A Motion Planning and Velocity Collision Avoidance Framework for Bilateral Manipulation

by

Isabelle André

Supervisors: Dr. Tim Salcudean, Alaa Eldin Abdelaal

ELEC 499: Undergraduate Thesis (6 credits)

BASc. Electrical Engineering

UNIVERSITY OF BRITISH COLUMBIA

Vancouver

December 30, 2021

Chapter 1

Introduction

1.1 Minimally Invasive Surgery (MIS)

Minimally Invasive Surgery (MIS) is a common surgical procedure decreasing the required size of skin and subcutaneous tissue incisions [1]. MIS is gaining prominence in the medical domain, replacing various surgical procedures once requiring larger-scale open surgery such as prostatectomy and hysterectomy [2]. While it may be more challenging to visualize the inside of a patient's body through incisions spanning centimeters, numerous benefits can be noted from using smaller incisions than would otherwise be used in conventional open surgery. The use of smaller incisions allows a faster recovery time, less scarring, and a lower probability of postoperative complications [3].

1.2 Robot-assisted Surgery

Laparoscopic surgery consists of manually inserting an endoscope and compact surgical tools through a small incision on the patient's skin to visualize the organs on an external monitor [4][5]. However, a counter-intuitive psychomotor challenge is introduced, as the surgeon's movements external to the patient's body result in the opposite maneuver being executed at the other end of the tool, with the incision acting as the point of rotation known as the fulcrum [6][7].

Robotics and telerobotics are an emerging advancement in the field of MIS, offering solutions to the lingering pitfalls of manual laparoscopic surgery [8]. Robot-assisted Surgery (RAS) introduces the use of computer-based robotic surgery to mitigate the limitations of the range of motion of the tools and endoscope control [8][5]. The instruments are held by a robot regulated by an external surgical console teleoperated by the surgeon, improving the poor ergonomics of lengthy or complex surgeries. Such a setup provides a more intuitive manner of manipulating the tools by offering a wrist-like motion and inversing the external controls programmatically to counter the fulcrum effect [9]. The robotic manipulators also facilitate accessing remote and difficult locations such as within the thoracic cavity during thymectomy surgery. Despite the numerous benefits that RAS has brought to modern surgical medicine, advanced teleoperated surgical techniques remain a dexterity challenge regardless of the surgeon's experience.

1.2.1 The da Vinci Surgical Robot

The da Vinci Research Kit (dVRK) developed by Intuitive Surgical Inc. [10] is an advanced master-slave platform with multiple robotic arms and manipulators remotely controlled by a console and a surgeon. Designed to provide surgeons with a greater level of dexterity, the da Vinci Surgical System provides seven degrees of freedom and a stereoscopic view at the tip of the endoscope [11][12].

The first-generation da Vinci Surgical System consists of three 7-DOF actuated Patient Side Manipulators (PSMs), one 4-DOF Endoscopic Camera Manipulator (ECM), and two Master Side Manipulators (MTMs). Each PSM manipulator of the da Vinci enters the body through an

incision acting as the fulcrum, constraining the motion akin to a spherical joint moving about a Remote Center of Motion (RCM) [11].

As the demands of manually tele-operating multiple robotic arms at the console remain a multi-tasking and dexterity challenge, there is an opportunity for automation to further improve the flow of surgical operations. In addition to controlling up to three PSMs at once, the surgeon must also accurately position the ECM for uninterrupted visual feedback of the tools [6]. While research has been conducted on the automation of various robot systems, the use of motion planning with the da Vinci robot in a surgical context is relatively new and requires further research and assessments before being safely practiced in surgery.

1.3 Motivation

The addition of a fully intelligent and automated system is one of many recent advancements in robotic surgery research [13]. Attempts to leverage the autonomous capabilities of multi-arm robots have been made by exploring the parallelism capabilities in developing execution models for surgical debridement [14] and multi-throw suturing sub-tasks. The concept of parallelism relies on the design of coordinated state machines to control the movement of manipulators in coupled or decoupled-motion sub-tasks [15]. Although robotic manipulators can be automated in parallel by coordinating their state machines, such a model may lack flexibility and responsiveness in a dynamic environment. Multi-arm robot automation in simple surgical sub-tasks with the da Vinci Surgical Robot may be optimized with the complement of motion planning.

The automation of surgical robots for integrated tasks introduces many challenges such as the avoidance of static and dynamic obstacles [16][17]. Furthermore, deadlocks are a known occurrence in concurrent systems requiring the use of mutually exclusive resources [18], and must be reliably detected and solved if a solution exists. As RAS involves a variety of different operations and manipulator movements, an extensive motion planning and collision avoidance framework would be needed to adapt to a variety of automated tasks and edge cases. The integration of motion planning and a dynamic collision avoidance method into autonomous multilateral task execution could improve task efficiency while conserving the parallelism aspect of the robotic system.

1.4 Objective

This thesis focuses on the development of an integrated task and motion planning (TMP) strategy for a multilateral automated model, providing a flexible task coordination approach in a dynamic environment. TMP is known as the problem of planning the actions of robots moving through an environment containing a number of objects, while changing the state of the environment or objects [19][16]. Motion planning will be implemented to generate a collision-free path for each PSM moving towards a point of interest. The position and range of each individual arm will also be tracked in order to avoid possible collisions using a Velocity Obstacle approach. The main idea of this approach is to compute collision-avoiding velocities with respect to other robots, such that each robot may continue on their path without changing their initial trajectory. The objectives of this work concern identifying and handling collisions with the static environment obstacles and potential collisions with PSMs dynamically without impeding the parallelism aspect of the robotic arm movements.

Chapter 2

Related Work

The integration of a motion planning mechanism in autonomous robotic surgery aims to improve the robustness of a task planning model by preventing robot self-collisions as well as robot collisions with the surgical environment. An effective multi-arm motion planner and coordination framework should be scalable, cooperative, and adaptable to multiple purposes [20]. Trajectories should be planned such that one arm does not impede the motion of another by colliding or blocking their target.

In this section, we examine previous work in multi-robot coordination, task planning, and motion planning algorithms in centralized and decentralized systems. Static and dynamic coordination methods are considered to formulate an optimal integrated task and motion planning framework. Collision avoidance methods are explored to integrate a dynamic avoidance mechanism suitable for a multi-arm robot system such as the da Vinci Surgical Robot.

2.1 Parallelism in Multilateral Automation

The concept of parallelism is used to leverage the capability to automate tasks in multilateral systems [15]. Multilateral execution models in which each robotic arm contributes to one or more tasks can be used to compare the efficiency of parallel and sequential execution methods. Previous work including the use of sequential models involve a single arm moving at once while blocking other arms from functioning until its individual motion is complete [21]. As sequential logic is single-minded, non-concurrent models often show a lack of performance during time-sensitive surgical tasks. Having the flexibility to be used in different configurations than what would be expected from a surgeon, concurrent multilateral models may alternatively be used to optimize task efficiency.

Abdelaal *et al.* demonstrate the concept of parallelism for the execution model of coupled-motion subtasks, in which multiple arms share the same resources, and decoupled-motion subtasks, where multiple arms execute a subtask independently from one another [15]. The concurrency of state machines in robotic subtasks with the da Vinci Robot allows two arms to move independently and in parallel using identical state machines to complete a debridement subtask more efficiently. It is observed that without a coordination or inter-arm communication model, the arms often collide over the receptacle due to the coupled-motion task and mutual-exclusive nature of the resource [15]. This case demonstrates the convenience of using a hierarchical structure to coordinate the individual state machines in a coupled-motion subtask, in which arms are expected to share resources without collision.

A Hierarchical Concurrent State Machine (HCSM) parallel execution model is proposed for coordinating manipulators and resource acquisition for a surgical debridement subtask with the two-arm da Vinci Surgical Robot. HCSMs possess multiple sub-state machines functioning concurrently coordinated using a “master” state machine or controller, allowing the system to be present in multiple states at once [22]. Ahmad *et al.* supports that using a hierarchical structure to coordinate and organize concurrent state machines permits behaviors to be grouped systematically. The act of encapsulating the logic of a single aspect in one state machine, such as picking up an object, simplifies the development, scaling, and validation of complex systems

[22][23]. Using external motion planning solvers and sampling, Wolfe *et al.* present a hierarchical planning system modeling primitive actions and representing finite continuous choices using a sampling based approach [20]. Reliability and failures are handled within the primitive actions on different levels of the hierarchy. Such a system increases the robustness of the hierarchical system such that an optimal plan can be found by exhaustive search.

While hierarchical structures offer a structured method of coordination for the robot and resources under optimal conditions, the collision avoidance mechanism remains only by design, and may not be as robust in a dynamic environment in which robotic arm workspaces overlap. Collision avoidance by design is less adaptive and not easily scalable to environment changes or additional arms, as additional state machines would need to be integrated in the controller. This suggests the possibility that if refined with a concrete dynamic collision avoidance method, concurrent state machine models, whether un-coordinated or hierarchical, could function with even more efficiency than an HCSM execution model.

2.2 Multi-Robot Coordination in Task Planning

One of the main challenges of coordinating the movements of multiple dynamic objects is maneuvering within the workspace without colliding with the static environment or other dynamic arms during the execution of a task. To simplify the problem, we may consider each mobile robotic element in the workspace as a single robot entity, together forming a Multi-Robot System (MRS) [24]. A MRS is defined as containing more than a single robot in a specified environment and may operate in cooperative or competitive behaviours. Within a surgical environment, the system works cooperatively as multiple manipulators interact together in order to complete a surgical subtask with the common goal of improving the system's outcome [24][25][26]. Typical issues relating to the task planning and coordination aspect of a cooperative MRS include resource conflicts, dynamic coordination, and robot communication mechanisms to avoid collisions dynamically.

This section describes the tradeoffs between static and dynamic coordination, and centralized and decentralized planners. An overview of common problems found in concurrent robotic systems is given, including resource conflicts and deadlocks.

2.2.1 Resource Conflict

The resource conflict problem is defined as a situation in which multiple robots attempt to access the same space or object in the event that their workspaces overlap. In order to prevent collisions, the mutual exclusiveness of resources and individual points in space should be preserved [24]. For instance, during a surgical debridement subtask with the da Vinci Surgical Robot, a resource conflict is encountered. The receptacle in which damaged tissue is dropped shown in Figure 2.2.1 is only large enough to allow a single PSM to access the resource at once without collision with other manipulators.

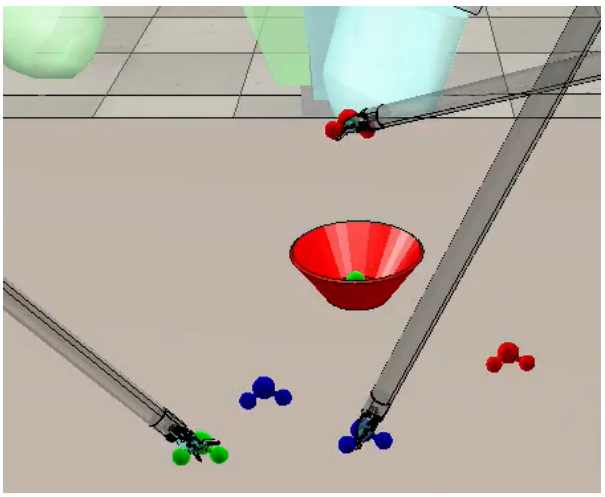


Figure 2.2.1: Trilateral tissue debridement task with the da Vinci Surgical Robot. Two PSMs wait until a PSM leaves the bowl, avoiding potential collisions at the bowl.

In a multi-arm robot system, the coordination of arms accessing a resource can be managed as a hierarchical model using a master centralized FSM. This case demonstrates the use of priority schemes and collision avoidance by design using a centralized machine to coordinate the entire system's actions. Chen *et al.* present a learning classifier based on dynamic allocation of priority methods to improve the performance of a MRS [27]. A high level hierarchical planner is introduced to resolve conflicts by assigning priority to individual robots. Using a priority scheme to determine which agent gets the right of way in accessing a resource has been shown to be effective in the resolution of inter-agent conflicts [27][28][29]. However, fixed priority schemes do not adapt well to different tasks and environments as they do not take individual robot task goals and constraints into account. A flexible framework assigning priority dynamically or using a reward based system may be a better fit in a dynamic MRS system. Similarly, Abdelaal *et al.* bypass the space conflict in a surgical debridement subtask by controlling each PSM of the da Vinci Robot using FSMs dictating different states such as picking, waiting, and dropping [15]. This allows the use of a resource sequentially using a deterministic coordination model, protecting the mutual exclusivity of the receptacle.

Introducing the concept of inter-robot communication as a solution to the resource conflict problem, Marcolino and Chaimowicz propose a decentralized coordination algorithm in which a MRS traffic is controlled by managing congestion as groups of robots move towards a common goal location [30]. The algorithm uses a probabilistic FSM to coordinate their own path by relying on local sensing and communication to warn their teammates about possible collisions, following which, robots are able to re-plan their individual paths to avoid clustering in one space. As opposed to previously stated fixed methods, each robot path is dynamically dependent on another's movements rather than restricted by a predetermined centralized source [31][32].

The use of a decentralized algorithm in conjunction with a method of inter-robot communication would be a flexible approach to the resource conflict problem in a dynamic environment. In this case, a robust decision making algorithm for each individual robot would be required to interpret and form decisions based on the implicit data collected from the environment.

2.2.2 Deadlocks

In an MRS, deadlocks are a state of equilibrium of system dynamics causing the robots to come to a stop before completing their task [18]. For instance, in the case of a tissue debridement surgical subtask executed by two manipulators, deadlocks would occur if a manipulator blocks the second in a way such that neither of them is able to continue along its trajectory without colliding.

As deadlocks are a well-known concept in concurrent systems, multiple methods of deadlock resolutions exist, although their suitability depends on the type of system implemented [18][33]. Alonso-Mora *et al*. present a method for motion planning applying to groups of robots performing tasks in a dynamic environment such as room exploration [34]. The approach is centralized for robots in the team and decentralized with respect to moving obstacles, meaning that robots are only required to measure the position and velocity of dynamic obstacles within a local range to solve the deadlock [34]. The proposed approach automatically generates alternative valid plans. However, due to the dynamic nature of some integrated tasks, there may exist no plan that guarantees the task's success [35][36]. Therefore the approach first identifies for which environment behaviour the plan is guaranteed to succeed, before replanning trajectories using local environment information. Nett and Schemmer coordinate the access to shared physical resources and deadlock resolution by enabling real-time explicit communication between robots about their trajectory [37]. By scheduling shared space and resource access and the intersection of their paths, dynamic robots can cross paths without collision. In the case of deadlock, robots would organize and communicate the deviation of their individual trajectories. The use of explicit communication as a means of information sharing between robots ensures the accuracy of the environment information between robots [37][33].

Jäger and Nebel present a decentralized coordination method to avoid deadlocks and potential collisions in a MRS by coordinating independently planned paths of the members [33]. As centralized approaches are computationally demanding, inflexible, and rely on a global communication network, distributed algorithms are used to limit communication to local pairs of robots [37]. The algorithm consists of tracking the distance between robots and triggering an exchange of information about their trajectories when the minimum distance drops below a certain threshold. In the case of deadlock, individual path planners detect the event by time elapsed since last movement, and re-plan their trajectory until the deadlock is resolved. This method demonstrates the use of conditional communication as the vehicle's trajectory information is only broadcasted once within a certain radius of another, allowing a conditionally explicit system to be formed [38][39]. The robots do not use any global synchronization nor do they interfere with each other, resulting in an adaptive coordination framework reactive to deadlocks, with lower communication demands.

2.2.3 Static and Dynamic Coordination

Multi-robot coordination is the core element of task planning and collision avoidance in a MRS. Coordination can be categorized as static or dynamic. Static coordination manifests as a predetermined set of rules or conventions designated prior to the start of the task execution [24]. Examples of such include controlling mobile robots by applying "traffic rules". Kato *et al.* apply traffic rules to a MRS, restricting the range of allowed movement using known environment information and information on the mobile robots, such as speed, size, shape and

quantity [40]. Collisions are avoided by setting rules and countermeasures for positions at risk of collisions, or restricting the range and directions of movements accordingly. Enforcing traffic rules to coordinate individual robots allows for a simple static system, such that the system does not need any form of inter-robot communication [40][41].

While static coordination has the advantages of fast and efficient computation times for complex tasks, it may perform poorly in real-time systems due to the non-deterministic nature of moving objects in the workspace. Once a motion is initiated, the robot will not respond to subsequent changes in the environment as a trajectory has already been computed according to the set traffic rules. Dynamic coordination provides a more accurate solution to real-time environments by enabling communication between robots [24][42]. An MRS may exchange information such as position and trajectory, or implicitly deduce such information using sensors, before taking adequate measures in order to prevent possible collisions from occurring [30]. This process describes the use of explicit inter-robot communication, further discussed in Section 2.2.3.

Dynamic coordination meets the demands of a real-time environment, providing adaptive solutions to late environment transitions, but may behave unreliably in complex systems, resulting in longer computation times or deadlocks. An example of this case in a tissue debridement simulation would be when an arm in a dropping state, attempts to move to a different location, but is blocked by a second arm wanting to drop an object in the bowl, and cannot compute a path without collision. The second arm cannot access the resource as the first is using it, and the first arm cannot leave the resource, as the second arm is blocking its path. In this case, arms are in a stale-mate and may remain unable to move until another condition clears the situation. Deadlocks occur in concurrent systems between one or more robots when one blocks the other from accessing the same resource such that no solution to the problem can be found [33]. The deadlock must reliably be detected and resolved, often by altering the timing of the robots or resetting the system. In an ideal system, deadlocks are a rare occurrence and measures to solve deadlocks are used as a last resort [33][43][44]. While deadlocks are a rare occurance in a robust system, a deadlock resolution algorithm should nonetheless be implemented to handle such instances by resetting or altering the timing of the system. A hybrid model can be further explored, employing static coordination when modeling an integrated task planning system to set basic rules, while handling real-time dynamic collisions by broadcasting the trajectory of the robots involved in the potential collision as incidents arise.

2.2.4 Summary

This section explored the multiple challenges and considerations of multi-robot coordination in a dynamic environment. The resource conflict problem was defined as a situation in which multiple robots attempt to access the same space or object in the event that their workspaces overlap. In order to prevent collisions, the mutual exclusiveness of resources and individual points in space should be preserved [24]. Deadlocks in an MRS context were defined as the state in which robots block each others' paths, preventing each other from completing their assigned task. A flexible approach to resource conflict and deadlock problems in a dynamic environment is the use of a decentralized algorithm in conjunction with an implicit inter-robot communication. In this case, a robust decision making algorithm for each individual robot would be required to interpret and form decisions based on local environment and robot data.

The core elements of task planning and collision avoidance were described as static or dynamic coordination. Static coordination is most commonly implemented as a predetermined set of conventions such as “traffic rules” to manage the traffic flow of robots through an intersection [40]. While offering fast and efficient computation times, static coordination may perform poorly in real-time systems due to unpredictable behaviours. Dynamic coordination can be used to provide an adaptable solution to real-time environments by enabling communication between robots. This allows an exchange of information between robots such as position and trajectory, after which individual robots may make appropriate decisions to avoid an imminent collision. A hybrid static and dynamic coordination model can further be explored as a means to handle real-time dynamic collisions while employing static coordination to set basic rules for the environment.
However, coordination in a MRS often implies the risk of deadlocks [33]. While deadlocks are a rare occurance in a robust system, a deadlock resolution algorithm should nonetheless be implemented to handle such instances by resetting or altering the timing of the system.
Static and dynamic coordination algorithms may be used individually or strengthened with an implicit or explicit communication method. However it is important to note that the complexity of the system increases with the number of components being integrated. When used in a moderate and controlled manner, inter-robot communication may complement a dynamic coordination model to form a robust task coordination system.

2.3 Centralized and Decentralized Motion Planners

2.3.1 Centralized Planners

The decision-making aspect of motion planners can be centralized or decentralized according to the architecture of the cooperative system. Being the most common type of motion planners, centralized planners benefit from a shared access to data from the entire robot system [45][46][47]. A central control agent in a centralized system architecture uses environment information as inputs in order to make coordination decisions and calculate trajectories for all robots within the system. Optimal plans can be generated using a global analysis of the centralized system architecture.

One of the first centralized architectures used in a MRS is the GOFER project [46], in which a centralized task scheduling system possessing global access to the environment controls the operation of several robots at once. The system integrates a task planning system to derive action plans from specified tasks and a task allocation framework to allocate certain tasks to robots. The GOFER robot system is only adapted to basic smaller-scale vehicular tasks such as box pushing and exploration [46]. The model would benefit from the development of an architecture triggering communication acts to coordinate more complex cognitive activities in the robot system.

Sanchez and Latombe describe experiments with a Probabilistic Roadmap Planner (PRM) in a welding setting with multiple robot manipulators performing centralized planning [48]. The model considers all robots as if they form a single multi-arm robot, by encoding all combined DOFs in a single configuration space. A collisionless path is found using the sampling-based PRM algorithm by searching the space for a free path between initial and goal manipulator configurations [49]. However computation times can be high due to the complexity of searching

for a valid configuration for each individual joint of the robot arms in a combined space with dozens of dimensions. The joint limits and range of each manipulator must be tightly restricted to the workspace to minimize the calculation time required.

Kehoe *et al.* use the Raven surgical robot in a centralized system to conduct the autonomous execution of a two arm robotic surgical debridement subtask [17]. The planning of the two robotic arms are coordinated in a centralized manner using a single 12 DOF planner rather than two 6 DOF planners for each arm. Therefore, a sophisticated optimization-based motion planning method is needed to meet the requirements of arm coordination in the centralized system. Using a Model Predictive Control (MPC) approach, the trajectory of each arm is replanned frequently to deal with the large kinematic uncertainty. Locally-optimal, collision-free trajectories are calculated simultaneously using Trajopt, a low-level motion planning algorithm [50]. After a change in target pose, each arm's motion is replanned by processing the target pose from each individual arm such that the centralized planner plans for both arms simultaneously [17]. Other studies involving a centralized architecture approach include [46][51][52]. As the planner must wait until it has received requests from both arms to replan before computing one arm's trajectory, the planning mechanism may be delayed until the other arm completes its plan. While this method allows collisionless trajectories to be replanned frequently using updated pose estimates, the parallelism aspect of the system is not conserved as one arm must wait for the other to complete its trajectory before activating the planner. A decentralized system could be explored in which trajectories are planned individually using the same MPC approach and coordinated dynamically by inter-robot communication, such that no additional planning delays are introduced in the coordination algorithm.

2.3.2 Decentralized Planners

The lesser used decentralized planner is treated as a multi-agent or multi-robot system with each robot arm controlled by its individual planner, receiving limited data from the system's state[45][53]. These individual motion planners only compute their own trajectory. A decentralized control structure is demonstrated by Khatib *et al.* in which objects of the task are represented as individual tasks for each cooperative robot [54]. Local feedback control loops are developed at grasp point and the design of controllers are created using augmented object and virtual linkage models. Virtual Linkage concerns the manipulation of a single resource by control of the internal forces in a multi-grasp manipulation model. The model connects grasp points in a set of virtual links with independent internal forces specified for joint. This system architecture is better suited for mobile manipulator systems as a redundant system is controlled using a dynamic coordination strategy, allowing its full bandwidth to be used. Similarly, Ha *et al.* support the use of decentralized planners in a closed-loop multi arm motion planner in which a decentralized policy is used to train and control a single robot arm to reach its target end-effector pose [45]. Multi-agent reinforcement learning (MARL) is used to learn a decentralized motion planning policy to cooperate using a rewarding system once all arms have reached their individual goal configuration. In this system, the use of a decentralized planner allows the cooperative training of a closed-loop, multi-arm motion planner that can easily scale with the number of robot arms.

As opposed to centralized planners, a decentralized architecture is more adaptive to unknowns and dynamic environments. While a decentralized system adds reliability, flexibility, and robustness, designing a controller for more complex tasks requiring simultaneous arm motion

may be challenging. When paired with a coordination method such as implicit dynamic coordination, decentralized motion planners must efficiently communicate to avoid collisions while cooperating to reach their goal, while only having control of their individual path.

2.4 Velocity Obstacles

Velocity Obstacles (VO) or velocity control uses the notion of computing collision-avoiding velocities with respect to other agents, specifying velocities that will direct an agent towards its goal [55]. This approach uses both the current position and velocities of other robots to compute their trajectories in order to avoid collisions in a dynamic environment.

Fiorini and Shiller present a dynamic robot motion planning method adding a time dimension to the robot's configuration space, knowing the bounded velocities and path of the obstacles [56]. Robot velocities are chosen outside of the Velocity Obstacles, represented by the set of robot velocities potentially resulting in a collision with a given obstacle moving at a certain velocity at some future time. The trajectory is initially computed using a search tree of feasible static avoidance maneuvers [57]. Dynamic obstacles are handled by considering a first-order approximation of robot velocities that would potentially cause collisions with a dynamic obstacle within a future time range [56][58][59]. This method is demonstrated with point and disk type robots as well as for automated vehicles in a highway scenario. Berg *et al.* introduce the acceleration-velocity obstacle (AVO), letting a robot avoid collisions with moving obstacles with a constrained acceleration. The concept of reciprocal collision avoidance is used, letting each robot take half of the responsibility of avoiding robot-to-robot collisions [55][58][60]. This approach is designed for holonomic robots and non-holonomic robots such as cars.

The combination of a motion planner to avoid static obstacles and Velocity Obstacles to avoid dynamic obstacles form a good pair in a dynamic MRS system. To adapt to a 3D environment for bilateral manipulation, an additional dimension would need to be added into the velocity collision avoidance problem, adding additional degrees of freedom and uncertainties. The model should be adaptable to a task oriented environment and support various robot actions or edge cases. Controlling the velocity of each dynamic agent rather than changing their paths to avoid dynamic obstacles in an MRS would ideally create minimal disturbances in each component's initial trajectory.

Chapter 3

Proposed Execution Model

3.1 Overview

Our method builds upon the concept of Velocity Obstacles as a means of solving dynamic collisions while minimizing disturbances to the path generated by a motion planner. As decentralized architectures are more adaptive to unknowns in a dynamic environment, we implement each arm as a state machine to control their own path individually. A combination of static and dynamic coordination is implemented in our framework to generate collision-free paths around obstacles during the planning phase, and handling collisions dynamically as they are detected.

The goal of this work is to present a flexible and robust path and motion planning framework for bilateral manipulation in a task specific environment. We demonstrate the benefits of using velocity control as a dynamic collision avoidance mechanism adapted for the execution of integrated tasks in a constrained pick-and-place simulation. The assignment of a multilateral task to a velocity obstacle method poses multiple new challenges which a velocity obstacle in continuous motion would rarely encounter. Such cases include the approach of a common goal location, as well as specific delays to execute certain actions in the context of the task. To provide an adaptable solution to a variety of robotic tasks, our framework must handle the various cases that may occur, including static obstacle avoidance, arm-to-arm collision detection, task specific delays, common goal trajectory planning, and deadlocks. A high level overview of the system components is shown in Figure 3.1.1.

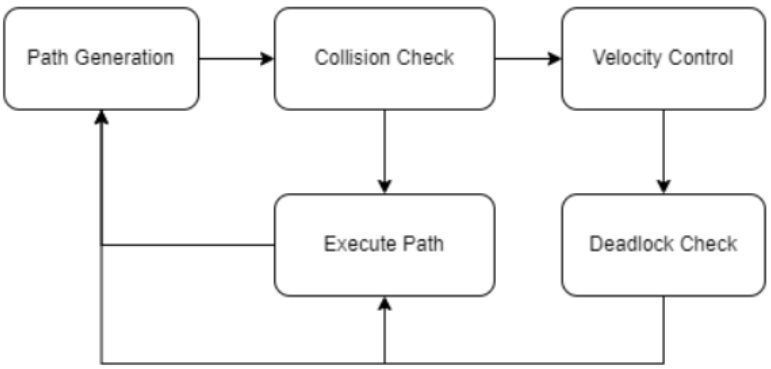


Figure 3.1.1: System components overview and interactions

3.2 System Architecture

Finite State Machines (FSMs) are used as a base structure to coordinate a single arm movement for a given task. As a decentralized architecture is used, each arm is assigned to its own FSM, planning and controlling their own individual path and task execution. We simulate six states to run a pick-and-place subtask. Arms first start in a planning state, in which optimal paths are generated using a motion planner, and collision checks are conducted. Once the path is executed, the arm approaches the nearest object, grabs the object, introducing a short delay, then recalculates its path to the bowl to drop the object and introduces another delay. The combined PSM state machine architecture is shown in Figure 3.2.1. A hybrid static and dynamic collision

avoidance mechanism is integrated in this system as a means to coordinate and control each arm's usage of environment space and resources. We handle real-time dynamic collisions while employing static coordination to set basic rules for the environment.

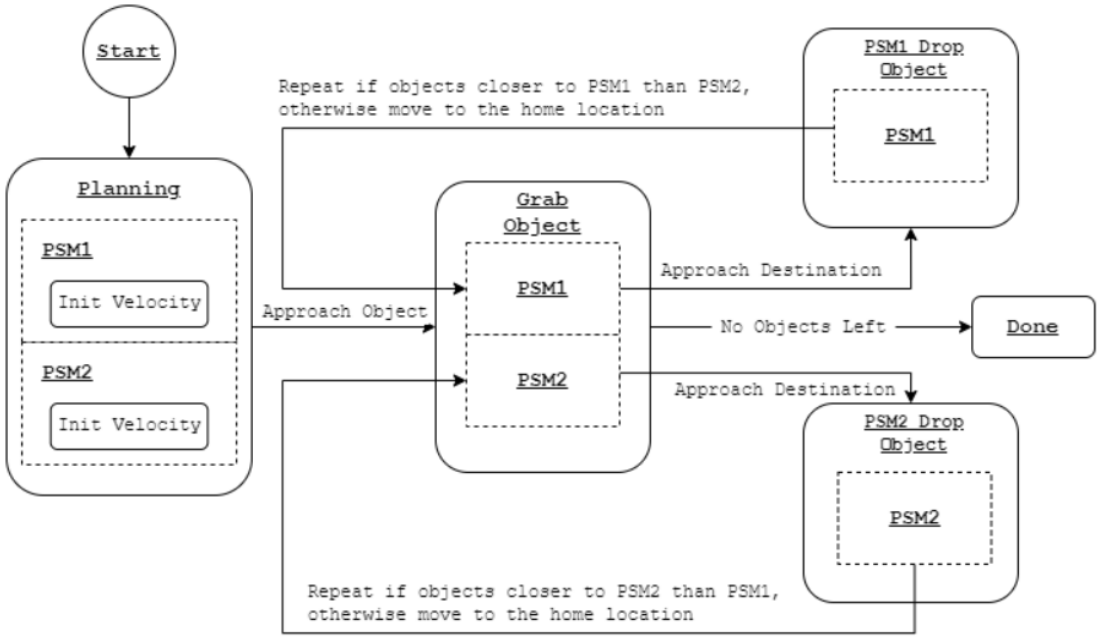


Figure 3.2.1: PSM1 and PSM2 basic pick-and-place state machines and task states

In order to use the arm state machines in Figure 3.2.1 while minimizing the probability of collisions, we integrate our motion planning and collision avoidance framework in the pick-and-place task. Each component of the algorithm is developed in isolation then gradually integrated after evaluating each component's performance.

Our proposed algorithm functions in real-time. The sampling based planner RRT* is used to calculate optimal and collisionless paths around static obstacles [61][62]. In the planning state, the two paths are inspected interactively at every point in time to check for possible collisions. Following collision inspection, there are three possible outcomes:

1. No collision detected
2. A possible collision detected along the path
3. A possible collision detected at a common goal location

In the case that no collisions are detected, arms proceed to their respective goal without any change in velocity or trajectory. Once the first arm arrives at its destination, we compute a new path and once again check for possible collisions with the remaining path left of the second arm.

A possible collision detected along the path in the second case implies that there exists a list of points in both paths which are under an arbitrary threshold at the same timestamp. The algorithm adjusts the velocity non-linearly to avoid the collision with minimal disturbance to the

initial trajectory and velocity. Once the fast arm reaches the velocity reset point, set as the first collision point in the path, we reset the velocity and check for collisions again.

The third case describes the situation in which both arms are approaching the same goal location, in our case the bowl. While this should be considered a possible collision, it is not necessarily detected as such, as it is not guaranteed that the two arms would arrive at the same time. However, the first arm arriving at the bowl may take a certain amount of time to complete a designed action before leaving the resource, causing the second to possibly collide at the goal location.. One again, we rely on a velocity control system to ensure that arms will never meet at the bowl location, relying on an emergency stop as a final resort to prevent an imminent collision. A flow chart describing the algorithm is shown in Figure 3.2.2.

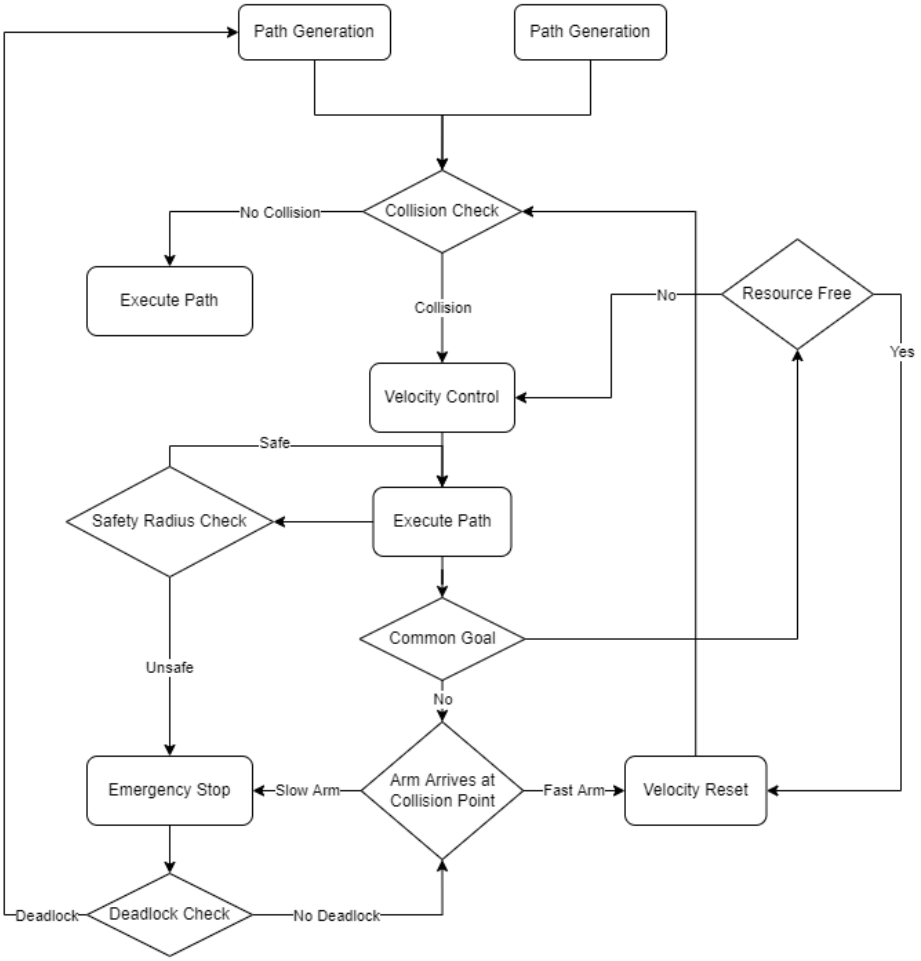


Figure 3.2.2: Algorithm feature flow chart

Common complications that may occur in decentralized concurrent systems are deadlocks. Deadlocks in our system may occur due to arms attempting to approach the same location and failing to coordinate its use, resulting in a stalemate caused by the algorithm's restrictions. Emergency stops may be used to assist in handling any unforeseen failures in the algorithm such as deadlocks. We use the number of emergency stops made in a row as a detection mechanism for deadlocks, before recalculating a different path to break the deadlock.

3.3 Motion Planning

Rapidly Exploring Random Trees* (RRT*) is a sampling based motion planner optimizing its predecessor, RRT, to achieve a shortest path [57][63]. The premise of any RRT based algorithm consists in randomly generating and connecting the nearest node. For every vertex generated, we check that the vertex does not encounter an obstacle. The algorithm comes to an end once a node is generated within the goal region or a solution is not found within a time limit.

We operate in a configuration space with a given state space denoted by $Q \subset R^n$ where n is the dimension of the search space $n \in N$. Obstacles are represented in the search space represented by $Q_{obs} \subset Q$ and unoccupied space is represented by $Q_{free} = Q/Q_{obs}$. The goal configuration is denoted by $q_{goal} \subset Q_{free}$ with the initial configuration as $q_{init} \subset Q_{free}$. q_{init} and q_{goal} are inputs to the planner by the algorithm, as well as the placement of obstacles Q_{obs}. RRT must find a collision free path from q_{init} to q_{goal} within Q_{free} within a set time limit $t \in R$ while minimizing distance cost.

The RRT planner's main objective is to find a valid path ρ_f: $[0, n]$ if one exists in $Q_{free} \subset Q$ such that $\rho_f(0) = q_{init} \subset Q_{free}$ and $\rho_f(n) \in q_{goal}$ within $t \in R$ and return false if a path cannot be computed.

RRT implements six main functions to build and compute the search tree:

- *Sampling:* The **Sample** function randomly samples a state $q_{rand} \in Q_{free}$ in a free space in the environment.
- *Nearest Neighbour:* **Nearest** finds the nearest neighbouring node from $T = (V, E)$ to q_{rand} using a cost function.
- *Steering:* The function **Steer** solves for the control input $u: [0, T]$ driving the system from $q(0) = q_{rand}$ to $q(T) = q_{nearest}$ along the path $q: [0, T] \rightarrow Q$.
- *Collision Check:* **ObstacleFree** checks whether a path $q: [0, T] \rightarrow Q$ lies in obstacle-free space $q \in Q_{free}$ for all $t \in [0, T]$.
- *Insert Node:* **InsertNode** adds q_{new} to V in the search tree *T = (V, E)*, connecting to its parent neighbouring node q_{min}.

The RRT planner tends to create irregular paths in nature as nodes can only be attached to their nearest neighbour, often resulting in a sub-optimal trajectory. These irregularities are addressed by RRT*

The RRT* planner's main objective is to build upon RRT to finding an optimal path ρ_{f*}: $[0, n]$ in $Q_{free} \subset Q$ such that the path cost $Cost(\rho_{f*})$ is a minimum.

$$Cost(\rho_{f*}) = \{min_{\rho s} Cost(\rho_f): \rho_f \in f\} \tag{3.1}$$

All of RRT's previously defined properties are inherited. RRT* further introduces two more functions into the search algorithm:

- *Rewiring:* The **Rewire** function checks that the cost of the set of nodes q_{near} is less when passing through q_{new} than that of its last cost. If the cost is found to be less, the parent node is changed to q_{new}.
- *Choose Parent:* **ChooseParent** finds the optimal parent node q_{new} among the nearby nodes.

The RRT* Algorithm is described in Algorithm 1.

Algorithm 1 $T = (V, E) \leftarrow \text{RRT}^*(q_{init})$

$T \leftarrow InitializeTree()$
$T \leftarrow InsertNode(\Theta, q_{init}, T)$
for $i = 0$ to $i = N$ **do**
  $q_{init} \leftarrow Sample(i)$
  $q_{nearest} \leftarrow Nearest(T, q_{rand})$
  $(q_{new}, U_{new}) \leftarrow Steer(q_{nearest}, q_{rand})$
  if $Obstaclefree(q_{new})$ **then**
    $q_{near} \leftarrow Near(T, q_{new}, |V|)$
    $q_{min} \leftarrow Chooseparent(q_{near}, q_{nearest}, q_{new})$
    $T \leftarrow InsertNode(q_{min}, q_{new}, T)$
    $T \leftarrow Rewire(T, q_{near}, q_{min}, q_{new})$
  end if
end for
return T

Algorithm 1: RRT* Algorithm description

The RRT and the optimized RRT* algorithms can be visualized in Figure 3.3.1. The blue lines display the explored states with the grey prisms representing environment obstacles. The green path shows the raw RRT path leading from start to goal state connected to neighbouring nodes and the orange line shows the the valid RRT* computed path from start state to goal state.

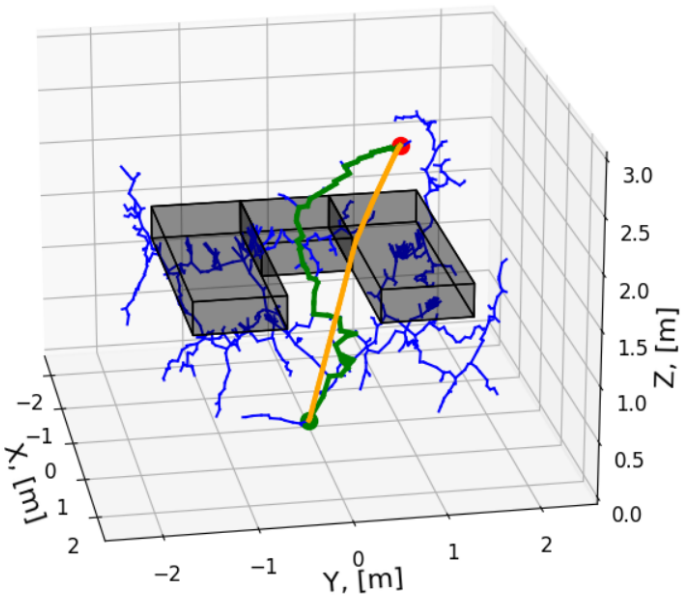


Figure 3.3.1: RRT (green) and RRT* (orange) generated paths.

3.4 Collision Detection

The arm paths at default velocity are represented as a set of equidistant points forming a polyline. As the probability of two polylines intersecting at exactly one point in a 3D space is very low, we define a possible collision as a list of points in the two paths, at which the euclidean distance is under a certain threshold. To detect potential collisions on two paths, we use a method of collision prediction using the known initial velocity v_0 and distance traveled for every time steps τ.

To compare paths of different lengths, we must first take the shortest path set length r_{min} between Path 1 as P_1 and Path 2 as P_2:

$$r_{min} = min\{r_{P1}, r_{p2}\} \tag{3.2}$$

Then starting at time $t = t_0$:

$$C_n = \{P_n(t) | \, norm(P_1(t), P_2(t)) \leq \delta_c\} \tag{3.3}$$

where C_n is the set of all potential collision points in Path n in which the euclidean distance of Arm 1 and Arm 2 at time t is less than a chosen collision threshold δ_c.

We proceed to check for collisions in the paths at every time step $t = t + \tau$ until:

$$P_n(t) = P_n(v_0/(S * r_{min})) \tag{3.4}$$

where S is the current path step size.

3.5 Velocity Control

As the raw polylines initially generated by RRT* are not constructed of equidistant points, we must first process the paths such that at every time step τ, arms move forward by a constant initial step size S_0, letting the arms move at constant initial velocity $V(t) = v_0$. For the current arm velocity at any time:

$$V(t) = S^*\tau \tag{3.5}$$

To interpolate the raw path with equidistant points, the n-th discrete difference is first calculated along every axis. For instance, the first difference would be given by $da = a[i+1] - a[i]$ along a given axis a.

For every point i in a path $P[i] = (x[i],\ y[i],\ z[i])$ of length l:

$$\begin{aligned} dx[i] &= x[i+1] - x[i] \\ dy[i] &= y[i+1] - y[i] \\ dz[i] &= z[i+1] - z[i] \end{aligned} \tag{3.6}$$

We then calculate the cumulative sum of the euclidean distance:

$$\lambda = \sqrt{dx^2 + dy^2 + dz^2} \tag{3.7}$$

$$u = \sum_{i=0}^{l} \lambda_i \tag{3.8}$$

Using the maximum value of u, an array k of evenly spaced values at a step size of S_0 is generated within the interval $[0, max\{u\}]$. Finally, we linearly interpolate the raw path for monotonically increasing sample points and return a one-dimensional piecewise linear interpolant with given discrete data points (k, u) evaluated at x, y, z.

Equidistant points are interpolated on the raw RRT* path such that the step size S between points in our path remains a constant value as shown in Figure 3.5.1, hence producing a constant velocity.

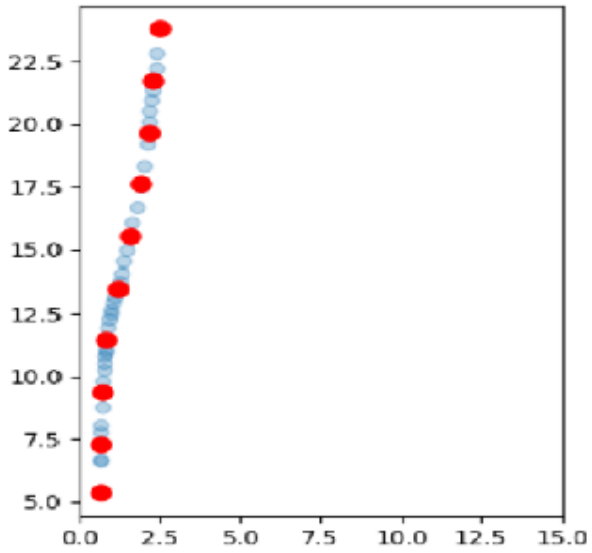


Figure 3.5.1: Processed RRT* path (red) at constant velocity and raw RRT* path (blue)

Before adjusting arm velocity to avoid a detected collision, the arm to adjust must first be selected to minimize the probability of collision, by maximizing the distance between arms as quickly as possible. Therefore, the algorithm chooses to slow down the arm whose goal is nearest to the other arm's current position. This lets the arm with the goal furthest from the other arm move away, gradually gaining distance and reducing chances of any further interactions between the two arms.

For instance, in a case such as in Figure 3.5.2, the blue arm's velocity is decreased in order to allow time for the green arm to clear the collision area, moving to a point away from the blue arm. If the green arm's velocity were to be decreased instead, the blue arm would continue to approach the green arm, potentially resulting in an emergency stop or a deadlock.

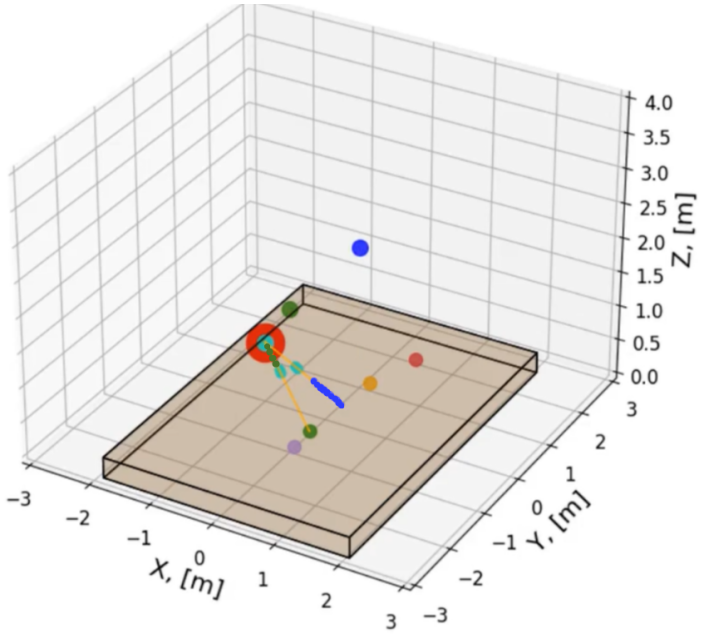


Figure 3.5.2: Arm 1 (blue) slowing down and moving towards bowl while Arm 2 (green) clears the area, moving away from Arm 1.

Having chosen an arm to decelerate, a Velocity Function is now integrated and applied to the arm's path. The purpose of the Velocity Function is to transform the path by interpolating points as needed with constant or varying step sizes in order to control the arm velocity along the path. We use a nonlinear Velocity Function to transform the path non-linearly. We assume that an ideal nonlinear function would allow the arm to continue moving near maximum velocity, only slowing down as necessary once closer to the collision point. A nonlinear function fitting this description is the natural logarithm as shown in Figure 3.5.3.

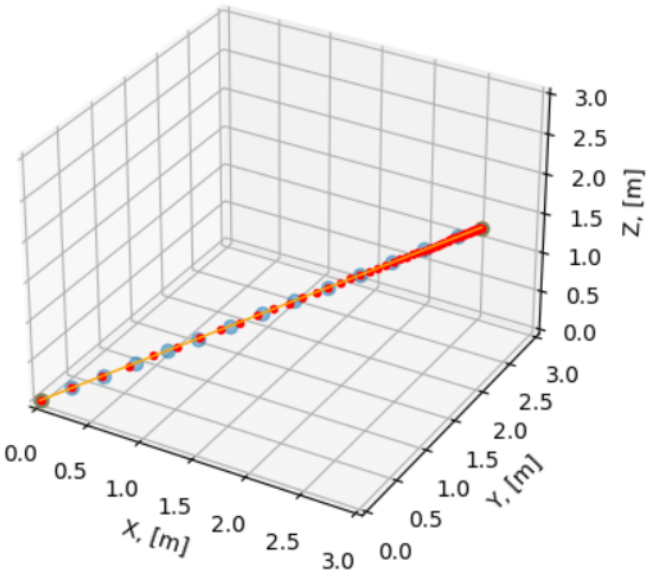


Figure 3.5.3: Arm path with equidistant points at constant velocity (light blue) and adjusted path transformed using a natural log function (red), where the first point represents the arm's start position, and the last point represents a collision point.

The arm gradually decreases in velocity, stopping at its collision point in the event that the fast arm does not pass its collision point in time. Letting the arm decelerate non-linearly allows minimum disturbance to the arm state, maintaining speed for as long as possible while avoiding a possible collision. An instance of the resulting velocity time graph showing a gradual decrease as a result of the path transformation is shown in Figure 3.5.4.

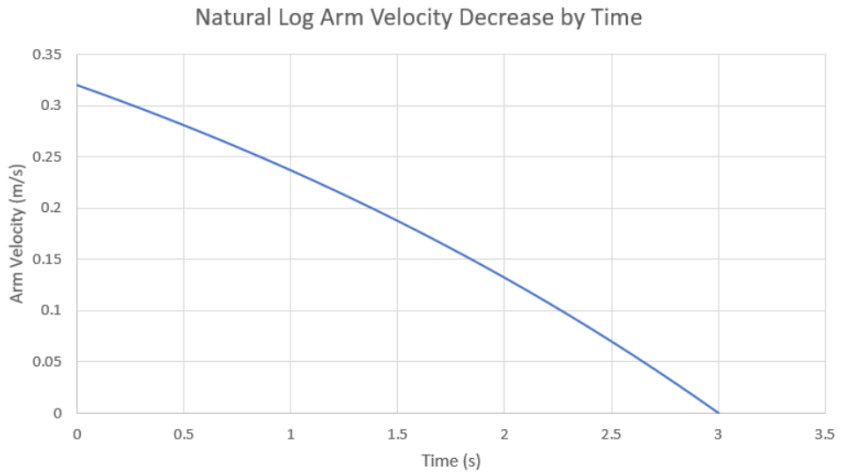


Figure 3.5.4: Velocity decay using a natural log function to transform arm path

The path is fit to a natural logarithm function with the step size *S* between points in our path changing non-linearly as such:

$$S = K(1 - e^{-t/T}) \tag{3.9}$$

where *K* is the gain of the log equation and *T* is the time constant. We consider the first order system steady state standard that after $4T$, we reach within ~2% of the asymptotic value [64], where 2% is the multiplier *m*. This indicates that after $4T$, steady state is considered to be reached at ~98% of the gain *K*.

To shape the Velocity Function as a gradual velocity decrease, we solve for the final distance value μ that we want to reach at $4T$ steady state by setting $K = 1 + m$ (in the case of the first order steady state standard, ~102%) of μ.

Knowing the initial velocity v_0, we can calculate the initial path step size S_0 using Equation 3.5 and use $t = 1$ and $K = 1.102\mu$ to form the following equation and solve for *T*:

$$T = 1/ln(1 - S_0/K) \tag{3.10}$$

K is a parameter that may be changed as needed if we would like the arm to reach the final point in the path sooner than $4T$ or later than $4T$ [1]. To control the gain *K*, we can change the value of the multiplier *m* and calculate *K* as such:

$$K = \mu(1 + m) \tag{3.11}$$

A higher multiplier results in a steeper velocity decrease towards the end, while a lower multiplier results in a more gradual velocity decrease.

3.6 Common Goal Location

Using a velocity obstacle method in a task driven concurrent system reveals many challenges. While two arms requiring the use of a single mutually exclusive resource means a possible collision, it is not necessarily detected as such, as it is not guaranteed that the two arms would arrive at the same time. However, due to the nature of integrated tasks, it is possible that the arm will idle for a certain amount of time at the goal location to complete an intended action, such as opening the manipulator jaws to drop an object into the bowl. Therefore, we set a collision point at the goal location as shown in Figure 3.6.1, and adjust the velocity to slow down the furthest arm, giving priority to the nearest arm to use the resource. As the collision is not considered avoided as long as both arms require the use of the same resource, the velocity is only reset to default once the arm frees the resource and begins executing its next path.

[1] The observation of controlling the gain to reach the last path point sooner or later than $4T$, allowing the control of the steepness of the velocity curve, was provided by W. Van Dam (2021)

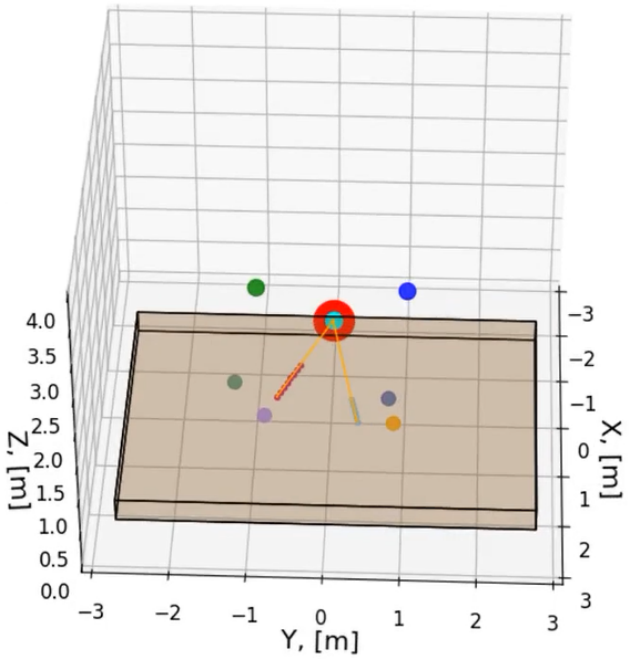


Figure 3.6.1: A collision point (cyan) is set on the goal location. Arm 1 (red) gets priority access to the bowl while Arm 2 (light blue) decreases in velocity until Arm 1 no longer needs the resource

3.7 Deadlock Resolution

Deadlocks are a possible occurrence in concurrent systems using mutually exclusive resources. Although deadlocks are a relatively rare occurrence, a deadlock resolution mechanism should still be included in the system to handle such a case. In our environment, any point in space is seen as mutually exclusive, as we must avoid collisions between the two arms. If arms are moving directly towards each other, collisions cannot be avoided using solely velocity control, as no matter the velocity, arms will continue to approach each other until reaching a stop at the first collision point. The fast arm will not be able to compute a collisionless path and remain stuck in a planning state, while the slow arm will simply stop at the first collision point until the fast arm clears the first collision point.

In order to break a deadlock, its occurrence must first be detected. As a deadlock is defined by both arms reaching a stalemate and being unable to move, we check at every sampling period whether the slow arm is in a state of emergency stop. If the emergency stop persists for a predefined continuous period of time, a deadlock is detected.

One of the most common methods of deadlock resolution is “cancelling” and restarting a process. We implement this method by resetting the fastest arm’s path and recalculating a path around the slow arm. To force the motion planner to calculate a path around the stationary slow arm, an imaginary boundary box is generated to enclose the slow arm and represent it as a static object, as shown in Figure 3.7.1.

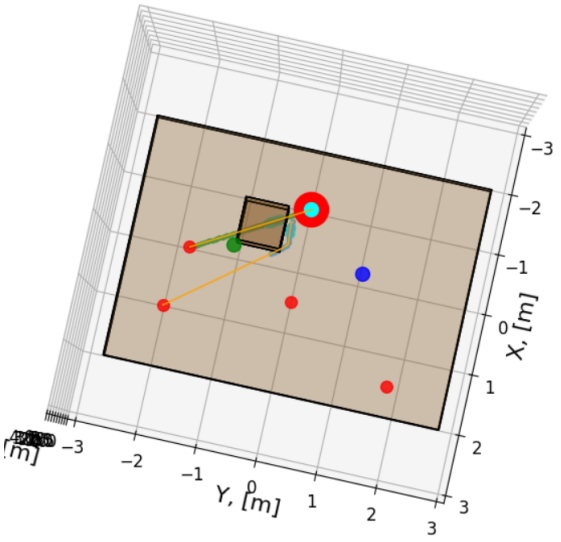


Figure 3.7.1: A deadlock is detected and a boundary box is generated around Arm 2 (green). Arm 1 (blue) resets its current path and computes a new path around the boundary box.

In a robot environment such as the da Vinci Surgical Robot, this box would need to enclose the entirety of the arm surface to minimize the possibility of any further deadlocks occurring on any part of the arm in this path.

As the RRT* motion planner attempts to find an optimal shortest path by nature, sections of path computed may lie directly along the surface or edges of the box. If the dimensions of the box were to be less than the collision detection range, collisions are likely to be detected early along the new path. As the arms are already near together, this may result in the slow arm reaching its collision point almost immediately, once again going into a state of emergency stop. The deadlock breaking process would repeat until arms are at a sufficient distance for a new path to be calculated without a collision detected.

In order to avoid these possible complications, the boundary box is set with dimensions equal or greater to that of the collision detection range r. Assuming the box is generated around an arm located at its centroid, we define the box dimensions to be $x \geq r,\ y \geq r,\ z \geq r$. An excessively large boundary box would also be inefficient, resulting in a lengthier path than necessary and increasing the tree search load on the RRT* motion planner. Therefore, we set the box dimensions at a value arbitrarily near that of the collision detection range, at $x \in [r,\ r + 0.05],\ y \in [r,\ r + 0.05],\ z \in [r,\ r + 0.05]$.

Chapter 4

Evaluation

4.1 Overview

Our motion planning and collision avoidance framework is evaluated using a pick-and-place task simulated in a Python environment, with arm trajectories and objects plotted in real-time on a 3D figure. Using this method, velocity changes and planned trajectories for each arm can be clearly observed, allowing more flexibility in testing and visualizing different parameters.

To demonstrate the robustness and effectiveness of the collision avoidance framework, three subsets of evaluations are conducted. The first type of evaluation consists of conducting experiments supporting the selection of certain features or parameters to reinforce the algorithm. Different functions are tested to decelerate an arm as well as the optimal position where the arm should return to its original velocity. The second type of evaluation is done by conducting stress tests by forcing the simulation into edge cases such as multiple intersection points, common goal locations, and deadlock resolution, and observing its behavior. This series of experiments demonstrates the framework's responsiveness and collision handling in a dynamic environment. The final evaluation compares the performance of our proposed execution model with parameters chosen according to the preceding evaluations' results, to a state of the art execution model in a pick-and-place simulation. We use Abdelaal *et al.*'s hierarchical state machine (HCSM) and Independent parallel FSM execution models as benchmarks [15].

4.1.1 Performance Metrics

We quantify the performance in the experiments described earlier using a combination of the following metrics:

- **Completion Time (s):** measures the total execution time in seconds from the starting position of the arms to the designed goal position for each trial.
- **Number of Collisions Avoided (Velocity Adjustments):** measures the total number of times that arm velocities are adjusted in order to avoid a possible collision, including common goal location adjustments, and detected collision adjustments. This metric is a simple benchmark to demonstrate the benefits of our algorithm and observe the number of collisions that would occur otherwise. Velocity adjustments resulting in deadlocks are not counted.
- **Number of Emergency Stops:** measures the number of times that arms must reach a stationary state after slowing down to avoid a collision, due to entering the other arm's safety radius. As we attempt to keep the parallelism aspect of the multilateral task, the number of stops should be minimized. This is also a measure of the suitability of the velocity function to choose a velocity at which arms will move continuously. Each deadlock encountered only counts as a single emergency stop.
- **Number of Path Recalculations (Resolved Deadlocks):** In the case when arms are moving towards one another and the collision cannot be avoided by controlling the

velocity of the arms, we attempt to recalculate the fast arm's path once the slow arm reaches a state of emergency stop and a deadlock is detected. If a valid path is found, the deadlock is resolved. This metric measures the number of successful path recalculations leading to the resolution of a deadlock, representing the complexity of the task set up.

- **Number of Unresolved Deadlocks:** measures the number of unsuccessful path recalculations leading to both arms reaching a stalemate, unable to move without colliding.
- **Number of Collisions:** measures the number of collisions between the two arms or between an arm and the environment.
- **Overall Task Error:** combines the metrics of the number of collisions and unresolved deadlocks, both of which are represented as critical failures in the system that must be avoided.

4.2 Parameter Tuning

Algorithm parameters refer to the constants used in our collision avoidance framework. The performance of our algorithm is dependent on its selected parameters, as evaluated according to the performance metrics described in Section 4.1.1. Experiments were conducted to compare a variety of parameters in order to choose the most suitable options/values, such as the velocity function used to adjust arm velocity, and the velocity reset point used to decide whether to reset velocity at the first or last collision point. Environment parameters represent the dimensions and positions of objects, and are chosen such that a fair comparison to a state-of-the-art multilateral task coordination model can be made against our proposed framework.
Table 4.2.1 and Table 4.2.2 provide a summary of the algorithm and environment parameters.

Parameter	Definition
Collision Detection Range	Defines the euclidean distance between two path points at which a collision can be detected in the planning state of the arm state machine.
Safety Radius	Defines the distance between the current positions of the two arms at which the slow arm must do an emergency stop until its velocity is reset.
Pause Time	Defines the idle time required for an arm to drop or pick up an object. It is selected to match the same action in the HCSM pick-and-place coordination model presented by Abdelaal *et al*. [15], against which a performance comparison is made.
Boundary Box Dimensions	Defines the invisible static obstacle enclosing the slow arm during deadlock resolution, allowing the fast arm to attempt planning a valid path around it. Box dimensions are measured using centroid distance to the outer face, and is chosen according to the collision detection range.
Default Velocity	Defines the initial velocity at which both arms move and are reset to after collision is avoided. This velocity is the same for

	both arms and is chosen to be the maximum velocity allowed by the robot.
Velocity Function	Defines the type of velocity adjustment used to decrease arm velocity, including Natural Log, Quadratic, or Step.
Velocity Reset Point	Defines the collision point in an array representing all points in space at which a collision is detected. Arm velocity is allowed to be reset either at the first or last point of the set.

Table 4.2.1: Algorithm parameters definition

Parameter	Definition
Object Positions	Objects may be generated within a certain distance on the table around the bowl location. Range is selected to match the same positional range in the HCSM pick-and place coordination model.
Bowl Position	Bowl is placed at a fixed position on the table, selected to match the same location in the HCSM pick-and place coordination model.
Home Position	Defines the start position of each arm, selected to match the same location in the HCSM pick-and place coordination model. Once an arm has picked all of its assigned objects, it should go back to its start location. Both arms having arrived at home position constitutes the end of the task execution.

Table 4.2.2: Environment parameters definition

4.2.1 Experimental Setup

The Overall Performance Comparison with alternative task coordination models in Section 4.6 will be conducted with the same algorithm and environment parameters. Therefore environment parameters have been chosen to match up to scale with the pick-and-place task experiment with the da Vinci Surgical Robot arm home positions, objects positions, and table size. Our pick-and-place Python simulation environment is shown in Figure 4.2.1, while the pick-and-place task with the da Vinci Surgical Robot simulated in V-REP [11] is shown in Figure 4.2.2. Optimal velocity function and reset point will be selected in this experiment to be used in Section 4.6. The Parameter Tuning experiments in Sections 4.2.2 and 4.2.3 were conducted using the following parameters in Table 4.2.3 over 20 trials.

Detection Range (m)	Safety Radius (m)	Default Velocity (m/s)	Boundary Box Size (m)
0.05	0.03	0.32	[0.06, 0.06, 0.06]

Table 4.2.3: Deadlock resolution experiment algorithm parameters

The experiment environment was set up according to the pick-and-place environment in the HCSM coordination model as follows.

- The Home positions for Arm 1 (Blue) and Arm 2 (Green) were fixed at [0.0, 1.0, 2.0] and [0.0, -1.0, 2.0] respectively.
- The Red bowl was positioned at [-1.0, 0.0, 0.8]
- Two Red objects were generated at random at x ∈ [-0.5, 1.0], y ∈ [-1.5, 1.5], and z = 0.33
- The table top dimensions are defined as 0.38x5.56x0.3 m

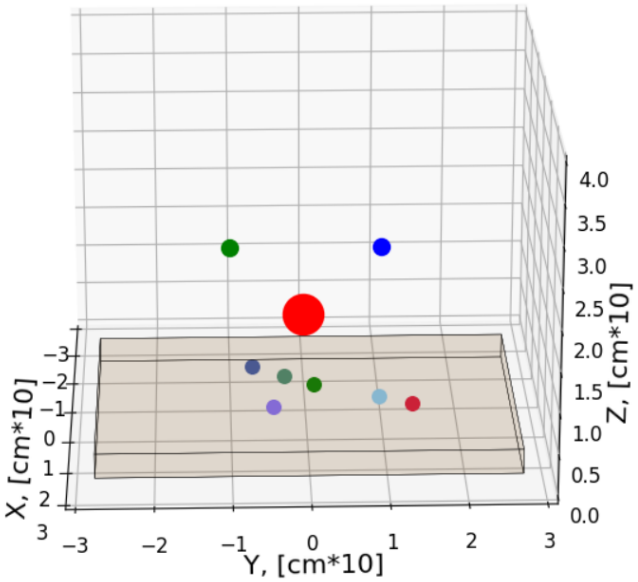


Figure 4.2.1: Main pick-and-place Python environment setup. Bowl is shown as a big red circle, Arm Home positions in dark green and blue above the bowl, and objects of different colors generated at random on the table.

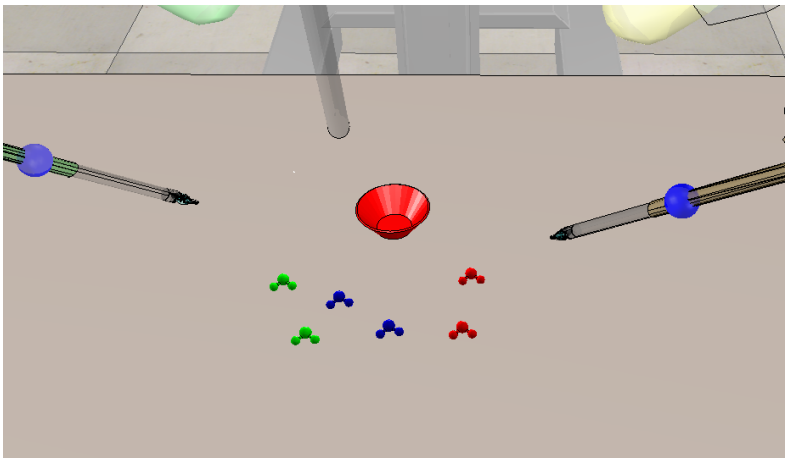


Figure 4.2.2: V-REP pick-and-place task simulation with the da Vinci Surgical Robot. Bowl is shown in Red and objects are represented as small water molecules generated at random on the table.

4.2.2 Velocity Function

As described in Table 4.2.1, the velocity function represents the type of transformation applied to decrease an arm's velocity in order to avoid a detected collision. This experiment compares and tests three velocity functions to find the best one for our collision avoidance framework.

The first velocity function is a step function reducing the path velocity by a constant value. The default velocity matching the da Vinci Surgical Robot's pick-and-place simulation was defined as V = 0.32m/s, and can be implemented in our simulation by calculating the path point step size using a known time step of τ = 0.0005s. The step size S of our path points is defined as:

$$S = V/\tau \tag{4.1}$$

This signifies that at every time step $\boldsymbol{\tau}$, the arm moves to the next point in its path, distanced at S from the previous. The path step size was calculated as S = 0.00016m with the paths retrieved from the RRT* algorithm interpolated accordingly. An arbitrarily shorter path step size of 0.00013m was chosen to adjust arm velocity in case of collision, resulting in a reduced velocity of 0.26 m/s. An instance of the resulting velocity time graph showing a velocity decrease by a constant, then a velocity increase to its initial value as a result of the path transformation is shown in Figure 4.2.3.

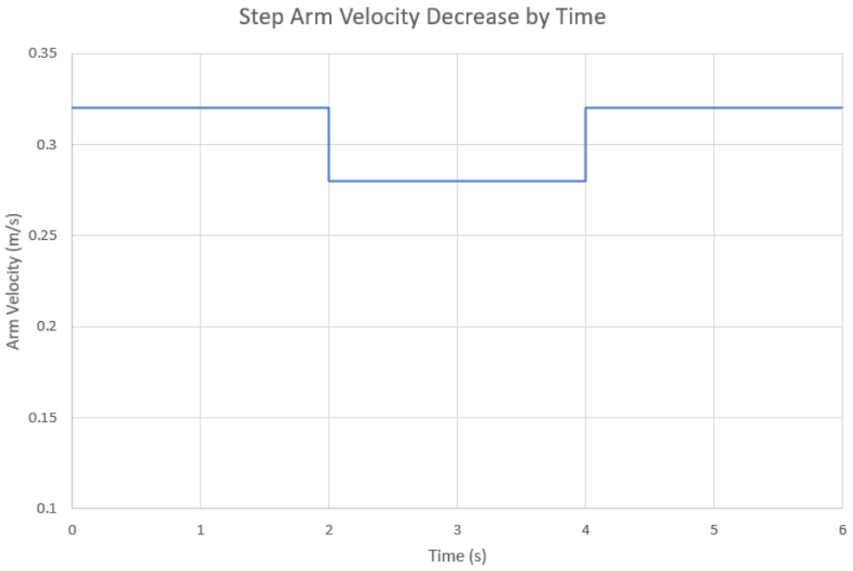


Figure 4.2.3: Velocity Step function. Arm velocity is decreased by a constant value for 2 seconds before returning to its initial velocity.

The second velocity function transforms and interpolates the path to fit a Quadratic function. The use of this function allows the arm to continue moving seemingly at default velocity, only slowing down noticeably once much closer to the collision point. The arm then shows a steep decrease in velocity, stopping just before its collision point in the case that the fast arm does not pass its collision point in time. Using a non-linear method of velocity adjustment allows minimum disturbance to the arm state despite a collision being detected, only reducing the velocity as needed to avoid the collision. The step size S between points in our path changes non-linearly as such:

$$S = ax^2 + bx + c \tag{4.2}$$

where x represents an iterated point of the final transformed path of length n, and a, b, and c are constants calculated as follows:

$$b = 2\mu - \sqrt{4\mu^2 - 4\mu S_0 + z^2} \tag{4.3}$$

$$a = S_0 - b \tag{4.4}$$

The parameter z represents the final path step size from *P[n-1]* to *P[n]* where n is the set length of the final transformed path. As we expect the slow arm to come to a stop upon reaching the collision point, $z = 0$ in our case. The parameter μ represents the upper bound of the path which is the total distance traveled along the path at the last path point, taken by calculating the cumulative sum of the distance of each point. The default step size is denoted by S_0.

The equations constants a and b are derived from the generic quadratic equation:

$$a(n-1)^2 + b(n-1) + c = \mu - z \tag{4.5}$$

where the set length of the final transformed path is calculated as follows:

$$n = (z - b)/[2(S_0 - b)] \tag{4.6}$$

Equation 4.2 is applied for every point x in range of the final transformed path of length n. An instance of the resulting velocity time graph showing a Quadratic velocity decrease to a stop as a result of the path transformation is shown in Figure 4.2.4.

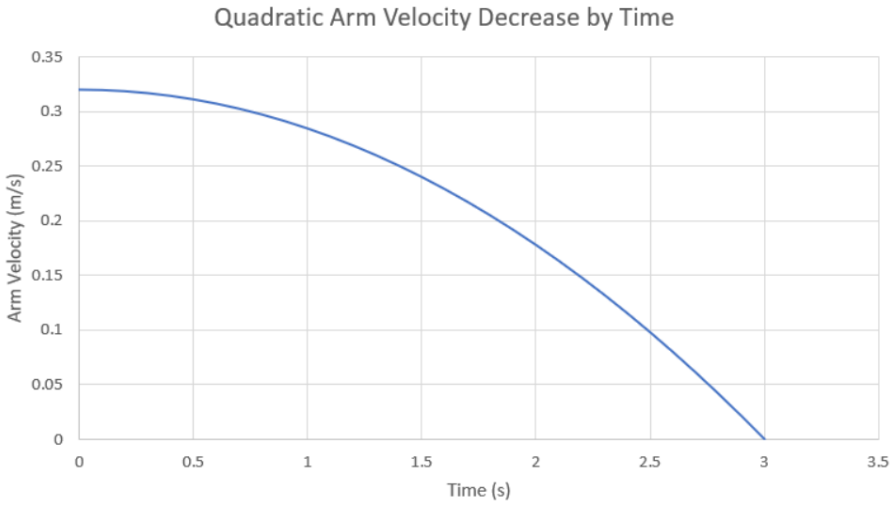


Figure 4.2.4: Velocity Quadratic function. Arm velocity is decreased quadratically for 3 seconds before coming to a full stop once the first collision point is reached.

The last velocity function in our evaluation is another non-linear interpolation, fitting the Natural Logarithm function defined in Section 3.5. Similarly to the Quadratic function, the interpolation of the arm path using a Natural Logarithm creates minimal initial disturbance to the velocity of the arm, only noticeably decreasing in speed once closer to the collision point.

However, the decrease in velocity is more gradual in nature and occurs visibly earlier than in the Quadratic interpolation.

The three velocity control functions are tested in conjunction with the velocity reset point parameter in Section 4.2.3 to optimize the final algorithm.

4.2.3 Velocity Reset Point

In the planning state of the arm state machine, a path is computed using the RRT* motion planner, followed by collision checking along every sampled point in this path. In this algorithm, an intersection is defined as an array of points at which the first arm's path euclidean distance from the second arm's path is within the collision detection range. As defined in Table 4.2.1, the velocity reset point represents the collision point in the array at which arm velocity is reset to its initial velocity once a collision is avoided. This parameter can be set as the first or last point of this array of collision points. Figure 4.2.5 displays an example of an array of collision points as a short trail of cyan points along each arm's path.

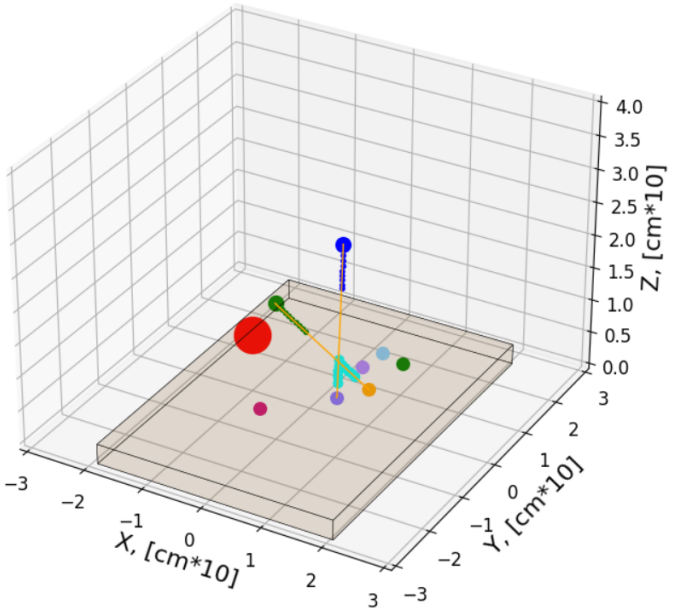


Figure 4.2.5 An array of collision points shown as a cyan trail along each arm's path (orange) in the main pick-and-place Python simulation

We test and compare the performance of our framework when using the first collision point of the array as our velocity reset point, against using the last collision point of the array. Furthermore, both options will be used as parameters to test each velocity function type in order to determine the optimal combination of parameters. These parameters will be used in the Overall Performance Comparison of our framework and a state-of-the-art model in Section 4.4.

4.2.4 Results

The Average Execution Time results observed for parameter tuning of various combinations of velocity functions and velocity reset points after 20 trials are shown in Table 4.2.4.

	Step Function	Quadratic Function	Natural Log Function
First Collision Point	13.58 ± 0.41 s	11.20 ± 0.62 s	10.52 ± 0.59 s
Last Collision Point	13.31 ± 0.37 s	12.98 ± 0.51 s	12.64 ± 0.48 s

Table 4.2.4: Average Execution Time per trial by Velocity Function and Velocity Reset Point

A comparison of the three velocity functions using the first collision point as the velocity reset point parameter in the algorithm is shown in Figure 4.2.6. The non-linear velocity control functions performed noticeably better than using a step function to decrease the velocity. While the number of unsolved deadlocks and paths recalculated were approximately the same, the natural log function had a lower number of emergency stops, showing a better affinity for this model.

The three velocity functions were again compared in Figure 4.2.7, using the last collision point as the velocity reset point parameter in the algorithm. Once again, the both nonlinear functions showed similar results, with the Natural Log function showing a lower number of emergency stops.

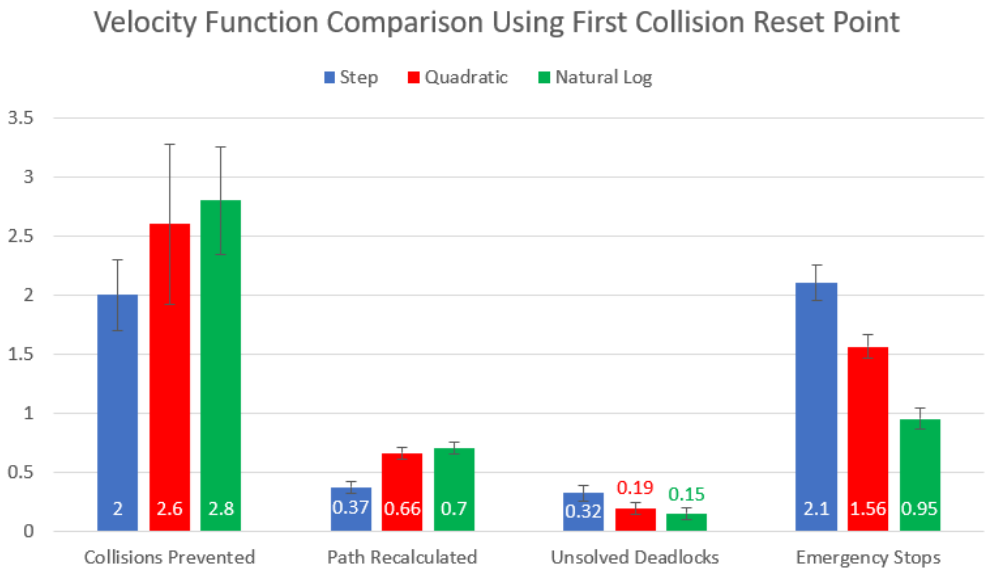


Figure 4.2.6: A comparison of the three velocity control methods with the Velocity Reset Point parameter initialized as the first collision point

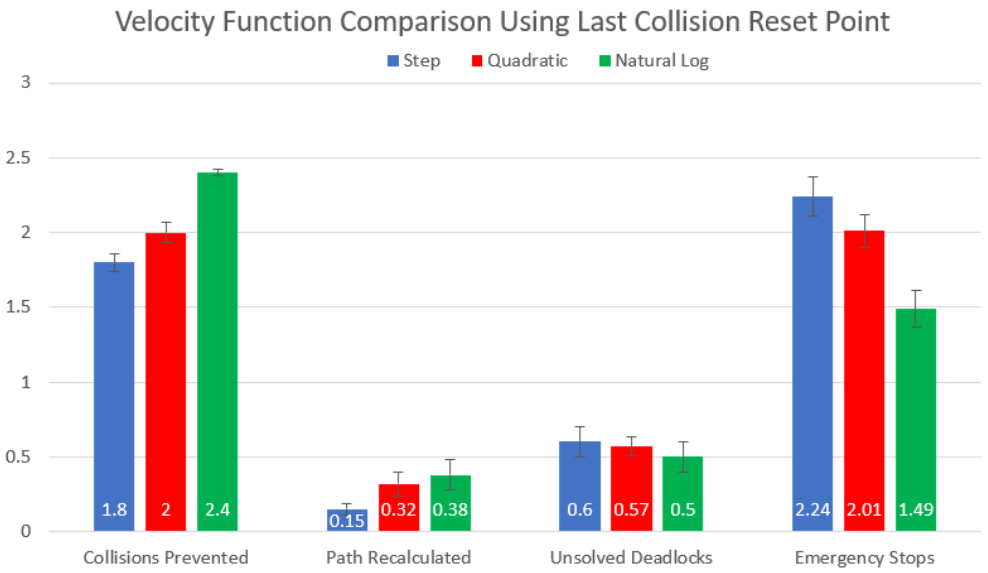


Figure 4.2.7: A comparison of the three velocity control methods with the Velocity Reset Point parameter initialized as the last collision point

Being the highest performing method of velocity control in both first and last collision reset point, the two latter settings were compared using the Natural Log Function. As shown in Figure 4.2.8 The last collision reset point parameter suffers from a low success rate in recalculating paths, and a large number of unsolved deadlocks, occurring 50% of the trials.

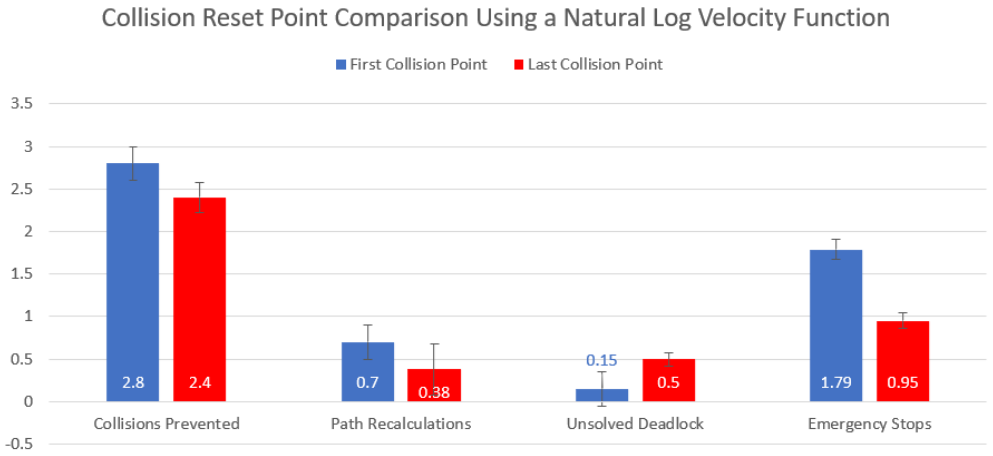


Figure 4.2.8: A comparison of Velocity Reset Point positions using a Natural Log Function as Velocity Control parameter

A plausible explanation would be the failure to consider the case of deadlocks, in which arms approach one another. As collision points are detected and set as the last collision point in the arrays, the slow arm will continue to approach its last collision point at which it plans to stop, bringing it closer to the fast arm. While designed with intent, there is no guarantee that the fast arm will reach its collision point before the slow arm. This leads to a higher risk of emergency stops caused by entering arm safety radius, and difficulties in recalculating trajectories to break deadlocks as discussed in the deadlock resolution evaluation in Section 4.3.3. Therefore, setting the velocity reset point parameter as the first collision point would give the optimal results in the main collision avoidance algorithm.

4.2.5 Discussion

This section discussed and evaluated different combinations of parameters to optimize the final motion planning and collision avoidance algorithm. The tuning of velocity functions and velocity reset points were explored, however there exists multiple other parameters that may be adjusted to optimize the algorithm. For instance, the collision detection range or safety radius could be increased or decreased depending on the desired sensibility of the detection algorithm. Furthermore, the velocity function parameters could be further tested. The Natural Log function can be adjusted by trying different multiplier values in order to tune a more or less gradual decrease in velocity. As the majority of parameters are currently determined by trial and error, the use of neural networks to find optimal algorithm inputs or even an optimal velocity function for the model could be explored in future work.

4.3 Stress Testing

4.3.1 Multiple Intersections

Overview

As the RRT* planner converges to an optimal solution in terms of the state space distance [57], the initial path computed for each arm is likely to be as simple as possible without any unnecessary deviation. However, when implemented in an environment with multiple obstacles, the planner may be forced to take detours, resulting in overlapping paths. These detours are made in order to avoid colliding with static obstacles while minimizing the total distance to the goal configuration. As the pick-and-place task lacks such obstacles to force the paths to deviate and overlap with each other more than once, this remains a less likely case. While a rare occurrence, this proposed method aims for an adaptive and robust solution to static and dynamic collision avoidance geared towards a variety of surgical tasks and unpredictable environments. Therefore, accounting for this case would be beneficial for a better understanding of the algorithm's limitations as well as its scalability for future multilateral works.

The purpose of this experiment is to observe the collision avoidance behavior when forced into a case with paths intersecting twice on their way to their goal configuration.

Four distinct types of collisions are taken into consideration in a double intersection problem:

- **Successive:** Only a single collision is detected initially at the first intersection as shown in Figure 4.3.1. Once paths are executed, the first collision is avoided by adjusting arm velocity. Once the fast arm reaches the first collision point, the slow arm's velocity is reset and paths are once again inspected for collisions. A new collision due to the change in timing is now detected at the second intersection seen in Figure 4.3.2 and handled accordingly.
- **Resolved:** A collision is initially detected at both intersections. Once paths are executed, the first collision is avoided by adjusting arm velocity. Once the fast arm reaches the first collision point, the slow arm's velocity is reset and paths are once again inspected for collisions. No collision is detected, as the second collision is avoided due to the change in timing. The arms proceed toward their goals at initial velocity.

- **Single:** Only a single collision is detected at one of the intersections as shown in Figure 4.3.1 in the case that no collision is detected at the second intersection. The other intersection poses no collision risk as arms cross the point at different timestamps.
- **None:** No collision is detected at either intersections, arms proceed through both intersections without any velocity changes.

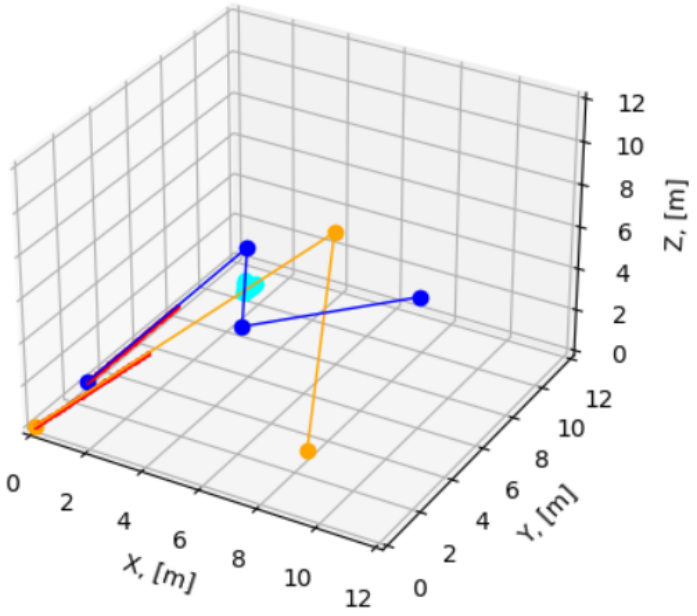


Figure 4.3.1: Single Collision or potential Successive Collision detected between Path 2 (Blue) and Path 1 (Orange), as arms (Red) approach the single collision point (Cyan). We do not know whether a second collision will be detected at the second intersection point yet.

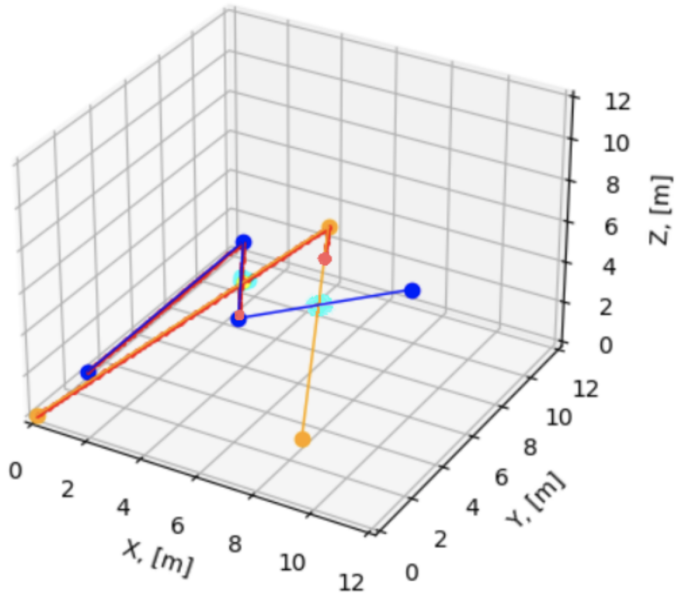


Figure 4.3.2: Successive Collision detected between Path 2 (Blue) intersects Path 1 (Orange), when arms (Red) pass the first collision point (Cyan)

Experimental Setup

As the case of a double intersection may be challenging to reproduce consistently in a pick-and-place simulation, we define the problem in a large empty space to allow more collision room. To simplify the problem and reduce computing time when generating two intersecting paths in a 3D environment, we define the first path as a polyline of 3 generated points, and the second path as a polyline of 4 generated points. In order to minimize the risk of deadlocks occurring due to paths moving in opposite directions, we fix the two arms' home location near each other at the arbitrary points of *p1* = [0, 0, 0] and *q1* = [2, 0, 3]. Each arm path point and line segments are generated such that the final paths intersect exactly twice.
This path behavior is forced by setting the following rules and restrictions:

First, a line segment *s1* of length $l1 \in [2, 18]$ is generated in the space by connecting two randomly generated points, *p2* and *p3* in $x \in [0, 12]$, $y \in [0, 12]$, and $z \in [0, 12]$. A second line, *s2,* connecting *p2* to *p1* is plotted, completing the first path, *P1*.

Two temporary points *t1* and *t2* are randomly generated anywhere on *s1* and *s2* to represent the intersection points of the second path, *P2*, with *P1*.

Another point, *q3,* is randomly generated in $x \in [0, 12]$, $y \in [0, 12]$, and $z \in [0, 12]$. This point must not intersect any lines, and represents the middle point between the two intersections.

From point *q3*, we generate two line segments, *r2* and *r3* of length $l2 \in [2, 18]$ and $l3 \in [2, 18]$ such that each begins at point *m1* and passes through one of *t1* or *t2*. Lastly, we connect *q1* to the nearest *r2* or *r3* line segment end point. In the case that one of the above steps are unfeasible, the paths are regenerated until two valid paths intersecting twice are computed as shown in Figure 4.3.3.

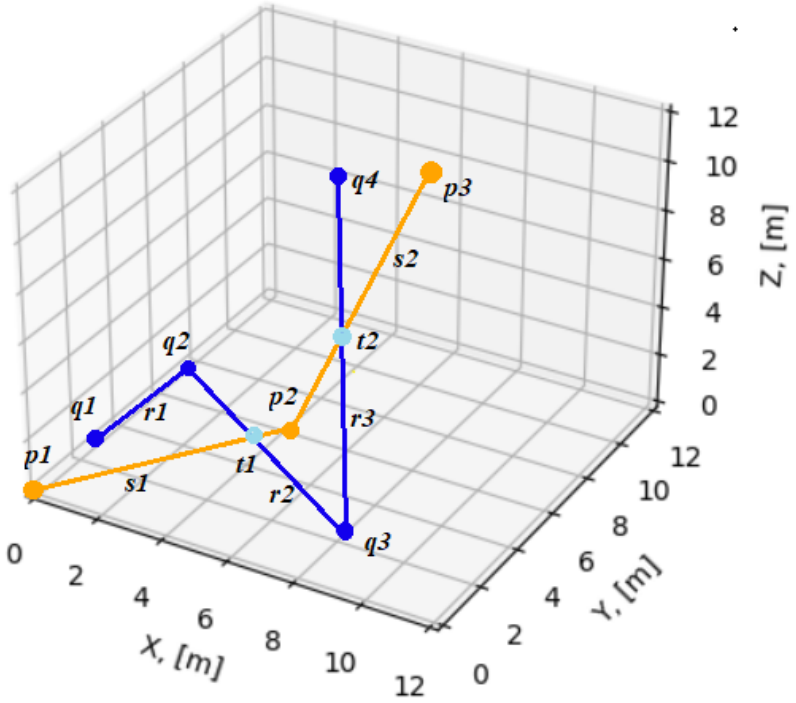


Figure 4.3.3: Path 2 (*P2* in Blue) intersects Path 1 (*P1* in Orange) at intersection points *t1* and *t2*

It is important to note that in this simplified two intersection simulation, an intersection is defined as a line intersecting with another line at a distinct point, such as *t1* or *t2*. In the final algorithm, an intersection may be defined as an array of points at which *P1*'s euclidean distance reaches a certain threshold from *P2*. In a 3D space, it is unlikely that lines will intersect exactly at one point, unless under forced circumstances such as this one.

The following parameters in Table 4.3.1 were used to tune the algorithm for this experiment over 20 trials:

Detection Range (m)	Safety Radius (m)	Default Velocity (m/s)	Velocity Function Type	Velocity Reset Point
1	0.3	0.32	Natural Log	First

Table 4.3.1: Double intersection experiment algorithm parameters

Results

The results observed in the double intersection experiment for each metric are shown in Table 4.3.2. Collisions were prevented by the emergency stops triggered when the slow arm reached the first collision point before the faster arm. A breakdown of the average execution time for each type of intersection is displayed in Figure 4.3.4. As successive collisions detect two collisions one after the other, requiring two velocity adjustments, the average execution time was found to be the highest. Resolved collisions had the second highest execution time, closely followed by single collisions, both of later only having to adjust arm velocity once. The total average execution time of a trial was found to be 5.69 seconds, with 0.9 velocity adjustments on average to avoid a collision in a trial.

Average Execution Time (s)	Collisions Detected	Velocity Adjustments	Emergency Stops	Collisions
5.69 ± 1.63	1.15 ± 0.34	0.9 ± 0.26	0.15 ± 0.11	0

Table 4.3.2: Average Collision data for two intersections per trial

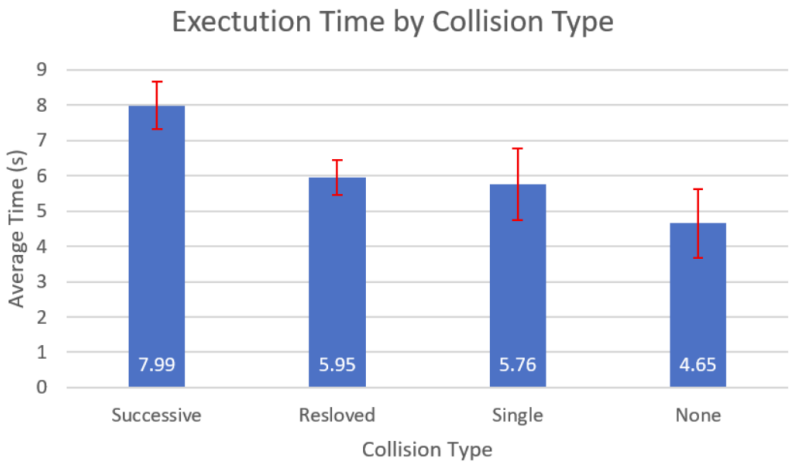


Figure 4.3.4: Average Execution Time (s) for each category of collision in a double path intersection problem

With 0.15 emergency stops and no collisions, it can be concluded that the velocity control algorithm is effective in handling two intersections in a path regardless of the number of collisions detected in these two intersections.

Discussion

As it is an unlikely edge case with only two manipulators, the generation of two intersecting paths has been executed by applying restrictions such as forcing a middle point to generate two intersection points on each side. This setup method shows a limited amount of free randomized variables with less flexibility in generating a variety of different paths. The path generation is therefore limited to using only 3 and 4 base points to shape the paths for simplicity, and represents only a small subset of possible paths that could be generated in this space.

Furthermore, the case of deadlocks in a double intersection simulation has not been covered, as it would be challenging to recreate consistently. This case is even further unlikely to occur in the main experiment with the addition of a motion planner. Deadlock in this experiment are prevented by fixing the start position of each arm at an arbitrary point near each other.

This velocity control algorithm has only been tested in the case of two path intersections, and does not cover the case in which a path intersects with itself, as this case is not plausible with a motion planner. However, it may open the way for further work with multiple manipulators and intersections.

4.3.2 Common Goal Location

Overview

In the context of a pick-and-place task, the manipulators start at a fixed home position before approaching their assigned object, then approaching their respective destination. In this case, both arms must drop their object into the same bowl. We assume that the bowl is not large enough to accomodate two arms dropping their object at once, hence representing the receptacle as a mutually exclusive resource. The arms cannot use the bowl at the same time, therefore velocity control is once again used to slow one arm and coordinate the use of the resource. As in a regular collision detection the algorithm would assume that the arm would not idle at the collision point, this collision type is handled slightly differently. The slow arm's velocity is not reset to its initial value until the other arm leaves the resource and begins approaching its next goal. This experiment tests the case in which the two arms are approaching a common goal location.

Experimental Setup

As this is likely to be a common occurrence in our main simulation, a similar environment set up and parameters were used to tune this experiment. In this experiment, two objects are generated at random on the table, representing the arm start positions. Following the regular arm state machine sequence and collision avoidance algorithm, arm paths are planned to the bowl to drop the objects. A common goal location is detected as shown in Figure 4.3.5, and the arm velocity is adjusted to ensure that arms will not meet at common goal. The first arm arriving at the bowl will pause for a certain amount of time to mock dropping the object. The slow arm's velocity continues to decrease until it either reaches a stop once within the fast arm's

safety radius, or until its velocity is reset by the event that the other arm has left the bowl. Once an arm has dropped its object, it approaches its assigned home location and ends its task.

The experiment environment was set up with parameters as follows:

- The Home positions for Arm 1 (Blue) and Arm 2 (Green) were fixed at [0.0, 1.0, 2.0] and [0.0, -1.0, 2.0] respectively.
- The Red bowl was positioned at [-1.0, 0.0, 0.8]
- Two Red objects were generated at random at x $\in$ [-0.5, 1.0], y $\in$ [-1.5, 1.5], and z = 0.33

The following parameters in Table 4.3.3 were used to tune the algorithm for this experiment over 20 trials:

Detection Range (m)	Safety Radius (m)	Default Velocity (m/s)	Velocity Function Type	Velocity Reset Point	Pause Time (s)
0.05	0.03	0.32	Natural Log	First	0.5

Table 4.3.3: Double intersection experiment algorithm parameters

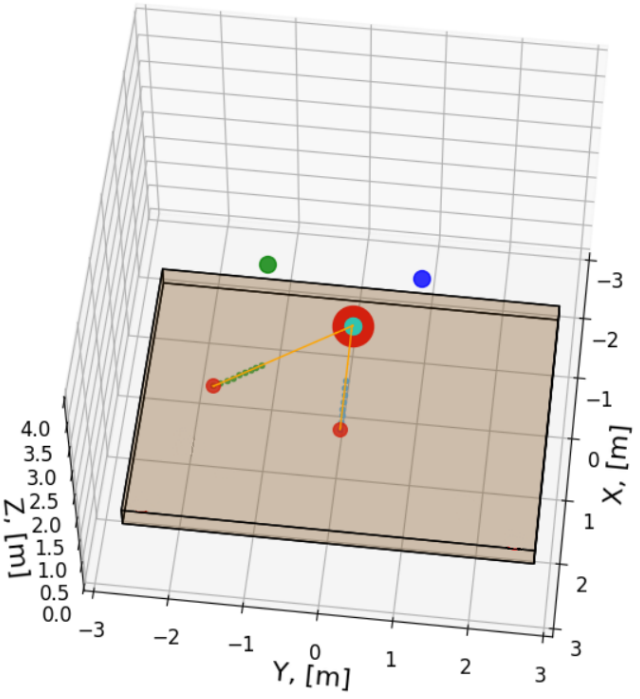


Figure 4.3.5: Two arms (green and blue trails) start from their object positions (two red points) and progress through their planned path (orange) towards a common goal (large red circle). A common goal location is detected, shown in cyan.

Results

This experiment aims at evaluating the algorithm's ability to handle a case in which arms approach a common goal location, as it is not guaranteed to be detected as a collision point. The experimental results over 20 trials are shown in Table 4.3.4. Once again, collisions were prevented using the algorithm's emergency stop function as a last resort when arms enter each other's safety radius. Despite adding a pause time at the bowl to drop the object, the slow arm's velocity continued to decrease until the other arm freed the common resource, hence preventing any collision. This shows the framework's effectiveness in the case of handling collisions coordinating multiple manipulators as they approach the same goal location.

Average Execution Time (s)	Emergency Stops	Collisions
4.04 ± 0.81	0.20 ± 0.08	0

Table 4.3.4: Average Collision Data for Common Goal Location per trial

4.3.3 Deadlock Resolution

Overview

Deadlocks are a much more common occurrence in our case due to the nature of the pick-and-place task, carrying objects to a common location, then moving back to pick up another object. This experiment evaluates our framework's ability to detect and handle deadlocks. Deadlocks are detected by checking the slow arm's state at every sampling time. If the slow arm remains in a state of emergency stop for a predefined continuous period of time, a deadlock is detected. The deadlock is then handled by setting an invisible boundary box on the slow arm and recalculating the fast arm's path as shown in Figure 4.3.6.

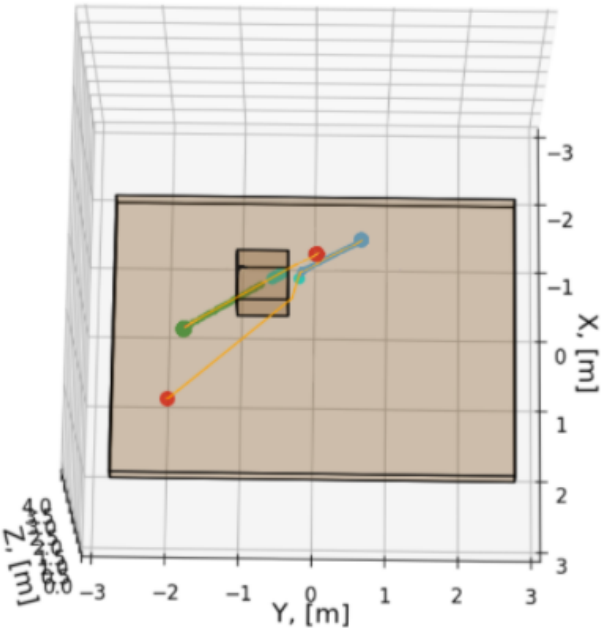


Figure 4.3.6: A deadlock is detected as arms are moving towards one another. Arm 1 (Blue) computes a new path around the boundary box enclosing Arm 2 (Green).

Experimental Setup

As deadlocks are not guaranteed to occur, this behavior is forced by restricting the generation of Arm 1's Start point to a certain distance from Arm 2's generated goal point, and Arm 2's Start point to a certain distance from Arm 1's Goal point. This allows the generation of two paths moving generally toward one another, increasing the probability of deadlock.

The Start and Goal positions for Arm 1 (Blue) are generated at random within $x \in [-0.5, 1.0]$, $y \in [-1.5, 1.5]$, and $z = 0.33$. Arm 2's Start and Goal positions are generated relative to Arm 1's Goal and Start positions within a distance of 0.1 m. The boundary box dimensions are defined as the distance between the centroid to the outer face on the *x*, *y*, and *z* axis.

The following parameters in Table 4.3.5 were used to tune the algorithm for this experiment over 20 trials:

Detection Range (m)	Safety Radius (m)	Default Velocity (m/s)	Velocity Function Type	Velocity Reset Point	Boundary Box Size (m)
0.05	0.03	0.32	Natural Log	First	[0.06, 0.06, 0.06]

Table 4.3.5: Deadlock resolution experiment algorithm parameters

Results

The results observed in the deadlock resolution experiment over 20 trials are shown in Table 4.3.6. Attempts to resolve deadlocks were made by recalculating paths after a certain amount of emergency stops in a row. In the case where a valid path could not be found, arms remained in a state of deadlock, unable to move and ending the simulation. The average number of unsolved deadlocks per trial was found to be 0.20, demonstrating the framework's effectiveness in handling deadlocks. The number of collisions is once again zero, as we choose to favor pausing manipulators indefinitely and resetting the system rather than letting the arms collide, preventing unknown consequences in the surgical environment.

Average Execution Time (s)	Unsolved Deadlocks
3.25 ± 0.23	0.20 ± 0.04

Table 4.3.6: Average Deadlock Data for Common Goal Location per trial

Discussion

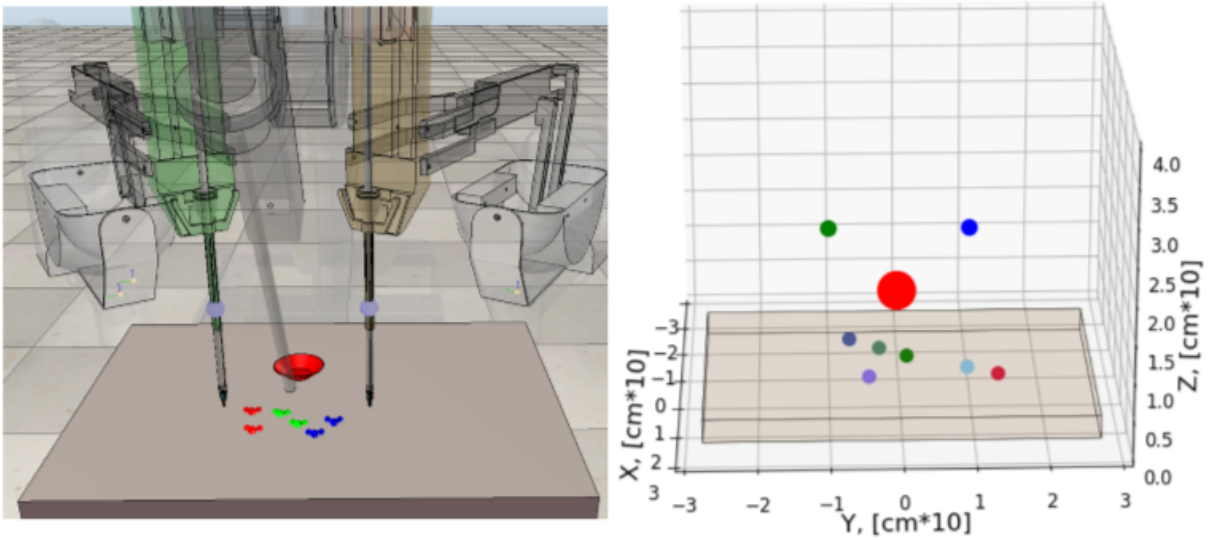
Using a path regeneration method, the collision avoidance is shown to be predominantly effective in resolving deadlocks as they occur. However, some cases remain when deadlocks may not be resolved and the motion planner fails to compute a valid path.
As there is ample space around each arm to compute a new path, It is implied that failure to compute a path occurs due to the fast arm being mistakenly enclosed inside the slow arm's boundary box. A boundary box lets the slow arm act as a static obstacle such that the motion planner can compute a path around it. The fast arm may be accidentally included inside the bounding box for a variety of inaccuracies.

An example of such inaccuracies occurs as the boundary box set on the slow arm is defined using the distance from centroid to outer face. The corners of the box may unintentionally enclose the fast arm, as they protrude outside of the intended radius. We cannot simply reduce the boundary box size as it must be larger than the collision detection range. Otherwise the fast arm would remain unable to move, as collisions with the slow arm would continue to be detected at its current position, within collision detection range. It is suggested that a cylindrical or spherical boundary volume of the exact collision detection range would be preferable and more accurate to reduce the number of unsolved deadlocks.

4.4 Overall Performance Comparison

4.4.1 Overview

Based on the metrics described in Section 4.1.1, our proposed execution model running a pick-and-place simulation was compared against two other bilateral coordination models by Abdelaal *et al* [15]. Using the experimental setup and optimized algorithm parameters determined in Section 4.2, we compare our average execution time per trial and collision data to that of the recorded results for an Independent FSM Parallelism Model and an HCSM Parallelism Model. These execution models mock a simulated first generation da Vinci surgical system patient-side cart with two PSMs and an endoscope shown in Figure 4.4.1 a), adapted in our simplified 3D plotted environment as shown in Figure 4.4.1 b).



a) The da Vinci simulator on V-REP running a pick-and-place subtask

b) Simplified 3D plotted Python environment running a pick-and-place subtask

Figure 4.4.1: Pick-and-place subtask experiment set up

The Independent FSM Parallelism Model consists of two decentralized identical arm state machines moving independently without any form of coordination. The two arms move in parallel to execute the pick-and-place task, but lack any form of coordination or collision avoidance. This case is used to quantify the effects of planning the motion of two arms and avoiding collisions while minimizing disturbances to the original path. An optimal collision

avoidance algorithm should aim to keep an average execution time near that of the Independent FSM Parallelism Model, while removing collisions risks as they occur.

The HCSM Model consists of multiple arm sub-state machines coordinated in parallel by a main centralized state machine as shown in Figure 4.4.2. The motion of the two robot arms are coordinated by having one arm pick up an object as the other drops its own, at the same time. This method does not have an implemented method of static or dynamic collision avoidance.

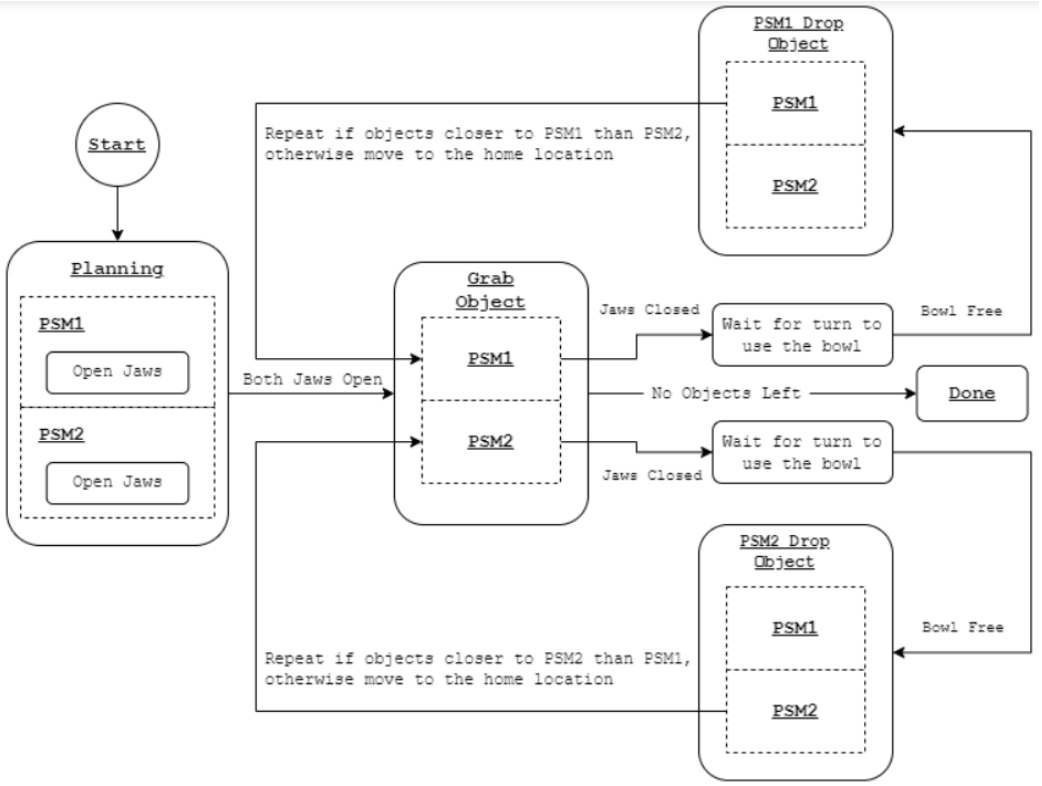


Figure 4.4.2: HCSM pick-and-place model

Our proposed framework aims to eliminate all collisions while offering a more flexible system by operating in a decentralized manner. We test our model's robustness to deadlocks, static obstacles, and arm collisions, by comparing its performance in the same pick-and-place simulation as the two previous models.

4.4.2 Results

The performance results of our execution models over 20 trials compared to the Independent FSMs and HCSM Parallelism methods are shown in Table 4.5.2. Our execution model took an average of 10.52 seconds per trial to complete a task, a slight increase in time from the Independent FSM model at 9.27 seconds. However, a noticeable increase in performance was shown in the collision avoidance aspect, resulting in an overall task error 95.31% less than that of the Independent FSM model.

Compared to the HCSM Model, our framework shows only a slight improvement in time execution time, reducing the execution time by 11.82%. Once again, a large improvement can be seen on the collision side, as collisions are completely eliminated, resulting in an overall task error of 84.21% less than that of the HCSM model.

As the Independent FSM execution does not have a coordination mechanism implemented, it is expected that its execution time would be lower than the two other execution models. Therefore, a slight increase in average execution time shows our success in creating an optimized algorithm. While our framework shows only a slight improvement in time execution compared to the HCSM Parallelism method, we include a broader range of collision measures to produce a robust system, including both static path planning and a dynamic avoidance mechanism. This framework arguably shows better adaptability to a variety of multilateral surgical tasks and robotic uses as the algorithm is not bound by a specific task.

Metric	Velocity Obstacle Method	Independent FSM Parallelism Method	HCSM Parallelism Method
Average Execution Time (s)	10.52 ± 0.59	9.27 ± 0.79	11.93 ± 0.72
Velocity Adjustments	2.8 ± 0.31	N/A	N/A
Unsolved Deadlocks	0.15 ± 0.09	N/A	N/A
Collisions	0	3.2 ± 0.98	0.95 ± 0.33
Overall Task Error	0.15 ± 0.09	3.2 ± 0.98	0.95 ± 0.33

Table 4.4.1: Overall performance comparison between our proposed framework, and two parallelism methods.

4.4.3 Discussion

While the access to the bowl resource is coordinated in the HCSM model such that only one arm is ever at the bowl location at once, this mechanism is only implemented at the bowl location. Collisions occurring as PSMs approach objects are not handled, resulting in the majority of collisions occurring away from the bowl. Our proposed execution model attempts to eliminate this problem by offering a dynamic collision avoidance mechanism operating at any point in space. Furthermore, the majority of collisions and deadlocks occur when multiple objects and the bowl are aligned, as both arms compute similar trajectories to move towards objects and back to the bowl.

While we combine unsolved deadlocks and collisions to form an overall task error metric, it is noted that there could exist multiple collisions in a single trial, but only one unsolvable deadlock per trial. In the case of an unsolvable deadlock, the trial would need to be reset to complete the simulation. We assume that forcing the simulation to stop would be prefered to having unknown consequences due to a collision in a surgical environment.

Chapter 5

Conclusions

5.1 Conclusions and Future Work

Our proposed method provides an effective method of dynamic collision avoidance by using the concept of velocity obstacle in a TMP context. The framework was simulated in a pick-and-place task to represent the process of tissue debridement in a constrained surgical environment. While using a velocity control approach successfully avoids the majority of collisions dynamically, there remains task specific cases requiring the additional coverage of deadlock resolution and common goal coordination. Several edge case solutions were explored and evaluated using high-level criteria into a constrained collision avoidance environment to force the behaviour consistently. The effectiveness of the framework was demonstrated in a pick-and-place subtask and could potentially be suited for a variety of tasks in similar conditions. Our collision avoidance framework was put through feature selection tests and stress tests, then compared against two existing parallelism task coordination frameworks to evaluate the overall optimized system performance.

The implementation of this framework consists in controlling the velocity of manipulators by decreasing the speed fitted to a natural log function once a collision is detected. Deadlocks resolving attempts are made by recalculating the path around the other arm. In the case that deadlocks are not resolved, the arms remain frozen in place until the system is reset to its initial state. With the elimination of collisions, we encounter complications with deadlocks, impeding the success rate of the algorithm. It is assumed that halting the task would be preferable to the unknown repercussions that an arm-to-arm collision may have in the environment. While collisions are undesirable, not all collisions may be considered fatal. Methods of identifying which collisions to avoid or ways of mitigating non-fatal collisions should they occur may at times be more preferable than resetting the system after an unsolvable deadlock.

In this simulation, we represent the robot manipulators as two points in space rather than a volume entity. In order to get an in-depth evaluation of the framework and its limits, this algorithm should be adapted to a simulation with the da Vinci Robot System such as the V-REP simulator [11], or ideally the robot itself. This would allow us to fully assess our collision avoidance features by integrating the collision detection range and safety radius over the entirety of the robot manipulator. Collisions should be detected over the entire arm surface rather than solely at the manipulator end point.

The velocity control algorithm updates the arm velocity by transforming fitting its planned path to a natural log function. Further research could be made regarding different methods of velocity deceleration in order to choose an adequate velocity to optimize execution time. We have tested velocity adjustment by decreasing the velocity by a constant, using a quadratic function, and a natural log function to fit the path, however velocity control could include any combination of functions. The algorithm could be made even more adaptive by implementing more environment states as function parameters. The integration of recurrent neural networks (RNN) [65] could also be used to solve for optimal algorithm inputs and velocity function for the model.

Multiple intersections were not observed in the overall algorithm performance, however this feature was nonetheless tested by forcing the generation of paths intersection twice. While this behaviour may not have been observed in the context of our pick-and-place task, further work with a greater variety of tasks may result in the occurrence of a double intersection. Increasing the number of manipulators to improve execution time would also greatly increase the chances of such a case as we now work with a trilateral model and the collision avoidance of three paths at once in a constrained 3D environment. This algorithm could be further stress tested against more than two intersection cases, with two manipulators or more.

5.2 Limitations

As our framework is simulated in a 3D plotted Python environment, this simulation provides adequate insight of the collision avoidance algorithm with the manipulators represented as a single point. The reported number of overall task errors may be lower than expected, as we do not consider the entire area of the arm in this execution model.

The implementation of the deadlock resolution mechanism could further be improved to reduce the number of unsolved deadlocks. In a simulation using a model of the da Vinci Surgical Robot, the boundary box generated to enclose a manipulator such that the other can replan its path around should be fitted to the arm's shape. A shape with protruding corners such as a cube may cause some inaccuracies such as both arms being enclosed in the box.

References

[1] T. A. Ponsky, A. Khosla, and J. L. Ponsky, “Minimally Invasive Surgery,” *Wiley Online Library*, 2012. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118321386.ch157

[2] A. C. Mason, M. J. Krasna, and C. S. White, “The Role of Radiologic Imaging in Diagnosing Complications of Video-Assisted Thoracoscopic Surgery,” *Chest*, vol. 113, no. 3, pp. 820–825, Mar. 1998, doi: 10.1378/chest.113.3.820.

[3] K. Mohiuddin and S. J. Swanson, “Maximizing the benefit of minimally invasive surgery,” *Journal of Surgical Oncology*, vol. 108, no. 5, pp. 315–319, Aug. 2013, doi: 10.1002/jso.23398.

[4]Y. Ikeda, H. Takami, Y. Sasaki, J. Takayama, and H. Kurihara, “Are There Significant Benefits of Minimally Invasive Endoscopic Thyroidectomy?,” *World Journal of Surgery*, vol. 28, no. 11, pp. 1075–1078, Oct. 2004, doi: 10.1007/s00268-004-7655-2.

[5] M. J. H. Lum *et al.*, “The RAVEN: Design and Validation of a Telesurgery System,” *The International Journal of Robotics Research*, vol. 28, no. 9, pp. 1183–1197, May 2009, doi: 10.1177/0278364909101795.

[6] G. H. Ballantyne, “The Pitfalls of Laparoscopic Surgery: Challenges for Robotics and Telerobotic Surgery,” *Surgical Laparoscopy, Endoscopy & Percutaneous Techniques*, vol. 12, no. 1, pp. 1–5, Feb. 2002, doi: 10.1097/00129689-200202000-00001.

[7] J. Sándor *et al.*, “Minimally invasive surgical technologies: Challenges in education and training,” *Asian Journal of Endoscopic Surgery*, vol. 3, no. 3, pp. 101–108, May 2010, doi: 10.1111/j.1758-5910.2010.00050.x.

[8] B. Hannaford *et al.*, “Raven-II: An Open Platform for Surgical Robotics Research,” *IEEE Transactions on Biomedical Engineering*, vol. 60, no. 4, pp. 954–959, Apr. 2013, doi: 10.1109/tbme.2012.2228858.

[9] J. Palep, “Robotic assisted minimally invasive surgery,” *Journal of Minimal Access Surgery*, vol. 5, no. 1, pp. 1–7, 2009, doi: 10.4103/0972-9941.51313.

[10] I. A. M. J. Broeders and J. Ruurda, “Robotics revolutionizing surgery: the Intuitive Surgical ‘Da Vinci’ system,” *Industrial Robot: An International Journal*, vol. 28, no. 5, pp. 387–392, Oct. 2001, doi: 10.1108/eum0000000005845.

[11] G. A. Fontanelli, M. Selvaggio, M. Ferro, F. Ficuciello, M. Vendittelli, and B. Siciliano, “A V-REP Simulator for the da Vinci Research Kit Robotic Platform,” in *2018 7th IEEE International Conference on Biomedical Robotics and Biomechatronics (Biorob)*, Aug. 2018, pp. 1056–1061. [Online]. Available: http://dx.doi.org/10.1109/biorob.2018.8487187

[12] Z. Chen, A. Deguet, R. H. Taylor, and P. Kazanzides, “Software Architecture of the Da Vinci Research Kit,” Apr. 2017. [Online]. Available: http://dx.doi.org/10.1109/irc.2017.69

[13] A. Pandya *et al.*, “A Review of Camera Viewpoint Automation in Robotic and Laparoscopic Surgery,” *Robotics*, vol. 3, no. 3, pp. 310–329, Aug. 2014, doi: 10.3390/robotics3030310.

[14] C. E. Attinger, E. J. Bulan, and Pa. Blume, “Surgical Debridement: the Key to Successful Wound Healing and Reconstruction,” *WB Saunders*, Nov. 01, 2000. https://www.researchgate.net/publication/12256428_Surgical_Debridement_the_Key_to_Successful_Wound_Healing_and_Reconstruction

[15] A. E. Abdelaal, J. Liu, N. Hong, G. D. Hager, and S. E. Salcudean, “Parallelism in Autonomous Robotic Surgery,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1824–1831, Apr. 2021, doi: 10.1109/lra.2021.3060402.

[16] C. R. Garrett *et al.*, “Integrated Task and Motion Planning,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 265–293, May 2021, doi: 10.1146/annurev-control-091420-084139.

[17] B. Kehoe *et al.*, “Autonomous multilateral debridement with the Raven surgical robot,” May 2014. [Online]. Available: http://dx.doi.org/10.1109/icra.2014.6907040

[18] J. S. Grover, C. Liu, and K. Sycara, “Deadlock Analysis and Resolution for Multi-robot Systems,” in *Algorithmic Foundations of Robotics XIV*, Cham: Springer International Publishing, 2021, pp. 294–312. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-66723-8_18

[19] W. Thomason and R. A. Knepper, “A Unified Sampling-Based Approach to Integrated Task and Motion Planning,” Jun. 03, 2019. https://www.researchgate.net/publication/342011117_A_Unified_Sampling-Based_Approach_to_Integrated_Task_and_Motion_Planning

[20] J. Wolfe, B. Marthi, and S. Russell, “Combined Task and Motion Planning for Mobile Manipulation,” *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 20, pp. 254–257, Jan. 2010, doi: 13436.

[21] S. Sen, A. Garg, D. V. Gealy, S. McKinley, Y. Jen, and K. Goldberg, “Automating multi-throw multilateral surgical suturing with a mechanical needle guide and sequential convex optimization,” May 2016. [Online]. Available: http://dx.doi.org/10.1109/icra.2016.7487611

[22] O. Ahmad, J. Cremer, J. Kearney, P. Willemsen, and S. Hansen, “Hierarchical, concurrent state machines for behavior modeling and scenario control.” [Online]. Available: http://dx.doi.org/10.1109/aihas.1994.390503

[23] L. P. Kaelbling and T. Lozano-Perez, “Hierarchical task and motion planning in the now,” in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1470–1471. [Online]. Available: http://dx.doi.org/10.1109/icra.2011.5980391

[24] Z. Yan, N. Jouandeau, and A. A. Cherif, “A Survey and Analysis of Multi-Robot Coordination,” *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, Jan. 2013, doi: 10.5772/57313.

[25] K. M. Wurm, C. Stachniss, and W. Burgard, “Coordinated multi-robot exploration using a segmentation of the environment,” Sep. 2008. [Online]. Available: http://dx.doi.org/10.1109/iros.2008.4650734

[26] C. R. Kube and E. Bonabeau, “Cooperative transport by ants and robots,” *Robotics and Autonomous Systems*, vol. 30, no. 1–2, pp. 85–101, Jan. 2000, doi: 10.1016/s0921-8890(99)00066-4.

[27] K.-Y. Chen, P. A. Lindsay, P. J. Robinson, and H. A. Abbass, “A hierarchical conflict resolution method for multi-agent path planning,” May 2009. [Online]. Available: http://dx.doi.org/10.1109/cec.2009.4983078

[28] Shuhua Liu, Yantao Tian, and Heping Lin, “A Multi-Level Conflict Resolution Method for Large-Scale Robot System,” 2006. [Online]. Available: http://dx.doi.org/10.1109/wcica.2006.1713742

[29] M. Bennewitz, W. Burgard, and S. Thrun, “Optimizing schedules for prioritized path planning of multi-robot systems.” [Online]. Available: http://dx.doi.org/10.1109/robot.2001.932565

[30] L. S. Marcolino and L. Chaimowicz, “Traffic control for a swarm of robots: Avoiding group conflicts,” Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/iros.2009.5354336

[31] V. Graciano Santos and L. Chaimowicz, “Hierarchical congestion control for robotic swarms,” Sep. 2011. [Online]. Available: http://dx.doi.org/10.1109/iros.2011.6094640

[32] N. Jouandeau and Z. Yan, “Decentralized waypoint-based multi-robot coordination,” May 2012. [Online]. Available: http://dx.doi.org/10.1109/cyber.2012.6392549

[33] M. Jager and B. Nebel, “Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots.” [Online]. Available: http://dx.doi.org/10.1109/iros.2001.977148

[34] J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit, “Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles,” *Autonomous Robots*, vol. 42, no. 4, pp. 801–824, Aug. 2017, doi: 10.1007/s10514-017-9665-6.

[35] G. A. S. Pereira, B. S. Pimentel, L. Chaimowicz, and M. F. M. Campos, “Coordination of multiple mobile robots in an object carrying task using implicit communication.” [Online]. Available: http://dx.doi.org/10.1109/robot.2002.1013374

[36] A. Anand, M. Nithya, and T. Sudarshan, “Coordination of mobile robots with master-slave architecture for a service application,” Nov. 2014. [Online]. Available: http://dx.doi.org/10.1109/ic3i.2014.7019647

[37] E. Nett and S. Schemmer, “Reliable real-time communication in cooperative mobile applications,” *IEEE Transactions on Computers*, vol. 52, no. 2, pp. 166–180, Feb. 2003, doi: 10.1109/tc.2003.1176984.

[38] U. Göllner, “Grassé, Pierre-P.: Fondation des Société – Construction. Termitologia. 2. 624 S., 452 Fig., 28 Tab., Masson, Paris, New York, Barcelona, Milan, Mexico, Sao Paulo, 1984,” *Deutsche Entomologische Zeitschrift*, vol. 32, no. 4–5, pp. 379–379, Sep. 1985, doi: 10.1002/mmnd.4800320420.

[39] N. Gildert, A. G. Millard, A. Pomfret, and J. Timmis, “The Need for Combining Implicit and Explicit Communication in Cooperative Robotic Systems,” *Frontiers in Robotics and AI*, vol. 5, Jun. 2018, doi: 10.3389/frobt.2018.00065.

[40] S. Kato, S. Nishiyama, and J. Takeno, “Coordinating Mobile Robots By Applying Traffic Rules.” [Online]. Available: http://dx.doi.org/10.1109/iros.1992.594218

[41] L. E. Parker, “Multiple Mobile Robot Teams, Path Planning and Motion Coordination in,” in *Encyclopedia of Complexity and Systems Science*, New York, NY: Springer New York, 2009, pp. 5783–5800. [Online]. Available: http://dx.doi.org/10.1007/978-0-387-30440-3_344

[42] D. Vail and M. Veloso, *In Multi-Robot Systems: From Swarms to Intelligent Automata*, vol. 2. Kluwer Academic Publishers, 2003, pp. 87–100. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1.6819

[43] P. A. O’Donnell and T. Lozano-Perez, “Deadlock-free and collision-free coordination of two robot manipulators,” 1989. [Online]. Available: http://dx.doi.org/10.1109/robot.1989.100033

[44] J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit, “Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles,” *Autonomous Robots*, vol. 42, no. 4, pp. 801–824, Aug. 2017, doi: 10.1007/s10514-017-9665-6.

[45] H. Ha, J. Xu, and S. Song, “Learning a Decentralized Multi-arm Motion Planner,” 2020. https://www.researchgate.net/publication/345351476_Learning_a_Decentralized_Multi-arm_Motion_Planner

[46] P. Caloud, Wonyun Choi, J.-C. Latombe, C. Le Pape, and M. Yim, “Indoor automation with many mobile robots.” [Online]. Available: http://dx.doi.org/10.1109/iros.1990.262370

[47] A. Yamashita, T. Arai, Jun Ota, and H. Asama, “Motion planning of multiple mobile robots for cooperative manipulation and transportation,” *IEEE Transactions on Robotics and Automation*, vol. 19, no. 2, pp. 223–237, Apr. 2003, doi: 10.1109/tra.2003.809592.

[48] G. Sanchez and J.-C. Latombe, “Using a PRM planner to compare centralized and decoupled planning for multi-robot systems.” [Online]. Available: http://dx.doi.org/10.1109/robot.2002.1014852

[49] T. Siméon, J.-P. Laumond, J. Cortés, and A. Sahbani, “Manipulation Planning with Probabilistic Roadmaps,” *The International Journal of Robotics Research*, vol. 23, no. 7–8, pp. 729–746, Aug. 2004, doi: 10.1177/0278364904045471.

[50] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, “Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization,” Jun. 2013. [Online]. Available: http://dx.doi.org/10.15607/rss.2013.ix.031

[51] R. Luna and K. E. Bekris, “Efficient and complete centralized multi-robot path planning,” Sep. 2011. [Online]. Available: http://dx.doi.org/10.1109/iros.2011.6095085

[52] C. M. Clark, S. M. Rock, and J.-C. Latombe, “Motion planning for multiple mobile robots using dynamic networks.” [Online]. Available: http://dx.doi.org/10.1109/robot.2003.1242252

[53] D. P. Losey, M. Li, J. Bohg, and D. Sadigh, “Learning from My Partner’s Actions: Roles in Decentralized Robot Teams,” Oct. 16, 2019. https://www.researchgate.net/publication/336639132_Learning_from_My_Partner%27s_Actions_Roles_in_Decentralized_Robot_Teams

[54] O. Khatib, K. Yokoi, K. Chang, D. Ruspini, R. Holmberg, and A. Casal, “Vehicle/arm coordination and multiple mobile manipulator decentralized cooperation.” [Online]. Available: http://dx.doi.org/10.1109/iros.1996.570849

[55] D. Wilkie, J. van den Berg, and D. Manocha, “Generalized velocity obstacles,” Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/iros.2009.5354175

[56] E. Owen and L. Montano, “Motion planning in dynamic environments using the velocity space,” 2005. [Online]. Available: http://dx.doi.org/10.1109/iros.2005.1545110

[57] I. Noreen, A. Khan, and Z. Habib, “A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms”.

[58] J. van den Berg, J. Snape, S. J. Guy, and D. Manocha, “Reciprocal collision avoidance with acceleration-velocity obstacles,” May 2011. [Online]. Available: http://dx.doi.org/10.1109/icra.2011.5980408

[59] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, “The Hybrid Reciprocal Velocity Obstacle,” *IEEE Transactions on Robotics*, vol. 27, no. 4, pp. 696–706, Aug. 2011, doi: 10.1109/tro.2011.2120810.

[60] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha, “Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles,” Oct. 2009. [Online]. Available: http://dx.doi.org/10.1109/iros.2009.5354821

[61] C. M. Clark, “Probabilistic Road Map sampling strategies for multi-robot motion planning,” *Robotics and Autonomous Systems*, vol. 53, no. 3–4, pp. 244–264, Dec. 2005, doi: 10.1016/j.robot.2005.09.002.

[62] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, Jun. 2011, doi: 10.1177/0278364911406761.

[63] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996, doi: 10.1109/70.508439.

[64] H. Niemann and R. Miklos, “A Simple Method for Estimation of Parameters in First order Systems,” *Journal of Physics: Conference Series*, vol. 570, no. 1, p. 012001, Dec. 2014, doi: 10.1088/1742-6596/570/1/012001.

[65] S. Otte, L. Hofmaier, and M. V. Butz, “Integrative Collision Avoidance Within RNN-Driven Many-Joint Robot Arms,” in *Artificial Neural Networks and Machine Learning – ICANN 2018*, Cham: Springer International Publishing, 2018, pp. 748–758. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-01424-7_73