## Abstract

The 2020 MOSAR (MOdular Spacecraft Assembly and Reconfiguration) project was initiated to develop innovative technologies aimed at standardising satellites and components by raising the degree of space system modularity. As part of the project, a ground demonstrator for on-orbit modular and reconfigurable satellites was created, consisting of reusable spacecraft modules and a repositionable symmetric walking robotic manipulator. This demonstrator can execute basic instructions to relocate the manipulator and move modules between positions. However, these instructions are currently generated and verified manually before transmission to the demonstrator.

In this project we develop a reconfiguration planning system to autonomously generate instructions for reconfiguring modular spacecraft. Despite the limited research in this field, we demonstrate the feasibility of developing a reliable reconfiguration planner that considers the physical constraints introduced by the mobile manipulator and local environment. This was achieved by dividing the discrete and continuous components of the reconfiguration planner into two separate systems that communicate through feedback strategies in a control-loop mechanism. Our results show the system's capability to solve complex reconfiguration problems and identifies areas for further development.

This project serves as a foundation for developing more advanced systems with enhanced capabilities, potentially benefitting the space industry in the assembly and reconfiguration of modular space systems.

## Acknowledgements

I would like to give a big thank you to my project supervisor, Dr Mark Post, for providing guidance and feedback throughout the project; Along with giving me the freedom to explore and develop what interested me most. My exceptional peers at university that always pushed me to a higher standard throughout my time at university; And my family for making my time at university to further develop myself possible and providing endless support.

## Ethics

After consideration of the University’s code of practice and principles for good ethical governance no ethical issues were identified in this project.

## Table of Contents

# Introduction

This section builds, and expands, on material previously included in the project Initial Report (see Appendix %)

## Background and Context

In recent years, there has been rapid development in space systems driven by a global push for increased commercial accessibility. Current commercial systems are designed with a focus on minimizing mass and launch costs, resulting in highly customized configurations that often lack robust maintenance and repair capabilities. Consequently, the population of aging satellites is expanding, and upon reaching the end of their operational life, they are either deliberately deorbited using atmospheric deconstruction methods or left in orbit, contributing to the accumulation of space debris.

At present, there is little available technology to overcome these conditions. The HORIZON 2020 EU-funded MOdular Spacecraft Assembly and Reconfiguration (MOSAR) project [%1] was therefore initiated to develop innovative technologies aimed at standardising satellites and components. Modularising and standardising space systems will benefit the European space industry by enabling mass production of standardised components, reducing assembly costs, shortening the time between customer orders and deployment in space, and facilitating direct in-orbit repair and component upgrades, thereby extending the lifetime of space systems.

MOSAR’s primary objective is to create modular and reconfigurable satellites that can be assembled and adjusted in orbit. The project has developed a demonstrator for reconfiguring cubic modules using a mobile robotic manipulator to simulate module movement. Currently, the manipulator receives fixed instructions for module mobility from software simulations on Earth [%]. This research aims to enhance the system by developing an algorithm to automate module reconfiguration, enabling self-repair and self-assembly. Once implemented, this technology could automate space system assembly and platform construction in space, overcoming current limitations in the space industry.

## Project Objectives and Specification

This project intends to enable autonomous assembly and reconfiguration of modular space systems by implementing a reconfiguration planning program made up of simple algorithms. This program, given the initial state and final state of a modular system, will generate a list of commands to be sent to a mobile manipulator to autonomously rearrange modules on a spacecraft or space platform. The planning program must account for physical constraints imposed by the mobile manipulator present on the modular system; therefore, this project will strive to explore methods of incorporating physical constraints into the planning process.

To achieve the research goal, the primary objective is to implement a functional planning program capable of autonomous module reconfiguration, which will be demonstrated through software simulation. If time allows, an additional goal is to physically demonstrate the planning program by integrating it with the available manipulator arm in the lab to reconfigure real modules.

To achieve the research objectives, the following sub-objectives have been identified:

1. Develop a reconfiguration planning program that generates module movement instructions for a mobile manipulator based on initial and final state configurations.
2. Enhance the reconfiguration planning program to integrate physical constraints imposed by the mobile manipulator.
3. Implement a display function to create reconfiguration slideshows or videos, allowing users to visualise the modular systems reconfiguration process.
4. Conduct systematic testing of the system with various inputs to analyse system performance during solution generation.
5. Demonstrate the system by integrating it with the laboratories robot arm to physically reconfigure real modules.

By pursuing these steps, the project aims to showcase the feasibility and effectiveness of the planning program for autonomous assembly and reconfiguration of modular space systems, potentially paving the way for practical applications in the space industry.

## Report Structure

This document serves as a comprehensive report of the research and development carried out during the Autonomous Re-Configuration of Modular Spacecraft with Manipulator Arm project. The report encompasses the following key components:

1. **Literature Review and Research:** A thorough examination of the current state-of-the-art in modular reconfiguration, including a review of relevant literature and existing technologies in the field.
2. **Detailed Design Development:** Creation of a detailed design plan outlining the implementation strategy for the reconfiguration planning program, specifying key components and methodologies
3. **Implementation Description and Specification:** Description and Specifications of the final implemented design, detailing the development and optimisation.
4. **Design Analysis and Results:** Analysis of the implemented design, records of performance metrics, solution generation times, and failure rates obtained through testing and simulation.
5. **Discussion of Results:** Interpretation and discussion of the analysis results, evaluating their significance and implications within the broader context of the area of study.
6. **Project Management Approach:** Examination of the project management methodology employed throughout the project lifecycle, documenting the evolution of the project plan and strategic adjustments made to achieve project objectives.
7. **Recommendations for Further Work:** Identification of potential areas for future research and development to build upon the findings and achievements detailed in this report, suggesting methods for expanding and refining the implemented system.

# Literature Review

This section builds, and expands, on material previously included in the project Initial Report (see Appendix %)

## Overview of Modular Spacecraft

Modular spacecraft represent a design concept where the overall space system consists of interchangeable modules, each fulfilling specific functions such as propulsion, communication, power generation, or sensing. These standardised modules enable easy assembly to form a unified system, allowing for module movement or replacement to optimize craft efficiency and extend system lifespan. Adopting a modular design approach offers several advantages over traditional methods, including enhanced flexibility, adaptability, and simplified maintenance.

Modules feature standardised interfaces that govern physical and electronic interactions, facilitating seamless integration of modules with different purposes or manufacturers into the overall system architecture. While module sizes and shapes may vary across designs, standardisation principles ensure compatibility for integration. The scalability of modular space system architectures depends on the types and quantities of modules used, providing versatility and cost-effectiveness as the system can be tailored to suit specific mission requirements without necessitating a complete redesign.

## State-of-the-art in Spacecraft Modularity and Automated Reconfiguration

This section explores existing cases of spacecraft modularity and reconfiguration technologies currently or previously in operation. Due to the challenges related to developing automated reconfiguration systems for space operations, there are limited existing cases of automated reconfiguration aside from the International Space Station (ISS). However, modular design principles have been integral in spacecraft development since the 1980s, notably with the introduction of the Multi-mission Modular Spacecraft (MMS).

### Multi-mission Modular Spacecraft (MMS)

The Multi-mission Modular Spacecraft (MMS) was designed and deployed by NASA in the 1980s and 1990s [%] with the intention of decreasing space mission costs. Intended to be recoverable/serviceable by the Space Shuttle Orbiter [%], It is one of the first cases of modular designs seen in the space industry and has paved the way for future innovations.

The MMS consisted of a small number of immobile modules, with the most basic deployed MMS containing only modules for altitude control, communications and data handling, and the power subsystems module [%].

The MMS flew only six missions through its lifetime which was vastly different from the thirty-one expected in the 1970s [%], it suffered limitation in the form of electronic technologies rather than mechanical restraints. NASA’s first Standard Spacecraft Computer (NSSC-1) [%] was developed to prevent requiring an entire redesign of onboard computers for each mission, requiring only a software redesign though this was still a heavy burden affecting the MMS’s mission flexibility. While no longer in operation as of 2006 [%], the system did show cost-savings in the range of 55% to 65% [%]. “The idea of a modular system serving many purposes was the pioneer of the leading systems within the space technology ecosystem today as it has left a lasting legacy” [%]. In the wake of the MMS’s legacy, new design techniques were developed such as the Modular, Adaptive, Reconfigurable Systems (MARS) system-level architecture [%] that has built the foundation for modern space systems.

### Modular Common Spacecraft Bus (MCSB)

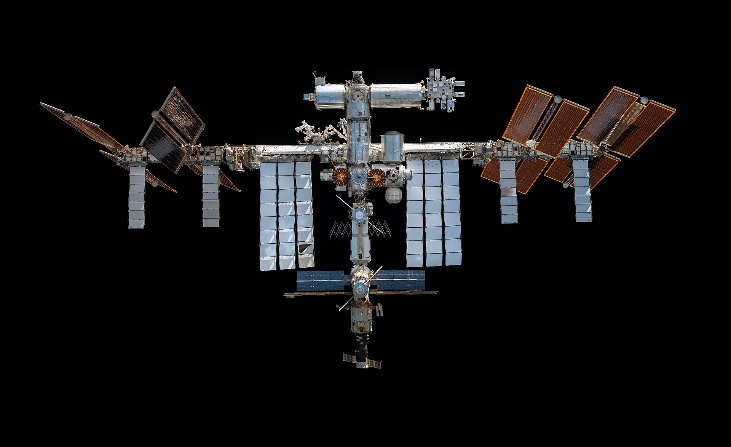
A diagram of a space module

Description automatically generatedThe MCSB is a fast-development, low-cost, general purpose spacecraft platform consisting of a series of 4-5 modules stacked on top of each other, each serving separate functionality [%]. According to NASA, “the spacecraft is roughly one tenth the price of a conventional unmanned mission and could be used to land on the Moon, orbit Earth, or rendezvous with near-Earth objects.” [%]

The MCSB system received the Popular Mechanics 2014 breakthrough Award for innovation in science and technology [%] and is proving to be at the forefront of existing modular space technologies, first deployed on the Lunar Atmosphere and Dust Environment Explorer (LADEE) mission in 2013 [%].

The MCSB system is an example of modularity being used to streamline and reduce costs of the initial development process of the craft, being able to carry up to 50kg of scientific equipment inside its payload module [%], though the end product is still a whole system that has limited in-operation service capabilities and is not capable of being reconfigured to adapt to mission requirements in-orbit.

### International Space Station (ISS)

The International Space Station (ISS) is the largest space platform ever built, created with the purpose of performing microgravity and space environment experiments. First launched in 1998 and expanded through the integration of additional modules and serviced by human occupants up until its planned de-orbit in 2031, it is a monument to advancements in the space industry.

The ISS is capable of reconfiguration using a robotic arm and automated docking with human oversight [%] unlike previous cases, though unsupervised automated reconfiguration is yet to be attempted due to the consequences of failure.

Although the examples provided are not exhaustive, they encompass significant cases of modularity in the history of space exploration. Currently, automated spacecraft reconfiguration remains unimplemented in the industry. This project aims to contribute towards the future widespread adoption of automated modular reconfiguration by developing a system that can be compared with other emerging systems, aiding to identify techniques that offer the most substantial benefits. These techniques can then be utilised to create increasingly advanced reconfiguration systems for space applications.

## Challenges and Limitations of Automated Reconfiguration in Space

The limited deployment of complex automated systems, like automated reconfiguration systems, in space is not due to a lack of interest, but rather stems from the formidable technical challenges and high-risk nature of space missions, which cannot afford failures due to their high cost and critical objectives.

Space systems must exhibit high reliability and operate effectively across a wide range of conditions. As system complexity increases, so does the number of potential failure points, making the validation, verification, and deployment of complex systems in the space industry a lengthy and costly process. Challenges that autonomous space systems face include:

* **Communication latency** – Delays in communications render real-time human intervention impossible, necessitating autonomous systems capable of operating independently without human oversight. Unlike terrestrial applications like self-driving cars that operate under human supervision, autonomous space systems must meet stringent autonomous reliability requirements.
* **Safety Requirements** – Systems will often be hosting valuable scientific equipment while operating in harsh, unpredictable environments with various hazards such as extreme temperature fluctuations, radiation, space debris, ice, and microgravity.
* **Limited Power Sources** – Autonomous systems rely on power sources that may not be constant or reliable. For instance, solar-powered crafts may experience power loss during eclipses or due to unexpected collisions with space debris. Autonomous systems must be capable of recovering from temporary power losses or have reliable backup power sources to prevent mission failure.
* **Isolation** – Unlike on Earth, space missions lack immediate external assistance or observation. Autonomous systems must possess robust sensing capabilities to self-diagnose issues, detect anomalies, and suspend standard operations when necessary to prevent further damage.

Overcoming these challenges demands cutting-edge technology, which has only recently become available, motivating research projects like this one. As computational power and materials sciences advance, we can expect a significant increase in autonomous systems within the space industry in the coming decades.

## Emerging Advancements in Reconfiguration Technologies

### MOSAR Project Outcomes

The MOSAR project has achieved several significant outcomes to date:

* Development of a standardized module framework utilizing the HOTDOCK adapter.
* Design and fabrication of a walking manipulator arm.
* Establishment of a related system architecture for remote control of the manipulator arm.
* Successful ground demonstration showcasing the manipulator arm's capabilities to move and connect modules.

At this stage, the MOSAR demonstrator could theoretically perform reconfiguration in orbit but currently requires manual transmission of reconfiguration instructions to the craft. Further work is needed to enable automated functionality, including:

* Automatic determination of a desired module configuration to meet mission requirements.
* Automated computation of manipulator instructions necessary to reconfigure the craft from one configuration to another.

The following review of automated reconfiguration literature will focus on identifying the best methods for the computation of manipulator instructions.

### Automated Reconfiguration

Automatic planners, algorithms that find a solution for which sequence of operations must be accomplished to achieve a specified goal, have been an area of development attracting wide-spread interest since the earliest days of robotics. Currently there are many different types of automatic planning techniques available. They encompass a large set of algorithmic requirements which trend towards purely discrete or purely continuous search space characteristics. The development of “Hybrid” automated planning approaches with search space characteristics that are not purely discrete or continuous, especially Task and Motion Planning (TAMP) algorithms, represent an area of study of which solutions are considered the most computationally difficult in theory [%]. Consequently, the application of automated planning algorithms to robotic assembly of modular satellites is a very recent development in which little work has been published that implements automatic reconfiguration algorithms while fully considering the range of real-world physical restraints and limitations presented by usage of a mobile manipulator arm in a low-gravity environment.

A diagram of a hybrid planning

Description automatically generated

Taxonomy of automated planning approaches based on their search spaces’ characteristics. Image from [%1].

#### Motion and Manipulation Planning

Motion Planning is finding solutions to move a robot “from one configuration to another configuration without colliding with the objects in the world” [%1]. It involves searching for paths within the robots reach which is a continuous configuration space limited by dimensions represented by the joints of the robot. These collision-free paths are important for robot motion but do not by themselves allow the robot to interact with the world. Further planning must be implemented to allow manipulation of objects through manipulation planning (known as Multi-Modal Motion Planning).

Due to the increased complexity of the problem presented by manipulation planning, the problem is best broken down into a hybrid discrete-continuous search problem of “selecting a finite sequence of discrete action types (e.g. which objects to pick and place), continuous action parameters (such as object poses to place and grasps), and continuous motion paths” [%1].

#### Task Planning

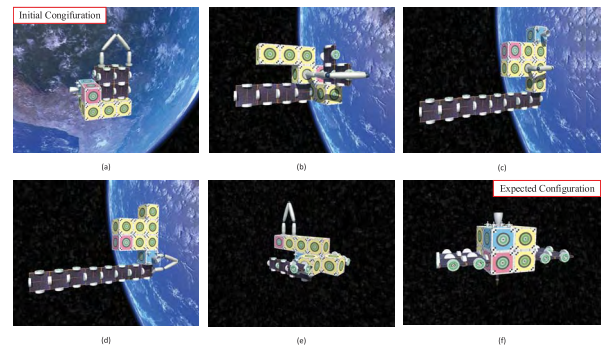
While Motion and Manipulation planning are seen as problems mainly within the robotics field, planning within large discrete domains such as in problems presented by task planning has been more deeply researched within the artificial intelligence (AI) community [%]. Task planning (also known as Action planning) referring to deducing a composition of symbolic actions to achieve a high-level goal (e.g. computing a sequence of actions required to stack boxes in a specified order). The discrete nature of the problem makes it particularly suitable for many machine learning techniques which have particularly advanced in recent years.

#### Task and Motion Planning

Current research in task and motion planning (TAMP) primarily aims to combine the robotics solutions for manipulation planning under physical constraints with the usually unrestricted machine learning approach to task planning. With the goal of deriving automated planning systems capable of reasoning symbolically with discrete “high-level” robotic action sets while geometrically taking into account continuous “low-level” robotic motion planning and restrictions. To date, several papers have developed algorithms for similar TAMP problems to the scenario of modular satellite reconfiguration that unfortunately are not compatible due to the method of module mobility, but act as a proof of concept that a solution is possible [%][%][%].

#### Related Work

The 2010 Intelligent Building Blocks for On-Orbit Satellite Servicing and Assembly (IBOSS) project [%] by DLR provided many advances in the area of satellite modularization with the development of standardised building blocks and interfaces [%]. Simple task planning techniques were implemented using Hierarchical task network (HTN) planning to produce high-level mobile arm instruction sets to then be verified through inverse kinematic checks and motion planning. This implementation solved the discrete and continuous planning problems separately, which simplified the problem however does not allow the separate systems to properly integrate. The system was not capable of efficiently solving more difficult tasks of identifying were solutions where not feasible.



Alternatively, another approach was taken here [%] through the implementation of the melt-grow algorithm [%]. The physical restraints of the robot were not including in the reconfiguration planning stage of the system, effectively reducing the problem to task planning. This reduces complexity though can only be achieved due to the behaviour of the melt-grow algorithm, which deconstructs (melts) the initial module configuration into chains of modules defined as the intermediate configuration, seen in configuration d in figure [%], before then reconstructing (growing) the modules into the expected configuration. The system then does not need to consider whether a move is possible for the mobile arm through manipulation planning as due to the algorithms inclusion of an intermediate state between the melting and growing operations, the algorithm essentially reconstructs the satellite instead of modifying the current state, all required moves are possible for the mobile manipulator and simply require motion planning. While proven to work, this method is shown to be highly inefficient for the mobile manipulator, especially as the number of modules increases in the system. Though, the paper [%] suggests this could be offset by the inclusion of additional manipulators which would consequently increase construction and operational costs.

More recent research has taken inspiration from these previous works to propose a comprehensive Task and Motion Planning (TAMP) problem solver [%] to intrinsically include the robot constraints into the system. The system, seen in figure [%], includes a logic layer, a physical layer, and a feedback system. Where the logic layer acts as a task planner finding a semantic solution by considering the solution as a sequence of states, with module movements defining the transition between states. A graph is developed to represent the possible states where nodes are system states and edges represent module movements which are verified by the physical layer which provides manipulation planning results through the feedback system. Using this graph, the shortest and hence most efficient set of operations to reconfigure the system into the desired state can be identified. The removal of the intermediate configuration present in the melt-grow algorithm improves the efficiency of the solution set of operations, especially as the number of modules in the system increases, requiring less movement from the mobile manipulator.

Diagram of a machine

Description automatically generated

“Architecture of the autonomous robot planning system. The system receives as inputs the start and goal satellite configurations, and iterates between the logic and physical layer until a solution is found.” [%]

The paper notes “the goal of this work was not to set a baseline for planning problems in terms of absolute times, but to demonstrate the usefulness of integrating feedback from the physical layer on the logic layer.” [%], suggesting that there is an opportunity for further research into the components of the planning system and the related feedback strategies to further advance the system towards space applications.

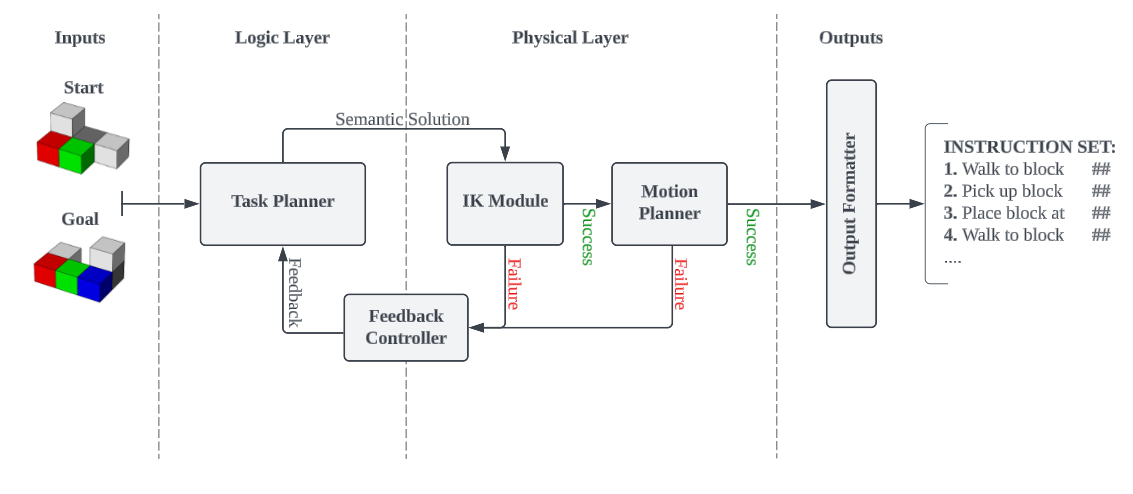
## Gaps and Opportunities

Modular reconfiguration defines a subclass of the generic planning problems usually addressed by TAMPs. Although research has previously demonstrated effective systems that can handle both symbolic and geometric reasoning, their application to robotic assembly and in particular robotic re-assembly is currently limited. There is additionally a distinct lack of discriminating modular blocks by type in existing algorithms which could potentially be implemented without a substantial hit to system performance.

The system proposed in figure [%] is promising due to the robustness of solutions and flexibility of the logic layer, however, there lacks the extensive performance testing required to recognise weaknesses and future improvements and identify why this system could not be used in real-world application currently.

# System Design and Development

## Overview



The Reconfiguration Task and Motion Planner (TAMP) program is designed using Python, leveraging existing Python implementations for controlling the robotic arm [%][%] available in the lab. The program inputs an initial and final state and outputs a list of instructions to reconfigure the initial state into the final state using a mobile manipulator.

The TAMP system comprises three main components: the logic layer, the physical layer, and the feedback controller. Each component has a distinct role, allowing the discrete and continuous aspects of the solution search to be managed separately while ensuring integration and communication between layers through the feedback controller.

* **Logic Layer:** Responsible for discrete task planning.
* **Physical Layer:** Manages continuous motion planning.
* **Feedback Controller:** Integrates the logic and physical layers, facilitating communication and iterative solution searching through feedback strategies.

The feedback strategies implemented by the feedback controller create a control loop behaviour, iteratively refining the solution. Once a feasible solution is found, it is formatted into the required instruction set by the output formatter.

## Logic Layer

### Overview

The logic layer of the TAMP program functions as a Task Planner, managing the discrete portion of the search to find semantic solutions. These solutions are sequences of state configurations, each differing by one module movement, that transform the initial state configuration into the goal state configuration.

While many contemporary task planners use machine learning techniques to find solutions, these are unsuitable for the space industry due to their black box behaviour. Instead, the TAMP program employs simple graph search techniques to ensure transparency and reliability.

### Searching the Graph

To search the graph and find the path to the goal state configuration, two major search algorithms are considered: Depth-First Search (DFS) and Breadth-First Search (BFS).

* **Depth-First Search (DFS):** In DFS, the algorithm explores a path to its full depth before backtracking and trying alternative paths. This can be implemented using a state priority queue that sorts states based on their proximity to the goal state, enabling quick solutions. However, the resulting path may not be the most efficient for the mobile manipulator, as each state transition involves additional movement.
* **Breadth-First Search (BFS):** In contrast, BFS explores all states at one depth level before proceeding to the next level. It searches all states one step away from the starting state, then two steps away, and so on. Although BFS is much slower than DFS and less scalable to larger numbers of modules, it guarantees finding paths with the fewest transitions to the goal state, making it more suitable for efficient reconfiguration.

Given the requirement for solution efficiency over planner speed, the Task Planner implements the BFS algorithm. The pseudo-code for the Task Planner algorithm is shown in Figure [%].

A computer code with text

Description automatically generated with medium confidence

### Generating States

To expand the graph, the task planner generates new states using the “GenNewStates()” function, referenced in the search algorithm pseudo-code(fig [%]). States are generated based on a set of rules that prioritize which modules to move, ensuring efficient state expansion. The priority rules for module movement are as follows:

1. Modules not yet in their final position
2. Modules adjacent to modules not yet in their final position
3. Remaining modules

These rules ensure that the task planner consistently generates new states while prioritizing the movement of modules that need to be repositioned first. This approach minimizes unnecessary movements and helps streamline the reconfiguration process.

A priority queue that prioritizes modules based on their distance from modules not yet in their final position was considered. This would allow for efficient repositioning of deeply embedded modules. However, this added complexity was deemed unnecessary for the current program's scope and would increase computation time for a relatively rare scenario. For larger structures, implementing such a priority queue could be beneficial to improve computation efficiency.

A diagram of a diagram

Description automatically generated

### Trimming States

When handling inputs with large numbers of modules, the search tree expansion can quickly result in a vast number of states, consuming significant memory, and computation time. To expedite the search process, generated states are sorted into a priority queue, and only the highest priority states are added to the search graph. The remaining states are discarded, as illustrated in Figure [%]. The states are prioritized based on their proximity to the desired goal configuration, using the following heuristics:

1. Number of modules already in their final positions.
2. Number of modules not in their final positions but occupying positions that are vacant in the goal state.
3. Sum of the Euclidean distances of the module positions from their final positions in the goal state.

These heuristics measure how far a state is from the goal state and allow comparison between states to determine which is closer to the goal. States are first sorted using the number of modules in their final positions. In the event of a tie, the second heuristic is used. If there is still a tie, the computationally intensive third heuristic is applied.

By primarily using the first two heuristics, the planner reduces the frequency of expensive calculations, thereby speeding up the state comparison process and improving overall efficiency.

### Physical Layer Feedback

When a semantic solution is identified, it is passed to the physical layer for verification. If the physical layer encounters a failure, the transition causing the failure is pruned from the search tree, and all subsequent transitions on that branch are removed. The search then resumes without the failing transition.

An alternative approach would involve performing physical layer checks for each move during state generation. However, physical layer calculations are significantly more computationally intensive compared to those in the logic layer. Thus, it is more efficient to focus on verifying only the transitions within the semantic solution, even if it means spending more time searching for these solutions. This approach balances computational load by leveraging the relatively quicker logic layer to identify potential solutions and reserving the intensive checks for validation.

## Physical Layer

### Overview

The physical layer is primarily composed of an inverse kinematics verifier and a motion planner. Its main function is to verify the physical feasibility of transitions in the semantic solution proposed by the logic layer. It processes a semantic solution and returns either a success or a failure.

* **Inverse Kinematics Verifier:** This module verifies whether the poses for grabbing and placing a module are possible from the current base position. If either pose is not feasible, the module attempts to find a base position that allows both poses. If both poses are feasible from the same base position, the verifier permits the semantic solution to proceed to the motion planner. If it fails to find such a position, it triggers an inverse kinematics failure and discards the semantic solution.
* **Motion Planner:** Upon receiving a transition, the motion planner determines a path from the start pose to the end pose that avoids collisions between the arm, the grabbed module, and other modules. If no collision-free path is found, the motion planner returns a motion planning failure, and discards the semantic solution.

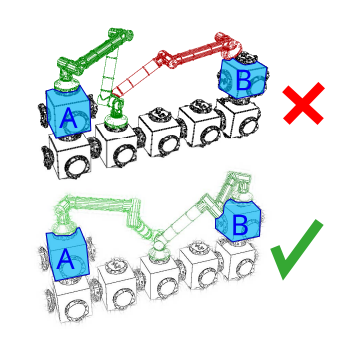
### Inverse Kinematics Verifier

Inverse kinematics (IK) determines the joint angles required to position a mobile manipulator's end-effector at a desired location and orientation. There are two primary methods for solving IK [%]:

* **Analytical Approach:** This method involves deriving a mathematical solution specific to each unique manipulator arm. It is complex because it requires detailed analysis and formulation. However, once the formulas are derived, calculations are extremely fast, making it suitable for frequent computations.
* **Pseudoinverse Jacobian Method:** This iterative method guesses the required joint angles and adjusts them incrementally to achieve the target position and orientation. It can be applied to any arm configuration without needing unique formulations but is computationally intensive.

Given the need for frequent IK calculations, the analytical approach is preferred. The mobile manipulator is represented using Denavit-Hartenberg (DH) parameters, which facilitates the derivation of analytical formulae for each joint. The formulae allow for efficient computation of IK, ensuring that the system can quickly verify and adjust the manipulator's poses.

### Manipulator Base Location Planning



The mobile manipulator present in the MOSAR [%] project can traverse the modular system. If the IK module fails to connect to a module from a position, moving the arm closer to the module that failed the inverse kinematic check could result in a successful solution like in figure [%]. When the inverse kinematics module fails a check, it attempts to move the base to another available surface in between the 2 movement points. As our available arm is stationary, our implementation will only include base location planning if time permits.

### Motion Planning

Originally, the plan was to implement a motion planner using the Rapidly-exploring Random Tree (RRT) algorithm used here [%] to generate collision-free motion paths for the robot arm. However, due to time constraints, a simpler approach is adopted.

In the lab scenario, where modules are reconfigured on a platform by a stationary robot arm, motion planning is simplified. We can assume that a module can be picked up if there are no modules above it and if it is within reach according to the inverse kinematic solution. Modules are then moved between positions by lifting them to the arm's maximum z-limit, moving above the new position, and lowering them into place.

Additional physical constraints are applied through feedback:

* Modules cannot be moved to negative-z values due to the platform.
* Modules must be placed on top of the platform or another module due to gravity.

These constraints will induce physical layer failures, leading to an expected increase in the time taken to generate results compared to operation in space.

### Failure Feedback

The physical layer reports various types of failures upon detecting conflicts, including:

* **Out of reach:** The starting and final positions of the module are unreachable for the current mobile manipulator.
* **No base location:** Although the starting and final positions of the module are within reach, there is no suitable base position on the module configuration to reach both points.
* **Collision**: There is no available path to move the module without colliding with another module.

## Feedback Strategies

Without feedback strategies, the system can find a step-by-step solution to reconfigure modules into the desired goal configuration and verify whether the mobile manipulator can execute the semantic solution. If the mobile manipulator cannot execute the solution, the system fails in its current state.

Feedback Strategies establish a control loop that facilitate communication between the logic and physical layers, enabling them to collaborate towards finding solutions that meet their respective goals. While various feedback strategies were considered, the project focuses on implementing those related to final output validation rather than computation time optimization. However, strategies for both aspects were explored in case of early project completion or for future work beyond the project's scope.

### Semantic Solution Verification

The primary feedback strategy implemented in this project involves simple verification of semantic solutions. Failed solutions are returned to the logic layer, and the problematic transition is removed from the search tree, along with all associated states. The logic layer then continues the search. There is concern that early failures in the physical layer may significantly reduce the search space, potentially hindering the discovery of reasonable solutions. Further testing and research are needed to address this issue and refine the feedback strategy.

### Failure Memory

Although implementing a failure memory, like the one developed in previous work [%], was considered, it was deemed beyond the project's scope due to its primary focus on time performance. The failure memory utilises a machine learning algorithm to predict physical layer failures based on past data, optimizing the system's runtime by reducing failures. Incorporating a failure memory would provide the system with scalability over time with the collection of failure data, enabling it to generate solutions for larger systems in a reasonable timeframe. Though currently it is unsuitable for space applications due to the reliance on black box machine learning algorithms.

# System Implementation and Specifications

## Hardware Specifications

### Processing Hardware

The current software implementation requires a simple processor but is constrained by its speed. The primary hardware limitation is the available RAM, with a recommended range of 2 to 4 GB for the planner, depending on the size of the input state configuration.

During the project, system timing results were obtained using a general-purpose desktop computer with the following specifications:

Operating System: Windows 10 Pro 64-bit (10.0, Build 19045)

Processor: AMD Ryzen 7 3700X 8-Core (16 CPUs) clocked at approximately 3.6 GHz

GPU: NVIDIA GeForce RTX 2070 Super with 8GB VRAM

RAM: 3 x 8 GB DDR4 Memory clocked at 3200 MHz

Memory: 1 TB M.2 SSD with read speeds of 7,300 MB/s and write speeds of 540 MB/s

### Mobile Manipulator

The mobile manipulator available in the lab is the Automata EVA, further details can be seen in Appendix D – Automata EVA Technical Specifications. The manipulator was not physically used during the project due to time constraints. Though, the arm specifications were used for simulation so the current state of the reconfiguration planner can be integrated with the arm in a future project.

## Software Specifications

### Software Architecture Overview

A diagram of a process

Description automatically generated

The modular structure of the final software implementation is illustrated in Figure [%]. The main file integrates the Logic Layer and Physical Layer, facilitating communication and applying feedback strategies between them. Once a solution is found, the Output Formatter is used to display the instructions to users (seen in Appendix [%]) and to create animations of state reconfiguration transitions for visual analysis of the process.

### Logic Layer

The Logic Layer consists of a Task Planner and its related methods, and classes to implement States, Modules and State Priority Queues

#### Task Planner

The Task Planner begins by verifying the start and goal states have the same number and composition of modules; And generates states utilising the State Priority Queue and State Classes. It returns an array of states representing the transitions required to reconfigure the start state into the goal state.

The Planner consists of two main methods, “FindPath()“ and “GenNewStates()”. The “FindPath()” method implements the search algorithm (as show in figure [%]), while “GenNewStates()” expands the tree based on the heuristics specified in the design [%] as can be seen in listing [%]. After generating states, “FindPath()” records their parent states, enabling the program to trace the transition path from the starting state to the goal state.

**GENERATE\_STATES**(state, goal\_state)

state\_queue <- new StateQueue(goal\_state)

from <- state.get\_non\_final\_modules()

to <- state.get\_available\_positions()

state\_queue.push(state.generate\_moves(from, to))

**if** State\_queue.empty() **do**:

from <- state.get\_adjacent\_modules(b)

state\_queue.push(state.generate\_moves(from, to))

**end**

**if** State\_queue.empty() **do**:

from <- state.get\_modules()

state\_queue.push(state.generate\_moves(from, to))

**end**

**return** state\_queue

#### State Priority Queue Class

The State Priority Queue class maintains a sorted list of states using a simple array data structure. States are ordered based on their proximity to the goal state, as determined by the heuristics detailed in design section x[%]. When a state is inserted, a binary search algorithm [%] is used to find the appropriate position in the queue to maintain the queue's priority order. Initially, a linear search algorithm was used for simplicity, but during optimization, the binary search was implemented, significantly reducing the overall insertion time.

#### State Class

The State Class represents a state configuration and stores module positions. It includes methods for state comparisons, measurements, validation, and visualization. Modules are stored in a dictionary, where keys represent module positions and values are module objects.

Initially, a position matrix was used for its simplicity in testing and modifying logic, which sped up development. However, it was later replaced by a dictionary for optimisation as unlike a position matrix, dictionaries do not store unnecessary zero values so increase memory efficiency of the system.

The class includes a verification function to ensure all modules in the state are connected. Originally, an out-of-the-box labelling algorithm from the scikit-image python package [%] was used with the position matrix to verify connectivity. After switching to a dictionary, a new search algorithm was implemented for state verification (shown in figure [%]).

VERIFY\_STATE(state)

found\_list <- []

search\_list <- [state.first\_module\_in\_dictionary]

**while** search\_list not empty **do:**

module <- search\_list.pop()

**for** neighbour in module.get\_neighbours() **do**:

**if** neighbour not in found\_list **do**:

**if** neighbour not in search\_list **do**:

search\_list.push(neighbour)

**end**

**end**

**end**

**end**

**return** found\_list.length() == state.num\_modules()

Several functions are implemented to measure the number of modules in final positions, the number of modules in free positions and the Euclidean distance between all modules in non-final positions and their final positions. Since these measurements are often requested multiple times for the same state, the calculated values are saved within the state after the initial computation and updated only if the goal state changes.

For visualizing reconfigurations and aiding in-depth testing and analysis, the State Class includes a display function. This function translates the dictionary into a position matrix, which is then displayed as a 3D matrix using Matplotlib [%]. Configurations, such as the one shown in figure [%], can be visualized. Additionally, this function is used to create reconfiguration videos, enabling clear visualization of the system's output.

A colorful cubes with different colors

Description automatically generated

As shown in listing [%], the State Class includes a function called GenerateMoves() for generating a list of states for mass movement of modules. This function takes two lists: one of module positions and another of positions the modules can move to. For each module, the function validates the state without the moving module to ensure the state remains intact during movement. It then moves the module to each of the possible positions, adding the new state to a return list if the state is valid after the movement. This custom mass movement function optimizes state generation during search tree expansion.

#### Module Class

The Module Class is a straightforward class that holds information about a module, primarily used for comparing modules through an equals function. This function can be modified to adjust which module properties determine equality. In its current implementation, colour is used to decide if two modules are equal. There is functionality to compare by module type or module identification number. However, comparing by colour simplifies analysis during testing, as it allows for visual differentiation of modules when displayed.

### Physical Layer

#### Inverse Kinematics Verifier

As detailed in design section [%], the implemented Inverse Kinematics Verifier uses an analytical solution to calculate the manipulator joint angles required to position the end-effector at a specified location. Initially, an analytical formula was developed specifically for the Automata EVA [%] arm available in the lab. This limited the compatibility of the reconfiguration planner to only that specific manipulator.

To increase compatibility, a Unified Robotics Description Format (URDF) file was created for the Automata EVA, as shown in appendix [%]. The IKPy package [%] was then used to generate an analytical solution for use by the Inverse Kinematics Verifier and Motion Planner. Users can now update the reconfiguration planner to work with different mobile manipulators by simply replacing or modifying the URDF file.

The inverse kinematics verifier is used to ensure that the start and final position of each module movement in the reconfiguration semantic solution is reachable by the mobile manipulator. In the case of a module being out of reach, the verifier returns the state transition that caused the failure. Otherwise, the semantic solution is sent to the motion planner for further processing.

#### Robot Description File

A URDF file is used to define the mechanical structure, dimensions, joint configurations, and physical constraints of the mobile manipulator the physical layer is simulating to verify the logic layers semantic solution. URDF files are an XML-based file format that is widely used in robotics [%] to describe robots to software systems. The file describes a robot as a collection of links and joints that can articulate around each other according to specified constraints. URDF files are also modular meaning they can include other URDF files, aiding in the design of particularly complex robots. This for example means that a user can develop a URDF file for an arm end-effector and simply include it in the already existing arm file to attach it to the arm.

URDF files also allow for the visualization of the defined arm joints, as seen in figure [%] which can be overlaid on top of our module state display to visualise mobile manipulator pose on the modular space system. Additionally available online packages such as urdf-loader [%] can display the visual meshes described in the URDF file to view the mobile manipulator in more detail such as seen in figure [%].

A graph of a line graph

Description automatically generated

A white robot with a black foot

Description automatically generated with medium confidence

#### Motion Planner

Due to time constraints during the project, instead of implementing an advanced motion planner, a simple motion planner was implemented that makes assumptions based on the lab environment surrounding the mobile manipulator available in the lab. Physical constraints of the robot arm and the environment are applied to the system during module movements using 2 basic rules:

1. Modules can only be picked up by the mobile arm if no blocks are above them.
2. Modules can only be placed at a supported position (above another module and on the ground) and cannot be placed at negative z values (below the ground)

The combination of these two rules applies the physical constraints of gravity and the presence of the ground. If either rule is broken, the planner returns the state transition causing the failure. Otherwise, it generates and returns an instruction set in the form of an array containing each manipulator action sequentially required to perform the state reconfiguration. The available instructions the motion planner can generate can be seen in fig [%]

|  |  |  |
| --- | --- | --- |
| Instruction | Supplied Information | Action |
| Connect | None | Signal the arms end-effector to grab/connect |
| Disconnect | None | Signal to the arms end-effector to drop/disconnect |
| Move to | Position, Joint angles | Move the arm to position the end-effector at the supplied position by transitioning the arms joint angles to the supplied joint angles |

### Feedback Strategies

Currently, the system employs a basic feedback strategy. Upon detecting a failure in the physical layer, the system returns the state transition responsible for the failure. Subsequently, the branch of the tree originating from the failing state transition is removed, eliminating states where the failing state was as an ancestor. The pruned search tree is then passed back to the logic layer to pursue an alternative semantic solution.

## Implementation Challenges

The first implementation iteration suffered from several initially unforeseen problems that required in-depth analysis and modifications to the system. These issues resulted in the system either being unable to find a solution, requiring more powerful hardware, or taking so long to find a solution that testing was not feasible.

### Memory Usage

The implemented data structure relied on a dictionary with positions serving as keys, storing only essential information, and facilitating straightforward modifications to internal logic and computations. However, during development, when testing configurations with a larger number of modules, the system encountered memory issues, halting the search prematurely despite having ample memory (24 GB). Upon investigation, it was discovered that the in-built copy module in Python not only replicated objects but also copied referenced objects within the object, leading to a duplication of the entire search tree with each transition. This oversight stemmed from a lack of familiarity with Python's memory management mechanisms. Subsequently, the development of a custom duplication method for the class significantly reduced memory usage and improved the efficiency of the search algorithm by 90%, enhancing the capabilities of the logic layer.

### Repeated Computations

During an analysis of the system's performance, the Generate\_States function emerged as one of the slowest processes. This function inserts states into a priority queue using a custom comparison function, which involves calculating multiple measurements for each state to determine priority. Initially, the linear search used to find the insertion index was suspected to be the bottleneck. However, after implementing a more efficient binary search algorithm, the expected improvement in insertion time was not fully realized.

Further analysis showed that when inserting a state, if there were many states in the queue, the inserting state would undergo many comparisons to find the insertion index required to maintain priority order. Each time the inserted state is compared against another state, it would calculate its own measurement values and ask the other state to also calculate its measurement values. As these measurement values are based on their module layout, composition, and the goal state, all of which does not change during runtime, there was no need to be repeating calculations.

To address this issue, we optimized the process by saving measurement values upon calculation for each state and recalculating them only when necessary or when the goal state changed. This approach, coupled with the implementation of a binary search, significantly reduced the overall number of calculations required during insertion, improving the efficiency of the system.

### Optimising Python

The greatest overall roadblock to further success and experimentation simply came down to the speed of the system and identifying slow processes. Due to the nature of the python programming language, often a considerable time-wasting process was a simple function that unknowingly had large overheads involved. The most successful method of improving the speed of the system during development was writing custom functions that utilise libraries built to utilise the C programming language. While there was no issue writing the custom utility functions, identifying where custom functions needed to be implemented took up most of the dedicated development time.

# Testing and Results

## Testing Method

To test the system, a generated set of module configurations were input into the system, and the total time spent in the logic layer and hardware layer were recorded separately for both the system with and without physical constraints applied. An average time is then calculated for configurations consisting of 4 – 9 modules. The branching factor was also varied throughout the test to analyse the effect on total time, failure rate and number of moves returned in the final solution.

## Performance Metric

To measure performance of the overall system, the testing methods measure:

* The number of failures encountered during the solution search with feedback strategies.
* The number of modules effect on failure count
* The number of search branches effect on failure count
* The number of search branches effect on number of moves in the solution.
* The number of search branches effect on search time.
* Total calculation time.

To Quantify the performance of the implemented feedback strategy, the number of semantic solutions that fail in the physical is used. The physical layer is the most computationally demanding section of the overall system, so it is desired to be used as little as possible. Time spent calculating results is recorded for comparison with other systems and to prove the systems capabilities for real-world use but is not considered a measure of project success.

## Analysis of results

A table with numbers and a number of objects

Description automatically generated with medium confidence

Tests were conducted on a system with the hardware specifications shown in section [%], results can be seen in appendix [%]. As expected, as the number of modules in the input configuration rises, so does the number of physical layer failures during the solution search as fig [%] shows. Interestingly, the more modules in the state, the higher the number of branches the task planner is using needs to be to completely avoid failing the search by reaching max recursions. Fig [%] shows a good example of this where failed tests are labelled DNF, and passed tests show the total time take to find a solution divided by the number of moves in the solution, giving an average time per generated move. The time it takes to find a solution seems to be almost completely irrelevant for analysis of the system, as it is dependent on the configuration of the state entered. For example, if a module must be moved out of its final position first to enable another module to be moved to its final position, it will take longer to find a solution than if all modules are already accessible.

# Discussion

## Interpretation of results

The results reveal that while the system is almost guaranteed to find a solution, if possible, the efficiency of the final solution and the time taken to find the solution are currently highly dependent on the logic layer maximum number of branches in the search tree. The optimal number of branches seems to be a function of the number of modules in the state configuration, with higher module numbers requiring more branches. More branches are required due to the increased number of physical layer failures present with a higher module count (seen in fig [%]) requiring more alternative paths to be searched to find a solution. Though, the time spent in the physical layer does not seem to increase according to a clear trend according to fig [%] suggesting that many failures are from the inverse kinematics verifier, preventing the expensive use of the motion planner.

As the logic layer takes exponentially more time to find a solution as the number of branches in the search tree increases, and the physical layer produces more failures according to the number of semantic solutions produced, the best way from this point forward to improve the system is to:

1. Optimise or improve the logic layer search algorithm to reduce the exponential nature in search time.
2. Enhance the feedback strategies used from physical layer failures to give more information to the logic layer allowing smarter trimming of the search tree. For example, when trimming a state from the search tree, if equivalent state exists as the result of a different path, append the trimmed branch to that state. This prevents the wastage of the searched paths along the trimmed branch that may otherwise be valid.

## Comparison to existing work

Solutions of the implemented system are fair more efficient than solutions proposed by the melt-and-grow algorithm [%] used previously, however the melt-and-grow algorithm can handle far more modules.

The system used as a primary source of inspiration [%] tested for a different set of performance metrics, so is hard to compare against. However, when conducting the 5-module test also conducted in the study we received much faster results as seen in fig [%]. The timing comparison is slightly unfair as our system does not implement an advanced motion planner or generate instructions for mobile manipulator walking. Therefore, only the logic layer timing can be fairly compared which is highly in favour of our system, likely due to more time put towards optimisation and different overall project goals.

|  |  |  |
| --- | --- | --- |
| 5 module test | Our result | Previous system result |
| Logic Layer Time | 0.008 s | 0.21 s |
| Physical Layer Time | 0.147 s | 46.89 (-+20.6) s |
| Total Time | 0.154 s | 47.10 (-+20.6) s |

## Implications – (potential impact of work on the field)

At present, the developed reconfiguration plan serves as a proof of concept and gives a modular base for further development to improve on. As the system is comprised of multiple parts working separately, its possible for multiple later projects to develop the system in parallel, focusing on developing separate portions of the overall system increasing development time of a more capable system.

It was suggested during project demonstration to industry professionals that at its current state, the system does not consider enough factors to reliably operate unsupervised. Though could be implemented in the manufacturing or construction industry shortly under supervision to for example, stack and track containers in a warehouse. The production of a less capable system for industry could be key to raising the funds required for further development and eventual adoption in the space industry.

# Planning and Time Management

## Project Management Procedures

To streamline the design and development of the project, the project followed a traditional engineering product development cycle consisting of 5 phases:

**Initiation** - The definition of the problem and the projects goals, requirements, and risks. This phase was completed by the given description of the project and further questioning of the project supervisor.

**Planning -** The definition of how to solve the problem by outlining the details and goals to meet the defined requirements. This phase was completed by the production of the initial report seen in appendix [%], the project plan seen in figure [%], and a conceptual high-level product design.

**Execution -** The working phase where the plan designed in the previous phase is put into action and the product is developed. This was completed according to the created project plan and was finished in its majority by the project demonstration day on the 29th of April 2024.

**Controlling & Monitoring -** This phase runs alongside the execution phase and involves tracking progress and adjusting the workflow to remove potential roadblocks.

**Closure -** Reflecting on the progress and results to officially end the project. This phase is conducted through analysis of project results, documentation of completed work and reflection of project success which is represented by this document.

Each phase was given a set of weeks to complete within the project plan, and every Friday a review of the plan was conducted to monitor progress and aid in modifying the plan in the case of unexpected roadblocks.

## Project Management Reflection

The project went according to plan through to the development of the physical layer. Due to unfamiliarity with robot kinematics, little in-depth design was created in the planning phase of the project with the assumption that with the knowledge of what each section of the physical layer needed to accomplish, figuring out how to accomplish it would not be a notable obstacle. This led to the physical layers’ development taking far longer than expected, over-running its planned development time by a week despite completing the logic layer a week earlier than expected. Due to overrunning the deadline, the project goal was instead completed by defining a simple physical layer rule to use for feedback such as “is the module at the top of the stack and hence, can be picked up in an environment with gravity by a stationary arm”; allowing the remainder of the project to be completed, making it possible to develop and analyse a range of feedback strategies without sacrificing time to develop a mostly unused and complex simulation.

Despite the delay, the final product does match what was planned at the beginning of the project, and as such the goals of the project have been filled. This can be attributed to appropriate levels of slack in task timing guidelines and creatively making use of out-of-the-box implementations to decrease production time drastically and reduce complexity.

## Risk Assessment

This section builds, and expands, on material previously included in the project Initial Report (see Appendix %)

Risk register:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Risk Description | Impact | Risk Probability | Mitigation of Risk | Effectiveness during project |
| 1 | Missing or corrupted documents | High | Medium | Documents are backed up to a GitHub repository | Highly effective |
| 2 | Ambitions for project are too great for the project time limit | High | High | Setting appropriate scope expectations from the beginning of the project | Slightly effective – did not properly take into consideration prior knowledge |
| 3 | Illness or work unavailability | High | Medium | Record illness and provide proper explanation for missing work in final report. Decrease scope to provide meaningful results | Highly effective – Illness affected several weeks of the project; However, scope was reduced appropriately |
| 4 | Losing test results | Medium | Medium | Produce lab reports to document progress | Highly effective |

## Evolution of Project Plan

The project plan saw little modification over the project. During progress reviews during the project, if it was seen that a section of the project would overrun its deadline, alternative methods of reaching a functional overall system were found that involved sacrificing small features such as including module orientation and module connectivity directivity direction. These features were still considered in the completed implementation allowing them to be designed and implemented with relative ease when the project is further developed in the future.

# Conclusion

This paper details the development of a hybrid reconfiguration planning system through the completion of the following sub-objectives:

1. Development of a task planner to create high-level semantic solutions to state reconfiguration.
2. Development of a motion planner to impose robot capabilities, geometry, and physical restraints on the semantic solution, and discard infeasible solutions and continue the search for solutions through feedback strategies.
3. Development of state and state reconfiguration plan visualisation functions to create videos of reconfiguration simulations.
4. Conduction testing of the system through various inputs to analyse performance.

We demonstrate that the system can produce efficient solutions and potentially can be integrated with the robot arm in the lab to complete sub-objective 5. Though the implementation of enhanced feedback strategies is needed to improve generation time. The base high-level plan for the system has great potential for further development.

# Further Work

This base system provides many opportunities for future work to enhance its capabilities. From minor changes such as support for the movement of multiple modules at once, multiple mobile manipulators or introducing module orientation and connection direction; To major changes like introducing a failure memory to predict physical layer failures, a purpose-built implementation utilising parallel programming, or the modification of the program to work in real-time so it can compensate for a non-stationary environment. To further identify areas of improvement, the next suggested development of the project would be the creation of a function to generate random goal states from a starting state, to be input into the system. This would allow the automation and conduction of mass testing to develop a data set for analysis.

# Appendix D – Code (need to cite libraries used) (multiple appendix’s)