadrotor UAV,” in *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA, USA: IEEE, Jun. 2016, pp. 252–260. doi: 10.1109/ICUAS.2016.7502565.
[35] E. D. B. Medagoda and P. W. Gibbens, “Synthetic-Waypoint Guidance Algorithm for Following a Desired Flight Trajectory,” *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 2, pp. 601–606, Mar. 2010, doi: 10.2514/1.46204.
[36] Z. Hong-han, G. Liming, C. Tao, W. Lu, and Z. Xun, “Global path planning methods of UUV in coastal environment,” in *2016 IEEE International Conference on Mechatronics and Automation*, Harbin, Heilongjiang, China: IEEE, Aug. 2016, pp. 1018–1023. doi: 10.1109/ICMA.2016.7558702.
[37] C. Goerzen, Z. Kong, and B. Mettler, “A Survey of Motion Planning Algorithms from the Perspective of Autonomous UAV Guidance,” *J Intell Robot Syst*, vol. 57, no. 1–4, pp. 65–100, Jan. 2010, doi: 10.1007/s10846-009-9383-1.
[38] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles,” 2016, doi: 10.48550/ARXIV.1604.07446.
[39] Z. Ai, M. Ma, and B. Zhao, “Distributed Multi-robot Collision Avoidance under Localization Uncertainty,” in *2021 3rd International Symposium on Robotics & Intelligent Manufacturing Technology (ISRIMT)*, Changzhou, China: IEEE, Sep. 2021, pp. 1–6. doi: 10.1109/ISRIMT53730.2021.9597113.
[40] R. Masinjila, “Multirobot Localization Using Heuristically Tuned Extended Kalman Filter,” 2016, doi: 10.20381/RUOR-447.
[41] S. Yang, “Active Sensing for Collaborative Localization in Swarm Robotics,” May 2020, doi: 10.20381/RUOR-24779.
[42] D. Wilkie, J. van den Berg, and D. Manocha, “Generalized velocity obstacles,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO, USA: IEEE, Oct. 2009, pp. 5573–5578. doi: 10.1109/IROS.2009.5354175.
[43] J. van den Berg, M. Lin, and D. Manocha, “Reciprocal Velocity Obstacles for real-time multi-agent navigation,” in *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA: IEEE, May 2008, pp. 1928–1935. doi: 10.1109/ROBOT.2008.4543489.
[44] J. Alonso-Mora, A. Breitenmoser, P. Beardsley, and R. Siegwart, “Reciprocal collision avoidance for multiple car-like robots,” in *2012 IEEE International Conference on Robotics and Automation*, St Paul, MN, USA: IEEE, May 2012, pp. 360–366. doi: 10.1109/ICRA.2012.6225166.
[45] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, “Reciprocal collision avoidance with acceleration-velocity obstacles,” in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China: IEEE, May 2011, pp. 3475–3482. doi: 10.1109/ICRA.2011.5980408.

[46] J. A. Douthwaite, S. Zhao, and L. S. Mihaylova, “A Comparative Study of Velocity Obstacle Approaches for Multi-Agent Systems,” in *2018 UKACC 12th International Conference on Control (CONTROL)*, Sheffield: IEEE, Sep. 2018, pp. 289–294. doi: 10.1109/CONTROL.2018.8516848.
[47] Yan Yongjie and Zhang Yan, “Collision avoidance planning in multi-robot based on improved artificial potential field and rules,” in *2008 IEEE International Conference on Robotics and Biomimetics*, Bangkok: IEEE, Feb. 2009, pp. 1026–1031. doi: 10.1109/ROBIO.2009.4913141.
[48] Y. Li, R. Cui, and D. Xu, “Multi-robot path planning based on the developed RRT* algorithm,” presented at the 32nd Chinese Control Conference, Xi’an, China, Jul. 2018.
[49] G. T. T. Bernardo *et al.*, “A-Star Based Algorithm Applied to Target Search and Rescue by a UAV Swarm,” in *2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*, São Bernardo do Campo, Brazil: IEEE, Oct. 2022, pp. 49–54. doi: 10.1109/LARS/SBR/WRE56824.2022.9996054.
[50] M. Alberri, S. Hegazy, M. Badra, M. Nasr, O. M. Shehata, and E. I. Morgan, “Generic ROS-based Architecture for Heterogeneous Multi-Autonomous Systems Development,” in *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, Madrid: IEEE, Sep. 2018, pp. 1–6. doi: 10.1109/ICVES.2018.8519589.
[51] R. Doriya, S. Mishra, and S. Gupta, “A brief survey and analysis of multi-robot communication and coordination,” in *International Conference on Computing, Communication & Automation*, Greater Noida, India: IEEE, May 2015, pp. 1014–1021. doi: 10.1109/CCAA.2015.7148524.
[52] G. A. Korsah, A. Stentz, and M. B. Dias, “A comprehensive taxonomy for multi-robot task allocation,” *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, Oct. 2013, doi: 10.1177/0278364913496484.
[53] A. Khamis, A. Hussein, and A. Elmogy, “Multi-robot Task Allocation: A Review of the State-of-the-Art,” in *Cooperative Robots and Sensor Networks 2015*, vol. 604, A. Koubâa and J. R. Martínez-de Dios, Eds., in Studies in Computational Intelligence, vol. 604. , Cham: Springer International Publishing, 2015, pp. 31–51. doi: 10.1007/978-3-319-18299-5_2.
[54] O. Cheikhrouhou and I. Khoufi, “A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy,” *Computer Science Review*, vol. 40, p. 100369, May 2021, doi: 10.1016/j.cosrev.2021.100369.
[55] T. Bektas, “The multiple traveling salesman problem: an overview of formulations and solution procedures,” *Omega*, vol. 34, no. 3, pp. 209–219, Jun. 2006, doi: 10.1016/j.omega.2004.10.004.
[56] A. Bockmayr and K. Reinert, “Concept: Types of algorithms,” in *Discrete Math for Bioinformatics*, 2010, pp. 1001–1011. [Online]. Available: http://www.mi.fu-berlin.de/wiki/pub/ABI/DiscretMathWS10/runtime.pdf
[57] R. Bellman, “Dynamic Programming Treatment of the Travelling Salesman Problem,” *J. ACM*, vol. 9, no. 1, pp. 61–63, Jan. 1962, doi: 10.1145/321105.321111.
[58] “A note on reproducibility of CPLEX runs.” Accessed: Mar. 29, 2023. [Online]. Available: https://www.ibm.com/support/pages/note-reproducibility-cplex-runs
[59] “MinMax multiple-TSP.” Accessed: Mar. 08, 2023. [Online]. Available: https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/
[60] Z. Zhang, S. Ma, and X. Jiang, “Research on Multi-Objective Multi-Robot Task Allocation by Lin–Kernighan–Helsgaun Guided Evolutionary Algorithms,” *Mathematics*, vol. 10, no. 24, p. 4714, Dec. 2022, doi: 10.3390/math10244714.
[61] V. Pacheco-Velencia, N. Vakhania, J. Hernandez, and J. Hernandez-Gomez, “A Fast Algorithm for Euclidean Bounded Single-Depot Multiple Traveling Salesman Problem†,”

presented at the International Online Conference on Algorithms, 2021. [Online]. Available: https://sciforum.net/manuscripts/10898/manuscript.pdf
[62] “IBM Client Case studies.” [Online]. Available: https://www.ibm.com/downloads/cas/VXLVVLVZ
[63] R. Necula, M. Breaban, and M. Raschip, “Tackling the Bi-criteria Facet of Multiple Traveling Salesman Problem with Ant Colony Systems,” in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, Vietri sul Mare, Italy: IEEE, Nov. 2015, pp. 873–880. doi: 10.1109/ICTAI.2015.127.
[64] X. Meng, J. Li, M. Zhou, X. Dai, and J. Dou, “Population-Based Incremental Learning Algorithm for a Serial Colored Traveling Salesman Problem,” *IEEE Trans. Syst. Man Cybern, Syst.*, vol. 48, no. 2, pp. 277–288, Feb. 2018, doi: 10.1109/TSMC.2016.2591267.
[65] E. Kivelevitch, K. Cohen, and M. Kumar, “A Market-based Solution to the Multiple Traveling Salesmen Problem,” *J Intell Robot Syst*, vol. 72, no. 1, pp. 21–40, Oct. 2013, doi: 10.1007/s10846-012-9805-3.
[66] V. M. Tran and T. H. N. Vu, “Leveraging CPLEX to Solve the Vehicle Routing Problem with Time Windows,” in *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*, Bangkok, Thailand: IEEE, Nov. 2021, pp. 1–6. doi: 10.1109/KSE53942.2021.9648591.
[67] H. Zhu, D. Liu, S. Zhang, S. Teng, and Y. Zhu, “Solving the Group Multirole Assignment Problem by Improving the ILOG Approach,” *IEEE Trans. Syst. Man Cybern, Syst.*, vol. 47, no. 12, pp. 3418–3424, Dec. 2017, doi: 10.1109/TSMC.2016.2566680.
[68] B. Dimitri, “Auction Algorithms.” Laboratory for Information and Decision Systems. [Online]. Available: https://web.mit.edu/dimitrib/www/Auction_Encycl.pdf
[69] C. Wei, Z. Ji, and B. Cai, “Particle Swarm Optimization for Cooperative Multi-Robot Task Allocation: A Multi-Objective Approach,” *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 2530–2537, Apr. 2020, doi: 10.1109/LRA.2020.2972894.
[70] V. Arya, A. Goyal, and V. Jaiswal, “An Optimal Solution to Multiple Travelling Salesperson Problem using Modified Genetic Algorithm,” *International Journal of Application or Innovation in Engineering and Management*, vol. 3, no. 1, Jan. 2014.
[71] X. Li, Z. Liu, and Y. Zhang, “A Novel Improved Ant Colony Algorithm for Multi-Robot Task Allocation,” in *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, Chongqing, China: IEEE, Dec. 2018, pp. 1629–1633. doi: 10.1109/ITOEC.2018.8740438.
[72] D. Bookstaber, “Simulated Annealing for the Traveling Salesman Problem,” 1997, doi: 10.13140/RG.2.2.26488.39682.
[73] S. C. Ozcan and H. Kaya, “An Analysis of Travelling Salesman Problem Utilizing Hill Climbing Algorithm for a Smart City Touristic Search on OpenStreetMap (OSM),” in *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, Ankara, Turkey: IEEE, Oct. 2018, pp. 1–5. doi: 10.1109/ISMSIT.2018.8567045.
[74] A. G. Gad, “Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review,” *Arch Computat Methods Eng*, vol. 29, no. 5, pp. 2531–2561, Aug. 2022, doi: 10.1007/s11831-021-09694-4.
[75] X. Kong, Y. Gao, T. Wang, J. Liu, and W. Xu, “Multi-robot Task Allocation Strategy based on Particle Swarm Optimization and Greedy Algorithm,” in *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Chongqing, China: IEEE, May 2019, pp. 1643–1646. doi: 10.1109/ITAIC.2019.8785472.
[76] Y. Chen, C. Du, J. Chen, and W. Yu, “Cooperative Task Allocation of Multiple UAVs Based on Greedy Algorithm,” in *2021 IEEE 4th International Conference on Computer and Communication Engineering Technology (CCET)*, Beijing, China: IEEE, Aug. 2021, pp. 408–413. doi: 10.1109/CCET52649.2021.9544170.

[77] S. Ejim, “Implementation of Greedy Algorithm in Travel Salesman Problem,” 2016, doi: 10.13140/RG.2.2.23921.48485.
[78] A. Vié, “Qualities, Challenges and Future of Genetic Algorithms,” *SSRN Journal*, 2020, doi: 10.2139/ssrn.3726035.
[79] M. Rohini., B. Manohari., and S. Adhithyan., “Genetic Algorithm Based Optimization in Solving Multi Robot Task Allocation Problems,” in *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, Greater Noida, India: IEEE, Nov. 2022, pp. 344–350. doi: 10.1109/ICCCIS56430.2022.10037687.
[80] M. Soleimanpour-Moghadam and H. Nezamabadi-Pour, “Discrete Genetic Algorithm for Solving Task Allocation of Multi-robot Systems,” in *2020 4th Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, Mashhad, Iran: IEEE, Sep. 2020, pp. 006–009. doi: 10.1109/CSIEC49655.2020.9237316.
[81] H. Zhou, M. Song, and W. Pedrycz, “A comparative study of improved GA and PSO in solving multiple traveling salesmen problem,” *Applied Soft Computing*, vol. 64, pp. 564–580, Mar. 2018, doi: 10.1016/j.asoc.2017.12.031.
[82] M. Dorigo and T. Stützle, “Ant Colony Optimization: Overview and Recent Advances,” in *Handbook of Metaheuristics*, vol. 146, M. Gendreau and J.-Y. Potvin, Eds., in International Series in Operations Research & Management Science, vol. 146. , Boston, MA: Springer US, 2010, pp. 227–263. doi: 10.1007/978-1-4419-1665-5_8.
[83] Y. Huang, Y. Zhang, and H. Xiao, “Multi-robot system task allocation mechanism for smart factory,” in *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Chongqing, China: IEEE, May 2019, pp. 587–591. doi: 10.1109/ITAIC.2019.8785546.
[84] L. Liu and D. Shell, “Optimal Market-based Multi-Robot Task Allocation via Strategic Pricing,” in *Robotics: Science and Systems IX*, Robotics: Science and Systems Foundation, Jun. 2013. doi: 10.15607/RSS.2013.IX.033.
[85] M. Badreldin, A. Hussein, and A. Khamis, “A Comparative Study between Optimization and Market-Based Approaches to Multi-Robot Task Allocation,” *Advances in Artificial Intelligence*, vol. 2013, pp. 1–11, Nov. 2013, doi: 10.1155/2013/256524.
[86] O. Al-Buraiki and P. Payeur, “Task Allocation in Multi-Robot Systems Based on the Suitability Level of the Individual Agents,” in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, Lyon, France: IEEE, Aug. 2021, pp. 209–214. doi: 10.1109/CASE49439.2021.9551466.
[87] H. Bruyninckx, “Bayesian Probability.” Dept. of Mechancial Engineering, K.U.Leuven, Nov. 2002. [Online]. Available: https://people.cs.kuleuven.be/~danny.deschreye/urks2to4_text.pdf
[88] W. Wu, “Object Recognition with Progressive Refinement for Collaborative Robots Task Allocation,” Dec. 2020, doi: 10.20381/RUOR-25803.
[89] J. Li, Q. Sun, M. Zhou, X. Yu, and X. Dai, “Colored Traveling Salesman Problem and Solution,” *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 9575–9580, 2014, doi: 10.3182/20140824-6-ZA-1003.01403.
[90] M. A. Al-Furhud and Z. Hussain, “Genetic Algorithms for the Multiple Travelling Salesman Problem,” *IJACSA*, vol. 11, no. 7, 2020, doi: 10.14569/IJACSA.2020.0110768.
[91] S. Sariel, T. Balch, and N. Erdogan, “Incremental multi-robot task selection for resource constrained and interrelated tasks,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, USA: IEEE, Oct. 2007, pp. 2314–2319. doi: 10.1109/IROS.2007.4399519.
[92] B. M. Baker and M. A. Ayechew, “A genetic algorithm for the vehicle routing problem,” *Computers & Operations Research*, vol. 30, no. 5, pp. 787–800, Apr. 2003, doi: 10.1016/S0305-0548(02)00051-5.

[93] A. Király and J. Abonyi, “Optimization of Multiple Traveling Salesmen Problem by a Novel Representation Based Genetic Algorithm,” in *Intelligent Computational Optimization in Engineering*, vol. 366, M. Köppen, G. Schaefer, and A. Abraham, Eds., in Studies in Computational Intelligence, vol. 366. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 241–269. doi: 10.1007/978-3-642-21705-0_9.
[94] R. Boeh, T. Hanne, and R. Dornberger, “A Comparison of Linear Rank and Tournament for Parent Selection in a Genetic Algorithm Solving a Dynamic Travelling Salesman Problem,” in *2022 9th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, Toronto, ON, Canada: IEEE, Nov. 2022, pp. 97–102. doi: 10.1109/ISCMI56532.2022.10068458.
[95] A. Otman and A. Jaafar, “A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem,” *International Journal of Computer Applications*, vol. 31, no. 11, Oct. 2011, [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1203/1203.3097.pdf
[96] S. Malik and S. Wadhwa, “Preventing Premature Convergence in Genetic Algorithm Using DGCA and Elitist Technique,” vol. 6, no. 4, p. 410 to 418, Jun. 2014.
[97] F. Yu, X. Fu, H. Li, and G. Dong, “Improved Roulette Wheel Selection-Based Genetic Algorithm for TSP,” in *2016 International Conference on Network and Information Systems for Computers (ICNISC)*, Wuhan: IEEE, Apr. 2016, pp. 151–154. doi: 10.1109/ICNISC.2016.041.
[98] “eil51, 5 Salesmen Benchmark Solution.” [Online]. Available: https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/figures/eil51(m=5)-GraphicTours.png
[99] “eil51, 3 Salesmen Benchmark Solution.” [Online]. Available: https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/figures/eil51(m=3)-GraphicTours.png
[100] “eil51, 7 Salesmen Benchmark Solution.” [Online]. Available: https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/figures/eil51(m=7)-GraphicTours.png
[101] “berlin52, 7 Salesmen Benchmark Solution.” [Online]. Available: https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/figures/berlin52(m=5)-GraphicTours.png
[102] “eil76, 7 Salesmen Benchmark Solution.” [Online]. Available: https://profs.info.uaic.ro/~mtsplib/MinMaxMTSP/figures/eil76(m=7)-GraphicTours.png
[103] “R Compute GINI Coefficient for Discrete Samples or Discrete Random Variable.” Accessed: Jan. 29, 2023. [Online]. Available: https://fanwangecon.github.io/R4Econ/math/func_ineq/htmlpdfr/fs_gini_disc.html
[104] B. Siciliano, Ed., “Chapter 2 - Kinematics,” in *Robotics: modelling, planning and control*, in Advanced textbooks in control and signal processing. , London: Springer, 2009, pp. 39–57.
[105] E. Maulana, M. A. Muslim, and A. Zainuri, “Inverse kinematics of a two-wheeled differential drive an autonomous mobile robot,” in *2014 Electrical Power, Electronics, Communicatons, Control and Informatics Seminar (EECCIS)*, Malang, Indonesia: IEEE, Aug. 2014, pp. 93–98. doi: 10.1109/EECCIS.2014.7003726.
[106] A. Xu, C. Viriyasuthee, and I. Rekleitis, “Optimal complete terrain coverage using an Unmanned Aerial Vehicle,” in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China: IEEE, May 2011, pp. 2513–2519. doi: 10.1109/ICRA.2011.5979707.
[107] A. Rosebrock, “AprilTag with Python,” PyImageSearch. Accessed: Nov. 24, 2022. [Online]. Available: https://pyimagesearch.com/2020/11/02/apriltag-with-python/
[108] “apriltag.” [Online]. Available: https://pypi.org/project/apriltag/
[109] “AprilTag.” [Online]. Available: https://april.eecs.umich.edu/software/apriltag
[110] S. H. Qi, J. Li, Z. P. Sun, J. T. Zhang, and Y. Sun, “Distance Estimation of Monocular Based on Vehicle Pose Information,” *J. Phys.: Conf. Ser.*, vol. 1168, p. 032040, Feb. 2019, doi: 10.1088/1742-6596/1168/3/032040.

[111] “Scikit Learn Clustering.” [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html#k-means
[112] “Motive: Optical Motion Capture Software.” [Online]. Available: https://optitrack.com/software/motive/
[113] “Prime x 13 - In Depth,” OptiTrack. Accessed: Apr. 13, 2023. [Online]. Available: http://optitrack.com/cameras/primex-13/index.html
[114] “OptiTrack Documentation.” Accessed: Feb. 09, 2023. [Online]. Available: https://docs.optitrack.com/
[115] “Documentation - ROS Wiki.” Accessed: Oct. 27, 2022. [Online]. Available: https://wiki.ros.org/
[116] “Q Ground Control.” [Online]. Available: http://qgroundcontrol.com/
[117] R. Wilson, “Definitions and Examples,” in *Introduction to Graph Theory*, Fourth., Prentice Hall, 1996, pp. 8–21. [Online]. Available: https://www.maths.ed.ac.uk/~v1ranick/papers/wilsongraph.pdf
[118] J. Alexandre and S. Martins, “MRSLAM - Multi-Robot Simultaneous Localization and Mapping,” Universidade de Coimbra, 2013.
[119] M. Garzón *et al.*, “Using ROS in Multi-robot Systems: Experiences and Lessons Learned from Real-World Field Tests,” in *Robot Operating System (ROS)*, vol. 707, A. Koubaa, Ed., in Studies in Computational Intelligence, vol. 707. , Cham: Springer International Publishing, 2017, pp. 449–483. doi: 10.1007/978-3-319-54927-9_14.
[120] ROBOTIS e-Manual. Accessed: Oct. 27, 2022. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/
[121] “Hector Quadrotor.” [Online]. Available: https://github.com/RAFALAMAO/hector-quadrotor-noetic
[122] “Qualcomm Flight™ RB5 5G Platform Drone Reference Design,” ModalAI, Inc. Accessed: Oct. 27, 2022. [Online]. Available: https://www.modalai.com/products/qualcomm-flight-rb5-5g-platform-reference-drone
[123] “PX4 Documentation.” [Online]. Available: https://docs.px4.io/main/en/
[124] “mavros - ROS Wiki.” Accessed: Oct. 27, 2022. [Online]. Available: http://wiki.ros.org/mavros
[125] “Introduction · MAVLink Developer Guide.” Accessed: Nov. 21, 2022. [Online]. Available: https://mavlink.io/en/
[126] “Gazebo.” Open Source Robotics Foundation. [Online]. Available: https://classic.gazebosim.org/
[127] “WSL2 setup.” [Online]. Available: https://github.com/albud187/wsl2/blob/main/ubuntu_gui_youtube
[128] “SciPy Linear Sum Assignment.” [Online]. Available: https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.linear_sum_assignment.html
[129] J. A. Douthwaite, S. Zhao, and L. S. Mihaylova, “Velocity Obstacle Approaches for Multi-Agent Collision Avoidance,” *Un. Sys.*, vol. 07, no. 01, pp. 55–64, Jan. 2019, doi: 10.1142/S2301385019400065.
[130] M. Sainte Catherine and E. Lucet, “A modified Hybrid Reciprocal Velocity Obstacles approach for multi-robot motion planning without communication,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 5708–5714. doi: 10.1109/IROS45743.2020.9341377.
[131] “data/m500/holybro2216_880kv_holybroblackprop_02252022.csv · master · voxl / Flight Core and PX4 / dyno_data · GitLab,” GitLab. Accessed: Oct. 27, 2022. [Online]. Available: https://gitlab.com/voxl-public/flight-core-px4/dyno_data/-/blob/master/data/m500/holybro2216_880kv_holybroblackprop_02252022.csv

[132] “RB5 Drone Camera Extrinsic Configuration File.” [Online]. Available: https://gitlab.com/voxl-public/voxl-sdk/utilities/voxl-mpa-tools/-/blob/master/misc_files/usr/share/modalai/extrinsic_configs/rb5_flight_v1.conf

Appendices

A1 – Scenario and Parameter Information for Test Cases

Table A.1 – Optimization Parameters for Task Allocation Algorithms

Variable	Parameter	Value	Unit of Measurement
n_p	Genetic Algorithm Population Size	10	-
p_m	Mutation Probability	0.7	-
p_c	Crossover Probability	0.7	-
s_{min}	Minimum Suitability Requirement	0.65	-

Table A.2 – Robots' Constant Parameters

Variable	Parameter	Value	Unit of Measurement
$R_i\ (\alpha)$	Inflated Radius for Type α Robots	0.7	m
$V_{max}(\alpha)$	Maximum Speed for type α Robots	0.5	m/s
$\omega_{max}(\alpha)$	Maximum Yaw Rate for type α Robots	2.5	rad/s
$R_i\ (\beta, \varepsilon, \tau)$	Inflated Radius for Robots of Types β, ε, τ	0.3	m
$V_{max}(\beta, \varepsilon, \tau)$	Maximum Speed for Robots of Types β, ε, τ	0.26	m/s
$\omega_{max}(\beta, \varepsilon, \tau)$	Maximum Yaw Rate for Robots of Types β, ε, τ	1.82	rad/s

Table A.3 – Tasks for 31T-6R Scenario

Index (j)	**Task Location X** (x_j)	**Task Location Y** (y_j)	**Task Type** (t_j)	**Actuation Cost** (c_j)	**Prerequisites** (P_j)
0	-4.711	5.841	F	0	[30]
1	-6.616	6.258	E	0	[30]
2	-6.001	4.576	F	0	[30]
3	3.027	0.691	E	0	[0]
4	-2.314	2.332	F	0	[]
5	3.109	5.388	E	0	[16]
6	-1.226	-2.447	F	0	[]
7	5.03	-2.214	E	0	[28]
8	7.573	3.331	F	0	[29]
9	5.512	2.319	F	0	[29]
10	7.476	1.469	E	0	[29]
11	-6.839	0.844	F	0	[27]
12	-4.949	0.777	E	0	[27]
13	-5.1	-0.626	F	0	[27]
14	-1.127	5.146	E	0	[16]
15	0.612	7.144	F	0	[]
16	1.025	3.758	D	16	[]
17	3.642	-2.163	F	0	[28]
18	5.487	-3.395	E	0	[28]
19	4.232	7.428	F	0	[]
20	-5.515	2.709	E	0	[]
21	9.653	6.37	F	0	[]
22	1.133	2.031	E	0	[16]
23	-2.649	7.655	F	0	[]
24	7.307	8.483	E	0	[]
25	-3.627	-7.308	F	0	[]
26	2.474	-7.203	E	0	[]
27	-5.500	0.000	D	12	[]
28	4.500	-2.500	D	12	[]
29	6.500	2.500	D	12	[]
30	-5.500	5.500	D	12	[]

Table A.4 – Robots for 31T-6R Scenario

Robot Agent	**Robot Type**	**Start Position** (x, y)	**Work Capacity** (u_{i_max})
r_0	β	(5, -8)	Infinity
r_1	ε	(6.5, -8)	Infinity
r_2	τ	(8, -8)	Infinity
r_3	β	(5, -6.5)	Infinity
r_4	ε	(6.5, -6.5)	Infinity
r_5	τ	(8, -6.5)	Infinity

Table A.5 – Tasks for 16T-3R Scenario

Index (j)	**Task Location X** (x_j)	**Task Location Y** (y_j)	**Task Type** (t_j)	**Actuation Cost** (c_j)	**Prerequisites** (P_j)
0	-3.32	-3.81	C	24	[]
1	2.18	1.01	C	24	[0]
2	4.91	6	C	24	[10]
3	-2.04	7.92	C	24	[]
4	12.52	4.28	C	24	[]
5	10.02	-12.01	B	0	[]
6	-11.83	0.489	B	0	[4]
7	-10.3	6.5	B	0	[]
8	-8	9.59	B	0	[]
9	-4.95	3.92	B	0	[0]
10	10.1	7.2	B	0	[]
11	10.51	0	B	0	[]
12	4.4	-3.4	B	0	[]
13	1.98	-8.3	B	0	[]
14	7.7	-6.4	B	0	[5]
15	12.56	-9.37	B	0	[5]

Table A.6 – Robots for 16T-3R Scenario

Robot Agent	**Robot Type**	**Start Position** (x, y)	**Work Capacity** (u_{i_max})
r_0	α	(-12, -12)	Infinity
r_1	δ	(-12, -15)	Infinity
r_2	ρ	(-15, -12)	Infinity

Table A.7 – Tasks for 24T-5R Scenario

Index (j)	**Task Location X** (x_j)	**Task Location Y** (y_j)	**Task Type** (t_j)	**Actuation Cost** (c_j)	**Prerequisites** (P_j)
0	2.982	-2.928	D	2	[1]
1	-2.107	-7.338	D	4	[]
2	-7.177	-9.795	E	0	[10]
3	-9.085	-5.043	F	0	[]
4	-7.673	2.794	D	2	[]
5	-2.921	2.007	D	4	[1,18]
6	6.461	8.59	E	0	[]
7	-3.828	9.265	F	0	[17]
8	3.253	-10.438	D	2	[1]
9	9.297	-5.118	D	4	[]
10	9.002	3.905	E	0	[]
11	2.241	10.508	F	0	[10]
12	-3.756	-13.068	D	2	[]
13	-5.449	-2.881	D	4	[18,22]
14	6.218	-7.514	E	0	[]
15	8.49	-9.776	F	0	[]
16	4.226	4.587	D	2	[10]
17	-8.431	8.449	D	4	[]
18	10.807	9.2	E	0	[1]
19	8.369	-0.391	F	0	[]
20	-8.875	-0.335	D	2	[13,16]
21	1.472	-6.577	D	4	[]
22	-0.468	6.415	E	0	[1]
23	10.76	1.444	F	0	[17]

Table A.8 – Robots for 24T-5R Scenario

Robot Agent	**Robot Type**	**Start Position** (x, y)	**Work Capacity** $(u_{i\ max})$
r_0	β	(0, 0)	Infinity
r_1	ε	(-1.5, -1.5)	Infinity
r_2	τ	(1.5, 1.5)	Infinity
r_3	β	(-1.5, 1.5)	Infinity
r_4	ε	(1.5, -1.5)	infinity

Table A.9 – Agent Task Suitabilities for all Test Scenarios

	Task Type (t_j)						
Robot Type (t_i)	A1	A2	B	C	D	E	F
α	0	1	1	1	0	0	0
δ	0	1	0.5	0.9	0	0	0
ρ	0	1	0.9	0.5	0	0	0
β	1	0	0	0	0.90	0.70	0.50
ε	1	0	0	0	0.70	0.50	0.90
τ	1	0	0	0	0.50	0.90	0.70

Table A.10 – Stationary Observation Tasks for Aerial Observation Proof of Concept

Order (k)	**Task Location X** (x_j)	**Task Location Y** (y_j)	**Observation Angle** (radians)
1	-0.38	-1.84	3.14
2	-0.38	-1.84	2.40
3	-0.38	-1.84	1.57
4	0.97	-0.10	3.14
5	0.97	-0.10	2.40
6	0.97	-0.10	1.57
7	2.69	1.75	3.14
8	2.69	1.75	-2.40
9	2.69	1.75	-1.57
10	4.10	-0.69	3.14
11	4.10	-0.69	2.40
12	4.10	-0.69	1.57
13	3.90	-1.50	1.57
14	3.90	-1.50	2.40
15	3.90	-1.50	3.14
16	2.30	-1.20	3.14
17	2.30	-1.20	2.40
18	2.30	-1.20	1.57

Table A.11 – Tasks for 6T-2R scenario

Index (j)	**Task Location X** (x_j)	**Task Location Y** (y_j)	**Task Type** (t_j)	**Actuation Cost** (c_j)	**Prerequisites** (P_j)
0	4.10	-0.64	F	0	[]
1	0.97	-0.10	D	2	[]
2	2.69	1.75	F	0	[]
3	2.30,	-1.20	D	2	[]
4	-0.38,	-1.84	F	0	[]
5	3.90	-1.50	F	0	[]

Table A.12 – Robots for 6T-2R Scenario

Robot Agent	**Robot Type**	**Start Position** (x, y)	**Work Capacity** (u_{i_max})
r_0	β	(-1.857, 1.315)	Infinity
r_1	ε	(-1.784, -0.581)	infinity

Table A.13 – Tasks for 11T-2R scenario

Index (j)	**Task Location X** (x_j)	**Task Location Y** (y_j)	**Task Type** (t_j)	**Actuation Cost** (c_j)	**Prerequisites** (P_j)
0	-0.75	-0.5	E	0	[]
1	-0.85	1.25	F	0	[]
2	-0.5	0.3	D	2.8	[]
3	0.5	-0.5	F	0	[]
4	-0.1	1	E	0	[0,2]
5	0.8	0.5	F	0	[2]
6	0.45	0.8	E	0	[4]
7	1.5	0.25	E	0	[5,6]
8	2	0.75	F	0	[7]
9	2	-0.25	E	0	[7]
10	2.5	1	F	2	[]

Table A.14 – Robots for 11T-2R Scenario

Robot Agent	**Robot Type**	**Start Position** (x, y)	**Work Capacity** (u_{i_max})
r_0	β	(-2.021, 1.461)	Infinity
r_1	ε	(-2.023, -0.661)	infinity

A2 – Optimality Evaluation Data

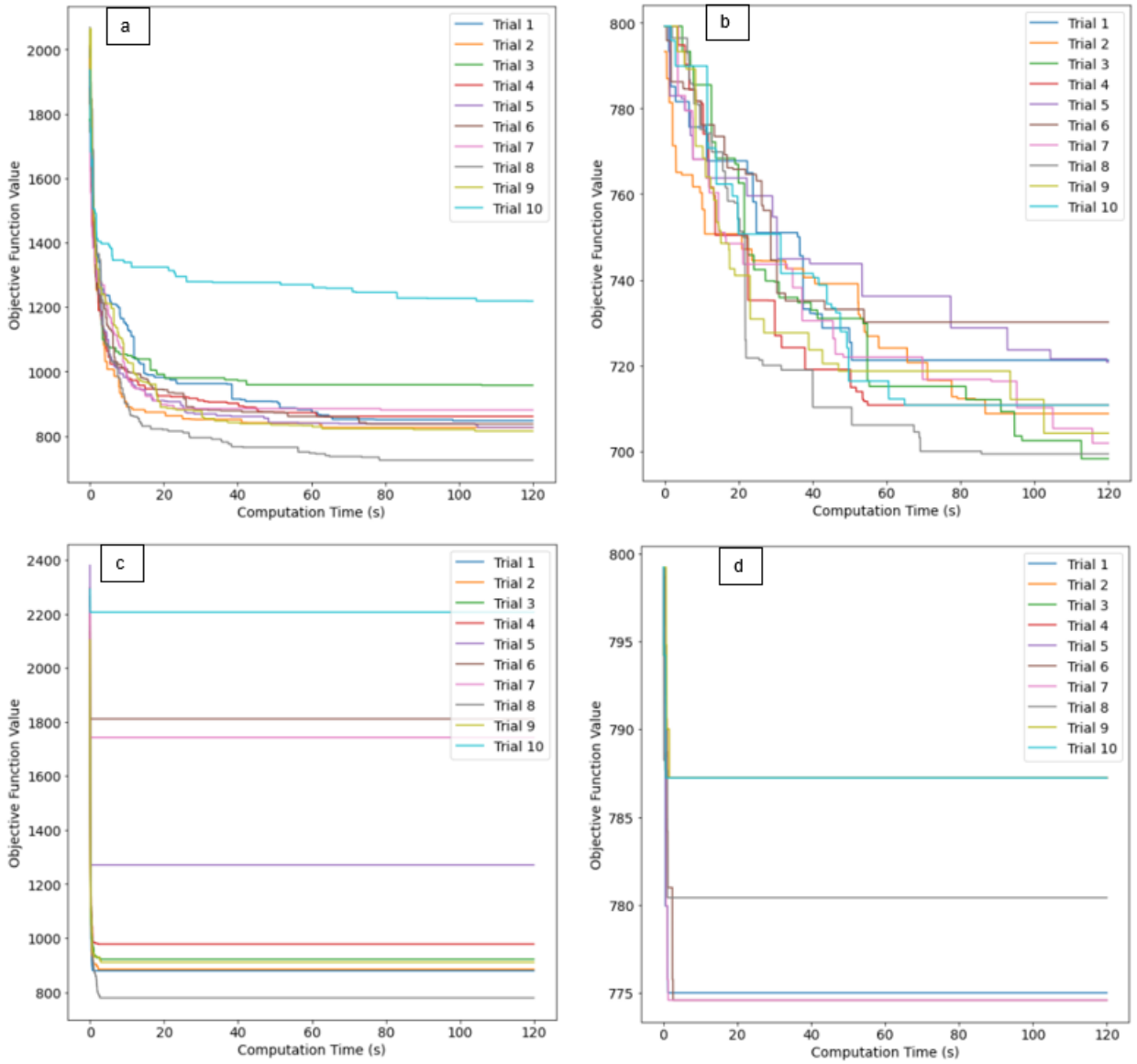


Figure A.1 – Global Objective Function Value over Computation Time for eil51 with 5 Salesmen: a) Genetic Algorithm Only, b) Greedy + Genetic Algorithm, c) Hill Climbing Only, and d) Greedy + Hill Climbing

Table A.15 – Task Allocation Algorithms' Objective Function Values at Final Iteration For eil51, 5 Salesmen Problem

	Objective Function (F) Value at 120 seconds			
Trial	$GenTA$	$GrdGenTA$	$HCTA$	$GrdHCTA$
1	847.63	721.21	878.91	775.01
2	823.29	708.72	883.89	787.23
3	952.54	698.19	921.92	787.23
4	860.50	710.72	977.68	787.23
5	826.91	707.04	1270.99	787.23
6	837.19	730.07	1811.37	774.59
7	880.64	701.82	1742.34	774.59
8	724.78	699.32	778.47	780.42
9	814.90	700.79	910.39	787.23
10	1218.70	710.72	2206.19	787.23

Table A.16 - Task Allocation Algorithms' Iterations Completed at 120 Seconds

	Iterations Completed in 120 seconds			
Trial	$GenTA$	$GrdGenTA$	$HCTA$	$GrdHCTA$
1	9739	9469	222169	208568
2	9969	9296	209748	208014
3	10057	9094	209369	206073
4	9308	9346	203786	208323
5	9986	9594	205291	207515
6	9537	9398	214053	208609
7	9646	9537	209886	208181
8	9991	9841	209858	207943
9	9710	9749	205441	206356
10	9973	9759	207749	203774

A3 – Motion Planning with Reciprocal Velocity Obstacles

This appendix describes the motion planner that generates collision-free velocities v_i^{opt} to drive robots to their desired positions.

A3.1 – Motion Planning Scheme

For this thesis, the motion planner only considered 2D motion for both UAVs and UGVs but can be generalized to 3D motion. The robots employ the method of reciprocal velocity obstacles (RVO) for multi-robot navigation [129]. Moreover, only the other robots are considered as obstacles in this thesis. In contrast, potential obstacles contained in the environment are not taken into consideration, given that environment modelling is beyond the scope of this work.

This approach calculates the velocities required to drive the robots to their destinations while avoiding collision with each other in a shared environment. The following velocities are defined:

- v_i^{id} is the ideal velocity of robot r_i. It would result in a straight-line path towards the goal position at maximum allowable speed.

$$v_i^{id} = \begin{bmatrix} \dot{x}_{id} \\ \dot{y}_{id} \end{bmatrix} \tag{A.1}$$

- v_i^{opt} is the optimal velocity of robot i, calculated from RVO. It is a velocity that does not result in a collision with other robots and has minimal difference from v_i^{id}.

$$v_i^{opt} = \begin{bmatrix} \dot{x}_{opt} \\ \dot{y}_{opt} \end{bmatrix} \tag{A.2}$$

- v_i is the actual velocity of robot i as per the robot's state estimate.

$$v_i = \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} = \dot{S} \tag{A.3}$$

The system architecture containing states and velocities is shown in Figure A.2 for robot r_i and robot r_k. This architecture can be expanded to larger teams of robots. Figure A.2 only shows two robots for ease of visualization.

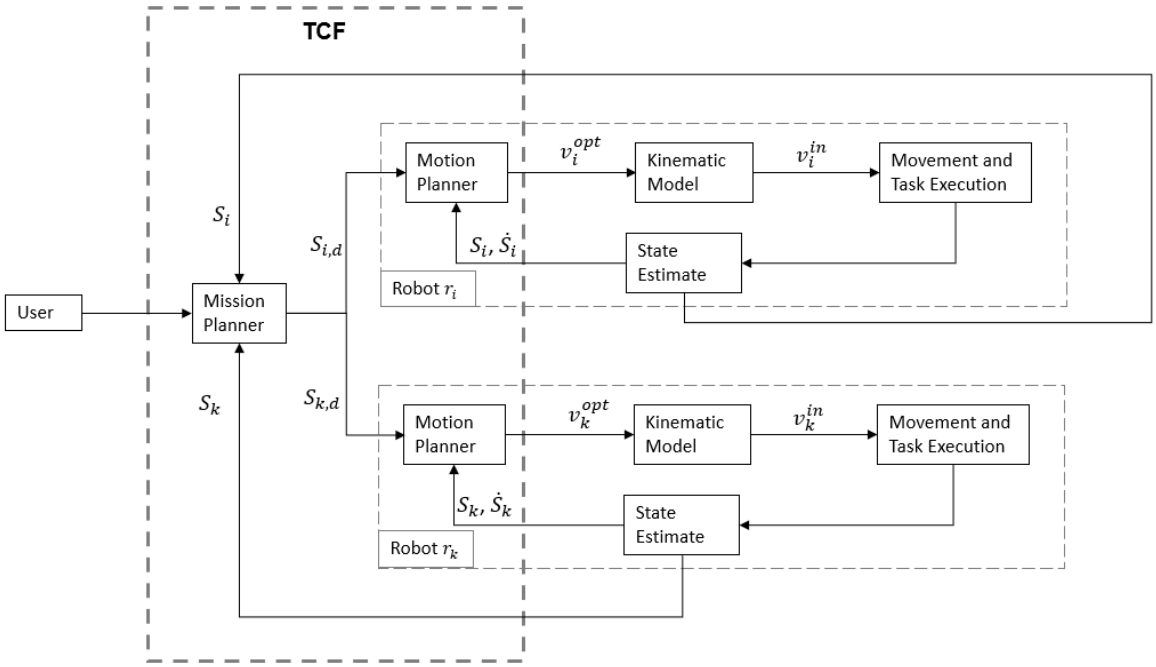


Figure A.2 – Motion Planning Loop

In Figure A.2, S_i and $\dot{S}_i$ correspond to pose and velocity of robot r_i; v_i^{opt} and v_i^{in} correspond to optimal velocity and control inputs for robot r_i; $S_{i,d}$ is goal pose for robot r_i; and likewise for robot r_k.

The mission planner assigns tasks to robots, from which goal poses S_d are extracted depending on the localization and type of task. Robots communicate their pose S_i to the mission planner. Robots communicate both their pose S_i and velocity, $\dot{S}_i$ to their own motion planner and other robots' motion planners (not shown in Figure A.2). Then, the Motion planner calculates optimal velocity v_i^{opt} with RVO to drive the robot i to its goal pose while avoiding collision with other robots.

A3.2 – Reciprocal Velocity Obstacles (RVO)

For every individual robot, the motion planner employs RVO to calculate robot i optimal velocity, v_i^{opt} as follows:

1) Calculate ideal velocity.
2) Calculate velocity obstacles of every other robot to robot i.
3) Iterate through a range of magnitudes and directions for possible velocities for robot i.
4) Calculate the intersection between possible velocities and velocity obstacles.
5) Select suitable velocities that do not intersect. Velocities that intersect are discarded.
6) If there are suitable velocities, select the one that differs the least from the ideal velocity. If there are no suitable velocities, select the one that maximizes time to collision (with the expectation that dynamic replanning at subsequent iterations will generate suitable collision-free velocities).

Calculation of ideal velocity, v_i^{id} entails calculating the unit vector from the robot's current position (x_i, y_i), to the goal (allocated task) position (x_d, y_d), and multiplying it by a scalar value V_{max} corresponding with the desired speed. This speed, V_{max} is user-defined and will vary based on the robots' dynamics, working environment and kinematic model.

$$v_i^{id} = \frac{V_{max}}{\sqrt{(x_i - x_d)^2 + (y_i - y_d)^2}} \begin{bmatrix} x_d - x_i \\ y_d - y_i \end{bmatrix} \tag{A.4}$$

The velocity obstacle [42] of robot k to robot i, $VO_{i,k}$ is calculated as a row vector describing its boundaries (A.5). Where $(x_i, y_i), (x_k, y_k)$ are positions of robots i and k, respectively, $(\dot{x}_i, \dot{y}_i)$, $(\dot{x}_k, \dot{y}_k)$ are velocities of robots i and k, respectively, and R_i is the inflated radius of robot i. The radius must be inflated to provide sufficient time for the robot to change its velocity from current velocity to optimal velocity through its kinematic model. The amount of inflation required depends on the robot's dimensions, kinematic model, and mobility. The variables in equations (A.5) to (A.7) are shown in Figure A.3.

$$VO_{i,k} = \begin{bmatrix} x_i + 0.5(\dot{x}_i + \dot{x}_k) \\ y_i + 0.5(\dot{y}_i + \dot{y}_k) \\ \cos(\theta_{ik} + \theta_R) \\ \sin(\theta_{ik} + \theta_R) \\ \cos(\theta_{ik} - \theta_R) \\ \sin(\theta_{ik} - \theta_R) \\ \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \\ 2R_i \end{bmatrix}^T = \begin{bmatrix} VO_{i,k}^1 \\ VO_{i,k}^2 \\ VO_{i,k}^3 \\ VO_{i,k}^4 \\ VO_{i,k}^5 \\ VO_{i,k}^6 \\ VO_{i,k}^7 \\ VO_{i,k}^8 \end{bmatrix}^T \tag{A.5}$$

The components of the velocity obstacle $VO_{i,k}$ boundaries are described as follows:

- $VO_{i,k}^1$ and $VO_{i,k}^2$ are the x and y components of the velocity obstacle. It is considered as a ray originating from robot i whose direction is the same as the average velocity between the robots i and k.
- $VO_{i,k}^3$ and $VO_{i,k}^4$ represent the left boundary of the velocity obstacle.
- $VO_{i,k}^5$ and $VO_{i,k}^6$ represent the right boundary of the velocity obstacle.
- $VO_{i,k}^7$ is the distance between the robots i and k.
- $VO_{i,k}^8$ is double the inflated radius. The inflated radius must be doubled to account for possible collisions along both the left and right sides of robot i.

$$\theta_{ik} = atan2(y_k - y_i, x_k - x_i) \tag{A.6}$$

$$\theta_R = \text{asin}\left(\frac{2R_i}{\sqrt{(x_k - x_i)^2 + (y_k - y_i)^2}}\right) \tag{A.7}$$

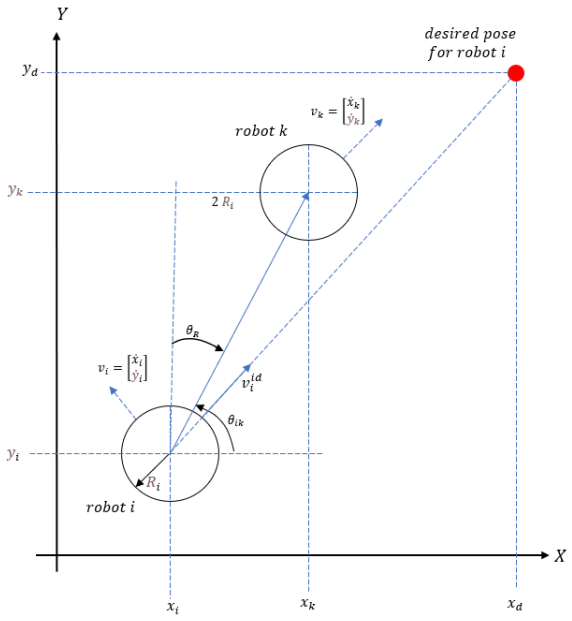


Figure A.3 – Velocity Obstacle Diagram

The calculation of $VO_{i,k}$ is repeated for all robots to obtain the velocity obstacle matrix VO_i (A.8) for all values k where $k \neq i$. In a multi-robot system consisting of m robots, the velocity obstacles of each robot, to robot i is denoted in (A.5). Each robot calculates VO_i to determine its own optimal velocity.

$$VO_i = \begin{bmatrix} VO_{i,1} \\ VO_{i,2} \\ \dots \\ VO_{i,m} \end{bmatrix} \tag{A.8}$$

We then generate the matrix of new possible velocities V_{new} where each row contains a set of velocities $v_{M,\phi} = M[\cos(\phi^{new}) \quad \sin(\phi^{new})]^T$, iterated from $\phi^{new} = 0$ to $\phi^{new} = 2\pi$ over several steps, with each step increasing the angle by n, where n is a step size such that there is sufficient granularity for the direction of motion. For the simulation and experimentation, a value of $n = 0.1$ was selected, which would iterate the heading of $v_{M,\phi}$ from $0,\ 0.1,\ 0.2$ ... radians until a direction of 2π radians. At the same time, the magnitude of $v_{M,\phi}$ was iterated from $M_1 = V_{min}$ to $M_{max} = V_{max}$.

$$V_{new} = \begin{bmatrix} \{v_{M_1,\phi^{new}} | \phi^{new} = 0, n, 2n \dots 2\pi\} \\ \{v_{M_2,\phi^{new}} | \phi^{new} = 0, n, 2n \dots 2\pi\} \\ \dots \\ \{v_{M_{max},\phi^{new}} | \phi^{new} = 0, n, 2n \dots 2\pi\} \end{bmatrix} \quad \text{(A.9)}$$

Each velocity $v_{M,\phi}$ is tested with each of the velocity obstacles in VO_i for suitability. From each possible new velocity, we calculate its intersections with each velocity obstacle, ϕ_l, ϕ_r, ϕ_Δ respectively. To facilitate this test, we calculate ϕ_l, ϕ_r and ϕ_Δ in equations (A.10) to (A.12).

$$\phi_l = atan2\left(VO^4_{i,k}, VO^3_{i,k}\right) \quad \text{(A.10)}$$

$$\phi_r = atan2\left(VO^6_{i,k}, VO^5_{i,k}\right) \quad \text{(A.11)}$$

$$\phi_\Delta = atan2(M \sin(\phi^{new}) + y_i - VO^2_{i,k}, M \cos(\phi^{new}) + x_i - VO^1_{i,k}) \quad \text{(A.12)}$$

$$\phi_\Delta = atan2(M \sin(\phi^{new}) - 0.5(\dot{y}_i + \dot{y}_k), M \cos(\phi^{new}) - 0.5(\dot{x}_i + \dot{x}_k))$$

- ϕ_l and ϕ_r are the left and right boundary angles of the velocity obstacle, respectively.
- ϕ_Δ is the heading of the difference between new velocity $v_{M,\phi}$ and average velocity between robot i and robot k.
- M and ϕ^{new} are magnitude and direction of new velocity $v_{M,\phi}$.

If $\phi_l \leq \phi_\Delta \leq \phi_r$ is true, then the velocity $v_{M,\phi}$ is unsuitable for that velocity obstacle, because it means the angle ϕ_Δ is in between ϕ_l and ϕ_r, and therefore would result in a collision if the velocities of robots i and k remain unchanged. If $\phi_r \leq \phi_\Delta \leq \phi_l$ is true, then the velocity $v_{M,\phi}$ is suitable for that velocity obstacle, meaning it would not result in a collision with that specific velocity obstacle. This test is repeated for all velocity obstacles. If $v_{M,\phi}$ is suitable for all velocity obstacles, then it is overall suitable for robot i. Else, that velocity is not suitable. This process generates the set of suitable and unsuitable velocities [43].

$$V_{suitable} = \{V_{new} |\ \phi_r \leq \phi_\Delta \leq \phi_l\ \} \quad \text{(A.13)}$$

$$V_{unsuitable} = \{V_{new} |\ v_{M,\phi^{new}} \notin V_{suitable}\ \} \quad \text{(A.14)}$$

If there are suitable velocities, our optimal velocity, v_i^{opt} is the suitable velocity that differs the least from our ideal velocity v_i^{id} (A.15).

$$v_i^{opt} = \underset{v_{M,\phi}}{argmin} \left|\left|v_{M,\phi} - v_i^{id}\right|\right|, \quad v_{M,\phi} \in V_{suitable} \quad \text{(A.15)}$$

If there are no suitable velocities, we calculate the velocity selection factor VSF to maximize the time to collision [43]. The calculation for VSF is described in equations (A.16) to (A.20). The velocity that minimizes VSF is selected as our optimal velocity, v_i^{opt}. This velocity is

selected to enable the robot to adjust its velocity, escape collision and then provide time to calculate a suitable velocity, while remaining as close as practicable to the ideal velocity.

$$v_i^{opt} = \underset{v_{M,\phi}}{argmin} VSF, \;\; v_{M,\phi} \in V_{unsuitable} \tag{A.16}$$

where:

$$VSF = \frac{w}{t_c} + \left| v_{M,\phi} - v_i^{id} \right| \tag{A.17}$$

$$w = weighting\ factor$$

$$t_c = \frac{|VO_{i,k}^7 * \cos(\phi_1)| - |VO_{i,k}^8 * \cos(\phi_2)|}{\sqrt{(M\cos(\phi^{new}) - 0.5(\dot{x}_i + \dot{x}_k))^2 + (M\sin(\phi^{new}) - 0.5(\dot{y}_i + \dot{y}_k))^2}}, \qquad t_c > 0 \tag{A.18}$$

$$\phi_1 = |\phi_\Delta - 0.5(\phi_l + \phi_r)| \tag{A.19}$$

$$\phi_2 = \text{asin}\left(\left|\frac{VO_{i,k}^7 * \sin(\phi_1)}{VO_{i,k}^8}\right|\right) \tag{A.20}$$

Equations (A.17) to (A.20) describe the process in which VSF is calculated. The weighting factor w adjusts the relative weights between time to collision t_c and difference between new velocity $v_{M,\phi}$ and ideal velocity v_i^{id}. A larger value of w would cause the RVO algorithm to assign more value on collision time instead of selecting a velocity close to the ideal velocity. This would reduce the likelihood of collision but could result in a longer trajectory.

The time to collision, t_c is calculated as relative distance (numerator) divided by relative velocity (denominator) in (A.18). The distance between the center of both robots is $VO_{i,k}^7$, however, the radius of the robot, must be considered, hence the requirement to factor in $VO_{i,k}^8$. The cosines of angles ϕ_1 and ϕ_2 must also be considered to factor in relative directions. The denominator in (A.18) is the magnitude of relative velocity between the new velocity and average velocity of robot i and robot k.

Once this optimal velocity v_i^{opt} is calculated, it is then converted into control inputs v_i^{in} using the respective robots' kinematic models [130], which will vary for different types of robots. The maximum speed V_{max} and Inflated radius R_i also need to be selected based on individual robots' dynamics, kinematic model, and capabilities.

Overall, the RVO algorithm effectively generated collision-free trajectories that drove the robots to their destinations. Validation scenarios were built on pre-set coordinates as per Table A.17, showing each robot's start and goal positions for 3 different scenarios. Parameters were as per Table A.18. The results of these 3 scenarios are shown in Figure A.4 to Figure A.6.

Table A.17 – Robots' Initial and Goal Positions

Robot	Scenario 1		Scenario 2		Scenario 3	
	Start	Goal	Start	Goal	Start	Goal
r_0	(2,2)	(-2,2)	(0,0)	(3,3)	(0,0)	(4,0)
r_1	(-2,-2)	(2,2)	(2,0)	(0,6)	(2,0)	(6,0)
r_2	(-2,2)	(0,0)	(4,0)	(0,4)	(4,0)	(0,0)
r_3	(2,-2)	(-2,2)	(6,0)	(0,2)	(6,0)	(2,0)

For this validation, the robots were treated as holonomic circles that can freely translate along the x, y plane (A.21). There was no consideration for orientation or kinematic constraints of any real robot. At each timestep Δt of this test, the robot's new position was calculated using the old position and moving at a constant v_i^{opt} calculated from RVO for that timestep (A.22). These conditions were used to validate the RVO algorithm, independent of any type of robot. However, the parameters in Table A.18 were inspired by the robots considered for experimentation, with a robot radius of $0.16\,m$, and inflating to $R_i = 0.21\,m$.

$$S = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad \dot{S} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = v_i^{opt} \tag{A.21}$$

$$S(t+1) = S(t) + \dot{S}\Delta t \tag{A.22}$$

Table A.18 – RVO Validation Parameters

Parameter	Description	Quantity
R_i	Inflated radius (m)	0.21
V_{max}	Maximum speed (m/s)	0.26
Δt	Duration of timestep (s)	0.25

Figure A.4 to Figure A.6 show each scenario's trajectories and minimum inter-robot distance. The trajectories show the robots' velocities adjusting to avoid collision with each other while reaching their destinations. Minimum inter-robot distance, indicated by d_min is the smallest distance between any 2 robots at that timestep. A collision occurs if d_min is equal or less than the uninflated radius of the robots, here 0.16 m. At all times, d_min was greater than the inflated radius of 0.21 m, which itself is greater than the robots' actual radius. Therefore, there was no collision in any of the tests. At the same time, we observe that none of the velocities indicate a straight-line ideal velocity, demonstrating velocity adjustments to avoid collision.

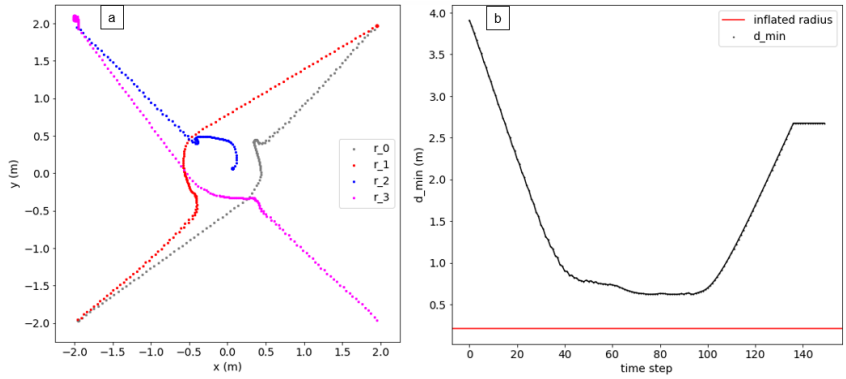


Figure A.4 – RVO Test Scenario 1: a) Robots' Trajectories, and b) Minimum Inter-Robot Distance

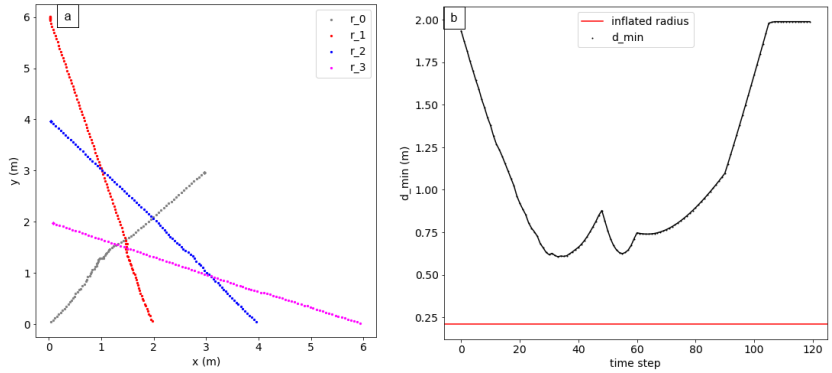


Figure A.5 – RVO Test Scenario 2: a) Robots' Trajectories, and b) Minimum Inter-Robot Distance

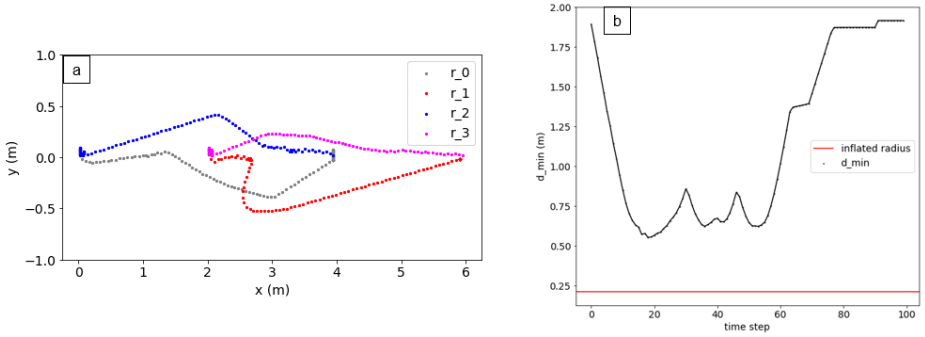


Figure A.6 – RVO Test Scenario 3: a) Robots' Trajectories, and b) Minimum Inter-Robot Distance

The velocities of each robot in each test scenario are also plotted in Figure A.7 to Figure A.9. These plots show the components of robots' velocities over time as they execute their maneuvers. There are periods of changing velocity as well as constant velocity. For example, in Figure A.7, timestep 0 to timestep 100, for "r_1 vel_x" (x component of v_i^{opt} of r_1) shows changing velocity while example timestep 101 to timestep 130 for "r_1 vel_x" shows constant (nonzero) x component velocity. The overall velocity is constant if the x and y components are constant. This indicates that r_1 is moving in a straight-line path towards its goal, and it can do so because there is no risk of collision with other robots; $v_i^{opt} = v_i^{id}$.

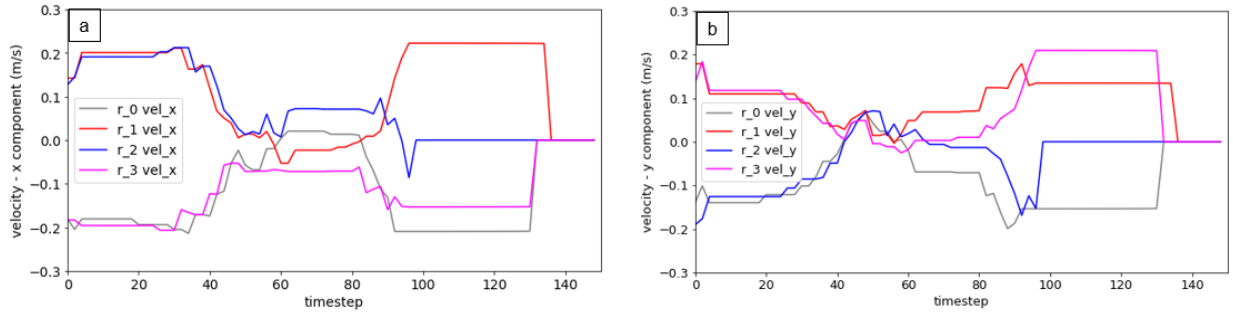


Figure A.7 – RVO test Scenario 1 Velocities: a) x Component, and b) y Component

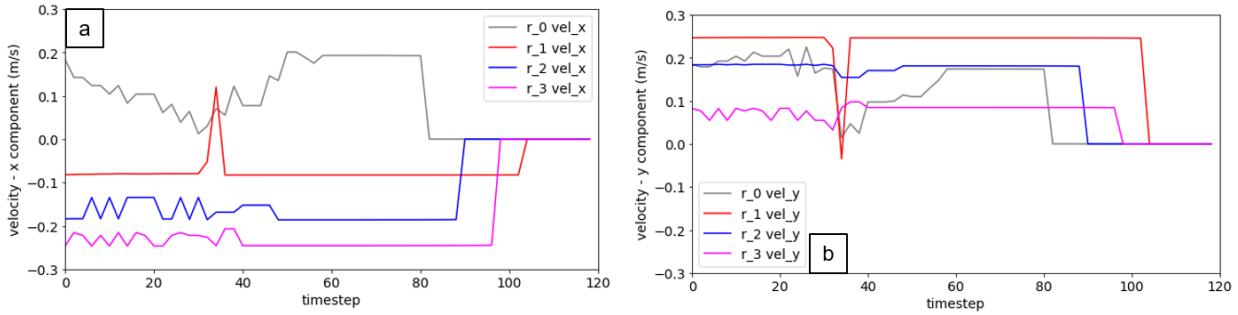


Figure A.8 – RVO test Scenario 2 Velocities: a) x Component, and b) y Component

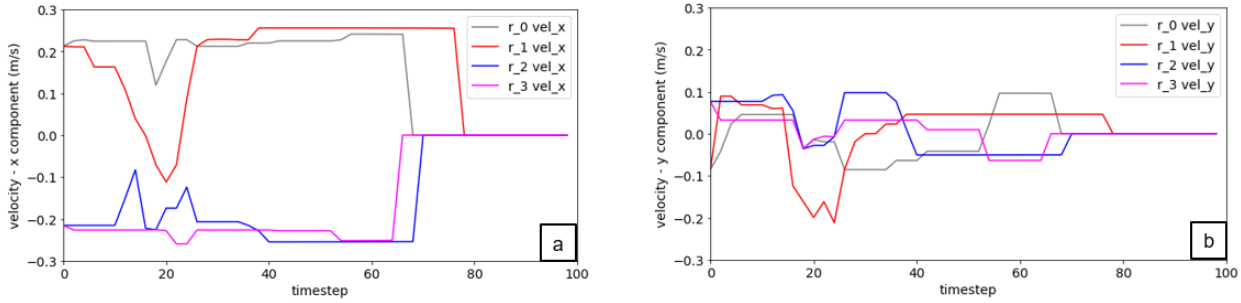


Figure A.9 – RVO test Scenario 3 Velocities: a) x Component, and b) y Component

A4 – Robots' Physical Descriptions

A4.1 – RB5 Drone Physical Description

The physical properties of the RB5 drone are listed in Table A.19.

Table A.19 – RB5 Drone Properties

Serial	Property	Unit of Measurement	Value
1	Size (length, width, height)	mm x mm x mm	730 x 591 x 187
2	Mass	kg	1.235
3	Electronic Speed Controllers (ESCs)		4x ModalAI VOXL 4-in-1 ESC V2
4	Motors		4x Holybro 2216-880kv Motors
5	Thrust per motor (100% throttle*)	N	9.896
6	Thrust per motor (50% throttle*)	N	3.068
7	Payload	kg	1
8	Flight time	minutes	20
9	Battery Type		4S LiPo
10	sensors		Cameras (stereoscopic, tracking, high resolution) IMU, barometer, GPS

*Throttle % is defined in (A.23).

$$\text{Throttle } \% = 100 * \frac{input\_PWM \ - \min\_PWM}{max\_PWM} \tag{A.23}$$

where $input\_PWM$ is the PWM signal given to the motor from the ESC, $\min\_PWM$ is the minimum range of allowable PWM signal and for this drone, $\min\_PWM = 1000. \max\_PWM$ is the maximum range of allowable PWM signal, and for this drone, $\max\_PWM = 2000$. The thrust data is available at [131], which shows the corresponding thrust values per motor at a given $input\_PWM$.

The homogeneous transformation ${}^{r}H_{CI}$ from the RB5 drone's robot body frame RF_r to the camera image frame RF_{CI} of its tracking camera is described in equations (A.24) to (A.26). The values for (A.24) and (A.25) were derived from the information in [132]. While the RB5 drone has several other cameras, only the tracking camera was downwards facing, which is evident in (A.24) and [132], describing a 45 degree or $\pi/4$ radian downwards tilt.

$$\begin{gathered} {}^{r}S_{CL} = [0.087 \quad 0 \quad -0.001 \quad 0 \quad \pi/4 \quad 0]^T \\ {}^{r}H_{CL} = \boldsymbol{HT}({}^{r}S_{CL}) \end{gathered} \tag{A.24}$$

$${}^{CL}S_{CI} = [0 \quad 0 \quad 0 \quad -\pi/2 \quad 0 \quad -\pi/2]^T \tag{A.25}$$

$$^{CL}H_{CI} = \boldsymbol{HT}(^{CL}S_{CI})$$

$$^{r}H_{CI} = {}^{r}H_{CL} * {}^{CL}H_{CI} \quad \text{(A.26)}$$

$^{r}H_{CL}$ is derived using the transformations from “body” to “imu_px4” and “imu_px4” to “tracking” in [132]. $^{r}H_{CL}$ represents the relative pose of the tracking camera to the drone. $^{CL}H_{CI}$ is a rotation required to orient x and y coordinates to match the camera image plane.

A4.2 – Wafflebot Physical Description

The physical properties of the Wafflebot are listed in Table A.20.

Table A.20 – Wafflebot Properties

Serial	Property	Unit of Measurement	Value
1	Size (length, width, height)	mm x mm x mm	281 x 306 x 141
2	Mass	kg	1.8
3	Max translational velocity	m/s	0.26
4	Max rotational velocity	rad/s	1.82
5	Operating time	hr	2
6	Charging time	hr	2.5
7	Single Board Computer		Raspberry Pi 3
8	Embedded Controller		OpenCR
9	Sensors		Raspberry Pi Camera, LiDAR gyroscope, accelerometer, magnetometer

A5 – RB5 Drone Setup and Configuration

This appendix contains instructions for safely setting up and flying the RB5 drone, both manually and programmatically, using ROS. A working level of experience with ROS and Ubuntu is assumed.

References:

https://docs.modalai.com/ - manufacturer documentation

https://forum.modalai.com/ - manufacturer forums

http://wiki.ros.org/mavros - general info on MAVROS

https://docs.px4.io/main/en/ - PX4 documentation

There are 3 overall steps to get the drone working out of the box to flying with ROS:

- OS/firmware setup
- PX4 and Q Ground Control (QGC)
- ROS setup

A computer with Ubuntu 18.04+ is required. USB connection between the computer and RB5 drone may be easier with a computer that is natively running Ubuntu. However, it is still possible with a virtual machine (VM).

A5.1 – OS and Firmware

Steps 1 to 5 cover updating the drone's operating system.

Steps 1 to 3: https://docs.modalai.com/Qualcomm-Flight-RB5-system-image/#voxl-sdk-for-rb5-flight

Steps 4 to 5: https://docs.modalai.com/Qualcomm-Flight-RB5-voxl-sdk-upgrade-guide/

1) Plug in RB5 Drone to computer and ensure it is detected. You can run **adb devices** and should see it. If you cannot see it, configure the VM or plug it into a machine that natively runs Ubuntu.

2) Go to https://developer.modalai.com/ to find the proper system image. It is in *protected downloads →RB5 Flight with VOXL SDK Platform Releases*. Download that file and unzip it using the command:
    - tar -xzvf {file_name}

Name	File Name	Upload Date	Category	Size	
Qualcomm Flight RB5 SDK (1.1.3)	rb5-flight-sdk-1.1.3.tar.gz	2/1/2022, 2:41:49 PM	RB5 Flight with RB5 Flight SDK Platform Releases	46.51 MB	Download
Qualcomm Flight RB5 System Image (v1.0.5-c)	1.0.5-M0052-9.1-perf-c.tar.gz	1/27/2022, 3:04:08 PM	RB5 Flight with RB5 Flight SDK Platform Releases	1419.77 MB	Download

3) While the RB5 drone is connected to the computer, cd into the folder with the unzipped system image and execute:

- sudo ./full-flash.sh

4) Back up camera calibration. Create a directory to store these files locally on your machine and execute adb pull to save them on your machine:
    - mkdir ~/cam_cal_files
    - adb pull /data/modalai/opencv_tracking_intrinsics.yml
    - adb pull /data/modalai/opencv_stereo_intrinsics.yml
    - adb pull /data/modalai/opencv_stereo_rear_intrinsics.yml
    - adb pull /data/modalai/opencv_stereo_extrinsics.yml
    - adb pull /data/modalai/opencv_stereo_rear_extrinsics.yml

5) On the host computer, navigate to where you downloaded the system image and install it by executing the following commands:
    - cd rb5_platform_1.3.1-0.8
    - ./install.sh

6) Push backup camera calibration files after installation is complete:
    - cd ~/cam_cal_files
    - adb push opencv_tracking_intrinsics.yml /data/modalai
    - adb push opencv_stereo_intrinsics.yml /data/modalai
    - adb push opencv_stereo_rear_intrinsics.yml /data/modalai
    - adb push opencv_stereo_extrinsics.yml /data/modalai
    - adb push opencv_stereo_rear_extrinsics.yml /data/modalai

A5.2 – PX4 and QCG

Before starting the steps in A5.2, download Q Ground Control (QGC).

Steps 1 to 5 enable wifi connectivity with the drone: https://docs.modalai.com/voxl-wifi/

Steps 6 to 11 enable MAVLINK and QCG communication with the computer

1) Setup a wifi hotspot. For example, you can use a cellphone and set SSID = “hotspot123” and password = “hotspot123”.

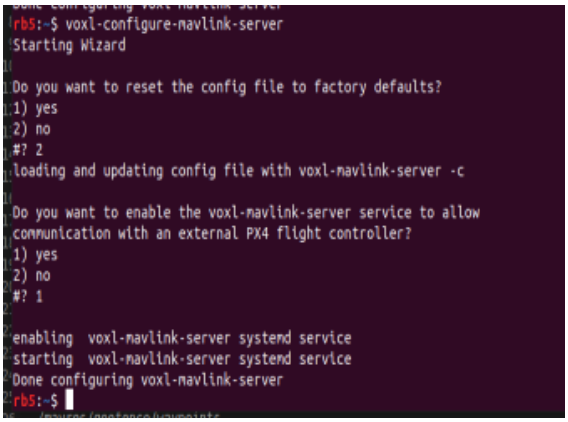
2) Connect the drone to the host computer via a USB cable, ensuring it is detected.

3) Open a terminal on the computer and execute the command:
    - voxl-wifi station <ssid> [password]

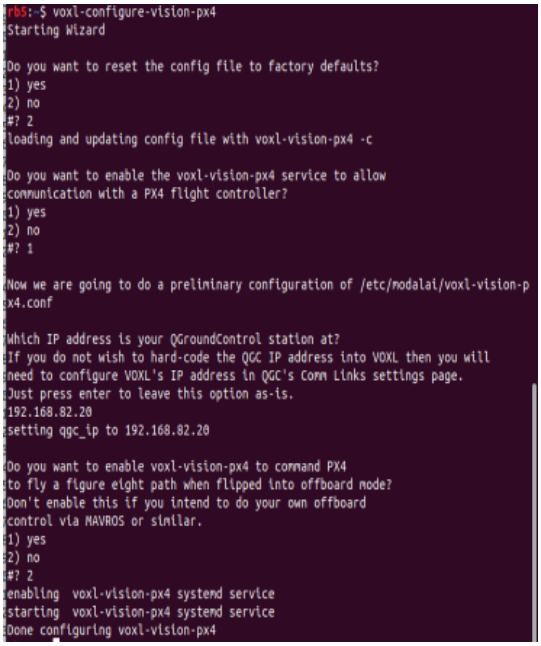
4) The next time the drone is rebooted, it should automatically connect to that network. Connecting to the drone via ssh on the host computer should be possible.

5) To connect to the drone via ssh, ensure both the computer and drone are on the same Wi-Fi network. On the computer, open a terminal and execute:
    - ssh root@{DRONE_IP}
        - password is *oelinux123*
        - DRONE_IP is found by going on connections → mobile hotspot and tethering → mobile hotspot → connected devices

6) Configure mavlink by executing the following command and following the prompts
    - voxl-configure-mavlink-server



7) Enable px4 and setup QCG IP by executing the following command and responding to the prompts as follows:
    - voxl-configure-vision-px4



QGC IP address is the IP address of the host computer. It can be found using the command:

- ifconfig -a

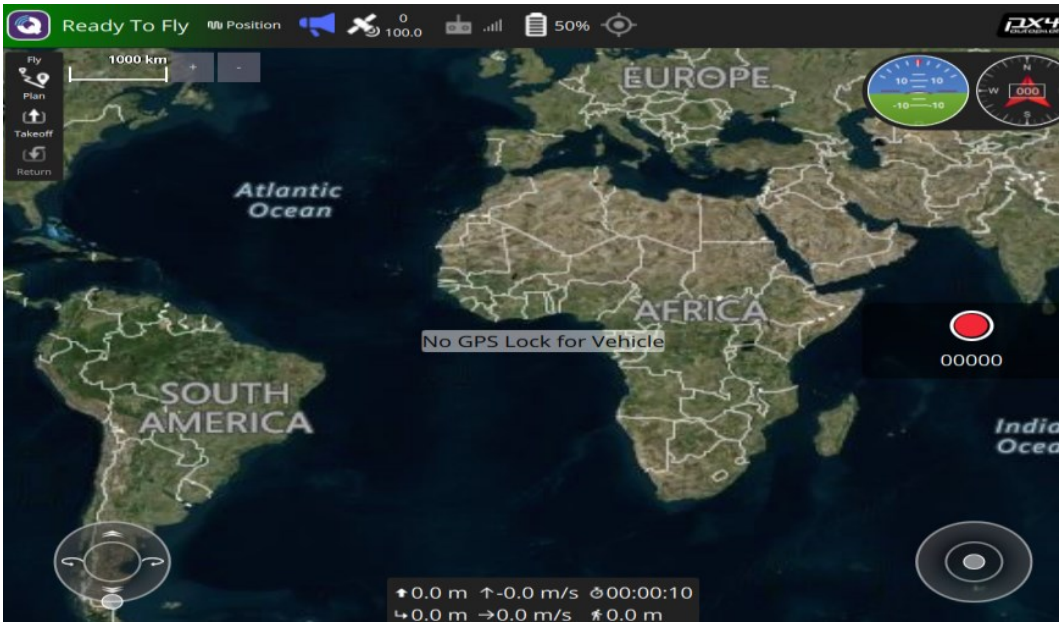
8) Update the .param file on QGC. It can be found at the following github repo: https://github.com/albud187/RB5_MAVROS_params as the filename “RB5_MAVROS_VIO_09-26.params”

9) Open QGC → vehicle setup → parameters → tools →load from file.

10) This .param file is required to allow the drone to arm and fly **without GPS.**

11) The drone can now be flown safely without GPS using **position mode** using the virtual joystick. If not, wait for this popup to appear on QGC, indicating that VIO has started.

After this popup, the drone should be able to switch to position mode. Position mode requires the drone to have a pose estimate, such as from VIO.



A5.3 – ROS setup

Steps 1 to 7: https://docs.modalai.com/setup-ros-on-voxl-2/

Steps 8 to 14: https://docs.modalai.com/mavros-voxl-2/

1) Open a terminal and ssh into the drone.

2) Execute:
   - sudo apt-get update
   - sudo apt install -y ros-melodic-ros-base ros-melodic-image-transport
   - voxl-configure-mpa -p -f rb5-flight

3) Open ~/.bashrc by executing:
   - nano ~/.bashrc

4) On ~/.bashrc add the following line:
    - . /opt/ros/melodic/setup.sh

5) Save and close the ~/.bashrc file. Then source it by executing:
    - source ~/.bashrc

6) Now, you can execute:
    - roslaunch voxl_mpa_to_ros voxl_mpa_to_ros.launch

7) The command in step 6) will start publishing the camera imagery as ROS topics.

8) Install MAVROS by executing:
    - Sudo apt-get install ros-melodic-mavros ros-melodic-mavros-extras ros-melodic-control-toolbox
    - cd /opt/ros/melodic/lib/mavros
    - ./install_geographiclib_datasets.sh

9) Build mavros test by executing:
    - cd /home
    - git clone -b simple-example https://gitlab.com/voxl-public/support/mavros_test.git
    - cd /home/mavros_test
    - ./build.sh

10) Configure ROS environment. Open ros_environment.sh by executing:
    - nano /home/mavros_test/ros_environment.sh

11) Under configure IPs, set them to the following:
    - export ROS_MASTER_IP=localhost
    - export ROS_IP=localhost
    - export ROS_MASTER_URI=http://localhost:11311/

12) Source the ROS environment on the ~/.bashrc so you do not have to source it each time. Open the .bashrc file by executing:
    - nano ~/.bashrc

13) Add the following line to the ~/.bashrc:
    - source /home/mavros_test/ros_environment.sh

14) Now you can launch MAVROS by executing:
    - roslaunch mavros px4.launch fcu_url:=udp://127.0.0.1:14551@:14551 tgt_system:=${PX4_SYS_ID}

A6 – Supplementary Experiments

A6.1 – RB5 Drone Localization

This experiment entails flying a single RB5 drone through a series of waypoints to quantify the accuracy of the RB5 drone's onboard visual inertial odometry (VIO) pose estimate, as introduced in Section 2.1.1. This onboard estimate yields $\hat{S}_c^{L'}$, which is the drone's estimated current pose in the initialization frame. Conversion from $\hat{S}_c^{L'}$ to $\hat{S}_c^G$ (A.27), that is the estimated pose in the global frame was facilitated through the transformation from the initialization frame to the global frame, ${}^{L}H_G$ at the instant the drone was powered up. Therefore, the estimated position of the RB5 drone in reference to the global frame $\hat{S}_c^G$ is denoted in (A.28).

$$\hat{S}_c^{L'} = \begin{bmatrix} \hat{x}_c^{L'} & \hat{y}_c^{L'} & \hat{z}_c^{L'} & 0 & 0 & \hat{\phi}_c^{L'} \end{bmatrix}^T \tag{A.27}$$

$$\hat{S}_c^{G} = \begin{bmatrix} \hat{x}_c^{G} & \hat{y}_c^{G} & \hat{z}_c^{G} & 0 & 0 & \hat{\phi}_c^{G} \end{bmatrix}^T$$

$$\begin{bmatrix} \hat{x}_c^{G} & \hat{y}_c^{G} & \hat{z}_c^{G} & 1 \end{bmatrix}^T = {}^{L}H_G * \begin{bmatrix} \hat{x}_c^{L'} & \hat{y}_c^{L'} & \hat{z}_c^{L'} & 1 \end{bmatrix}^T \tag{A.28}$$

The RB5 drone flew through a series of 5 setpoint positions from its start location, as listed in Table A.21. The start position was measured to be $(-2.18, 0.04, 0.118)$ from the motion capture (MoCap) system. It flew this series of setpoint positions 5 times over 4 minutes and 30 seconds. The drone's trajectory, estimated (black dots) and MoCap (red dots), along with the setpoints, are shown in Figure A.10. These positions are in reference to the global frame. While ground truth positions were unavailable, VIO accuracy was obtained by comparison with MoCap pose estimates. The MoCap system generated highly accurate measurements with less than 0.1 mm error [112].

Table A.21 – Setpoint Positions for RB5 VIO Test

Setpoint Number	Coordinates (x, y, z) m
1	$(-1, -1, 0.75)$
2	$(1, -1.5, 1.25)$
3	$(2, 0, 0.75)$
4	$(1, 1.5, 1.25)$
5	$(-1, 1, 0.75)$

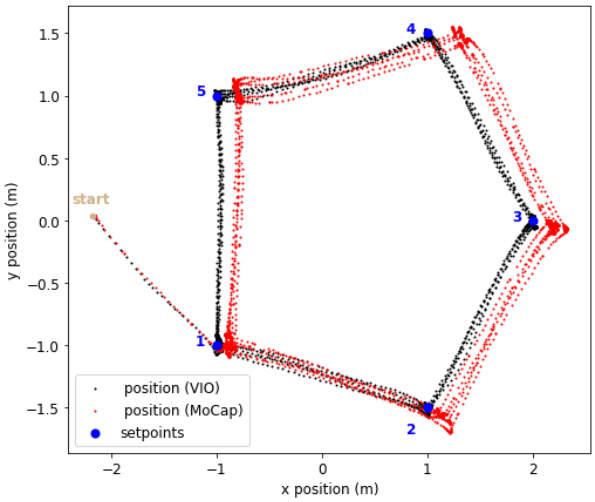


Figure A.10 – RB5 Drone Trajectory over XY plane

Figure A.11 to Figure A.13 show the components of setpoint position, compared with the estimated position from VIO, MoCap measurement, and localization error. Localization error is the difference between the pose estimate from VIO and the measured pose from MoCap. The RB5 drone was autonomously flying through its setpoints between timesteps 50 and 2500, responding to setpoint positions during this timeframe. This timeframe of autonomous flight is shown in Figure A.11 to Figure A.13 between the two gray dashed vertical lines. At timestep 2500, the drone was no longer flying autonomously. It was at waypoint 1 and then landed manually. The landing was completed at 2750, shown by the black dashed vertical line. From these figures, the drone's estimated pose converges onto the desired setpoint pose with a high degree of responsiveness and little oscillation.

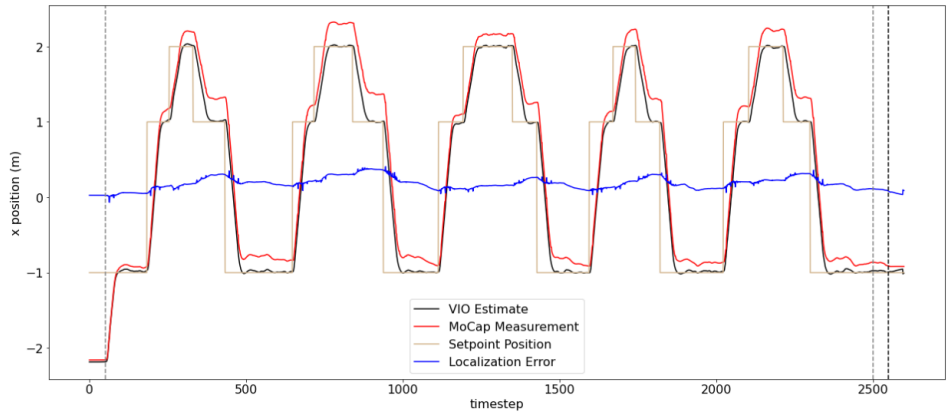


Figure A.11 – X Position Estimate, MoCap Measurement, Setpoint and Localization Error

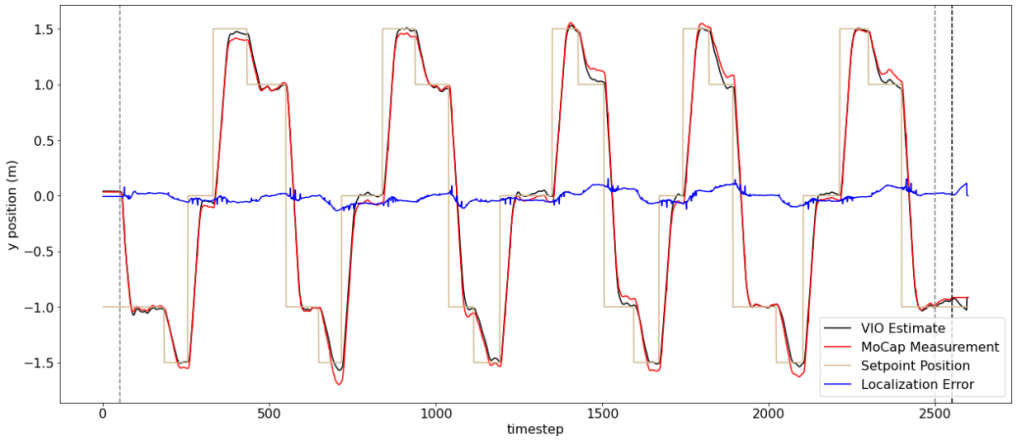


Figure A.12 – Y Position Estimate, MoCap Measurement, Setpoint and Localization Error

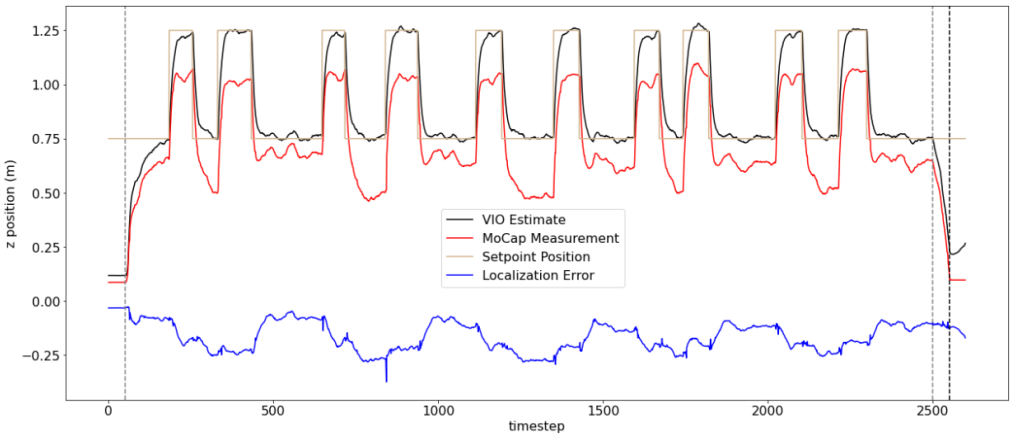


Figure A.13 – Z Position Estimate, MoCap Measurement, Setpoint and Localization Error

Note that delays between changing the desired setpoint and the drone reaching that setpoint are due to the travel time required for the drone to fly to the desired position. For aerial localization, the RB5 drone's maximum speed was set to $0.5\ m/s$. Increasing this speed would result in the RB5 drone reaching setpoint positions faster. However, it may be detrimental to image quality for aerial observation.

The 3D localization error over the distance travelled is shown in Figure A.14. The 3D localization error is the 3D distance between the measured position from MoCap and the estimated position from VIO.

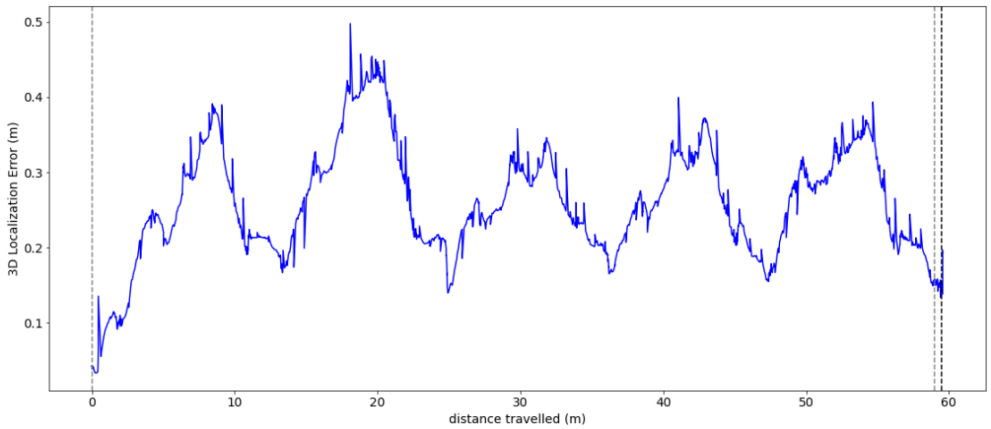


Figure A.14 – 3D Localization Error Over Distance Travelled

Figure A.13 shows that the error remains between 0.15 m to 0.5 m throughout the entire maneuver in which the drone travelled 60 m, corresponding to an error of $0.25\ \%$ to $0.83\ \%$. Alternatively, with a total of 4 passes, each pass through all 5 waypoints had a travel distance of $15\ m$, which means the average error per pass ranged from $1.0\ \%$ to $3.33\ \%$.

The error increases and decreases cyclically, corresponding to repeated iterations of the same setpoints, but does not appear to drift during this trajectory. This cyclical pattern can be attributed to the difference between the RB5 drone's orientation and the direction of movement. This difference in orientation affects the relative movement of matched features on successive camera images in the VIO algorithm, which may affect pose estimate accuracy. Overall, the RB5 Drone's VIO algorithm is quite robust; while it is not 100% accurate, it is close to the MoCap measurement (which differs less than 0.1 mm from the ground truth) and maintains consistent performance throughout the flight test.

A6.2 – Wafflebot RVO

The reciprocal velocity obstacles (RVO) algorithm for motion planning described in Appendix A3 was tested using two real-life Wafflebots. The test scenarios are described in Table A.22. The results of each test scenario are shown in Figure A.15 to Figure A.17. Similar to Section 6.1.1, these figures show the robots' trajectories, minimum inter-robot distance d_{min} and the robots' final positions.

Table A.22 – Scenarios for Real-Life Wafflebot RVO Tests

Robot	**Scenario 1**		**Scenario 2**		**Scenario 3**	
	Start	**Goal**	**Start**	**Goal**	**Start**	**Goal**
r_0 (r_0)	(-2,1)	(-2,-1)	(-2,-1)	(2,1)	(2,1)	(2,-1)
r_1 (r_1)	(-2,-1)	(-2,1)	(-2, 1)	(2,-1)	(2,-1)	(2,0)

From each real-life test, the robots' trajectories can be observed to reach the goal position while deflecting to avoid collision. The minimum inter-robot distance at all times remained greater than the inflated radius (which in itself is greater than the robots' real-life radii). Therefore, no collisions occurred.

Localization for this test was facilitated via MoCap to provide accurate pose estimates. The Wafflebots' onboard wheel encoder odometry uses only dead reckoning, rendering it susceptible to drift. This would yield inaccurate pose estimates.

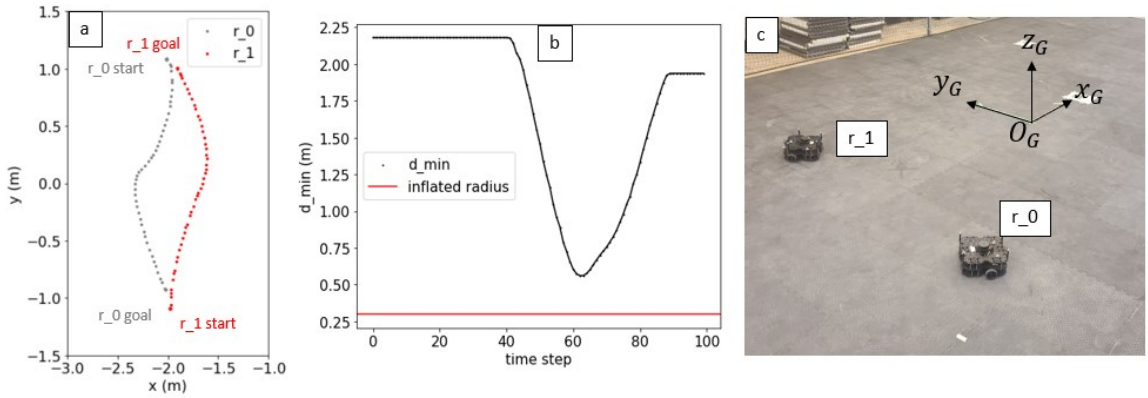


Figure A.15 – Real Life Wafflebot RVO Simulation Test 1: a) Respective Paths Over the Workspace, b) Minimum Inter-Robot Distance Over Time, and c) Final Positions in Test Environment

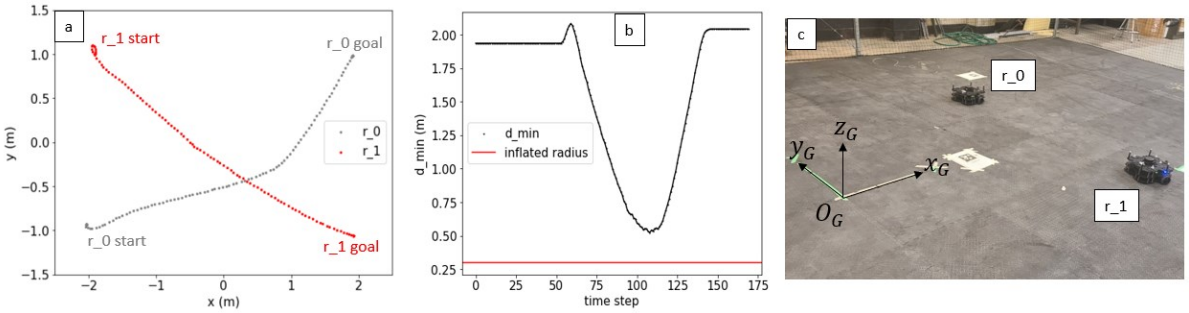


Figure A.16 – Real Life Wafflebot RVO Simulation Test 2: a) Respective Paths Over the Workspace, b) Minimum Inter-Robot Distance Over Time, and c) Final Positions in Test Environment

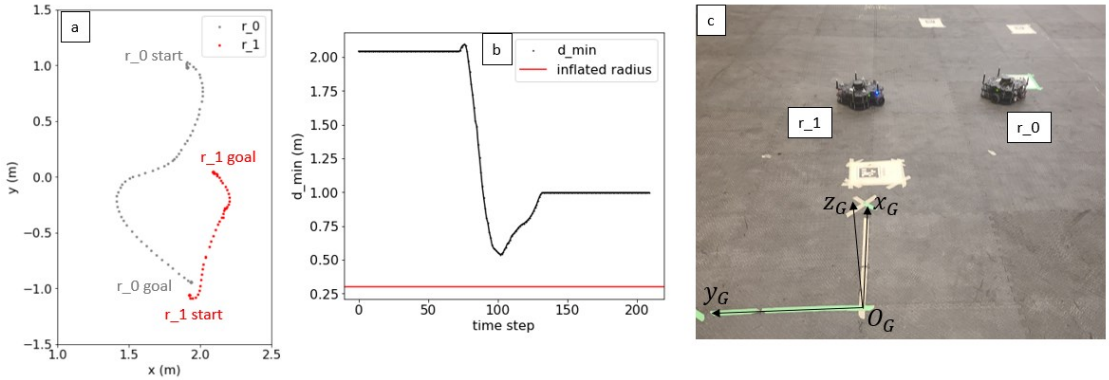


Figure A.17 – Real Life Wafflebot RVO Simulation Test 3: a) Respective Paths Over the Workspace, b) Minimum Inter-Robot Distance Over Time, and c) Final Positions in Test Environment