Reconfigurable Autonomous Surface Vehicles: Perception and Trajectory Optimization Algorithms

by

Banti Henricus Gheneti

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

Author . . . .
Department of Electrical Engineering and Computer Science
February 1, 2019

Certified by . . . .
Daniela Rus
Andrew (1956) and Erna Viterbi Professor, CSAIL Director
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chairman, Department Committee on Graduate Theses

Reconfigurable Autonomous Surface Vehicles: Perception and Trajectory Optimization Algorithms

by

Banti Henricus Gheneti

Submitted to the Department of Electrical Engineering and Computer Science
on February 1, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

Autonomous Surface Vehicles (ASV) are a highly active area of robotics with many ongoing projects in search and rescue, environmental surveying, monitoring, and beyond. There have been significant studies on ASVs in riverine, coastal, and sea environments, yet only limited research on urban waterways, one of the most busy and important water environments. This thesis presents an Urban Autonomy System that is able to meet the critical precision, real-time and other requirements that are unique to ASVs in urban waterways. LiDAR-based perception algorithms are presented to enable robust and precise obstacle avoidance and object pose estimation on the water.

Additionally, operating ASVs in well-networked urban waterways creates many potential use cases for ASVs to serve as re-configurable urban infrastructure, but this necessitates developing novel multi-robot planners for urban ASV operations. Efficient sequential quadratic programming and real-time B-spline parameterized mixed-integer quadratic programming multi-ASV motion planners are presented respectively for formation changing and shapeshifting operations, enabling use cases such as ASV docking and bridge-building on water. These methods increase the potential of urban and non-urban ASVs in the field. The underlying planners in turn contribute to the motion planning and trajectory optimization toolbox for unmanned aerial vehicles (UAVs), self-driving cars, and other autonomous systems.

Thesis Supervisor: Daniela Rus
Title: Andrew (1956) and Erna Viterbi Professor, CSAIL Director

Acknowledgments

First, I would like to thank all the Professors at MIT without whom I would not have developed the sufficient capacity, curiosity, and confidence in a diverse range of perspectives to the interdisciplinary field of Robotics to complete this undertaking. Professor Rus served as a stalwart advocate and advisor during my time at the Senseable City Lab and drove me to study important research questions and produce results I would not otherwise have thought probable. Professor Tedrake elucidated the dynamics of underactuated systems and piqued my interest in optimization-based methods in robotics, greatly shaping the technical direction of this thesis in the process. Professor Karaman taught me state space control systems and gave me the opportunity to serve as a lab assistant for the Robotics Science and Systems course.

Second, I would like to thank all the members of the broader Roboat team and Sense-able City Lab who made my time as a masters student at MIT colorful and mind-altering. I would like to thank Professor Ratti for welcoming me into this amazing project. In particular, I would like to thank Wei Wang and Luis Mateos for getting me involved with the Roboat project in the first place and working with me on the quarter scale autonomous prototype of the Roboat. I greatly appreciate Shinkyu Park for being an amazing mentor and collaborator, helping me to understand research and become a better researcher. I would also like to thank my other collaborators, Ryan Kelly and Drew Meyers, and other members of the roboat team and lab, including Cheng Dai, Bill Cai, Xiaojiang Li, Rachel Seavey and Rose Silva, who helped me in many ways during my time at the lab and were great company to be around. I would also like to extend my gratitude to Erin Schenk and Fabio Duarte. They provided me with many opportunities and responsibilities beyond those of a researcher, allowing me to gain many invaluable lessons about what running a large global project entails that I would never have expected to learn during my Masters.

Lastly, I would like to thanks my friends, family and other inspirations from MIT, NYC, AMS, SSDH and beyond. In particular I would like to thank my parents, Henriette Keijzers and Yeshanew Gheneti, for nurturing my freedom and eccentricities and serving as beacons with their values, dedication and grit. I would like to thank Chloe Yang for her unconditional care and awesome ingenuity and guts, helping me through hurdles and jumps during the thesis process and beyond. Finally, on a more lighthearted note, I would like to thank Benjamin Booker for serving as a great soundtrack to the thesis writing experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the advent of field robots, autonomy has been a core goal in the field of robotics. Unmoored from fixed positions, robots can move through space, increasing the complexity of their environments and the interactions they have with those environments. This raises important questions of how mobile robots independently perceive their environments, reason about them and act upon them. Better understanding the design of autonomy system components and how they interact to enable autonomy in a mobile robot could potentially expand the roles robots can safely and confidently take to serve humans within society.

1.1 Recent Developments

While autonomous mobile robots have been used in relatively tightly scoped robotics competitions, deployed in purpose built warehouses and sailed in the open oceans, it was only recently with the DARPA Grand Challenge and initiatives following it that quad-rotors could deliver packages in China and autonomous cars could drive on public roads in California, albeit with supervision [30, 10, 53]. These developments raise important questions about which challenges remain unconquered, what high-level design principles and novel algorithms have been employed in building autonomy systems for cars, and how these newfound capabilities translate to other kinds of autonomous systems such as autonomous surface vehicles.

1.2 Motivation

This thesis envisions an autonomous surface vehicle (ASV) capable of performing multi-boat operations in challenging urban environments. While significant advances, have been made for self-driving cars, ASVs have limited autonomy capabilities and only in riverine, coastal and sea environments. While research groups and companies are actively developing self-driving cars on busy and challenging urban roadways, autonomous surface vehicles have seen no deployments in similarly challenging inland and urban waterways.

In the same way rain and weather affect self-driving cars, waves and reflections on the water make perception difficult for ASVs. Where cars are able to use reliable odometry for localization, ASVs cannot and have to rely more on radar and GPS which provide only rough measurements. These difficulties expand the safety margins that need to be factored in for planning, yielding systems that can only navigate safely in large, relatively open areas. The challenges experienced by each autonomy module make it challenging to operate ASVs in urban settings, let alone multi-ASV operation in urban-settings. I demonstrate an ASV system design, robust perception algorithms and novel multi-boat planning algorithms, tackling the challenges facing ASVs in urban settings at multiple levels.

By putting together the building blocks outlined in this thesis, I hope to enable new kinds of autonomy on water. First, I want robots to achieve on water what has been achieved on land, safely navigating with ease and autonomously providing new kinds of on-demand services, like ride-sharing and municipal waste-collection. Second, I hope to enable ASVs to collaborate and reconfigure as needed, one moment carrying out individual missions across the city and the next latching together to form a bridge.

1.3 Roboat

The Roboat is an ASV being built according to this vision as part of a collaboration between various labs at MIT, including the Senseable City Lab and Distributed Robotics Lab, as well as local partners AMS and Waternet. After centuries of importance to the society and economy of Amsterdam, the canals are presently used in a more limited capacity. The Roboat project envisions strengthening the civil and commercial purposes of the canals through the creation of an autonomous boat platform that will enable a range of new uses for the canals in the areas of transportation, municipal waste collection and food distribution. To fully realize these use cases, the Roboat must navigate through the canals, avoid collisions with boats and other objects on the water, and in some cases, interact with such objects by more intelligent means [63].

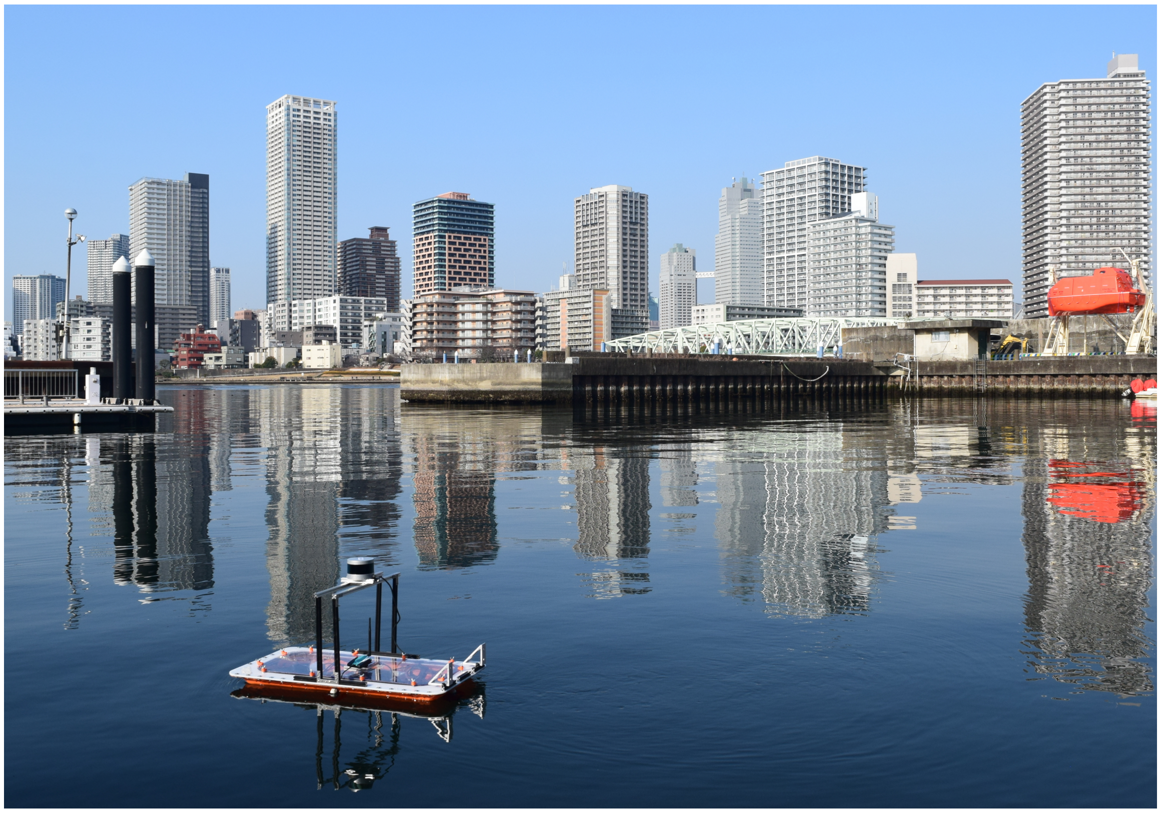


Figure 1-1: Roboat in the Tokyo Harbor

1.4 Contributions

.

Drawing from perception and planning approaches developed in self-driving car and unmanned aerial vehicle research, I present an urban autonomy system for Roboat, as well as a novel ASV for urban waterways, focusing particularly on contributions in robust LiDAR-based perception and multi-robot trajectory optimization.

1.4.1 Urban Autonomy System

The design of the Roboat's autonomy system, the selection of and interaction between software and hardware components, is a crucial part of creating an infrastructure capable of executing the use cases mentioned in the Roboat subsection above. It is important that the computations and actions necessary can be performed by the system components, and that the components interact in a fast, reliable and interpretable manner to perform these use cases [5, 24, 38].

While there are many ASVs for coastal and marine applications [18, 34, 33], these have limited autonomy in perception and planning. Few besides the Roboat meet the hardware and software requirements necessary to work in challenging urban conditions. [33, 59, 29, 42].

1.4.2 LiDAR-based Perception

As part of this system, it is important that the Roboat have sensors to record its environment and software components to interpret and represent sensor input. A Roboat must perceive to determine where it is and avoid collisions.

There are few ASVs with the requisite hardware to perform reliable obstacle detection and avoidance in narrow environments, and no other such ASVs are able to dynamically avoid obstacles and perform precise object pose estimation in narrow quarters [29, 42, 59].

1.4.3 Multi-Robot Trajectory Optimization

Equipped with LiDAR-based perception, the Roboat is prepared to understand its environment and plan appropriately in urban contexts. In the real world, the Roboat must plan to safely and expediently complete its objective, whether that be taking a tourist from the IJ to the Rijksmuseum, collecting municipal waste and dropping it off at the waste to energy plant, or coordinating with other Roboats to form a temporary bridge. A Roboat must plan, considering its state and goals to determine its actions.

While many ASVs operate individually and some act as part of multi-robot systems, which involves sharing sensor measurements and state information or coordinating plans, there is only limited research on modular reconfigurable ASVs [33, 40, 50]. In reconfigurable robotic systems, robots physically make contact and interact with each other to change the physical dimensions and dynamics of the system as a whole. Roboat implements a modular and reconfigurable design to enable the re-purposing of urban infrastructure when the system is used to create platforms or bridges on water. In their paper, Ohara *et al.* present a system for creating a structure out of boats, but this system requires that part of the configuration be attached to a docking structure and does not consider the task of reconfiguring the boats from this docked structure to another one [40]. This thesis presents novel contributions to the field which allow for truly reconfigurable modular ASVs. I present a real-time reconfiguration motion-planner for multi-robot systems, drawing from trajectory optimization formulations for UAVs. [9, 28, 62, 32].

1.4.4 Summary

In summary, this thesis contributes the following:

- A Complete Surface Vehicle Urban Autonomy System Architecture
  - Precise trajectory planning and tracking
  - Reliable performance in highly dynamic urban environments

- LiDAR-based Perception Algorithms
  - Reliable detection and avoidance of static and dynamic obstacles
  - Precise object pose estimation on water
- Multirobot Trajectory Optimization Algorithms
  - Efficient concurrent formation changing with SQP optimization
  - Real-time modular robot shapeshifting with MIQP and SQP optimization
  - Shapeshifting on water

1.5 Outline

In Chapter 2, I extensively discuss prior work related to the contributions made in this thesis. In chapter 3, I outline the design of the Urban Autonomy System for the Roboat, focusing particularly on its software architecture and ROS packages. In Chapter 4, I discuss various perception algorithms for filtering, clustering and model fitting I integrated and implemented to perform robust obstacle detection and object pose estimation on water. In Chapter 5, I present the two multi-boat planning scenarios of formation changing and shapeshifting, and I demonstrate efficient optimization-based methods for solving them. Finally, in Chapter 6, I conclude with lessons learned from my undertakings and outline sketches of where research in urban ASVs can take us.

Chapter 2

Related Work

2.1 Urban Autonomy System

Current USVs are usually deployed for coastal and marine applications [18, 34, 33]. These USVs, shown in Table 2.1, have limited maneuverability (especially in shape and actuation), and limited autonomy (especially in perception and planning), making them unable to meet the autonomy requirements for applications in narrow and crowded urban environments.

Table 2.1: Autonomy Analysis of Typical USVs ([18, 34, 33])

USVs	**Shape**	**Actuation**	**Localization**	**Perception**	**Planning**	**Control**
ARTEMIS	V-hull	Under	GPS+IMU	N	N	fuzzy
ACES	Catamaran	Under	GPS+IMU	N	N	fuzzy
SCOUT	Kayak	Under	GPS+IMU	RF	N	PID
AutoCat	Catamaran	Under	GPS+IMU	RF	N	PID
Owl MK USVs	V-hull	Under	GPS+IMU	RF	N	cluster-space
Seadoo Challenger 2000	V-hull	Under	GPS+IMU	radar	N	PID
Wave Glider	V-hull	Under	GPS+IMU+acoustic	N	N	Y, unclear
Springer	Catamaran	Under	GPS+IMU+speedlog	N	N	LQG
Blackfish	V-hull	Over	GPS+IMU	N	N	backstepping
Roaz II	Catamaran	Under	GPS+IMU	radar+camera	C&C	Y, unclear
Minion 2016[59]	Catamaran	Under	GPS+IMU	LiDAR+camera	Y, unclear	Y, unclear

N stands for none. Here, Perception and Planning refer to object detection and obstacle avoidance respectively.

As highlighted by Liu *et al.* the shape of the hull can affect an ASV's vulnerability to capsizing in rough water and its thruster configuration may determine whether it is underactuated or not which has strong implications for its maneuverability [33]. Besides the Blackfish, few USVs are fully or over-actuated, making them hard to maneuver in tight urban settings. Furthermore, most employ catamaran-shaped or V-hulls, making them fast and stable in ocean settings, but not easily sideways-maneuverable in urban settings. Here, I demonstrate a system that has superior maneuverability characteristics for narrow urban settings.

To safely navigate in urban waterways, a USV should localize itself with decimeter-level accuracy. Current USVs typically use GPS and IMU (fused by an extended Kalman filter (EKF) or unscented Kalman filter (UKF)) for meter-level precision [33]. These GPS-IMU-based approaches can be unstable in urban waterways, where GPS signals are often severely attenuated. Naeem *et al.* proposed a reliable multi-sensor navigation system which includes GPS, compass, speed log, and a depth sensor to account for sensor failure, but it cannot guarantee high accuracy [38]. To date, there is no feasible solution for accurate urban USV localization. Here, I propose a system which depends on LiDAR, camera, and IMU to provide a decimeter-level precision in GPS-attenuated dynamic urban waterways.

2.2 LiDAR-based Perception

2.2.1 Obstacle Detection and Tracking

In the context of perception for ASVs, existing studies mostly consider detection between cooperating vehicles [3, 57], while other environmental obstacles are commonly ignored expect for these several studies [20, 13, 66, 59]. For example, Heidarsson *et al.* [20] employed a profiling sonar in shallow water, but the low resolution of the single beam limits the obstacle detection accuracy[20]. Moreover, vision-based systems have also been used, but are usually designed for imprecisely detecting obstacles at more than 100 meters [66, 13]. In addition, Thompson *et al.* utilized a support vector machine (SVM) classifier on LiDAR pointclouds [66]. Nevertheless, moving obstacles cannot be classified due to limitations of their occupancy grid. Hence, I propose to use LiDAR as the main sensor and employ the Euclidean clustering and contour tracking algorithms to obtain reliable and accurate state estimation of static and moving obstacles.

2.2.2 Object Pose Estimation

In the broader context of perception, precisely determining the pose of a specific known object is a desirable property for many operations, in addition to being able to roughly detect any object. I extend this to precise pose-detection for ASVs. There are some recent methods that have gained traction in camera and LiDAR-based pose estimation, including signed distance function based optimizations to fit the model configuration to point-cloud data, QR tags to determine pose from an RGB data, and deep convolution neural networks to infer pose from RGB, RGB-D and point-cloud data [51, 41, 17, 12, 43]. In Dense Articulated Realtime Tracking(DART)[51], a Kalman filter is composed with a optimization-based measurement update that minimizes a signed distance function for observed object pointclouds straying outside the model's bodies. This framework reliably tracks articulated joints such as robotic arms and hands, but requires a manual initialization and relies on dense pointcloud data, which our VLP-16 cannot provide outside of a small lab environment. Apriltags are another popular framework for determining the pose of an object, by recognizing unique tags embedded on the object with a camera [41]. While such tags are simple to use, and reliable in favorable conditions, changes in lighting such as reduced visibility, low-light and orientation make them much less reliable. In recent years, deep convolutional neural networks, have also become popular for predicting pose [17, 12, 43]. While they can accurately detect pose over a wide range of object variations, they require significant amounts of training data and can't infer on or detect samples out of the training distribution. Finding concise input data representations that reduce this burden is still an open area of research.

As a preliminary test I attempted using iterative closest point (ICP), a classical approach that attempts to fit a segmented Roboat pointcloud to a model pointcloud. The fit was subpar, as shown in Fig. 2-1, because of the low density, water noise, and LiDAR vantage point induced nonuniform model-sampling inherent in the segmented Roboat pointcloud.

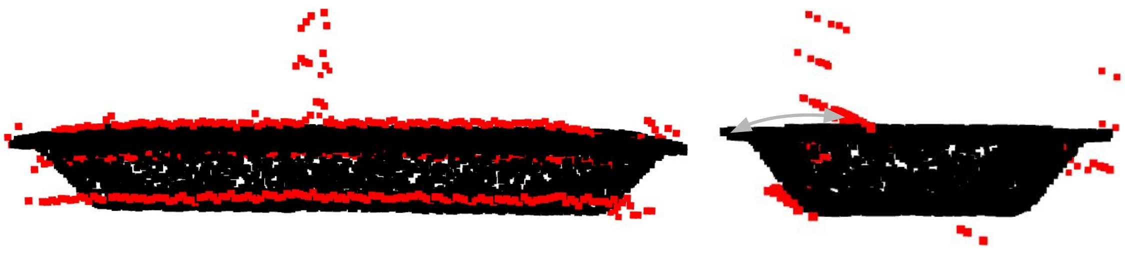

(a) Side View (b) Front View

Figure 2-1: ICP Pose Estimation: LiDAR measurement in red, Model in black. A grey arrow shows the difference between them, indicating ICP has a large bias.

Wasik *et al.* demonstrate [65] a simple, efficient and robust approach to detecting and tracking the position of circular robots in medium-density pointclouds that employs optimization-based circle-fitting, but is unable to to determine the orientation of robots. I adapt the clustering approach I use in obstacle detection for an even-simpler random sample consensus(RANSAC) circle-fitting of multiple features on a boat to determine a precise position and orientation.

2.3 Multi-Robot Trajectory Optimization

Motion planning is another area that is critical to advancing ASV autonomy. While in many instances agents or subsets of agents can be considered independently and standard planning approaches such as A*, dynamic programming (DP), sampling-based methods and trajectory optimization can be applied independently to each agent to obtain kinematically or dynamically feasible trajectories (eg. when flying a few quadrotors over a large field) as agents collaborate, operate more densely or the number of agents increases the light coupling between agents grows stronger and new methods need to be considered. This is the case for multi-ASV planning in tight urban environments.

2.3.1 Formation Changing

In the context of formation changing, when there are many agents operating closely together and the potential for collisions grows, robots cannot be considered independently. One approach to this coupling is to maintain decoupled plans, but replan based on undesired interactions. One algorithm taking such an approach is Conflict Based Search (CBS), which calculates individual plans but replans with added constraints when conflicts are predicted to obtain an overall solution [55]. While CBS performs well in situations with a few conflicts or bottlenecks, it is NP-hard and does not scale well to the prospect of many conflicts. Another strategy is to explicitly consider agents as one coupled system and plan over the entire system. This could be extremely costly, especially when using methods guaranteeing global optimality, such as DP or Mixed Integer Programming(MIP), as the number of state variables scales linearly, and the compute time exponentially with the number of agents [52]. Additionally, while MIP over a system of agents can handle a continuous state space, DP and CBS operate over a discretized state space. While practical optimal time formulations have also been found using network flow methods, discretization and growing computational intractability still underlay these approaches [67]. Given these considerations, trajectory optimization and sampling based planning methods, provide likely solutions to finding feasible and low-makespan paths as they operate over a continuous space and are able to provide locally optimal and asymptotically optimal plans, in the cases of sequential quadratic programming (SQP) and randomly expanding random trees star (RRT*) respectively[25].

SQP optimizations have been successfully used before to plan robust trajectories over decoupled agents [7]. I share a trajectory optimization based approach to minimum makespan formation planning for boats that takes into account the above considerations.

2.3.2 Shapeshifting

Another important motion planning opportunity for using optimization on the Roboat is shapeshifting, the task of reconfiguring the shape of Roboats latched together. While shapeshifting is an exciting area of research and there is significant research on modular robotics and shapeshifting with sheets, mobile robots, cubes and UAVs [19, 46, 45, 68], there is no prior research in shapeshifting on water. Ohara *et al.* present an algorithm for assembling shapes on water but provide no means of reassembling the structure nor dealing with localization-less boats [40]. Saldana *et al.* demonstrate a way to assemble and change the formation of boats in a decentralized manner, but demonstrate no experimental results and make similar assumptions about robot sensor homogeneity [50]. Thus, I endeavour to present reliable and experimentally validated algorithms generating shapeshifting trajectories on water, using a mixed integer programming optimization.

Other mixed integer approaches employ polynomial basis functions [9], requiring costly semi-definite constraints to contain the trajectory in obstacle-free regions, or optimize the placement of linear trajectory segment, requiring a large number of integer variables [37]. Our approach manages to use B-Splines in a MIP regime, using less variables to generate smooth paths and requiring no expensive SDP constraints.

Our formulation is similar to that employed by Usenko *et al.* for high-speed replanning of a quadrotor's trajectory. They use an unconstrained optimization approach that runs faster than our but does not guarantee a solution and has a much higher failure rate [62]. Flores and Milam cleverly formulate the trajectory generation in the presence of obstacles problem as a non-linear program, representing their trajectory using non-uniform rational b-splines (NURBS). NURBS generally allow for more expressive trajectories but in their case obstacle avoidance is guaranteed by fixing the placement of control points and optimizing over the weighting of their points [11]. In our case, a simpler representation is used and the control points are directly optimized over, not unnecessarily restricting the convex hulls of the path prior to the optimization. Additionally, I provide runtimes demonstrating the real-time nature of our system.

Other polynomial-based approaches exist but do not rigorously address collision avoidance [32, 6]. Liu *et al.* sample points along curves to guarantee collision avoidance for a sum of polynomial basis functions trajectory but are unable to guarantee such properties over the entire path. [6] Chen *et al.* provide a real-time solution search-based quadrotor trajectory planner with a similar parameterization, but iteratively trajectory checking and waypoint addition is needed to guarantee collision avoidance. Our solution guarantees obstacle avoidance directly from the MIQP and provides more optimal results with a SQP smoothing pass over the first result while also running in real-time.

Overall, the approaches to ASV system design, perception and multi-robot planning presented in the subsequent chapters advance the state of the art in autonomy for ASVs, expanding their applicability in urban settings and the broader horizons of what they are capable of achieving.

Chapter 3

Urban Autonomy System

To begin to tackle the challenges faced by ASVs operating in dense urban waterways, instead of coastal, marine or large inland waterways, one must start by taking a holistic look how an ASV's system design shapes its performance for an intended task. As highlighted by Liu *et al.* the shape of the hull can affect an ASV's vulnerability to capsizing in rough water, its thruster configuration may determine whether it is underactuated or not which has strong implications for its maneuverability, and its sensor choice may determine the reliability of localization in different environmental conditions [33]. In addition to these factors considered by Liu, there are also software architecture design factors, the choice of third party packages and configuration of purpose-built software modules mapped to their intended function in the system, that place similar constraints on system functionality and performance [56]. I look at these questions through the lens of operating ASVs in the Amsterdam canal environment for the use cases highlighted in the introduction. Towards the goal of creating such an ASV, we constructed an approximately 1/4 scale prototype of the Roboat.

This chapter provides an overview of the prototype platform I used in experiments and on which the algorithms described in subsequent chapters are deployed on, first addressing the hardware and then the software. The methodology followed and unique system design aspects will be highlighted in the corresponding hardware and software sections.

3.1 Hardware

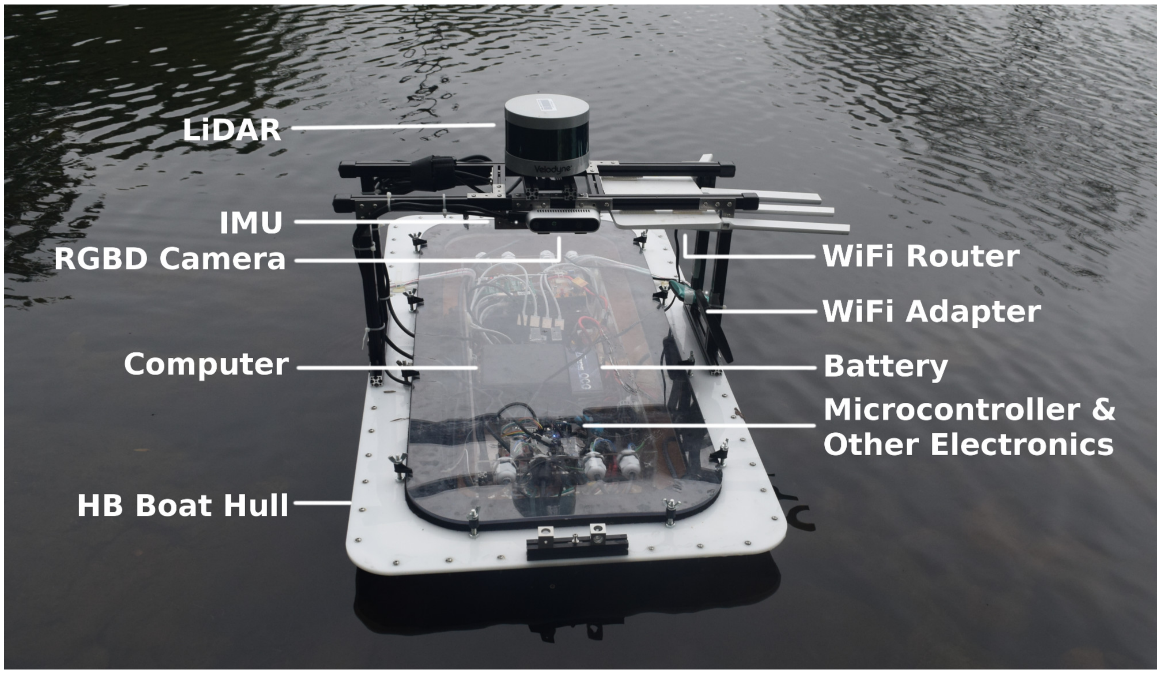
3.1.1 Hull and Actuators

The hull and thruster configuration on the Roboat, pictured in Fig. 3-1b, was designed with carrying capacity and maneuverability in mind. The full scale Roboat will ultimately transport 2 metric tons, in passengers or municipal waste. The prototype is similarly transporting a significant mass in components, given its size. Thus, a barge-like rectangular shape was chosen for maximal capacity. The rectangular shape is also easy to tile in the situation that multiple boats are latched side by side to make a floating platform [63]. Maneuverability is also important in urban scenarios. The curvature in the hull on all sides allows for relatively easy motion in all directions, as well as rotation, making the Roboat easy to navigate quickly in tight spaces [35]. The flat shape and uniform curvature would not be suitable for marine environments, as turbulent water would cause the boat to oscillate heavily. The Roboat though is operating in a calmer and shallower urban canal environment and this trade-off in favor of carrying capacity and maneuverability is well worth it [35, 14, 33].

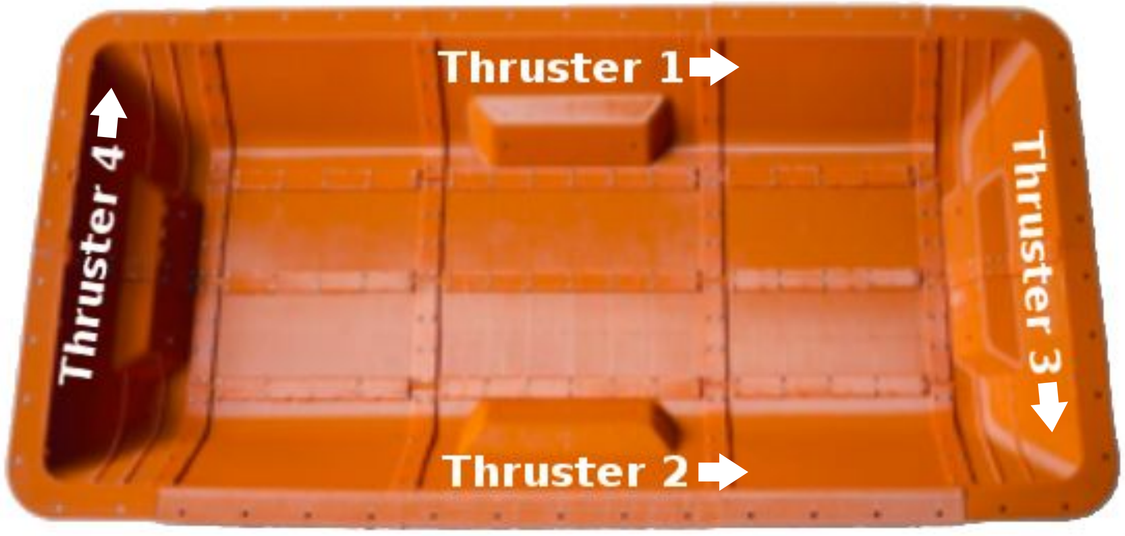
Similar considerations were made in choosing the thruster configuration. The Roboat is outfitted with 4 thrusters, attached to the hull in the directions indicated by the arrows at the mount points shown in Fig. 3-1b. The direction of thrusters 1 and 2 is perpendicular to that of thrusters 3 and 4. Some ASVs have all thrusters in the same direction, letting them move faster, but that significantly limits their maneuverability [8]. Following such a convention on the Roboat would be especially unwarranted as Amsterdam's canals have a low speed limit of 6 km/h. Our thruster configuration makes the Roboat fully actuated, allowing it rotate and accelerate its position in any direction, regardless of its orientation, on a water surface.

3.1.2 Sensors

The Roboat is outfitted with several sensors to maximize performance in relatively dense city environments with obstacles at a distance of centimeter to tens of meters.



(a) Roboat in the water with labeled components



(b) Roboat hull without other components. Thruster mount points are labeled.

Figure 3-1: Roboat Hardware Overview

While many marine ASVs rely primarily on GPS or RADAR for global positioning this would not work for the Roboat which needs to operate in the dense and potentially GPS denied environment of the city [33, 38]. Also sonar has a very low resolution for precise mapping in shallow water and urban environments as demonstrated here [20]. Additionally, most marine systems rely primarily on a mono or stereo camera for perception, but this can only provide low precision measurements at several meters [66, 13]. In an urban canal environment, where obstacles are much closer, sub-centimeter level precision is preferable. Instead, a unique combination of a sixteen beam LiDAR (Velodyne VLP-16) and IMU(Microstrain GX5-25) are primarily used to provide high precision localization and perception. The LiDAR measures the relative 3D position of obstructed points in the sensor's range.

As Liu mentions there is only minimal research into ranging sensor use on ASVs [33]. The IMU provides 3 axis angular velocity. Together, the LiDAR pose estimation and the IMU measurements are filtered in an EKF. From this an MPC controller, can use the localization to follow a trajectory. This EKF-MPC architecture is further described in our paper [64].

Cameras and GPSs are still useful as secondary sensors. An RGB-D camera (Intel Realsense D435) is mounted on the boat for higher resolution, but slightly lower precision, perception. This augments but does not replace the LiDAR's use for perception, which suffers from lower vertical resolution. Similarly, a GPS, may be added to augment the LiDAR's use for localization, which suffers in the open environments that are rarely encountered.

3.1.3 Electronics

The electronics were selected to optimize for real-time localization, perception, planning and control, while increasing the operating-time of the Roboat. A powerful yet efficient Intel NUC (Intel Core i7-8650U) with 8 threads, outfitted with 32 GB of memory is used as the main computer. A computer with 8 threads can process many processes in parallel,w which is important to our software system made up of many modules running concurrently. The 8 threads running at 1.9 GHz, along with the

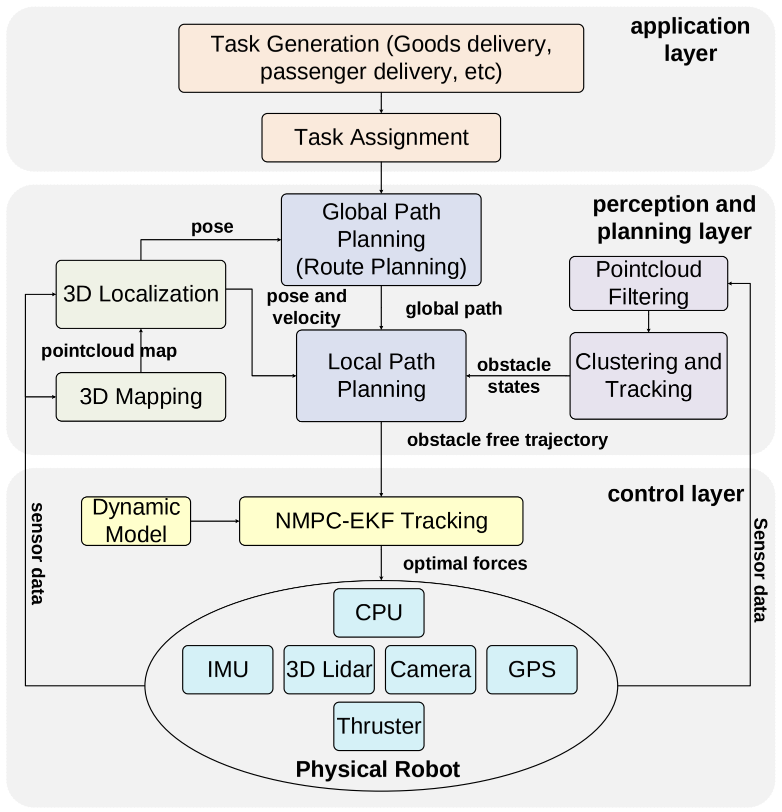
32GB of RAM are also a good fit for processing significant amounts of pointcloud data. Using a GPU to process the perception data would be even faster, but would consume 200-300 Watts compared to the less than 60 watts consumed by the NUC, thus causing the battery to deplete much more quickly.

In addition to the NUC, a 32-bit auxiliary processor, STM32F103, is used for converting the forces from the main controller to the corresponding actuator signals.

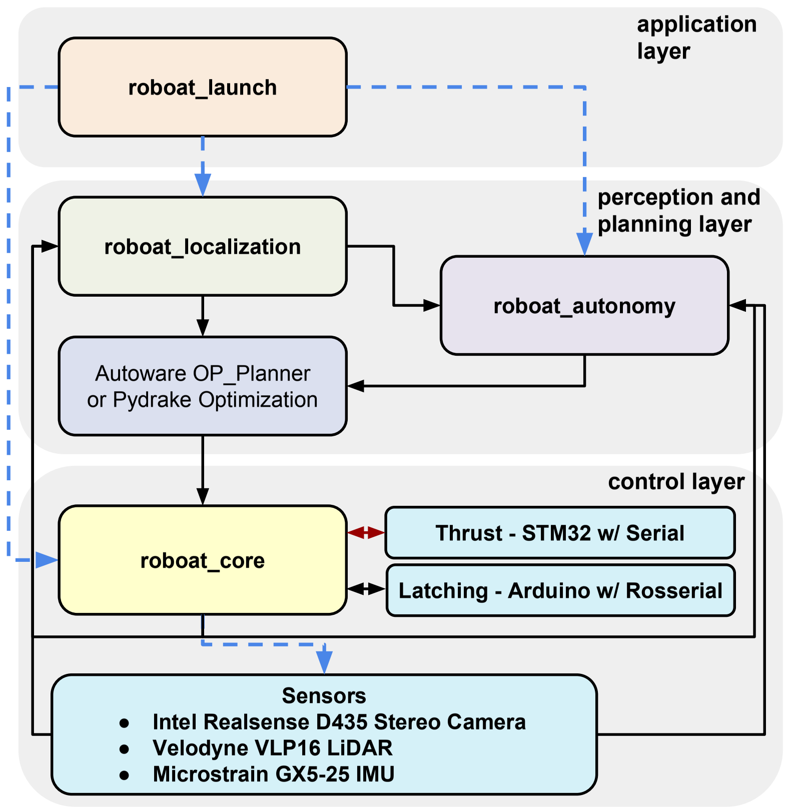
3.2 Software

I designed the Roboat's software with the widely used and performant Robotics Operating System (ROS) middleware, segmenting modules into ROS packages with a hierarchical structure [2]. The task of perception and planning in a dense urban environment with many obstacles, necessitates our system to perform more operations at a higher frequency than a typical ASV, closer to that of Autonomous Cars. Based on the 10Hz suggested rate of the primary sensor, the LiDAR, as well the response time of the physical boat to control inputs, I designed most components to run faster than 10Hz and the entire system to perform feedback control at 10Hz with minimal delay. The system described below runs on the onboard computer, which is accessible via SSH from a WiFi-connected laptop.

In architecting the software system for the Roboat, the reusability and clean mapping from functional components to their design in the implementation were prioritized, using the ROS best practices and tips for large projects. This includes the use of top level launch files for the application layer in the roboat_launch package, as well an additional top level launch file for each package. Arguments and parameter files can easily be changed to alter the flow of the launch files and ROS message topics, making the the code easily reusable. I also used existing third party libraries and packages where they fit our needs and added minimal complexity to the codebase. Where they did not, I built concise, reusable and easy to maintain software, or refactored existing related code to be more general, depending on ease [31, 1, 56].



(a) Functional System: Conceptually split int the application layer, perception and planning layer and control layer. Each layer contains many functional modules. This functional diagram is implemented by the implementation diagram on the right.



(b) Software Implementation: Blue arrows show components launching other components at the start of execution. roboat_launch, the Autoware planner, latching and thrust components are executed by the user. ROS topics generally match the flow of the black arrows. The flow of non-ROS message serial communication shown in red.

Figure 3-2: Roboat Autonomy System Diagram

3.2.1 Functional Layers

On a conceptual level, the operations of the autonomy system are split into three layers: the application layer, perception and planning layer and control layer. These layers are shown in Fig. 3-2a. First, the application layer contains the use case specific operations and manages the execution of the other layers. This layer determines the operational mode of the boat, whether the boat is carrying goods to a point or latching to another boat, and coordinates the execution of the other necessary components in the lower layers. Second, the perception and planning layer contains algorithms for processing sensor data, updating the state of the world, and planning trajectories and commands to be carried out by controllers in the even lower planning layer. Third, the control layer contains hardware components as well as algorithms for generating actuator commands from detailed plans.

3.2.2 Roboat Packages

The corresponding ROS package implementations are shown in 3-2b. Using ROS message topics, nodes within these packages can easily communicate with other nodes in the same or other packages. Note that there is a simple clean mapping from the functional layers to the packages implementing layer functionalities. [54, 56]. Although some functional sub-modules in these layers are implemented in the same ROS packages to combine functional sub-modules requiring similar dependencies (eg. 3D Localization and 3D Mapping are both implemented by roboat_localization), their underlying software class and function implementations, not shown in the figure, are cleanly separated. Listed here is a description of main the the packages developed for the Roboat and their use.

- **roboat_autonomy** filters LiDAR data based on the boat environment and task
- **roboat_core** manages sensors, low-level serial communication and actuator control
- **roboat_launch** coordinates the execution of a task setup amongst by calling other

packages with the relevant configuration

- **roboat_localization** localizes the boat with LiDAR, IMU and Camera data. It manages Autoware's NDT for LiDAR scan matching and employs robot localization for EKF filtering and rtabmap visual odometry
- **roboat_utils** contains utility scripts for commanding the boat with a joypad from a laptop, configuration files and an installer script for the Roboat setup

The packages can easily be used independently and stacked for more complex tasks. Simple tasks like trajectory tracking can be launched with roboat_launch and will only call roboat_core and roboat_localization. Similarly, LiDAR pointcloud filtering is accomplished by only launching roboat_core and roboat_autonomy with roboat_launch. With more complex tasks such as path planning with obstacle avoidance, requiring trajectory tracking, as well as pointcloud filtering and planning, roboat_launch can coordinate the execution of all the system packages.

3.2.3 Third Party Packages

In addition to the above purpose built packages, the Roboat software stack makes heavy use of the following excellent open source packages.

- **ACADO Toolkit** generates efficient MPC controllers in c for Roboat_core [22]
- **autoware** scan matches LiDAR pointcloud to map and plans paths [26]
- **drake** robotics toolbox with python bindings used for trajectory optimization
- **graph-tool** constructs and computes on large graphs [44] [58]
- **point cloud lib** manipulates and processes LiDAR data in Roboat autonomy [47]
- **shapely** manipulates and analyzes planar geometric objects [16]

A more extensive list of packages used can be found in appendix A.

3.2.4 System Usage

The system described in this chapter was used for all experiments in [63], [64] and [36], as well as those in the subsequent perception chapter. The experiments from the multi-robot trajectory optimization chapter were carried out on a pared down prototype multi-boat system architecture based on that used by Park *et al.* [49]. In the future, those functionalities too could be integrated into this architecture, giving the multi-boat setups additional perception and localization autonomy capabilities, for real-world multi-boat system deployment.

Chapter 4

LiDAR-based Perception

4.1 Obstacle Detection and Tracking

For an autonomous surface vehicle like the Roboat to safely navigate in areas with unmapped obstacles, obstacle detection and tracking of other vehicles and objects in the canal environment is crucial. This allows it to understand its environment so that it may appropriately plan and act.

4.1.1 Approach

Because of the narrow and crowded environments, navigating in urban waterways necessitates that both the object detection and tracking are reliable, which is typically not the case for ASVs in open waters. On the other hand, driverless cars always consider that objects like pedestrians and bicycles may enter their roadways from the outside. However, ASVs don't need to pay attention to this. Hence, I crop the waterway before representing obstacles. In particular, I use LiDAR to perceive the surroundings in three steps: cropping to filter out the irrelevant objects and noise, euclidean clustering to detect obstacles and contour tracking to track obstacles over time. These are shown in Fig. 4-1 and laid out in detail as algorithm 1.

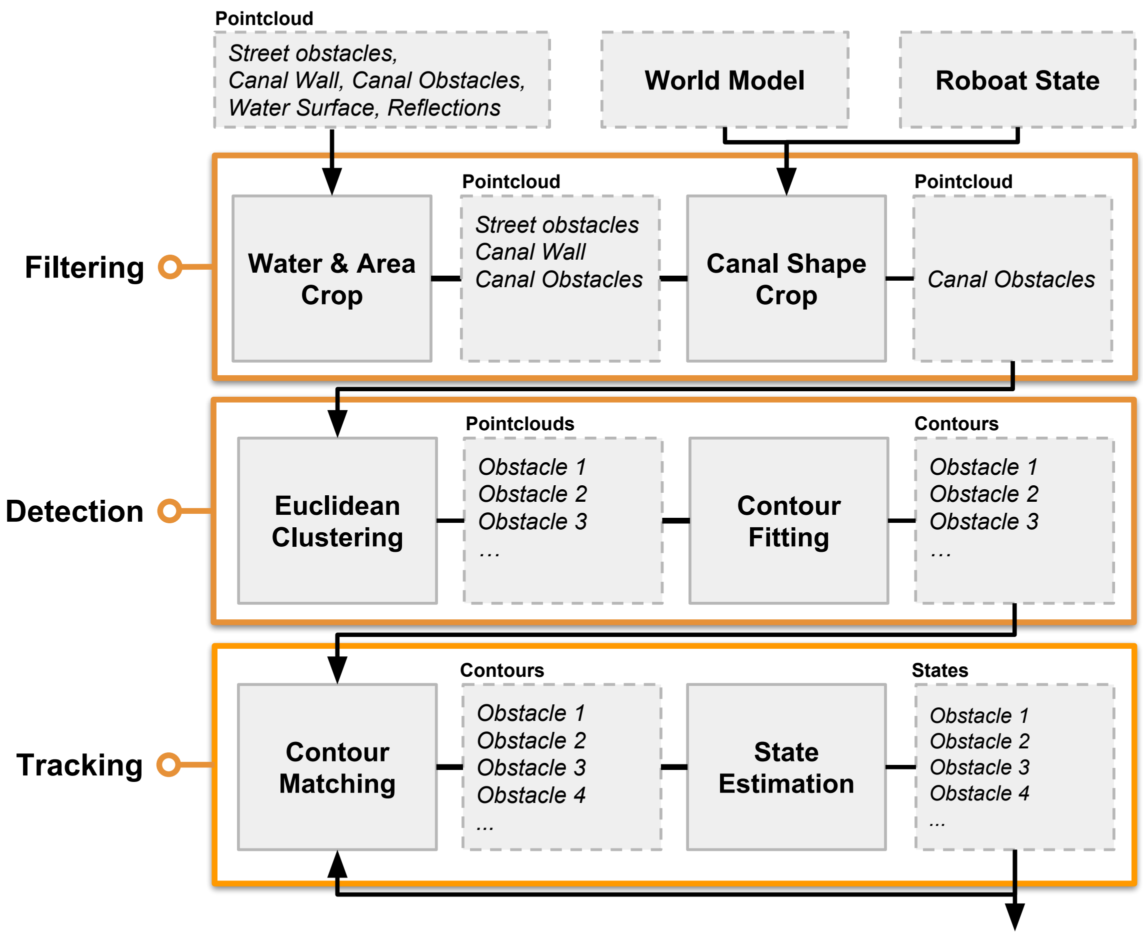

Pointcloud
Street obstacles, Canal Wall, Canal Obstacles, Water Surface, Reflections
World Model
Roboat State
Filtering
Water & Area Crop
Pointcloud
Street obstacles Canal Wall Canal Obstacles
Canal Shape Crop
Pointcloud
Canal Obstacles
Detection
Euclidean Clustering
Pointclouds
Obstacle 1 Obstacle 2 Obstacle 3 …
Contour Fitting
Contours
Obstacle 1 Obstacle 2 Obstacle 3 …
Tracking
Contour Matching
Contours
Obstacle 1 Obstacle 2 Obstacle 3 Obstacle 4 …
State Estimation
States
Obstacle 1 Obstacle 2 Obstacle 3 Obstacle 4 …

Figure 4-1: Roboat Perception Pipeline Diagram: the filtering, detection and tracking modules are used to obtain the real-time obstacle contours from the LiDAR pointcloud

Algorithm 1: Obstacle Detection and Tracking Algorithm

```
Input: pointcloud, contours
1  filteredPointcloud = [];
2  for p in pointcloud do
3      if $|p.x|<\Delta$ and $|p.y|<\Delta$ and $z_{th}<p.z$ and inCanal(p) then
4          filteredPointCloud.append(p);
5      end
6  end
7  clusters = cluster(filteredPointcloud, $c_{\text{th}}$, $c_{\text{min}}$);
8  for cluster in clusters do
9      contour = polygonContour(cluster);
10     for oldContour in contours do
11         if not oldCountour.matched and match(oldCountour, contour, $\iota_{th}$, $\kappa_{th}$)
             then
12             oldContour.update(contour);
13             break;
14         end
15     end
16     if not contour.matched then  contours.append(contour) ;
17 end
18 for contour in contours do
19     if time-contour.updateTime $> T_{th}$ then  contours.remove(contour) ;
20     contour.matched = False
21 end
22 ;
```

Crop Filtering

Given an input LiDAR pointcloud P_{in}, I use cropping filters to constrain the detection space within the waterway in lines 2-6 of the algorithm. This makes the obstacles I

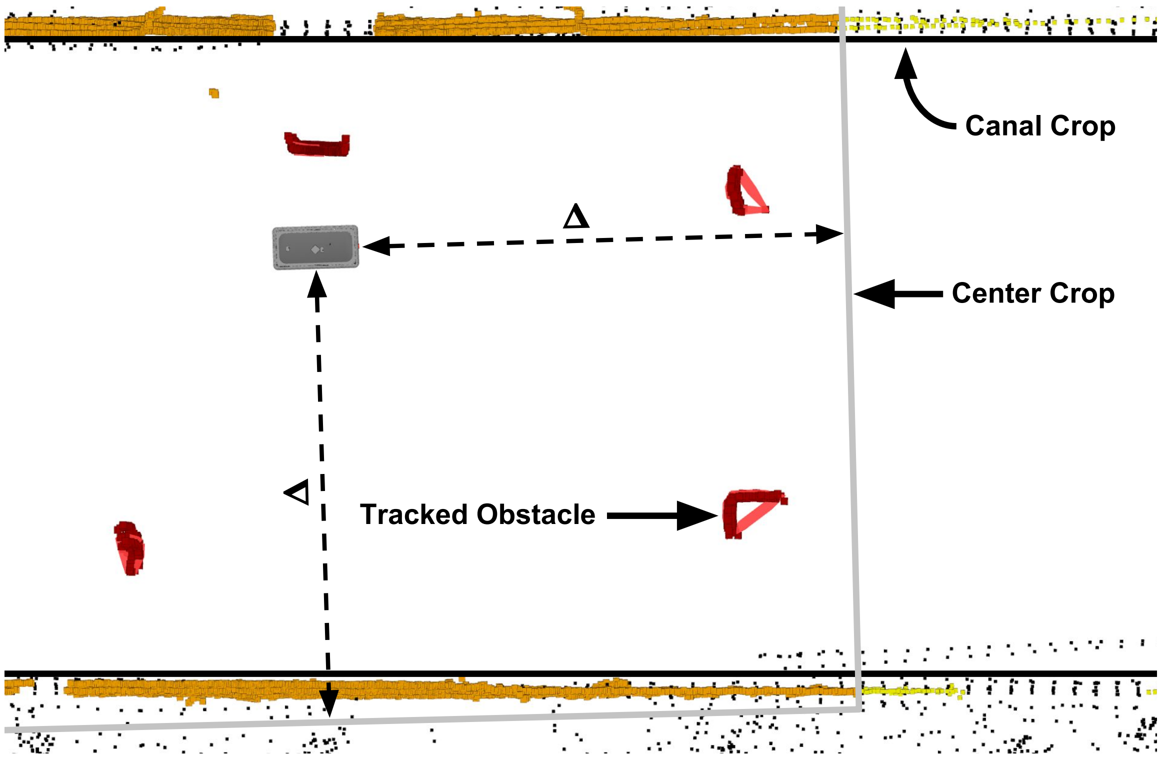


Figure 4-2: Pointcloud Filtering: Sequential crop filters applied prior to Euclidean clustering (top down view). Pointcloud map in grey, measured points in yellow, points also in 2Δ cropping area in orange and points also in waterway in red.

detect more relevant and reduces the computational load on the system. As shown in Fig 4-2, I first crop P_{in} to a size of $2\Delta \times 2\Delta$ in the $x - y$ plane and apply a minimum threshold z_{th} to the z-axis to obtain P_{crop}, centered on the boat-fixed frame. It should be noted that the z_{th} threshold is important because the ground filters used by autonomous cars are not usable on the non-uniformly registered water texture. Afterwards, I transform P_{crop} from a body-fixed frame to an earth-fixed frame with the transformation matrix $\boldsymbol{T}$, yielding $\tilde{P}_{\text{crop}}$. Last, I filter $\tilde{P}_{\text{crop}}$ based on the waterway edges to obtain candidate points $\tilde{P}_{\text{out}}$ lying within the boundaries. Thus, I only retain points belonging to proximal obstacles.

Pointcloud Clustering

To identify obstacles present in $\tilde{P}_{\text{out}}$ in line 7 of the algorithm I employ the Euclidean clustering implementation in Point Cloud Lib (PCL), an effective and computationally efficient density based clustering algorithm by Rusu [48]. It represents $\tilde{PC}_{out}$ as a kd-tree and iteratively associates a point with other points within a threshold L_2 distance d_{th} to obtain cluster candidates. Cluster candidates containing more than a threshold number of points c_{min} are labeled as clusters. Clusters with centroids within a threshold L_2 inter-cluster distance c_{th} are merged.

I tuned the clustering radius d_{th}, minimum cluster size c_{min} and inter-cluster merging distance c_{th}. I set d_{th} relatively high because of the low density of VLP16 LiDAR points. I use a relatively large c_{th} to cluster non-convex obstacles such as kayakers as one. I discriminatively set c_{min} to enable clustering small obstacles, while filtering out noise from waves. Considering these factors in our parameter choice enables our ASV to cluster obstacles on the water of varying sizes, even in choppy conditions.

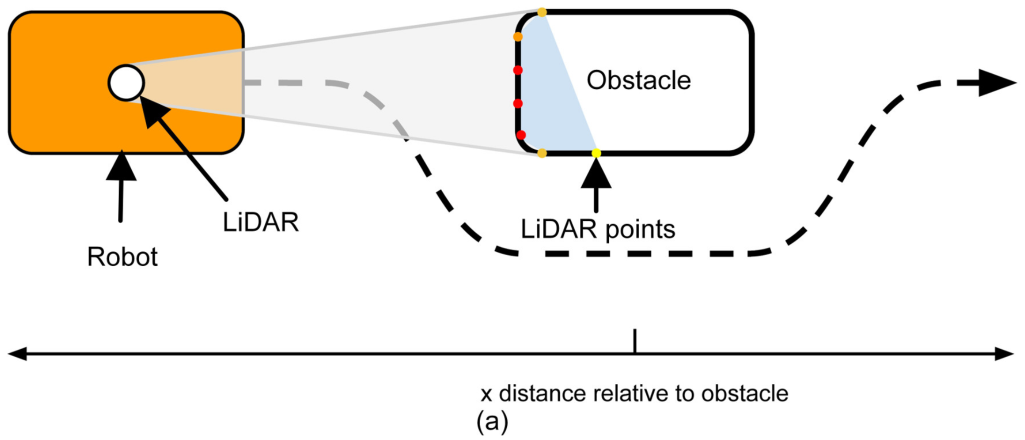
Obstacle Tracking

I use a contour tracker adapted from that proposed here [26] in lines 8-21 of the algorithm, representing the convex hull of each cluster with a polygon in the $x - y$ plane. Each obstacle is given a unique ID and tracked over time. An obstacle in one time step is matched to one from the previous time step if the centroid moves less than ι_{th} and the detected area changes by no more than a factor of κ_{th}. Lastly, an obstacle is remembered for T_{th} at its last reported location, if it is not observed, and remove after that time. This allows for robust tracking, even in the most difficult and noisy detection environments, and enables safe planning when passing close to

Table 4.1: Parameter Values Used in Obstacle Detection and Tracking

Condition	Parameters							
	Δ	z_{th}	d_{th}	c_{th}	c_{min}	ι_{th}	κ_{th}	T_{th}
Indoor	5.0	-0.4	0.5	1.5	20	1.0	3.0	5
Outdoor	10.0	-0.4	0.5	1.5	40	1.5	3.0	10

Note: Δ, z_{th}, d_{th} and c_{th} in meters. T_{th} in seconds. ι_{th}, κ_{th} and c_{min} no units.



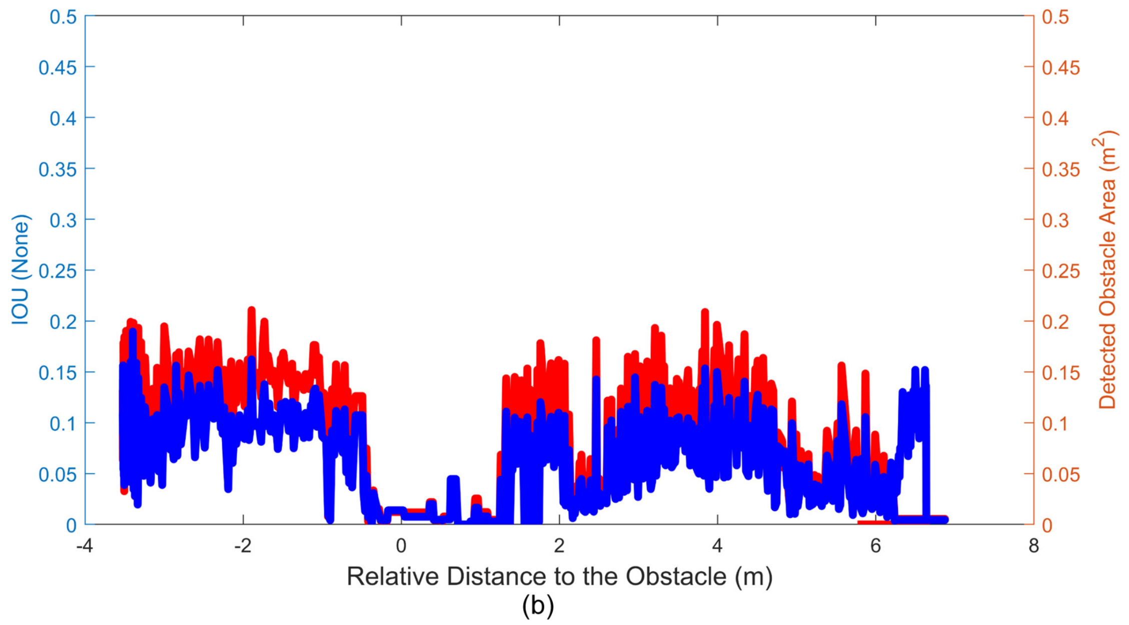


Figure 4-3: Detection IoU Values: results for Roboat moving along an obstacle-free path: Path (a) is followed during the test yielding the IoU values in (b) along it

obstacles in urban waterways.

4.1.2 Experimental Evaluation

I carried out experiments along the path in Fig. 4-3(a) to determine the accuracy of tracked contours at different x distances relative to an obstacle. The boat system in Chapter 3 was used during these experiments. The accuracies in Fig. 4-3(b) are measured in terms of the intersection over union (IoU) or Jaccard Index, using the detected contours and the ground truth contour. From Fig. 4-3(b), it is visible that the robot consistently detects the obstacle (IoU > 0), except for the blind zone of the LiDAR when the robot is too close to the object. In these regions the last detected contour is used by the planner. The noisy IoUs indicate that the swaying of the robot, caused by the water disturbances, affects sensor visibility a lot across different object surfaces.

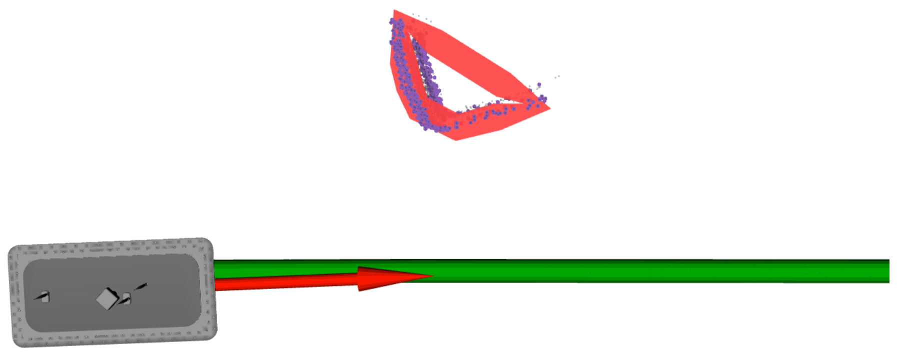


Figure 4-4: Roboat Visualization: Roboat following path in green. Detected obstacle pointcloud shown in purple with resulting tracked obstacle countour in red.

Taking a look at the position of the Roboat, clustered points and obstacle contour from one time step along the path in Fig. 4-3, it is visible that the contour well represents the contour of the relevant pointcloud cluster in this time step. It does not, however, well match the contour of the box obstacle. To alleviate these effects, aggregating obstacle pointclouds and contours from different vantage points across time, using scan-matching, might help. This will allow us to obtain a more consistent contour with better IoU. Additionally, detecting obstacles based on a library of candidate shape priors might help the tracked contours more closely match detected obstacle shapes. Overall, our solution for obstacle detection and tracking is effective because it always consistently detects and tracks obstacles in real-time. This opens the way for transplanting ASVs from open water to urban waterway applications.

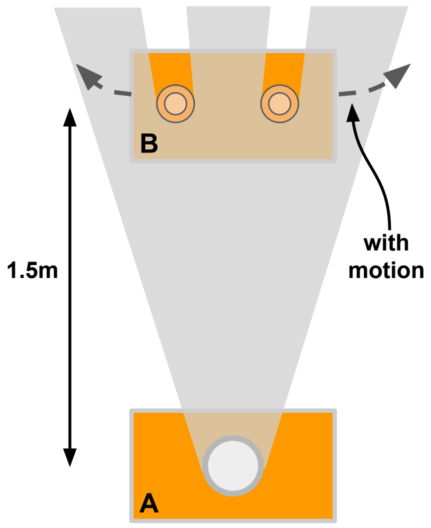
4.2 Object Pose Estimation

While for many objectives such as obstacle avoidance, detection and tracking will suffice, more complicated objectives such as connecting with another robot or planning through contact require a more precise estimate of an other robot's state to be reliably completed. Here, I present a revision of the perception pipeline for determining the pose of another Roboat, which is closely related to the obstacle avoidance task in the previous section.

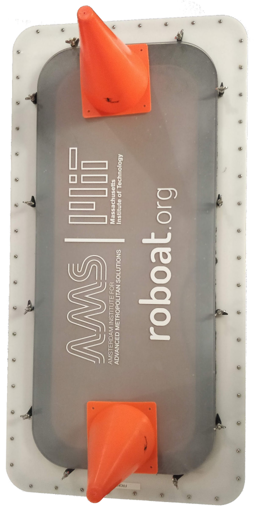
In the obstacle avoidance scenario, I made no assumptions about what kinds of objects could be encountered and obtained position estimates with a relatively simple algorithm and model with few parameters. All I desired was a quick position estimate with decimeter precision for obstacle avoidance planning. Obtaining a more precise estimate is difficult, unless I make assumptions about the objects I will encounter or create a more complicated model. In the latching scenario for which knowing the pose of another Roboat is desired, I can limit the scope to just obtaining a precise pose for other Roboats, thus simplifying the task.

4.2.1 Approach

In Fig. 4-5a Roboat A is sensing Roboat B with a LiDAR and determining its pose based on the detection of a feature in the pointcloud, 2 cones on Roboat B, as showing in 4-5b. By detecting two cones on boat with two cones, and precisely knowing their position, a reliable pose for that boat can be inferred. This is the principle behind the RANSAC method I am using. Similar to the obstacle avoidance scenario, the perception pipeline for Roboat A, is broken down into three components: Filtering, Detection and Tracking.



(a) Experiment Setup



(b) Roboat B With Cones

Figure 4-5: Object Pose Estimation: boat A with LiDAR detects boat B.

Filtering and Euclidean Clustering

Just as for obstacle detection and tracking, filtering and clustering are required to focus on relevant areas of the input point-cloud and segment relevant objects. A similar water and area crop, as well as canal shape crop are used to reduce the pointcloud to objects contained in the canal or pool area. In addition to that, the pointcloud is cropped to a limited number of beams to limit the scope to the relevant cone height, and improve the clustering speed.

Detection

A similar clustering approach is used as in the previous section, but RANSAC is used to fit circles to the cones and determine their position. For clustering, euclidean clustering was also used here but different model parameters shown in table 4.2 were employed to cluster small cones, instead of obstacles of variable size.

RANSAC is a popular method whereby data points are randomly sampled to fit a model, and the fitted model is accepted if it reliably fits a large fraction of the unsampled points. The RANSAC function in PCL was used to fit a circle to points belonging to a candidate cone cluster. A minimum radius, r_min, and maximum

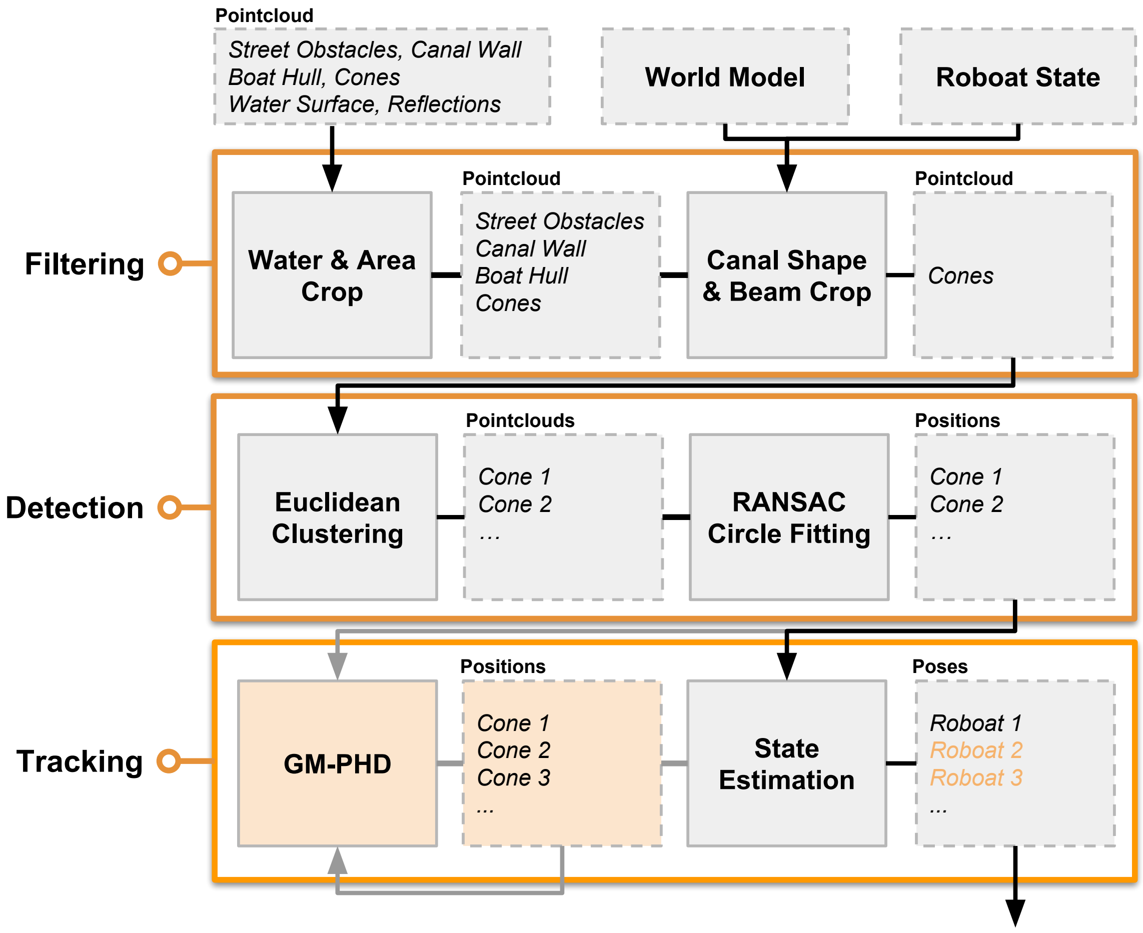


Figure 4-6: Roboat Pose Estimation Perception Pipeline Diagram: the filtering, detection and tracking modules are used to obtain real-time object pose estimates from the LiDAR pointcloud. Future components shown in light orange.

radius, r_max, are set for the circle. A threshold $d2_{th}$ is used for choosing inliers. Other methods such as fitting a cone, instead of a circle were also tried but yielded noisy and error-prone results due to the increased larger of model parameters relative to the low number of sample points, a similar issue to that experienced by ICP, which DART and could potentially also be prone to. As shown in Fig. 4-7a this method yields precise results for the center positions of the two cones on the boat, despite the sparsity of the pointcloud around the cones.

Tracking

As of now the two cone detections are averaged to yield the center of the Boat, and the vector between them is used to determine the orientation of the Boat. This approach works for one boat, but would need to be modified for multiple boats. Additionally, tracking would be needed, to expand this to detecting cones over multiple boats or with occlusion between frames changes. Using the contour-based matching from the previous section would not provide the results needed for precise pose tracking.

Table 4.2: Parameter Values Used in Object Pose Estimation

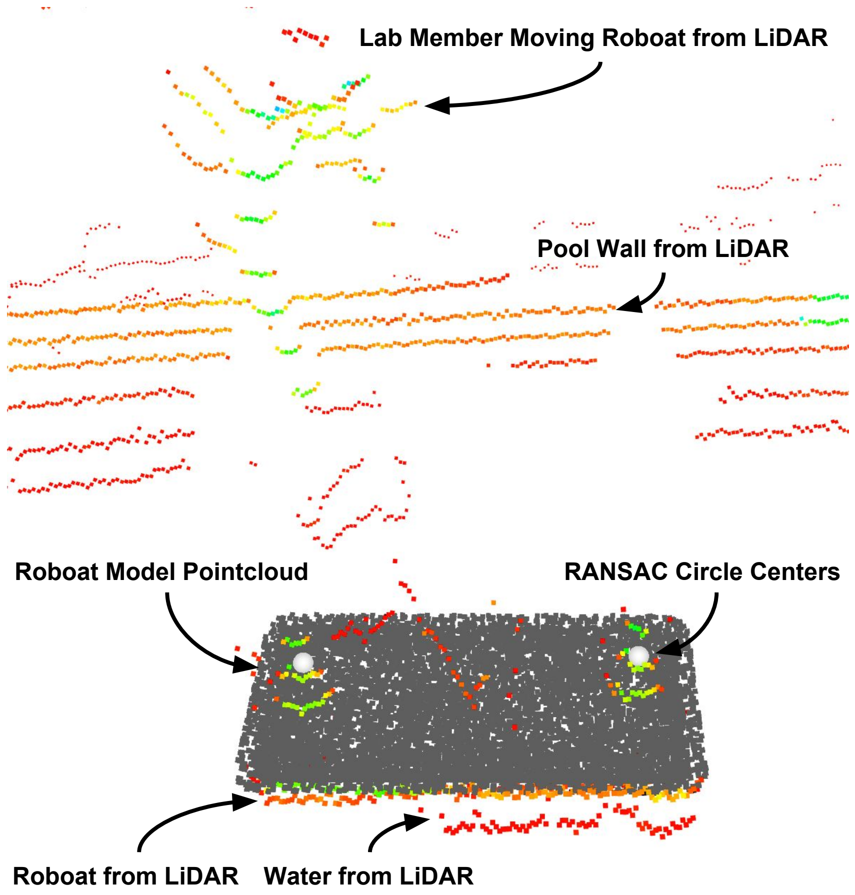
Δ	z_{th}	d_{th}	c_{th}	c_{min}	c_{max}	r_{min}	r_{max}	$d2_{\text{th}}$
5.0	0.03	0.05	0	20	40	0.025	0.08	0.04

Note: Δ, z_{th}, d_{th}, c_{th}, r_{min} r_{max} and $d2_{\text{th}}$ in meters. c_{min} and c_{min} no units.

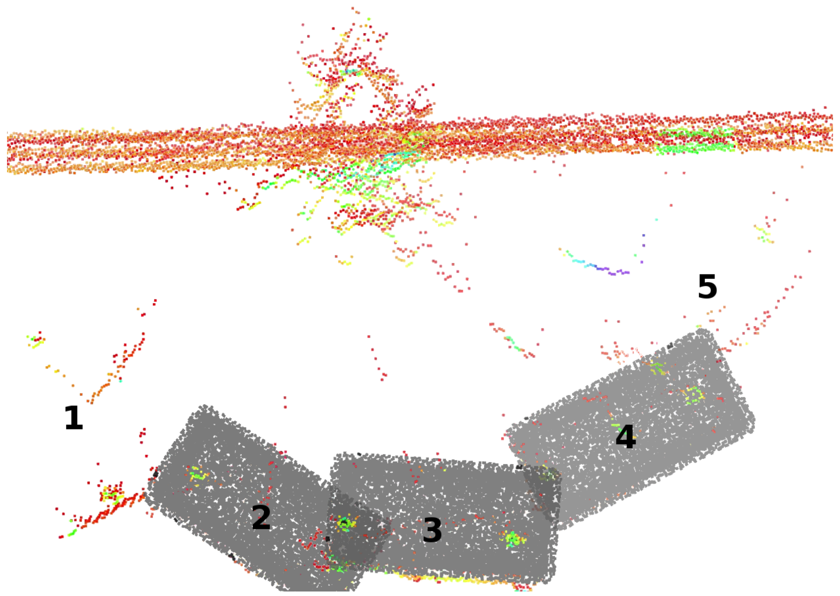
4.2.2 Experimental Evaluation

Two experiments were tried with the configuration shown in Fig. 4-5a. In these experiments, Roboat A is equipped with the autonomous setup described in chapter 3 and observes Roboat B with a LiDAR. Roboat B is simply a handheld hull with cones. Roboat B is static in experiment one and moves along an arc in experiment two.

The resulting fits at the static point and along the arc can be observed in 4-7. For both the static and moving cases, the detected results show a model fitting well with the LiDAR points. the Roboat edges line up well with the colored contours in the



(a) Static Roboat: visible features in the pointcloud, circle centers fit to cones, and the resulting Roboat model fit are labeled.



(b) Moving Roboat: accumulated pointcloud and Roboat model fits at different time points while Roboat moves along an arc.

Figure 4-7: RANSAC Circle Fitting-based Pose Estimation Results

Table 4.3: Number of Successful Pose Estimation Steps on All Pointclouds

Experiment	Pointclouds		
	All	Successful Clustering	Successful RANSAC
Static	321	320	232
Moving	871	334	143

Note: success indicates operation produced a result for both feature cones

pointcloud and the cone on the pointcloud well matches the relative position of the cone on the model. The difference in results is noticeable however, at time points 1 and 5 in the moving case where there is no model fit. In those situations the density of the pointcloud at the cone or the occlusion of one or more of the cones by the others precluded matching the cones to the model. Overall, out of all the pointclouds for the first experiment both cones were clustered in 320 instances and fit in 232 instances. Out of all the pointclouds for the second experiment both cones were clustered in 334 instances and fit in 334. In the cases where there are sufficient clusters, a lower quality pose can still be obtained if the fit is not within the 4cm threshold for $d2_{th}$. What the plots demonstrates is that my method and representation can provide precise results for detecting the pose of another Roboat, but have difficulty with consistent results in occlusion-causing configuration extrema brought on by motion. Possible amends for these factors, including tracking are discussed below.

To further the results demonstrated here to more boats, and improve robustness to occlusions, tracking on multiple levels and combining RANSAC with other methods will be explored in future work. A limitation of the current system is the inability to deal with a situation where there are multiple other boats as the current method used two cones to determine another Roboat's center and orientation. An expectation maximization algorithm to estimate the correspondence of cones to boats, similar to [23], would be an interesting approach to extend to the challenging task of on-water precise pose estimation in future work.

Another difficulty the current system encounters is its vulnerability to occlusion. A potential limitation, as discussed above, which could be causing this is the lack of tracking for the cones. In the future a filtering approach such as a GM-PHD filter could be employed to reliably track a varying number of candidate cones, as opposed to an EKF which would require fixing the number of cones and continually observing them. Additionally, given an initial guess from RANSAC and tracking, and further filtering of a pointcloud's outliers, given this initial guess, it might be feasible to use ICP or DART to yield a more precise fit in the context of occlusion.

The last current difficulty, is detection from a slight distance where pointcloud density around the cones drops, in these contexts a more nuanced model that is able to distinguish cones with less points could be suitable. Given preliminary segmentation using euclidean clustering, and the limited amount of data for encoding a cone, shallow convolution neural networks could be a good candidate for yielding a small yet more refined cone detection model and would be an interesting area to explore.

Chapter 5

Multi-Robot Trajectory Optimization

5.1 Formation Changing

An important aspect of planning in multi-robot systems is coordination. In formation movement, multiple robots coordinate to move together at a certain orientation or distance from each other. A challenging aspect of moving in formations is handling situations where the formation has to change. In this section, I present a formulation for executing such formation changes with ASVs such as Roboats.

5.1.1 Approach

First, an assignment from agents to goals is calculated, which I find using the Hungarian algorithm. The assignment method is derived from work done by Turpin, Michael and Kumar which demonstrates a provably optimal assignment and plan for holonomic agents. The plan is able to achieve an instantaneous maximum velocity for a mean squared distance cost, given sufficient clearance between all initial and goal positions [60]. While these conditions do not hold for our agents, trajectory optimization surmounts these difficulties and shows good results nonetheless.

Second, as a crucial step, an initial trajectory consisting of states and velocities for each agent is calculated to drastically improve compute time for trajectory optimization. Two methods are explored: a linear interpolation and a novel shape-based interpolation. In the realm of formation control and planning, shape has often been used as a reference for designing and stabilizing reference trajectories over relative positions of agents or the distances between them [39]. Here, for the shape-based interpolation, an approximate transformation from the end to start formation frame is found and used as a reference for assignment and linear interpolation of a trajectory initialization.

Third, to find a final trajectory, an SQP is executed starting from the initialized trajectory. This combination of techniques performs robustly when finding trajectories for dense formation change and takes under two minutes in most cases, which I demonstrate on up to 9 boats in simulation.

In subsection 2, the I describe the assignment scheme in detail. In subsection 3, I provide an overview of the two trajectory initializations provided to the trajectory optimizer in different plans, as well as adjustments to the assignment scheme that were made in the context of shape-based initialization. In subsection 4, I discuss the system dynamics and the trajectory optimization that I use to generate trajectories. In subsection 5, I show the robust simulation results my methodology produces across a range of formation changes. In subsection 6, I show experimental results.

5.1.2 Goal Assignment

To find a trajectory which moves an unlabeled formation of boats from an initial formation to a final formation in a timely fashion, I formulate the problem as finding the minimum makespan, the minimum time required for the final formation to be complete. As such, the problem is not well formulated for a nonlinear program, as the final formation constraints are ill defined without a clear assignment of boats to goal position in the final formation (Figure 5-1).

$$\mathbf{X}_i^N = \mathbf{G}_j \iff \boldsymbol{\phi}_{i,j} = 1 \tag{5.1}$$

$$\mathbf{X}_i^N = \mathbf{X}_j^N \implies i = j \tag{5.2}$$

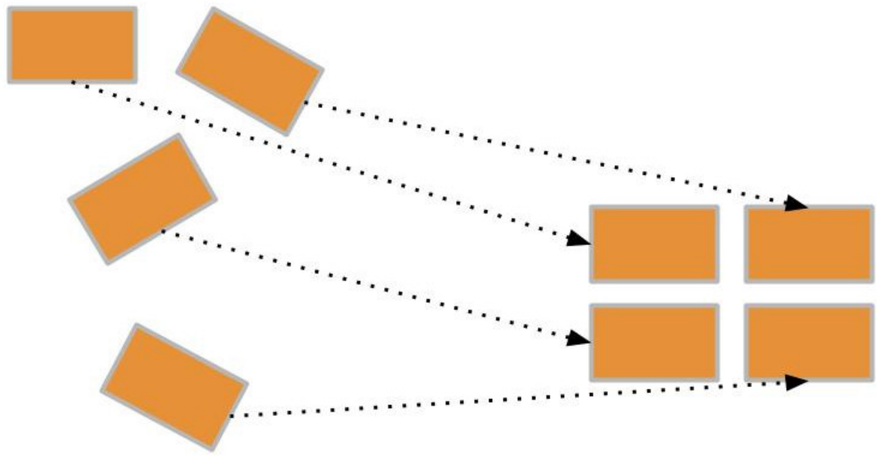


Figure 5-1: Assignment: matching initial and final boat configurations

With $\mathbf{X}_i^N$ representing the state of boat i at time N, let's use ϕ to represent an assignment of boats to goals G such that no two boats share the same final position (5.1) and (5.2). For this, I employ the Hungarian algorithm, an $O(n^3)$ algorithm which minimizes the following cost over $\boldsymbol{\phi}$ given costs $\mathbf{C}_{i,j}$.

$$\min_{\boldsymbol{\phi}} \sum_{i,j} \boldsymbol{\phi}_{i,j} \mathbf{C}_{i,j} \tag{5.3}$$

As demonstrated by Turpin, Michael and Kumar, minimizing the cost of the mean square distance between start and end position assignments can directly provide constant velocity, collision free trajectory by interpolating the position for holonomic robots with radius R if all positions in $\mathbf{X}^0$ and $\mathbf{X}^N$ are at least $2\sqrt{2}R$ apart [60]. This does not hold for all possible boat formations and formation changes, but it has the potential to provide a good initialization for finding a collision free trajectory.

While using a second order norm is possible, in a later paper Turpin *et al.* demonstrate that by using a higher order norm, the cost approaches the minimum makespan assignment, albeit providing collision free trajectories with additional steps [61].

$$\mathbf{C}_{i,j} = ||\mathbf{G}_j - \mathbf{X}_i^N||_{50} \tag{5.4}$$

As an optimization is later performed on these trajectories, this is not a worry. As such, the cost function above is used (5.4).

5.1.3 Trajectory Initialization

To calculate initial trajectories, two methods were considered: linear interpolation and shape based interpolation.

Linear Initialization

In the case of linear interpolation, the initial and final formations are interpolated over to determine trajectory initialization (5.5) and obtain linear trajectories as shown in Fig. 5-2. $\mathbf{X}^1$ and $\mathbf{X}^{N-1}$ are equal to $\mathbf{X}^0$ and $\mathbf{X}^N$, respectively, to allow the boat to accelerate and decelerate in a compatible way according to the Euler method state integrals used for the trajectory optimization. Given the assignment from above and minimal collisions, this proximity to dynamically feasible trajectory places the initialization in a relatively convex space close to minima, making it a good candidate.

$$\mathbf{X}^i = \frac{(i-1)\mathbf{X}^N + (N-i-1)\mathbf{X}^i)}{N-2} \quad \forall i \in [1, N-1] \tag{5.5}$$

$$\dot{\mathbf{X}}^i = \frac{\mathbf{X}^{i+1} - \mathbf{X}^i}{\Delta t} \quad \forall i \in [0, N-1] \tag{5.6}$$

I determine velocity for a state in a way similar to how it is defined in the Euler method state transition constraints, by calculating a constant velocity between the present and subsequent position (5.6). Lastly, Δt was initialized to 0.5.

Shape-based Initialization

In the case of shape-based initialization, the goal is to get the starting formation into the final formation's shape and then move it to the final position as shown in Fig.

5-2. Drawing from formation strategies, the goal here is to bring the robots toward a shape, after which a trajectory can be robustly followed without collisions. While this trajectory is not explicitly used, the goal is to produce highly feasible trajectories and initialize trajectory optimization near desired convex minima.

$$\mathbf{x}^N = \frac{\sum_i \mathbf{X}_i^N}{K} \tag{5.7}$$

$$\mathbf{X}^S = R\dot{(}\mathbf{X}^N - \mathbf{x}^N) + \mathbf{T} + \mathbf{x}^N \tag{5.8}$$

I first determine an approximate offset $\mathbf{T}$ and rotation $\mathbf{R}$ between $\mathbf{X}^N$ and $\mathbf{X}^0$ frames with the mean position of boats in $\mathbf{X}^N$ subtracted, where K is the number of boats (5.7). This subtraction helps with finding a rotation about the center of $\mathbf{X}^N$ and a translation of this center, resulting in a shorter initialization trajectory. The calculations of $\mathbf{T}$ and $\mathbf{R}$ are performed with 10 iterations of the iterative closest point (ICP) algorithm [4]. Using this initialization method, the assignment cost in (5.4) is computed between the start position X^0 of the boat and end shape transformed to the start position $\mathbf{X}^S$, where S is 50 for all my plans (5.8).

$$\mathbf{X}^i = \frac{(i-1)\mathbf{X}^S + (S-i-1)\mathbf{X}^i)}{S-1} \quad \forall i \in [1, S] \tag{5.9}$$

$$\mathbf{X}^i = \frac{(i-S)\mathbf{X}^N + (N-i-1)\mathbf{X}^S)}{N-S-1} \quad \forall i \in [S, N-1] \tag{5.10}$$

Paralleling the linear trajectory initialization approach, I also use linear interpolation for the shape-based initialization, but this is performed between X^0, X^S and X^N (5.9) and (5.10). Velocity is determined in the same way as the linear interpolation case (5.6).

5.1.4 Nonlinear Programming

I obtain trajectories by taking one of the initializations and performing a direct transcription trajectory optimization. The state transition, input magnitude and collision

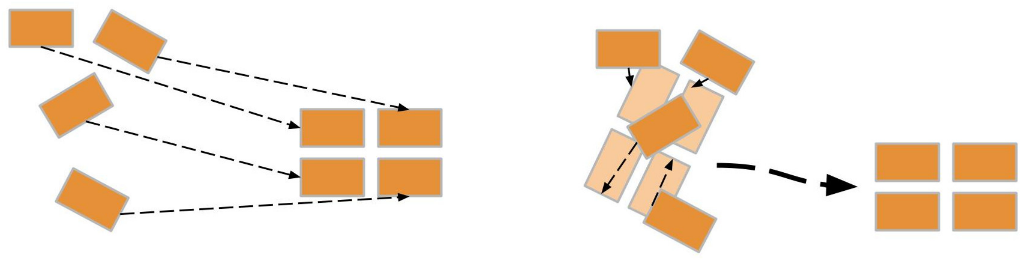

(a) linear interpolation (b) shape-based interpolation

Figure 5-2: Different Trajectory Initialization Strategies for Optimization

avoidance constraints, as well as a min Δt objective, are framed as an SQP and solved using the drake python framework with the SNOPT solver [27, 15]. In all optimizations, the number of knot points was $N + 1$ for $N = 100$.

State Transition Constraints

I follow the notation developed by Fossen for marine vehicles and employed by Wang *et al.* [12, 63]. In the model used here, $\mathbf{V}_b^i$, the velocity of each boat b at time i in the boat frame, is represented using a manipulator equation where $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{D}$ are the mass, coriolis and drag matrices respectively (5.11).

$$\mathbf{M}\dot{\mathbf{V}}_b^i + \mathbf{C}_b(\mathbf{V}_b^i)\mathbf{V}_b^i + \mathbf{D}\mathbf{V}_b^i)\mathbf{V}_b^i = \mathbf{B}\mathbf{U}_b^i \tag{5.11}$$

Given the state $\mathbf{X}_b^i = [x\ y\ \psi]^T$, a transformation matrix $\mathbf{R}(\psi)$ from the boat body frame to the inertial frame can be computed (5.12).

$$\mathbf{R}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.12}$$

This in turn allows me to represent the state derivative $\dot{\mathbf{X}}_b^i$ corresponding to the

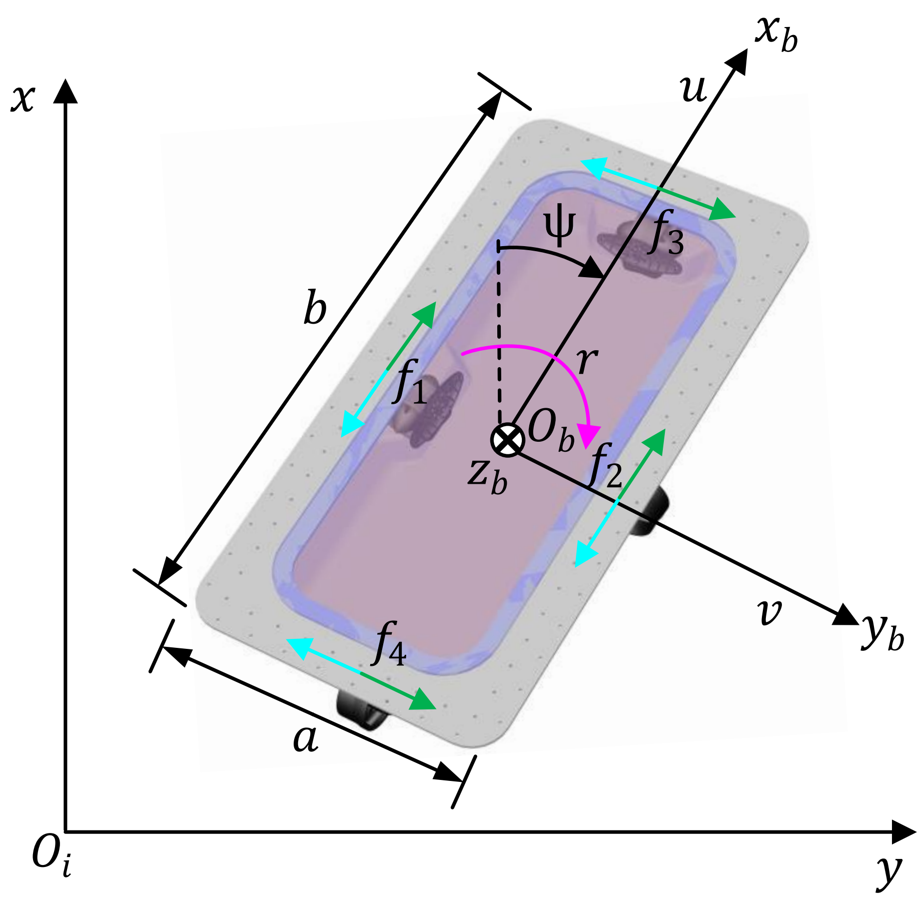


Figure 5-3: Oblique View of Autonomous Boat. I use two coordinate systems: inertial coordinates $\sum_i$, O_i-xy and body-fixed coordinates $\sum_b$, O_b-$x_b y_b$. Green arrows stand for positive force and blue arrows stand for negative force. [63]

body frame velocity $\mathbf{V}_b^i$, thus leading to (5.13).

$$\dot{\mathbf{X}}_b^i = \mathbf{R}(\psi)\mathbf{V}_b^i \tag{5.13}$$

Finally, the Euler method approximation state update constraints for $\mathbf{V}_b^i$ and $\mathbf{X}_b^i$ used in the trajectory optimization are represented thus.

$$\mathbf{V}_b^{i+1} = \mathbf{V}_b^i + \Delta t(\mathbf{B}\mathbf{U}_b^i - \mathbf{M}^{-1}(\mathbf{V}_b^i(\mathbf{C}(\mathbf{V_b^i}) + \mathbf{D}(\mathbf{V_b^i}))) \tag{5.14}$$

$$\mathbf{X}_b^{i+1} = \mathbf{X}_b^i + \Delta t\mathbf{R}(\psi)\mathbf{V}_b^i \tag{5.15}$$

Input Constraints

Torque limits of 4 newtons are placed on every input on every boat to attain reasonable actuator behavior (5.16).

$$|\mathbf{U}_b^i| < 4 \tag{5.16}$$

Collision Constraints

All above constraints would be commonplace in any trajectory optimization context with this boat model. To frame this problem as a multiagent one and avoid conflicts between trajectories of different boats, collision constraints are added. The boats are represented as circles with a radius derived at their widest orientation and constrained to maintain at least this distance as well as a minimum clearance from each other (5.17).

$$||\mathbf{X}_{b_1}^i - \mathbf{X}_{b_2}^i||_2 >= \sqrt{W^2 + H^2} + C \quad | \quad b_1! = b_2 \tag{5.17}$$

Lastly Δt is constrained between 5 and 100 seconds.

Together, the minimum time objective, state update constraints, input constraints and collision constraints provide a sufficient framework to obtain optimized feasible trajectories.

5.1.5 Simulation Results

I conducted planning on 9 different formation changes show in Fig. 5-4. While trajectory optimization without a defined initialization only finds trajectories for some of the formation changes involving four or less boats, the linear interpolation initialized optimizations is able to find trajectories in all cases and the shape-based one is able to find trajectories in most.

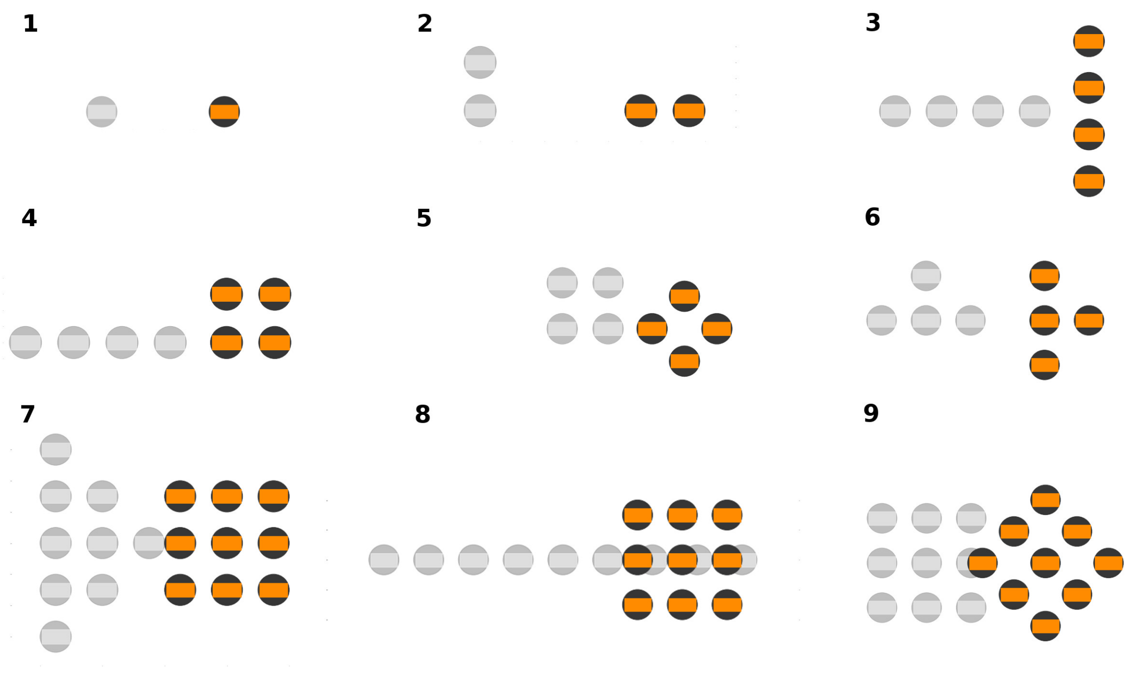
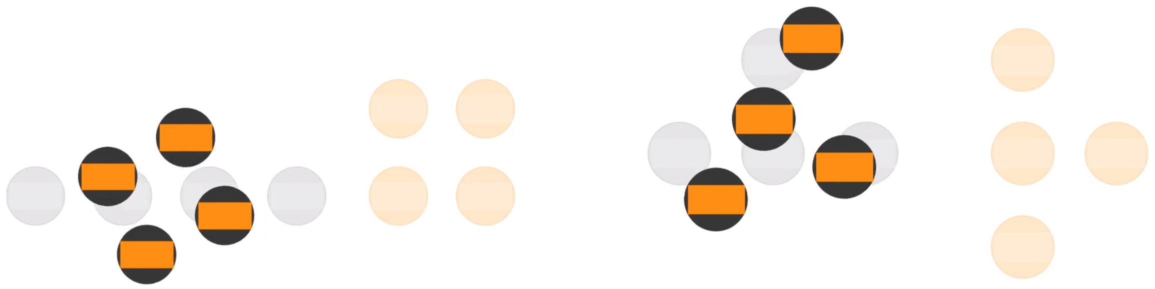


Figure 5-4: Formation Change Scenarios: 9 different formation changes with a diverse range of geometries, constant spacing between boats and constant mean state offset. Initial state in light gray, end state in orange. A dark gray region around the rectangular boats represents the clearance constraint region.

Trajectory Initialization

The initializations for the linear interpolation case need no further explaining as they are strictly defined, given the above specifications. Here I give an analysis of the calculated transformations and initializations for the shape-based initialization that employed ICP. ICP finds optimal or close to optimal fits for most final formations to initial formations, but does not guarantee an optimal fit and failed in some instances. As can be seen in Fig. 5-5 ICP finds practically optimal fits, even in many cases where no perfect transformation between initial and final points exists, but fails in some cases where a perfect transformation does exist. For plan 6 in Fig. 5-4 a perfect fit between the initial and final triangular formation does exist but ICP instead finds a local minima in Fig. 5-5.

In some other cases, such as plan 5, a perfect fit is found but ICP cannot find the minimum rotation trajectory, turning in the direction requiring a larger rotation. These defects in the shape-based initialization using ICP most likely affect trajectories


(a) close ICP fit

(b) suboptimal ICP fit

Figure 5-5: ICP Fitting: Initial and end state in gray and light orange respectively, with the ICP fit of the end state to the initial state in dark orange and gray. While ICP is able to find close to optimal fits in many situations (a), it does not guarantee an optimal fit or transformation (b).

optimized from this initialization and the performance of this method relative to the linear interpolation.

Simulation Evaluation

Performing direct transcription trajectory optimization works reliably for determining trajectories for boats, even in the more difficult plans with up to 9 boats as is shown in table 5.1. While SQP without an initialization converges to a solution in plans 1 and 4, linear initialization and shape-based initialization result in convergence to a solution in 8 and 7 plans out of 9 respectively.

While trajectory optimization without an initialization works for the 1 boat plan and one of the 4 boat plan, it is unable to cope reliably with multiple boats in a highly non-convex space. Additionally, in the 4 boat case where it succeeded it took over 90 seconds to compute a solution, a time comparable to finding trajectories from initializations for 9 boats. As can be seen in Fig. 5-6 from the trajectory found with a linear initialization for plan 9, solutions consist of significant crossings and overlaps between trajectories of different boats. The SQP optimization is unable to find a solution from no initialization and needs a close guess to work well. Linear and shape-based initializations provide this close guess.

Observing the results in Table 5.1, it can be noted that linear initializations converge to a solution in more cases and finds solutions faster that offer shorter time trajectories than shape-based ones, only hitting an iteration limit in plan 8, and even in that case providing a feasible trajectory upon closer inspection. Further analysis also revealed no significant difference in the input costs of boat initializations' final trajectories. The sum of the average squared input across all boats nor the maximum of the average sum squared input over all boats between both trajectories, indicating that both methods performed comparably in terms of actuator torques, although this

Table 5.1: Results of Trajectory Optimization

Plan	initialization	final time	solve time	solver status
1	-	6.45	0.68	Solution Found
	line	6.45	0.66	Solution Found
	shape	6.45	0.60	Solution Found
2	-	10.75	8.25	Iteration Limit
	line	9.43	4.04	Solution Found
	shape	9.43	4.28	Solution Found
3	-	12.52	55.30	Iteration Limit
	line	9.44	24.41	Solution Found
	shape	9.44	56.40	Solution Found
4	-	9.13	92.24	Solution Found
	line	9.13	25.64	Solution Found
	shape	11.41	49.67	Iteration Limit
5	-	11.87	76.50	Iteration Limit
	line	6.43	26.07	Solution Found
	shape	7.87	31.71	Solution Found
6	-	44.52	70.39	Iteration Limit
	line	8.36	29.95	Solution Found
	shape	8.36	24.86	Solution Found
7	-	47.75	120.36	Iteration Limit
	line	8.36	268.88	Solution Found
	shape	11.95	898.94	Solution Found
8	-	51.53	131.65	Unknown Error
	line	7.64	83.33	Iteration Limit
	shape	11.03	371.34	Iteration Limit
9	-	48.38	180.19	Iteration Limit
	line	7.64	83.33	Solution Found
	shape	13.31	136.88	Unknown Error

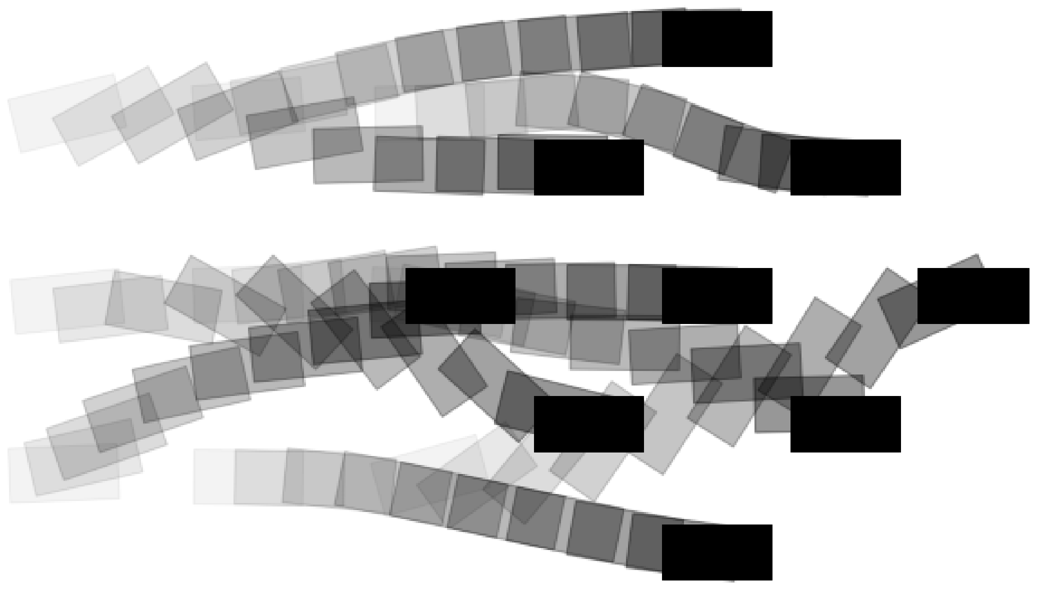


Figure 5-6: Superimposed States Near X^N for Plan 9: Boats manage to avoid collisions despite their concurrent motion and significant overlaps in their configuration spaces over all time.

was not explicitly placed in a cost function. It should be noted that the variance across input magnitudes at any instance for the final trajectories was larger for the linear interpolation than the shape-based initialization; this is likely because the shape-based initialization interpolates a common offset over all boats which contributes most of the velocity for the smaller formations.

In Fig. 5-7 the initialized and final trajectories for the linear and shape-based cases show the big differences in the initializations and solutions for both methods. First, a different assignment between boats and end points is found as assignment happens directly over start and end points in the linear case, but over the ICP shape in the shape-based initialization case. While the linear interpolation starts straight and converges to a curve close to the line, the shape-based initialization starts with significant curvature and converges further away from the initialization to straighter trajectories with close to constant curvature. Observing other trajectories also shows that the shape-based initialization undergoes more change in most, but not all tra-

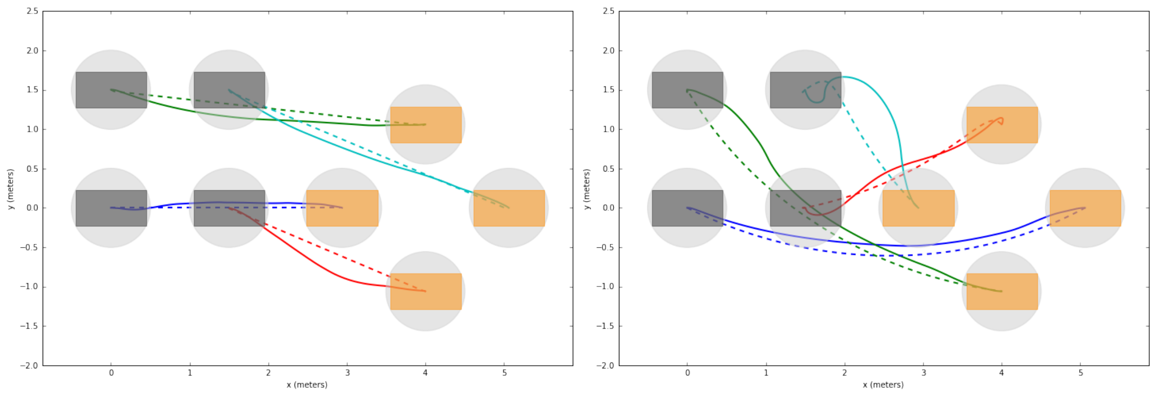

(a) linear initialization and final trajectory (b) shape-based initialization and final trajectory

Figure 5-7: Formation Change Trajectories: Initial and final formations shown in gray and orange respectively for plan 5. Dashed lines show the initialized trajectory and solid lines show the final trajectory.

jectories, indicating that the initialization is further from any minima.

While linear initialization performs better in all observed cased in terms of compute time and the minimum makespan objective, it should be noted that the shape transformation matrices found with ICP were suboptimal in many cases and could have contributed to the simple linear interpolation initialization outperforming shape-based initialization according to all considered metrics. Importantly, it must be noted that these are preliminary results and the resulting trajectories were not evaluated for robustness to disturbances. Shape-based initialization have the potential to be valuable when considering robustness or other metrics not evaluated her, as well as when optimizing over other objectives. Additional results are available online.[1].

The Hungarian algorithm for goal assignment, linear and shape-based interpolation for trajectory initialization and trajectory optimization, using direct transcription, from initial to goal formations proves to be a winning combination. The efficacy of this method was demonstrated on up to 9 simulated boats in very close proximity, making this a promising combination in the class centralized coupled planning algorithms.

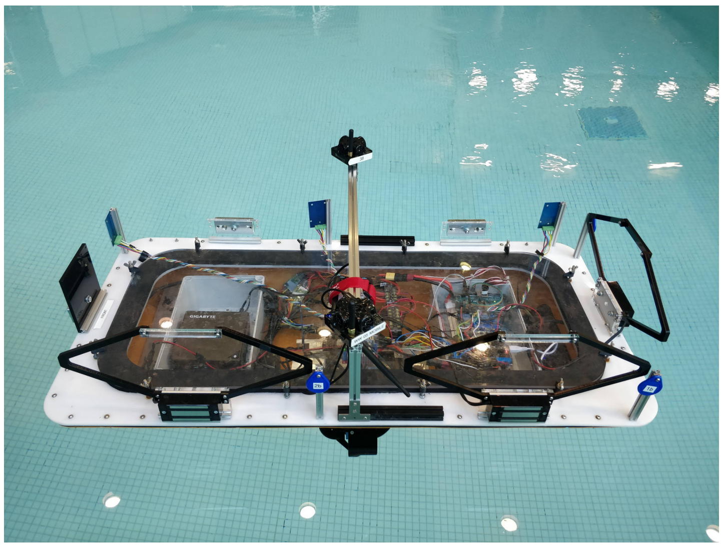
All initializations perform significantly better than the no-initialization scenario

[1]A video demonstrating the formation changing approach and simulation results is visible here: `https://youtu.be/kd0PPfe8hwg`

and linear initialization provides superior results to shape-based initialization for the minimum makespan objective. My results demonstrate that fast and concurrent trajectories close to the linear trajectories considered by Turpin *et al.* can be achieved even in conditions where all the constraints of C-CAPT do not hold and when a higher order assignment cost is used.

Nonetheless, the novel shape-based initialization explored her performs well. Variants of ICP, employing gradient-descent methods such as BFGS instead of SVD for ICP fitting, should be considered in the future to provide a more optimal transformation for the shape-initialization. Such an approach may yield superior results than those demonstrated here. Additionally, it must be noted that shape methods underlay much of formation control and evaluating shape-based initialization of trajectory optimization from a robustness perspective when executing trajectories is an avenue for exploration.

5.1.6 Experimental Evaluation


(a) Roboat: simple setup with range-finding beacons above the boat and a magnetic latching system around the sides

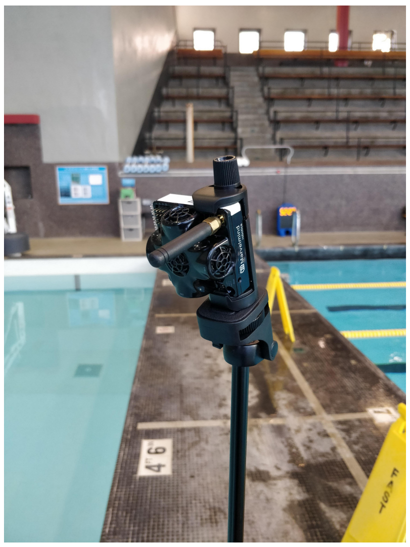

(b) Beacon System: units are mounted around the pool to help the Roboat localize

Figure 5-8: Setup for Formation Changing Experimental Evaluation

Two generated trajectories for simulated plan 7 with shape initialization were

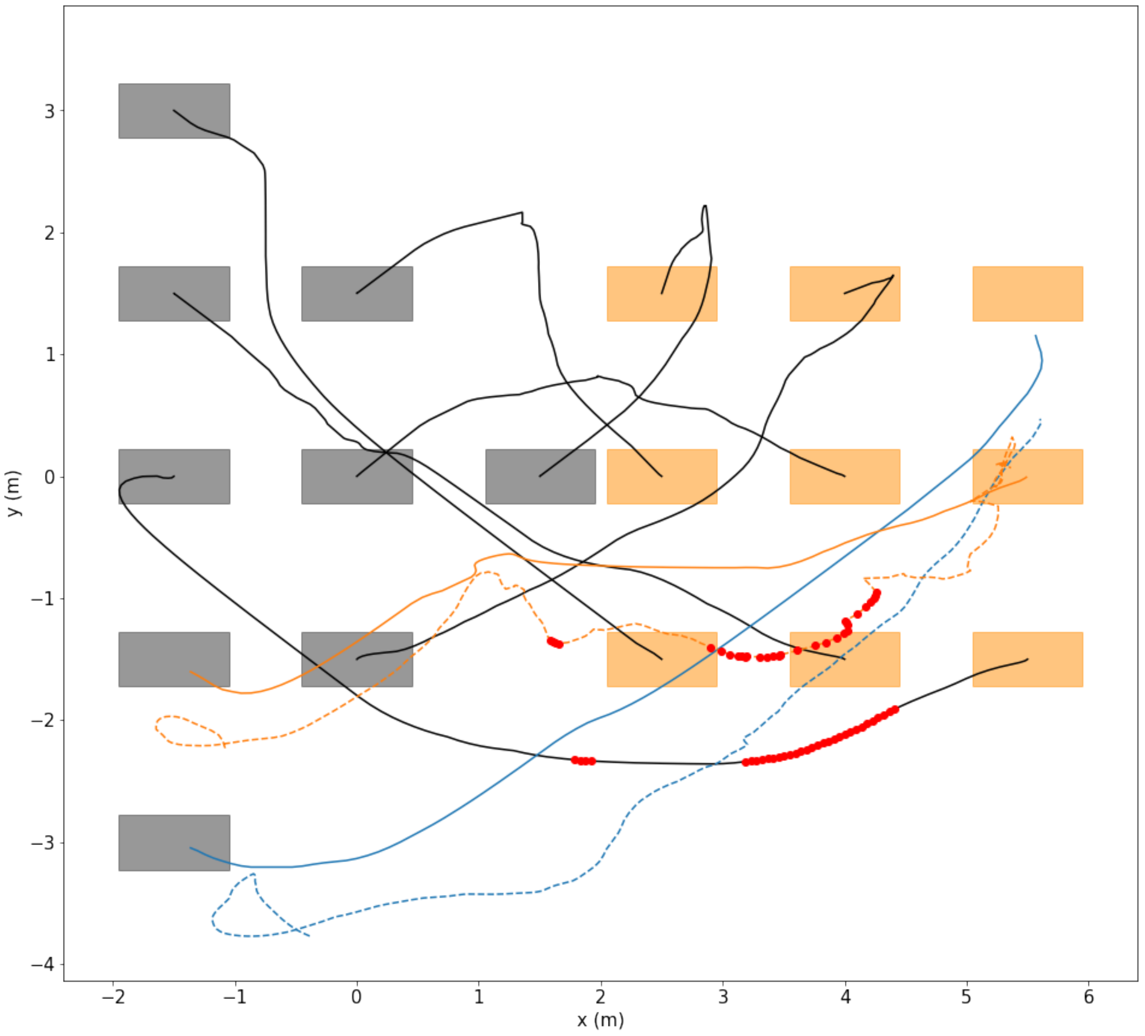


Figure 5-9: Formation Changing Pool Test for Plan 9: simulated trajectories shown in black, two tracked trajectories shown in blue and orange with dashed lines showing the real trajectory. Red points indicate evaluated collisions. A collision is observable between a tracked and simulated trajectory.

evaluated in the pool with the setup shown in Fig. 5-8. A simple beacon system was used for localization, to validate the trajectories. In the future they could be employed on the system described in chapters 3 and 4. For more details on the setup and controller please refer to the following paper by Park *et al.* [49].

All 9 trajectories for a large triangle to square formation change are presented in Fig. 5-9. Two trajectories, shown in blue and orange were tested in the pool, with dashed lines indicating the tracking result. The two boats are able to safely track the trajectory, without collisions between each other. When evaluating the tracking result against the other simulated trajectories, however, a collision can be observed between the real boat with an orange trajectory and a simulated boat, for parts of the trajectory shown in red. Despite the trajectory abiding by the input constraints described earlier and boat following a scaled trajectory not exceeding 0.3m/s, the controller has difficulty tracking the trajectory accurately in the pool environment. An average tracking error of 0.4m and 1.2m are experienced respectively on the orange and blue trajectories. This is likely because the boat was heavily modified between the current and previous setup, invalidating the model parameters. Besides the observance of collisions between one pair of real and simulated boats, the formation changing worked well. With a well tuned controller this method should be scalable to testing all 9 boats in water.

5.2 Shapeshifting: Reconfiguration Planning

Another important opportunity for using optimization on the Roboat is shapeshifting, the task of reconfiguring the shape of Roboats latched together. In such a scenario, pictured in Fig. 5-10, Roboats must plan an unlatch move and re-latch to change their shape. The boats used can be heterogeneous, meaning that in the case of the figure, only the ones labeled in red have localization abilities but manage to guide the other boats because of their attachment. This has many potential use-cases, including changing the shape of a large on-water platform or bridge and rearranging a Roboat in a convoy to fit through a canal opening.

In the previous section, we used sequential quadratic programming to effectively generate motion for one or multiple boats and considered the distance between boats' centers to apply collision constraints. In a shapeshifting scenario, where even tighter motion is desired, more precise non-convex constraints, considering each of the boats' edges, would need to be used to achieve close collision free motion. While there are nonlinear-programming approaches that directly grapple with such pairwise constraints between edges of rotating shapes, this becomes more difficult as the shapes become more complex, the likelihood of an NLP not finding a solution increases.

There exist other approaches that attempt to deal with this non-convexity splitting the original problem into convex sub-problems or asking a different non-convex problem. Mixed integer programming is a favorable methodology I will consider in this chapter for addressing such problems. Non-convex situations can be dealt with in a Mixed Integer Program (MIP) by partitioning them into convex problems and using integer variables to combine such sub-problems. Given such a formulation, Mixed Integer Programming is guaranteed to find the optimal solution to the problem given sufficient time. While it may often take a significant amount of time to solve a Mixed Integer Programs, there are approaches for generating tighter, more quickly solvable formulations. In our case, an MIP is used to find a valid, close to optimal trajectory in a planning corridor, and this solution is passed to an NLP solver with a slightly different formulation for refinement.

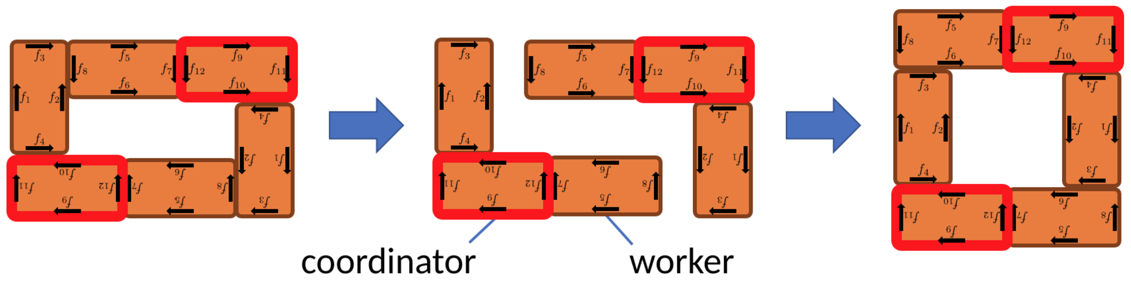


Figure 5-10: Shapeshifting: unlatching and latching to change the shape of Roboats. All localization-less workers remain connected to a worker.

In this section, I find the trajectory for a shape-shift by representing the trajectory as a quadratic B-Spline and employing a Mixed Integer Quadratic Programming (MIQP) formulation for the placement of the spline's control points. Constraints are employed on the spline, and these constraints maintain the spline's expressiveness while ensuring collision avoidance during transitions between different convex hulls in the configuration space. This provides a new way to generate smooth trajectories in near real-time.

5.2.1 Approach

My approach is broken into the steps shown in Fig 5-11. First, a pre-processing stage generates a configuration space graph. This is completed by providing the necessary steps in the shapeshift, calculating the Minkowski sum for static "obstacle" shapes, and generating a graph of linked convex polygons which the configuration space can be partitioned into. Second, in a trajectory optimization stage, a set of states and control inputs are found. This is completed by searching the graph for a free corridor in which the moving shape can move, optimizing a B-spline trajectory through this corridor, and scaling and discretizing the generated trajectory.

5.2.2 Pre-processing: Representing the Configuration Space

A configuration space for the reconfigured portions of shape is found by taking its Minkowski sum with the stationary portion of the shape over four cardinal orienta-

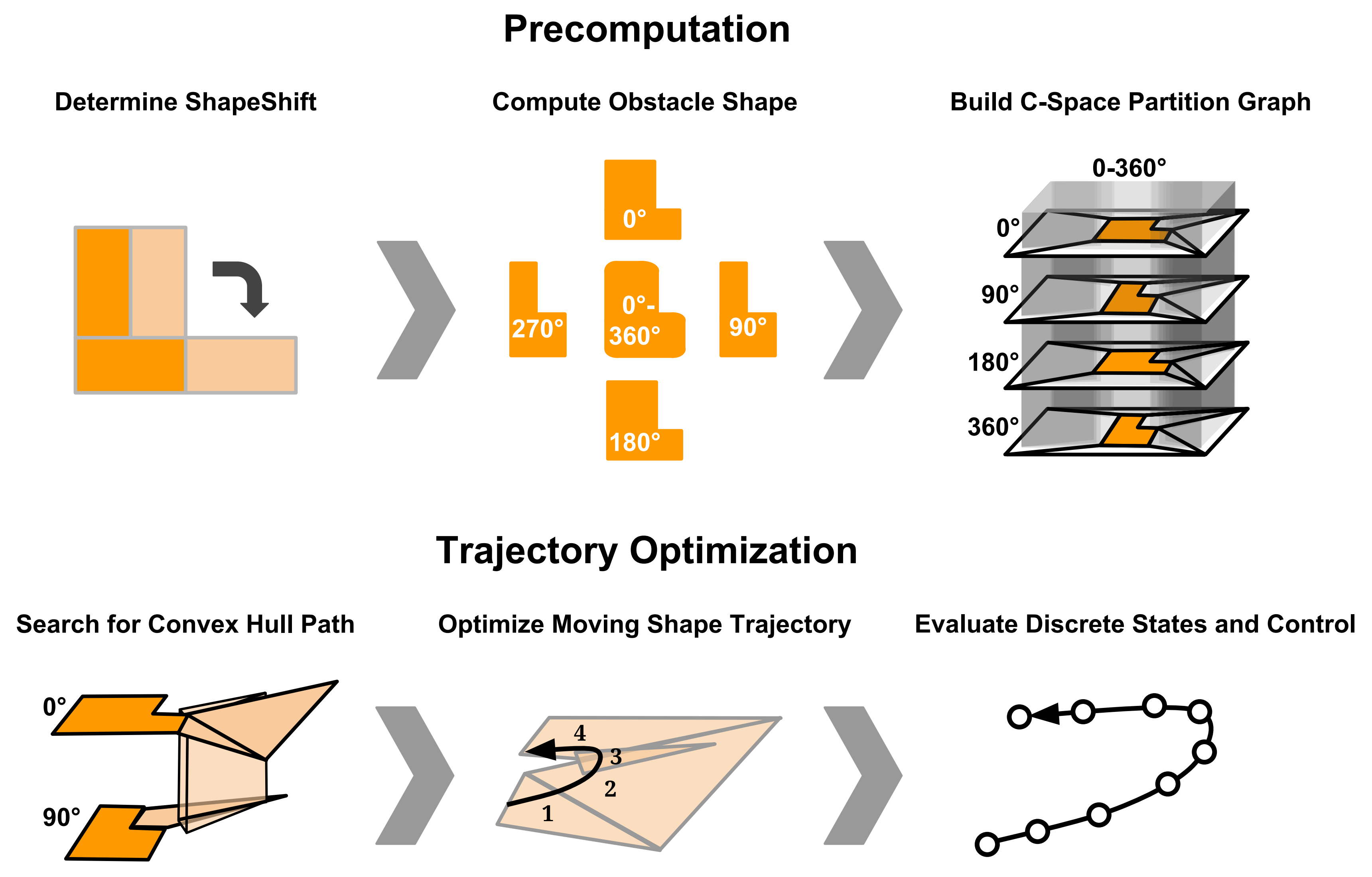


Figure 5-11: Steps Involved in Trajectory Generation for Shapeshifting: A precomputation layer generates C-Space partition graphs for each obstacle shape. These are then used by the trajectory optimization. In the future these could be used by the trajectory optimization layer for re-planning or use in other similar shapeshifts.

tions. Additionally, a Minkowski sum is taken between a 360 degree rotated sweep of this reconfigured portion and the static shape to determine the obstacle free regions where the boat can rotate unimpeded. A slight buffer is also added around these shapes, except for near the latch points to provide for safety. While other approaches use many more slices, this simple representation of the configuration space in four slices, instead of the convention of using many more over all rotations [28] allows for a speedy calculation of the trajectory optimization.

Minkowski Sum of Polygonal Shapes

Given two shapes of boats, the Minkowski sum of the stationary shape with the moving one is found. This operation is completed for all four directions the boat can face, as well as the sweep obtained from rotating the boat. Unlike other approaches that require generating a map over many more angles, by only considering the angles the boat will latch at and the places a boat can rotate in, a more efficient, albeit suboptimal, solution is found.

Configuration Space Partitioning into Polygons

We use the Hertel-Mehlhorn algorithm to partition the configuration space into convex regions. This method guarantees a speedy partitioning consisting of no more than four times the optimal number of regions in $O(n^3)$ in the number of points [21].

Graph Representation

The partitioning is represented as a graph with the convex hulls represented as vertices and shared edges and overlaps as edges. Connections are made in $\mathrm{O}(n^2)$ time in the number of hulls between the layers, representing the different rotations of the Roboat. A depth first search over this graph finds a corridor over the configuration space to use in the trajectory optimization [44].

5.2.3 MIQP Trajectory Optimization

To optimize over the space of all trajectories for a boat shape through the polygons delineated in the configuration space we cannot simply rely on the same kind of optimization used in the last section since we will have several OR constraints which cannot be represented with sequential quadratic programming. Hence, we turn to Mixed Integer Quadratic Programming (MIQP), an optimization in which the objective is quadratic, the constraints are all linear and both can also contain integer variables.

Mixed Integer Quadratic Programming

We use a simplified state and transition model of the boat from the last section which does not include rotation. In the last section, rotation was included to represent the entire boat shape being moved as one boat. The pydrake optimization framework was used to implement the optimizations [58].

Acceleration Cost In the MIQP, a fixed time interval is set for the state transitions, as nonlinear terms are not permitted in the constraints. Therefore, I optimize the acceleration in this case to obtain a smooth path.

$$\min \sum_i \ddot{\boldsymbol{x}}_i(t)^T \ddot{\boldsymbol{x}}_i(t) \tag{5.18}$$

Collision Constraints Unlike the previous section where pairwise constraints were placed on the distance between boats, here we only have a boat shape and configuration space of polygon regions. Thus we constrain each state in the path to lie in a valid place in the configuration space by creating $\boldsymbol{H}$, a binary matrix. There's N rows for each transition between states, M columns for each polygon. Rows summing to 1, as well as a 1 at an index, indicates that the boat is in that column's polygon for that row's state transition.

$$\boldsymbol{H} \in \{0,1\}^{M \times N-1} \tag{5.19}$$

$$\sum \boldsymbol{H^i} = 1 \mid 0 \leq i \leq N+1 \tag{5.20}$$

I can constrain the physical state of the boat shape to lie in the right polygons during the state transitions by constraining the start and end state for each to lie in the polygons, as shown in Fig. 5-12 from Deits and Tedrake [9].

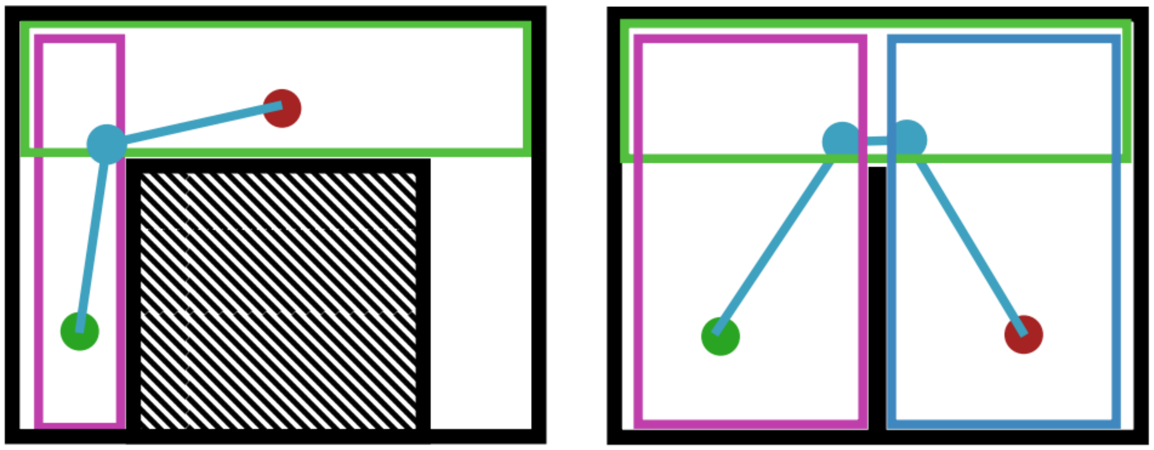


Figure 5-12: A trajectory in which each linear segment is required to remain entirely within one of the convex obstacle-free regions indicated by the colored boxes. This requirement ensures that the entire trajectory is collision-free [9].

This works because the boat shape is represented as a transition between states along lines, and the polygons are convex, guaranteeing that a line within them connecting two points is contained in the polygon.

This is represented with the following equation

$$\boldsymbol{B}_h \leq \boldsymbol{A}_h \boldsymbol{x}_{i+j} \mid \boldsymbol{H}_h^i = 1, j \in \{0,1\} \tag{5.21}$$

which is implemented in the MIQP as follows

$$M(\boldsymbol{H}_i^h - 1) + \boldsymbol{B_h} \leq \boldsymbol{A}_h \boldsymbol{x}_{i+j} \mid j \in \{0,1\} \tag{5.22}$$

5.2.4 Adding Rotation

As is, we can obtain x-y paths, but rotation for the boats is not defined. An additional step is produced to yield smooth rotations. The angle $\boldsymbol{\theta}$, angular velocity $\dot{\boldsymbol{\theta}}$ and angular acceleration $\ddot{\boldsymbol{\theta}}$ are defined as follows.

$$\boldsymbol{\theta} \in \mathbb{N}^{N+1} \tag{5.23}$$

$$\dot{\boldsymbol{\theta}} \in \mathbb{N}^{N} \tag{5.24}$$

$$\ddot{\boldsymbol{\theta}} \in \mathbb{N}^{N\times 2} \tag{5.25}$$

Angle transition constraints $\boldsymbol{\theta}$ and $\dot{\boldsymbol{\theta}}$ represent the angle and angular velocity, respectively, at the control points, while $\ddot{\boldsymbol{\theta}}$ represents a constant acceleration for the first and second half of each state transition. Based on this, the angle transition constraints are defined as follows.

$$\begin{aligned}
\boldsymbol{\theta}_i &= \boldsymbol{\theta}_{i-1} + \dot{\boldsymbol{\theta}}_{i-1} + \frac{1}{2}((\frac{1}{2})^2\ddot{\boldsymbol{\theta}}^0_{i-1}) + \frac{1}{2}(\frac{1}{2}\ddot{\boldsymbol{\theta}}^0_{i-1} + (\frac{1}{2})^2\ddot{\boldsymbol{\theta}}^1_{i-1}) && (5.26)\\
&= \boldsymbol{\theta}_{i-1} + \dot{\boldsymbol{\theta}}_{i-1} + .375\ddot{\boldsymbol{\theta}}^0_{i-1} + .125\ddot{\boldsymbol{\theta}}^1_{i-1} && (5.27)
\end{aligned}$$

$$\dot{\boldsymbol{\theta}}_i = \dot{\boldsymbol{\theta}}_i + \frac{\ddot{\boldsymbol{\theta}}^0_{i-1} + \ddot{\boldsymbol{\theta}}^1_{i-1}}{2} \tag{5.28}$$

Angular acceleration cost As was done with the x-y state, a quadratic cost is applied to smooth the angle as follows. This has no effect on the x-y state as that was obtained in the previous optimization.

$$\sum_i \min \boldsymbol{\theta}_i^T \boldsymbol{\theta}_i \tag{5.29}$$

Angle collision constraints Similar to the prior MIQP optimizations, the angle at control points is constrained to lie in valid regions. As there are no spline guarantees here, additional constraints are placed on the angular velocity. Together with the angle constraints, this ensures that state transitions remain in the acceptable angle ranges for each polygon.

$$\forall j \in \{0,1\}, i \in \mathbb{N} \mid 0 \leq i \leq N \tag{5.30}$$

$$\boldsymbol{\theta}^h_{min} + M(\boldsymbol{H}^h_{i+j} - 1) \leq \boldsymbol{\theta}_i \leq M(1 - \boldsymbol{H}^h_{i+j}) + \boldsymbol{\theta}^h_{max} \tag{5.31}$$

$$(\boldsymbol{\theta}^h_{min} - \boldsymbol{\theta}^h_{max}) + M(\boldsymbol{H}^h_{i+j} - 1) \leq \dot{\boldsymbol{\theta}}_i \leq M(1 - \boldsymbol{H}^h_{i+j}) + (\boldsymbol{\theta}^h_{max} - \boldsymbol{\theta}^h_{min}) \tag{5.32}$$

Quadratic B-Spline Trajectory Representation

While the MIQP can be easily implemented with linear transitions between x-y states, this requires an exceeding number of states to create a smooth path. This is acceptable in an SQP implementation which runs in polynomial time, but not for an MIQP which generally runs in a time exponential to the number of integer variables. Therefore, a curve-based parameterization is desirable for the MIQP.

Quadratic B-Splines B-splines are an ideal solution for representing trajectories as they provide an efficient way of specifying a smooth path, with spline segments capable of representing any polynomial of degree B with B+1 control points. Additionally, compared to other smooth parameterizations, B-splines have a convexity guarantee which ensures that spline segments lie in the convex hull of control points defining them.

This is done by recursively defining scaling factors for the ith control point and kth order along the trajectory with the Cox-de Boor formula.

$$B_{i,k}(t) = \frac{t - T_i}{T_{i+k-1} - T_i B_{i,k-1}(t)} + \frac{T_{i+k} - t}{T_{i+k} - T_{i+1} B_{i+1,k-1}(t)} \tag{5.33}$$

$$B_{i,1}(t) = \begin{cases} 1 & \text{if} \quad T_i \leq t < T_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{5.34}$$

The sum of the control points multiplied by their scaling factors yields the following B-spline equation:

$$\boldsymbol{p}_k(t) = \sum_i B_{i,k}(t)\boldsymbol{\alpha}_i \tag{5.35}$$

In this thesis, a uniform quadratic B-spline will be used. Any second order polynomial equations can be described with B-splines of order $k = 3$. Uniform B-splines have evenly spaced T_i in Eq. 5.34, which means that $\boldsymbol{p}(t)$ can be easily represented as the following:

$$\boldsymbol{p}(t) = \boldsymbol{p}_3(t) = \begin{pmatrix} 1 \\ \frac{t}{\Delta} - i \\ (\frac{t}{\Delta} - i)^2 \end{pmatrix}^T \boldsymbol{M}_3 \begin{pmatrix} \boldsymbol{\alpha}_{i-1}{}^T \\ \boldsymbol{\alpha}_i{}^T \\ \boldsymbol{\alpha}_{i+1}{}^T \end{pmatrix} \mid i \leq \frac{t}{\Delta} < i+1 \tag{5.36}$$

where in the case of quadratic B-splines, M_3 is constant and

$$\boldsymbol{M}_3 = 0.5 \begin{pmatrix} 1 & 1 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \tag{5.37}$$

Initial and Final State Constraints To constrain the B-spline to start at the initial state and end at the final state, the additional control points are defined before and after the the 0th as well as from Nth control points and are respectively constrained to equal the initial and final states.

$$\boldsymbol{\alpha} \in \mathbb{N}^{N+5\times 2} \tag{5.38}$$

$$\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_{-1}, \boldsymbol{\alpha}_{-2} = \boldsymbol{x_0} \tag{5.39}$$

$$\boldsymbol{\alpha}_N, \boldsymbol{\alpha}_{N+1}, \boldsymbol{\alpha}_{N+2} = \boldsymbol{x_N} \tag{5.40}$$

Acceleration Cost As in the standard MIQP formulation, a cost is placed on the acceleration to force the boat shape to follow the smoothest possible trajectory in the allotted time.

$$\min \int_{-1}^{N+1} \ddot{\boldsymbol{p}}(t)^T \ddot{\boldsymbol{p}}(t) dt \tag{5.41}$$

The acceleration can be simplified, as demonstrated by Usenko *et al.* [62]. Δ is set to 1 during the optimization but scaled later to generate the final trajectory for tracking.

$$\ddot{\boldsymbol{p}}_i(t) = \frac{1}{\Delta^2} \begin{pmatrix} 0 \\ 0 \\ 2 \end{pmatrix}^T \boldsymbol{M}_3 \begin{pmatrix} \boldsymbol{\alpha}_{i-1}{}^T \\ \boldsymbol{\alpha}_i{}^T \\ \boldsymbol{\alpha}_{i+1}{}^T \end{pmatrix} \mid i \leq \frac{t}{\Delta} < i+1 \tag{5.42}$$

$$\ddot{\boldsymbol{p}}_i(t) = \frac{1}{\Delta^2} b^T \boldsymbol{M}_3 \begin{pmatrix} \boldsymbol{\alpha}_{i-1}{}^T \\ \boldsymbol{\alpha}_i{}^T \\ \boldsymbol{\alpha}_{i+1}{}^T \end{pmatrix} \mid i \leq \frac{t}{\Delta} < i+1 \tag{5.43}$$

As $\boldsymbol{M_3}$ and $\boldsymbol{b}$ are constant, this yields the following quadratic equation to be minimized in the optimization, yielding a smooth trajectory for the boat shape:

$$\underset{p}{\operatorname{argmin}} \int_{-1}^{N+1} \boldsymbol{p}(t)^T \boldsymbol{p}(t) dt = \tag{5.44}$$

$$\underset{p}{\operatorname{argmin}} \sum_{i=-1}^{N+1} tr(\begin{pmatrix} \boldsymbol{\alpha}_{i-1}{}^T \\ \boldsymbol{\alpha}_i{}^T \\ \boldsymbol{\alpha}_{i+1}{}^T \end{pmatrix}^T \boldsymbol{M}_3^T \boldsymbol{b}^T \boldsymbol{b} \boldsymbol{M}_3 \begin{pmatrix} \boldsymbol{\alpha}_{i-1}{}^T \\ \boldsymbol{\alpha}_i{}^T \\ \boldsymbol{\alpha}_{i+1}{}^T \end{pmatrix}) \tag{5.45}$$

Control Point Edge Constraints The strong convex hull property of B-splines guarantees that spline segments lie in the convex hull of control points defining them. In our quadratic B-spline case, this means that any point on the trajectory segment between knot points is guaranteed to lie in a triangle if the three control points defining it lie in the polygon.

As in the standard MIQP formulation where constraints were placed on the states to contain the state transitions in convex regions 5.22, constraints are placed on the control points in the quadratic B-Spline formulation.

$$M(\boldsymbol{H}_i^h - 1) + \boldsymbol{B_h} \leq \boldsymbol{A}_h \boldsymbol{\alpha}_{i+j} \mid j \in \{0, 1\} \tag{5.46}$$

Polygon Transition Edge Constraints Using the Strong Hull Property The above constraint is sufficient if the trajectory is entirely contained in one polygon. Special additional precautions need to be taken in the MIQP case when the trajectory moves from one polygon to another, since the convex hull of three consecutive polygon transition control points will not be contained in one polygon. When this occurs, we ensure that the triangle made by these three control points is fully contained in the union of two polygons.

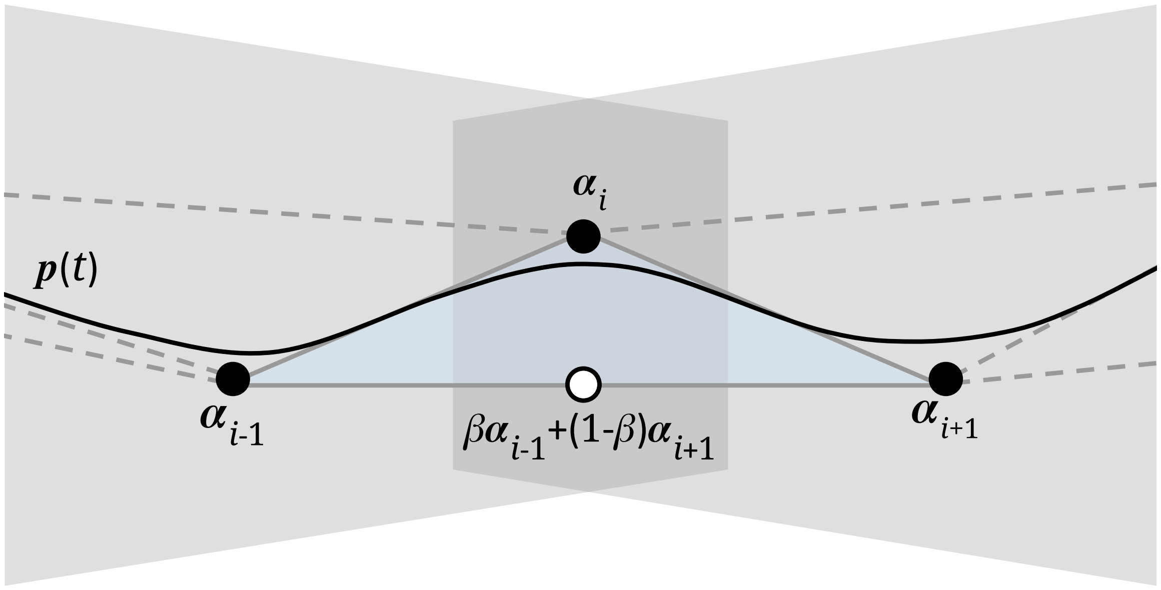

$\boldsymbol{\alpha}_i$
$\boldsymbol{p}(t)$
$\boldsymbol{\alpha}_{i\text{-}1}$
$\beta\boldsymbol{\alpha}_{i\text{-}1}+(1\text{-}\beta)\boldsymbol{\alpha}_{i+1}$
$\boldsymbol{\alpha}_{i+1}$

Figure 5-13: Collision Avoidance Constraints at Polygon Transitions: constraining the control points $\boldsymbol{\alpha}_{i-1}$ and $\boldsymbol{\alpha}_{i+1}$ to pass through the blue triangle insures $\boldsymbol{p}(t)$ stays in the union of the grey polygons.

Theorem 5.2.1 *Given three consecutive points $\boldsymbol{\alpha}_{i-1}$, $\boldsymbol{\alpha}_i$ and $\boldsymbol{\alpha}_{i+1}$, with $\overline{\boldsymbol{\alpha}_{i-1}\boldsymbol{\alpha}_i}$ lying in one convex polygon and $\overline{\boldsymbol{\alpha}_i\boldsymbol{\alpha}_{i+1}}$ lying in another, the convex hull of these points lies in the union of the polygons if $\overline{\boldsymbol{\alpha}_{i-1}\boldsymbol{\alpha}_{i+1}}$ intersects with their intersection.*

Proof 5.2.1 *If the line segment $\overline{\boldsymbol{\alpha}_{i-1}\boldsymbol{\alpha}_{i+1}}$ passes through the intersection of two convex hulls, some point $\boldsymbol{x}$ along this line segment lies in the intersection, since $\boldsymbol{\alpha}_i$ and*

$\boldsymbol{x}$ lie in both polygons as does $\overline{\boldsymbol{x}\boldsymbol{\alpha}_i}$. Therefore, $\boldsymbol{x}\boldsymbol{\alpha}_i\overset{\triangle}{\boldsymbol{\alpha}}\boldsymbol{\alpha}_{i-1}$ lies in the first polygon and $\boldsymbol{x}\boldsymbol{\alpha}_i\overset{\triangle}{\boldsymbol{\alpha}}\boldsymbol{\alpha}_{i+1}$ lies in the second polygon, showing $\boldsymbol{\alpha}_{i-1}\overset{\triangle}{\boldsymbol{\alpha}_i}\boldsymbol{\alpha}_{i+1}$ lies in the union of the polygons. □

At such polygon transition points shown in Fig. 5-13, I constrain a point between $\boldsymbol{\alpha}_{i-1}$ and $\boldsymbol{\alpha}_{i+1}$ to lie in the intersection between the polygons they respectively lie in. Thereby, according to 5.2.1, $\boldsymbol{\alpha}_{i-1}\overset{\triangle}{\boldsymbol{\alpha}_i}\boldsymbol{\alpha}_{i+1}$ lies in the union of the polygons, and according to the strong hull property of B-splines, the Quadratic B-Spline $\boldsymbol{p}(t)$ will remain in $\boldsymbol{\alpha}_{i-1}\overset{\triangle}{\boldsymbol{\alpha}_i}\boldsymbol{\alpha}_{i+1}$ for the segment defined $\boldsymbol{\alpha}_{i-1}$, $\boldsymbol{\alpha}_i$ and $\boldsymbol{\alpha}_{i+1}$.

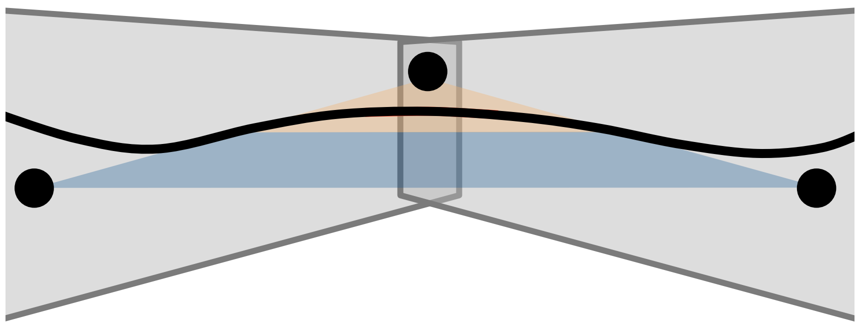
In the MIQP optimization, we simplify the condition such that we require the halfway point between first and third transition control points to be contained in both the pre and post transition polygons. This allows us to formulate the condition as the following linear constraint.

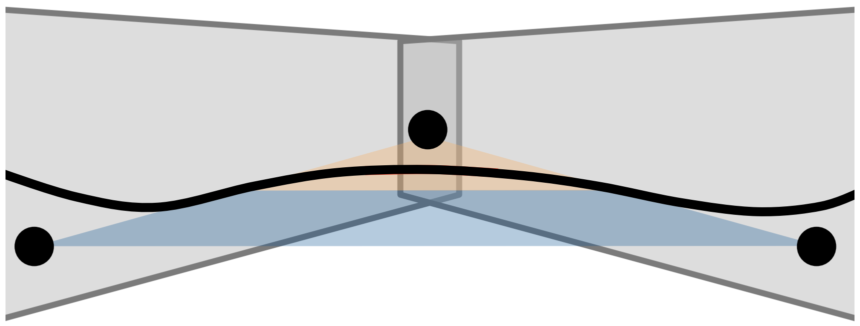
This can be represented with the following equation:

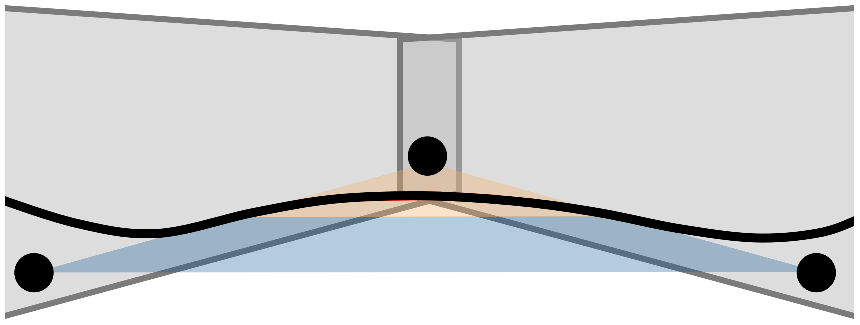
$$\boldsymbol{B}_h \leq \boldsymbol{A}_h \frac{\boldsymbol{\alpha}_{i-1} + \boldsymbol{\alpha}_{i+1}}{2} \mid \boldsymbol{H}_h^{i-1} + \boldsymbol{H}_h^{i} = 1 \tag{5.47}$$

A More Optimal Polygon Transition Constraint While a constraint on $\boldsymbol{\alpha}_{i-1}\overset{\triangle}{\boldsymbol{\alpha}_i}\boldsymbol{\alpha}_{i+1}$ yields a trajectory contained in the C-Space polygons, relaxing this constraint does not necessarily mean that the B-spline is not constrained in the C-Space polygons, as shown in Fig. 5-14. In the case of Uniform Quadratic B-splines, where control points are evenly spaced in time, at polygon transition points, $\boldsymbol{p}(i)$ is tangent to both $\overline{\boldsymbol{\alpha}_{i-1}\boldsymbol{\alpha}_i}$ and $\overline{\boldsymbol{\alpha}_i\boldsymbol{\alpha}_{i+1}}$(eg. $\boldsymbol{p}(i) = \frac{\boldsymbol{\alpha}_i+\boldsymbol{\alpha}_{i-1}}{2}$ and $\boldsymbol{p}(i+1) = \frac{\boldsymbol{\alpha}_i+\boldsymbol{\alpha}_{i+1}}{2}$). The acceleration is also a constant $\overrightarrow{\boldsymbol{\alpha}_i\boldsymbol{\alpha}_{i+1}} - \overrightarrow{\boldsymbol{\alpha}_{i-1}\boldsymbol{\alpha}_i}$ during this interval, according to Eq. 5.43, yielding a quadratic curve that stays above the line segment $\overline{\boldsymbol{p}(i)\boldsymbol{p}(i+1)}$. Requiring this cut hull, $\boldsymbol{p}(i)\overset{\triangle}{\boldsymbol{\alpha}_i}\boldsymbol{p}(i+1)$, instead of the larger convex hull $\boldsymbol{\alpha}_{i-1}\overset{\triangle}{\boldsymbol{\alpha}_i}\boldsymbol{\alpha}_{i+1}$ it is contained in, to lie in the union of convex polygons allows a wider range of B-splines to be considered, producing a more optimal result.

Thus, in my final formulation, a point along the line segment $\overline{\boldsymbol{p}(i)\boldsymbol{p}(i+1)}$ is constrained as shown in Fig. 5-15. This can be represented with the following equation.


(a) Control Point Hull and Cut Hull in Polygon Union


(b) Only Cut Hull in C-Space Polygon Union


(c) Control Point Hull and Cut Hull outside Polygon Union

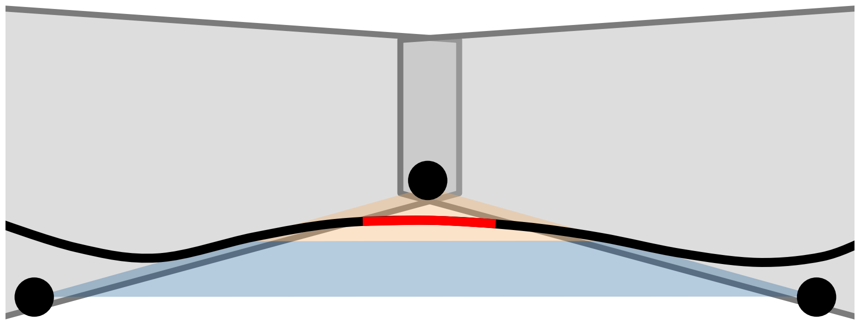

(d) Control Point Hull, Cut Hull and B-Spline outside Polygon Union

Figure 5-14: Comparing Different Hulls for Containing B-Spline in C-Space: keeping the cut hull triangle (orange) in the union of the convex hulls (grey outlined) provides a tighter constraint than the control point triangle (blue) for keeping the B-spline (black) in the union of the convex hull polygons. As shown in (c), it is still possible for both triangle to stray outside this union, but the B-Spline to remain inside.

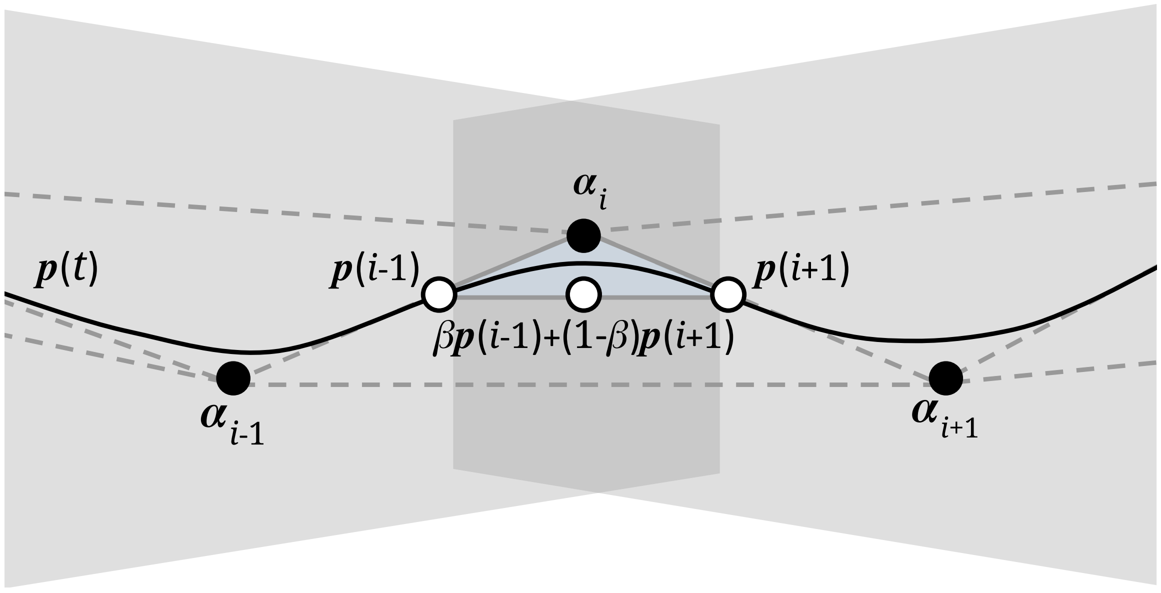


Figure 5-15: Constraints at Polygon Transitions: constraining the line between points $\boldsymbol{p}(i-1)$ and $\boldsymbol{p}(i+1)$ to pass through the blue triangle is a tighter constraint than that shown in 5-13 for ensuring $\boldsymbol{p}(t)$ stays in the union of the grey polygons.

$$\boldsymbol{\beta} \in (0,1)^{N-2} \tag{5.48}$$

$$\boldsymbol{B}_h \leq \boldsymbol{A}_h(\boldsymbol{\beta}_i \frac{\boldsymbol{\alpha}_{i-1} + \boldsymbol{\alpha}_i}{2} + (1-\boldsymbol{\beta}_i)\frac{\boldsymbol{\alpha}_i + \boldsymbol{\alpha}_{i+1}}{2}) \mid \boldsymbol{H}_h^{i-1} + \boldsymbol{H}_h^i = 1 \tag{5.49}$$

By setting $\boldsymbol{\beta}$ to 0.5, I can represent this in my optimization as the following linear mixed integer constraint.

$$M(j\boldsymbol{H}_{i-1}^h - j\boldsymbol{H}_i^h - 1) + \boldsymbol{B_h} \leq \boldsymbol{A}_h \frac{1}{2}(\frac{\boldsymbol{\alpha}_{i-1} + \boldsymbol{\alpha}_i}{2} + \frac{\boldsymbol{\alpha}_i + \boldsymbol{\alpha}_{i+1}}{2}) \mid j \in \{1, -1\} \tag{5.50}$$

Scaling

As a final step, the B-spline trajectories are scaled in time to meet the desired kinematic constraints. This is easily done by changing Δ in Eq. 5.43 after the optimization to evaluate the trajectory.

5.2.5 SQP Smoothing

This last transition constraint restricts the shape of the B-spline in unnecessary ways, since setting $\boldsymbol{\beta}$ to 0.5 may not necessarily be optimal. Therefore, a new scheme is considered, in addition to the Line and B-spline-based MIQP formulations. In this scheme, the MIQP B-spline optimization result is used to determine the integer variables. This result is used in an SQP optimization, using the resulting control points from the MIQP optimization as an initialization. In this second optimization, $\boldsymbol{\beta}$ is set by directly constraining it according to Eq. 5.49, resulting in refined control point placements and a more optimal B-spline for minimal extra computation.

5.2.6 Simulation Evaluation

I compared the three representations described previously in a series of simulation experiments presented in table 5.2 and Fig. 5-16. No scaling was done to evenly compare the optimization results. The experiments cover a range of trajectories involving shape changes over a varying number of boats, from 2 to 12. A fixed number of control points, 11, were used in each simulation. The solve times and average costs over the control points are listed, with the fastest and lowest cost values highlighted respectively for each experiment when there is not a three way tie.

As can be observed, the simple line representation generally yields the fastest total solve times, which is only surpassed by the MIQP only B-spline representation for one experiment. While it is generally faster, the B-spline representation achieves comparable results on accuracy, sometimes surpassing the Line representation, although it generally takes longer due to the more complicated representation. In the Refined B-spline, this MIQP result is taken, fixing the control point hull placements, and optimizing their precise in hull positions, yielding a generally lower cost result, while only taking slightly longer. This benefit comes despite the fact that the B-spline representations yield a twice differentiable continuous trajectory, maintaining stronger obstacle avoidance guarantees. Thus, running in 0.1-0.3 seconds, this refined B-spline optimization yields high quality real-time trajectories.

Table 5.2: Solve Times and Costs For Different Optimization Strategies: best in bold

Representation	Plan	Solve Time (s)			Average Cost	
		Initial	Final	Total	Position	Angle
Line	1 line to L	-	0.120	**0.122**	1.540	0.000
	2 T to block	-	0.110	0.110	**12.226**	0.000
	3 line to block	-	0.110	**0.115**	4.530	0.000
	4 line to L 2	-	0.120	**0.121**	2.010	9.740
	5 Ls to block	-	0.060	**0.057**	**8.892**	38.980
	6 line to block 2	-	0.190	**0.188**	5.610	0.000
	7 rhombus to block	-	0.100	0.100	**56.658**	0.000
	8 past	-	0.110	**0.105**	7.700	0.000
	9 pool a	-	0.110	**0.110**	2.390	0.000
	10 pool b	-	0.100	**0.101**	4.660	9.740
B-Spline	1 line to L	-	0.230	0.230	1.930	0.000
	2 T to block	-	0.110	**0.106**	28.150	0.000
	3 line to block	-	0.200	0.200	6.210	0.000
	4 line to L 2	-	0.180	0.180	1.730	**8.700**
	5 Ls to block	-	0.060	0.060	62.460	**34.801**
	6 line to block 2	-	0.240	0.240	7.500	0.000
	7 rhombus to block	-	0.040	**0.044**	1008.360	0.000
	8 past	-	0.170	0.170	6.260	0.000
	9 pool a	-	0.250	0.250	3.340	0.000
	10 pool b	-	0.130	0.130	5.100	**8.700**
Refined B-Spline	1 line to L	0.230	**0.037**	0.270	**1.034**	0.000
	2 T to block	0.100	**0.024**	0.130	12.510	0.000
	3 line to block	0.200	**0.025**	0.230	**3.590**	0.000
	4 line to L 2	0.190	**0.032**	0.230	**1.324**	**8.700**
	5 Ls to block	0.090	**0.023**	0.110	17.340	**34.801**
	6 line to block 2	0.230	**0.033**	0.260	**4.143**	0.000
	7 rhombus to block	0.040	**0.027**	0.070	82.860	0.000
	8 past	0.210	**0.026**	0.240	**5.794**	0.000
	9 pool a	0.240	**0.029**	0.270	**2.185**	0.000
	10 pool b	0.260	**0.026**	0.280	**3.305**	**8.700**

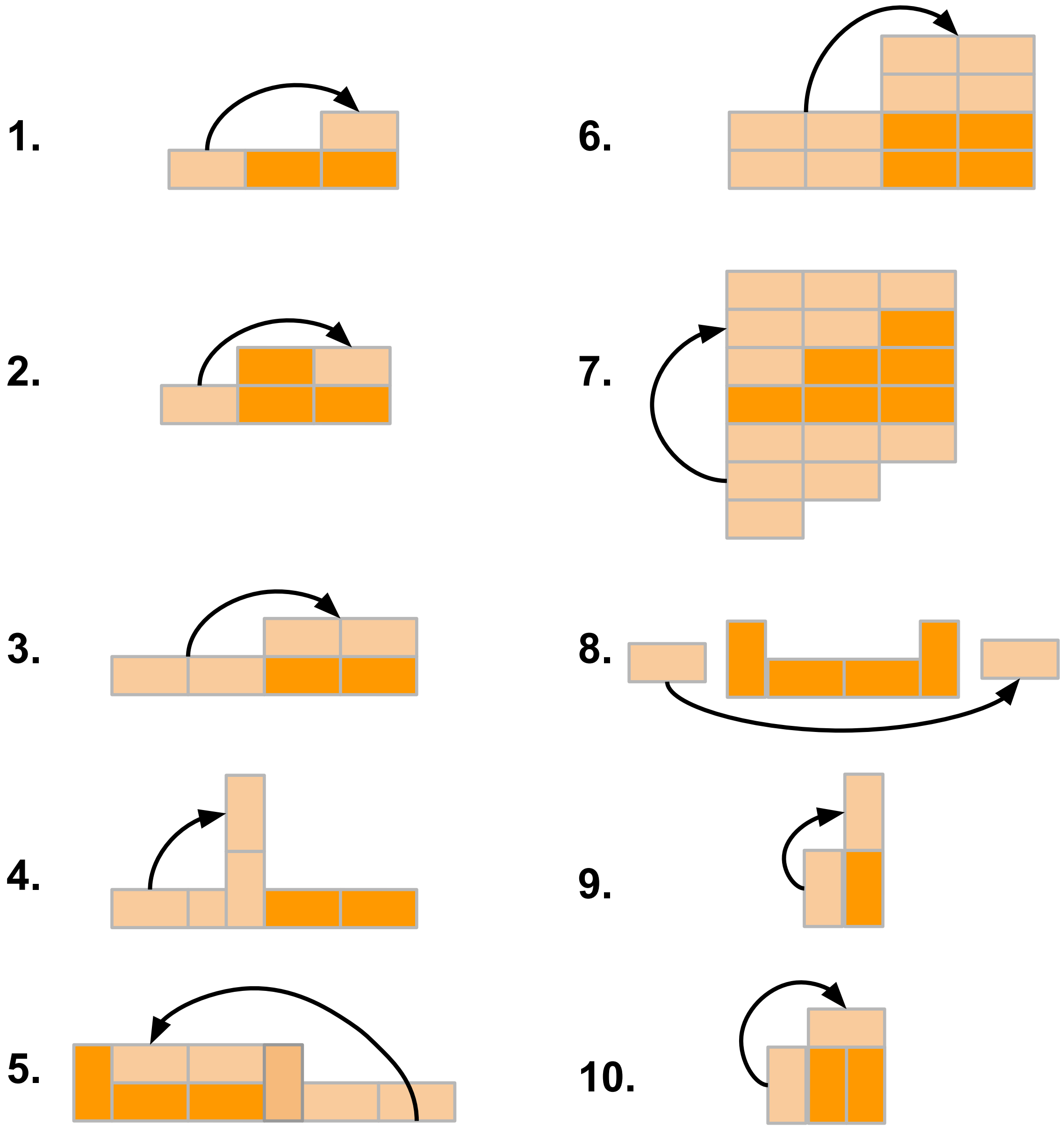


Figure 5-16: Visualization of Shapeshifts Optimized in Table 5.2: dark orange rectangles are static boats and light orange blocks are moving boats for which a trajectory is found.

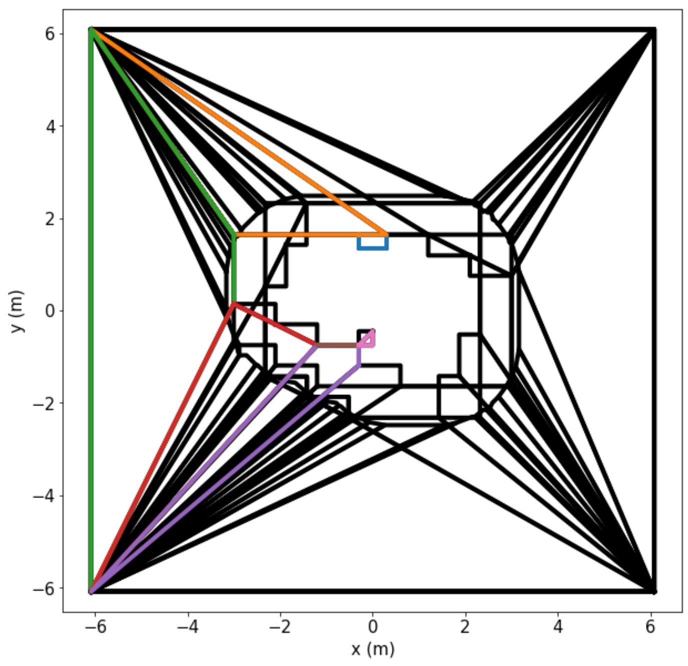
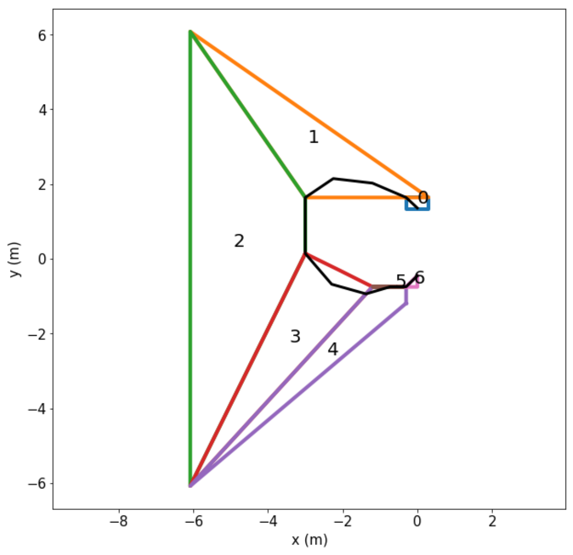


Figure 5-17: Shapeshifting Plan 7 Graph-search: selected corridor hulls colored

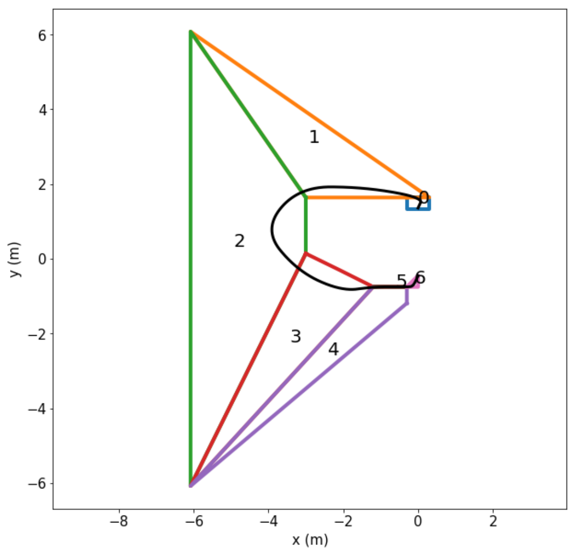
I also reviewed the trajectories for plan 7 in detail, the rhombus to block path that involves a shape of 12 boats separating into two triangles, with one of the triangles moving to generate a rectangle. The selected hulls from the graph search are demarcated in Fig. 5-17. The resulting trajectories for each optimization variant for plan 7 are shown in Fig. 5-18. As expected, the line-based trajectory produces the most jagged trajectories, which would tracking it hard and potentially impossible. The B-spline trajectory, is more continuous but takes a relatively long path which is slow. This is unnecessary, as a safety buffer has already been accounted for. The refined B-spline yields the best combination of these factors, producing a smooth yet short path. The resulting collision-free trail of the boats from the refined B-spline path is visible in Fig. 5-19.

5.2.7 Experimental Evaluation

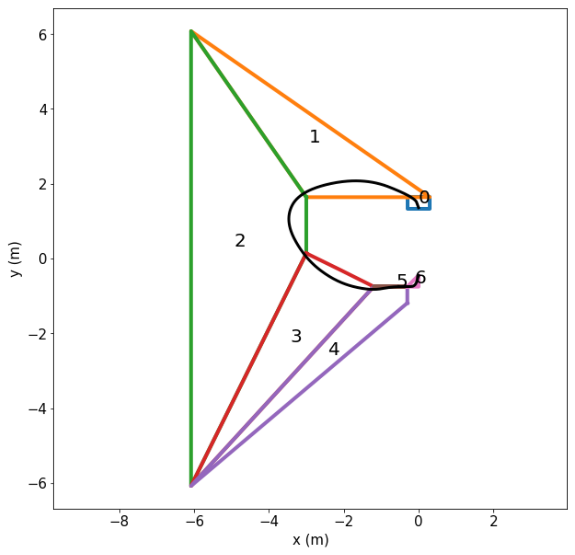
I carried out pool experiments with a setup similar to the previous formation changing section to evaluate the superior refined B-spline method in practice. During these tests, the generated trajectories from simulated plans 9 and 10 were used on the



(a) Line trajectory through selected hulls



(b) B-spline trajectory through selected hulls



(c) Refined B-spline trajectory through selected hulls

Figure 5-18: Shapeshifting Trajectories for Plan 7 Using Different Representations: Line trajectory is short but yields jagged results, while Refined B-spline trajectory is smooth and short

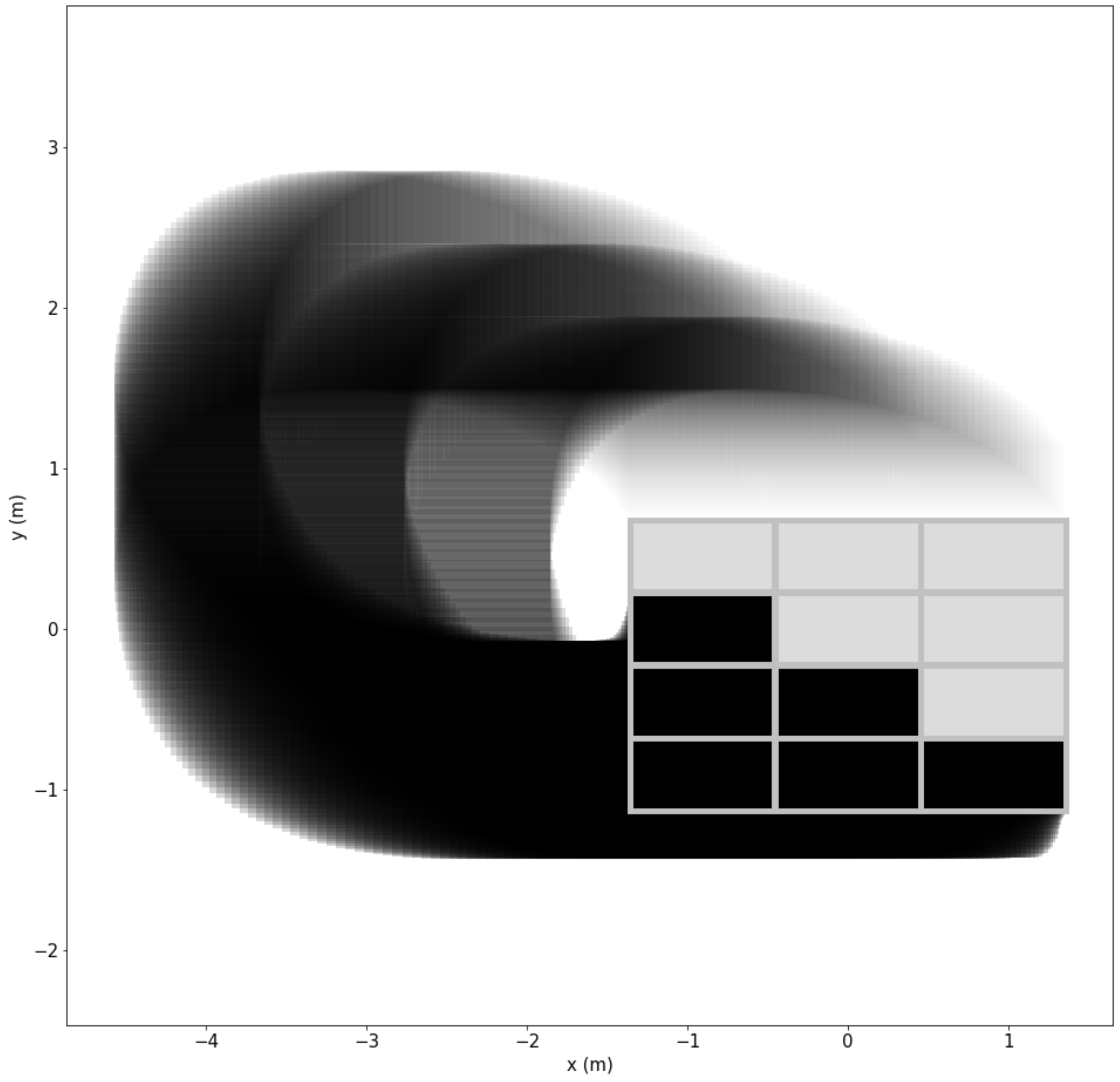
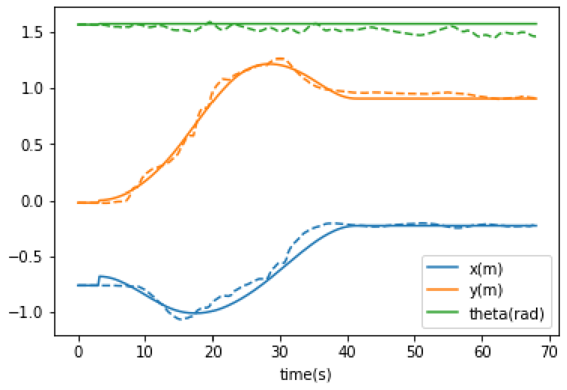


Figure 5-19: Boat Trails During B-spline Trajectory for Plan 7: moving boats in black and static boats in light grey. Final state of moving boat in solid black.

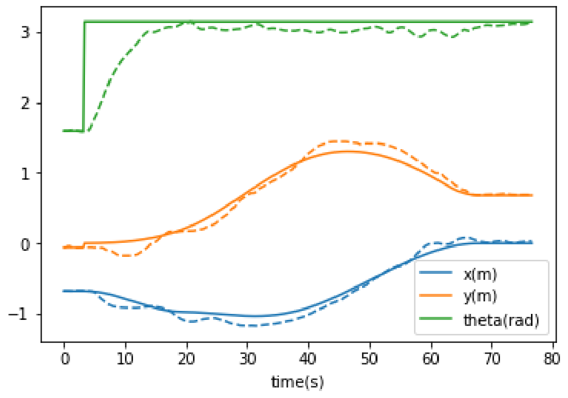
moving boat, and the static boats were fastened to the wall of the swimming pool. In the scaling stage, the trajectories were scaled to a velocity of no more than 0.1 m/s.

In the first test, a formation of two boats shapeshifts from a square to a line, as shown in Fig. 5-21. Five tests were conducted. Four went successfully, and one ended with boats getting stuck right before latching due to friction between the latching parts. The position trajectories, followed by the moving boat during one such test, is pictured in Fig. 5-23.

Next, I carried out pool experiments, changing the shape of two boats from one rectangle to another, as shown in Fig. 5-22. This experiment requires a total of three boats, instead of two, and makes the moving boat rotate. Five tests were conducted and three went successfully. On two occasions, a slight collision occurred between the static boats and moving boat when the moving boat rotated. The position trajectories followed by the moving boat during a successful test is pictured in Fig. 5-24.



(a) Plan 9: 5.9cm, 0.06 radians avg. error

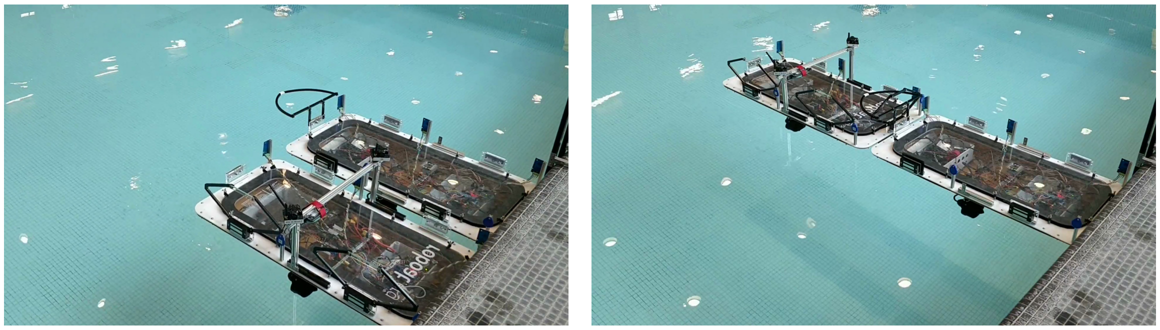


(b) Plan 10: 10.8cm, 0.26 radians avg. error

Figure 5-20: Shapeshift Trajectory Tracking: references solid, actual values dashed

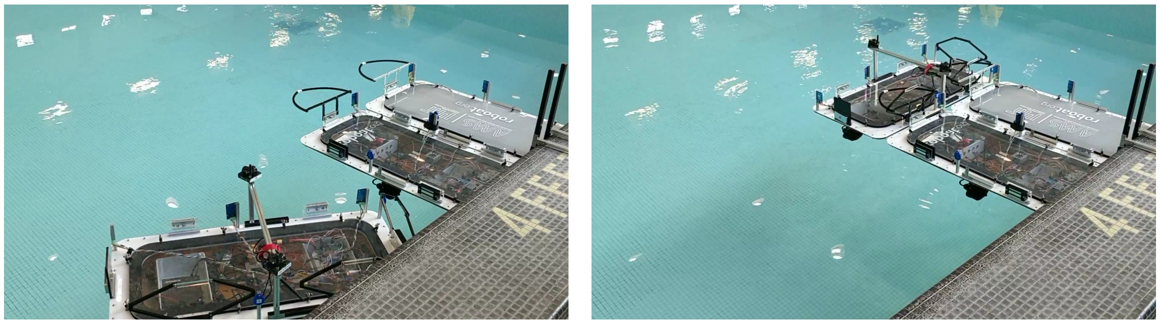
The first series of tests was almost entirely successful, while the second series was more challenging. Looking at the trajectory plots in Fig. 5-23 and 5-24 and evaluating the tracking quality for both tests in Fig. 5-20, tracking is noticeably better in an experiment for the first shapeshift compared to an experiment from the second shapeshift. Particularly apparent is that the average angle tracking error increased from 0.06 to 0.26 radians. Errors in the untuned model, also experienced in the formation changing section, and a relatively fast rotation could be contributing factors to the witnessed light collisions. These can easily be fixed by changing the time scaling used and spending more experimental time identifying the model parameters for the modified Roboat.

Overall, the method described works really well. My formulation of the shapeshifting problem and the trajectory optimization method are robust, even in the presence of tracking error, allowing for trajectories that successfully latch and shapeshift on the water.



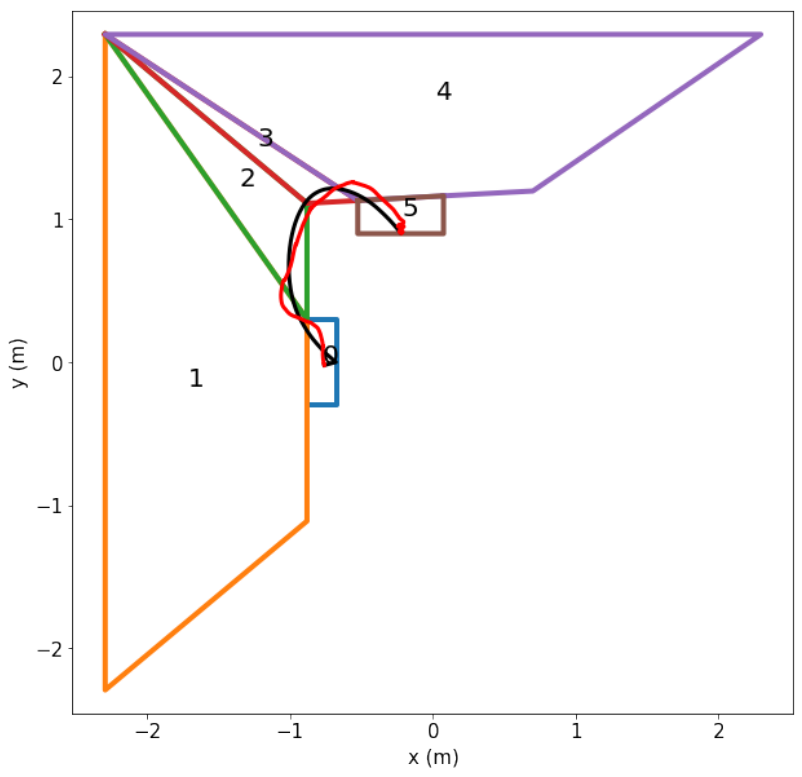
(a) Roboats in Square (b) Roboats in Line

Figure 5-21: Shapeshifting Plan 9: closest boat in (a) tracks the generated trajectory. This takes the Roboats from a square, shown in (a), to a line, shown in (b).

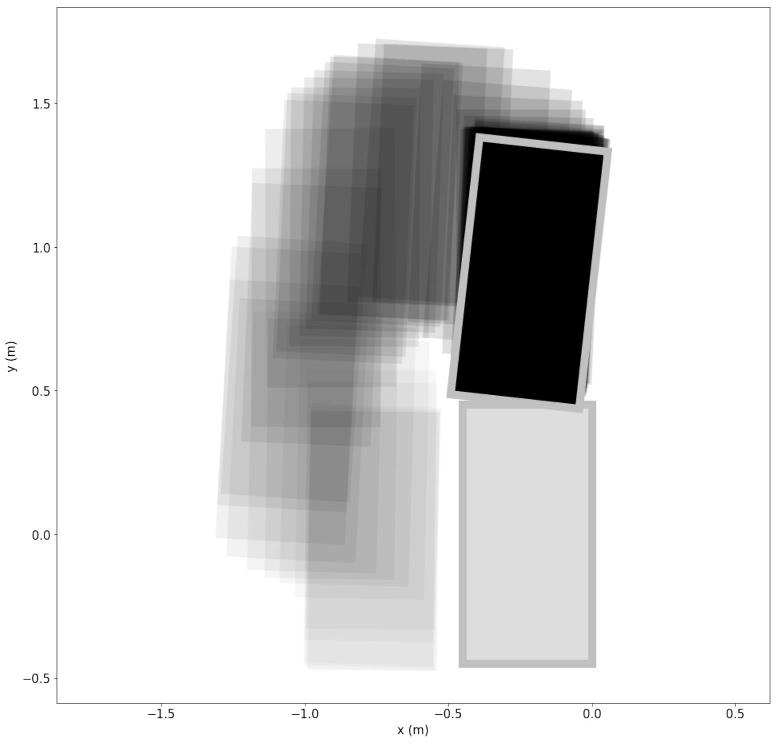


(a) Roboat turning from first rectangle (b) Roboats in new rectangle

Figure 5-22: Shapeshifting Plan 10: closest boat in (a) tracks the generated trajectory. This takes the Roboats from on rectangle to another

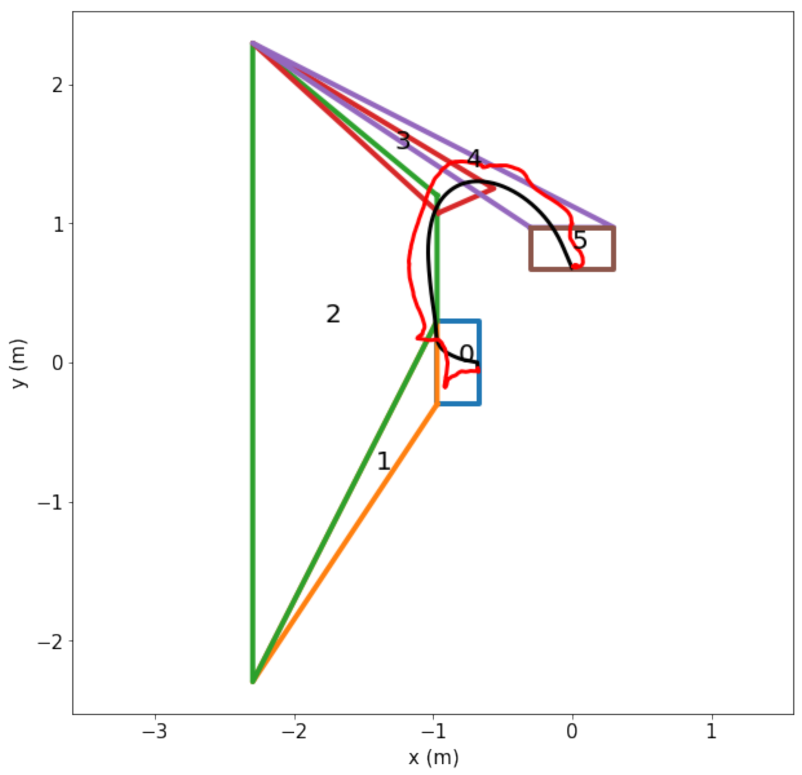


(a) Trajectory Plot: planned B-spline in black, actual in red, with search corridor convex hulls in other colors

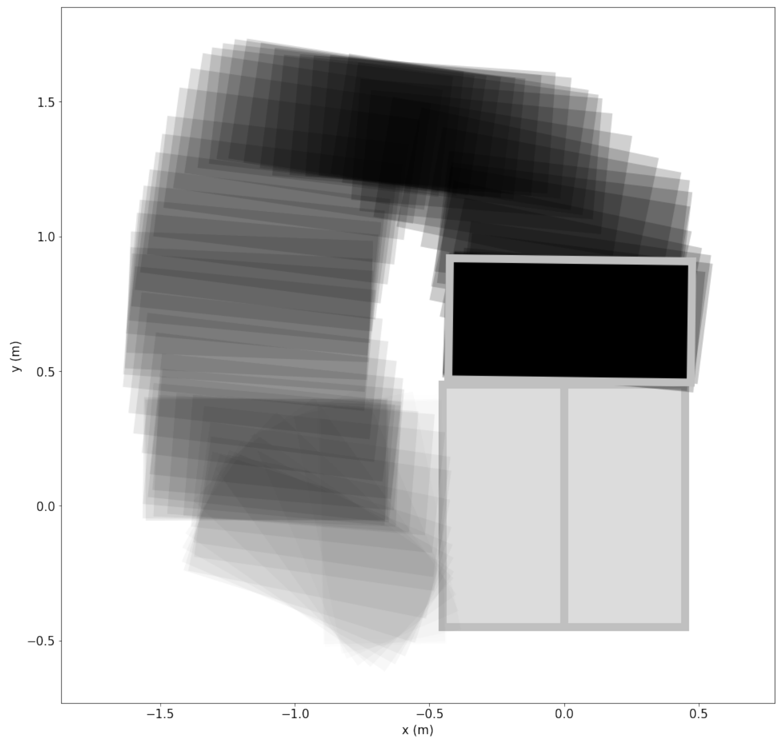


(b) Boat Trails During the Above Actual Trajectory: moving boats in black and static boats in grey

Figure 5-23: Boat Motion During Shapeshift Plan 9



(a) Trajectory Plot: planned B-spline in black, actual in red, with search corridor convex hulls in other colors



(b) Boat Trails During the Above Actual Trajectory: moving boats in black and static boats in grey. Moving boat rotates before relocating

Figure 5-24: Boat Motion During Shapeshift Plan 10

Chapter 6

Conclusion

In this thesis I worked towards developing the system design, as well as perception and planning algorithms to bring reliable autonomy to single and multi-robot autonomous surface vehicle (ASV) systems in urban environments.

In the third chapter, I examined the hardware and software needs of an urban ASV, and contribute to the sensor choice and software architecture necessary for developing full autonomy capabilities in ASVs. The Roboat, our ASV, is able to operate in highly dynamic urban environments and perform precise trajectory planning and tracking, leveling my contribution to the overall design and my sole contribution to the software system design.

In the fourth chapter, I demonstrated a sequence of perception algorithms for filtering, detecting and tracking obstacles LiDAR pointclouds, as well as a variation of this sequence of algorithms for even more precise object pose estimation. The contributions of this chapter include reliable dynamic obstacle detection and tracking, as well as precise object pose estimation on water. Enabled by this perception subsystem, the Roboat is able to reliable avoid obstacle and navigate in narrow urban canal environments.

In the fifth chapter, I presented trajectory optimization algorithms for planning in multi-robot scenarios on water. A multi-boat minimum-makespan formation planning algorithm is presented that allows ASVs to efficiently change from one formation to another using sequential quadratic programming. This work contributes a valuable exploration of concurrent and close motion for formation changing on water. I then conclude by designing an efficient trajectory-planner for the closely related shapeshifting problem, explored for the first time on water. My contribution of a real-time mixed integer programming B-spline-parameterized trajectory planner aids in generating smooth and efficient trajectories and is broadly relevant to all kinds of mobile robots.

As a whole, by drawing on technologies and methods present in other autonomy platforms and designing systems and algorithms for water-based use cases in urban environments, this thesis makes a significant contribution in advancing ASV autonomy and making ASVs more broadly relevant.

6.1 Future Work

From a systems perspective this thesis makes headways in single-ASV system design. This provides a solid groundwork for designing multi-ASV systems, especially those leveraging tight linkages with each other and wider data sources and computing devices in the urban environment to provide cohesive services to the urban populace. Such a cohesive multi-robot systems have been demonstrated in factory and warehouse environments. Further advancements in the design of such systems on the water could enable new kinds of infrastructure. This could play a major part in seamlessly providing citizens with municipal services and giving citizens and planners insights into the underlying infrastructure, as imagined by the Roboat project.

While LiDAR-based perception was heavily explored in this thesis, there is much room for research in the area of multimodal perception on water. LiDAR and RGB-D data are used separately in the current system and could be fused for outlier rejection and increased precision and resolution in detection and tracking. For the pose-estimation task there is room for improvement, given more precise tracking algorithms on water that could for improved robustness to occlusions and low density distant pointcloud returns. In both obstacle detection and tracking, and pose detections good results were obtained, but the limitation of simple features and hand-tuned classical computer vision features were evident. Exploring concise yet expressive models for pattern recognition, such conditional random fields and neural networks, is an exciting avenue of research on water. This could improve tracking and pose detection, and enable object recognition. Improving the robot's ability to reliably recognize features in its environment is critical to improving the Roboat's planning performance in all non-trivial environments.

Lastly, this thesis combines and employs SQP optimization-based motion planning approaches in new domains and shows novel contributions in MIQP spline-based parameterization. Firstly, while trajectory planning across multiple robots was explored in this thesis, the ultimate goal of this system is to provide municipal services. These services are currently contained in the separate application layer, but a Task and Motion Planning optimization approach could achieve synergies between application and trajectory planning, a rarely explored area on the water. Similarly the current approach assumes a separate and reliable localization system, but there is room for making the perception-planning boundary more porous and researching active perception approaches to plan paths that improve perception. Considering perception in planning could allow heterogeneous robots to exceed their shapeshifting abilities and plan motion across an entire unlatched formation, by planning paths that allow blind-robots to stay localized.

As a whole, the development of the Roboat urban autonomy system and novel algorithms described in this thesis, leave open many exciting avenues for exploration in Amsterdam's and other cities' waterways.

Appendix A

Software Packages

Included here is an extensive but non-exaustive list of software packages used. There are several minor depedendencies that are installed as subdependencies and may not be listed here.

A.1 Urban Autonomy System and Perception

- System Packages

  ACADO toolkit C++ optimization and control toolkit with C code generation functionality

  eigen C++ library for linear algebra

  fast csv parser C++ header-only library for reading csv files

  git version control tool

  robot operating system (ROS) Kinetic recent LTS version of popular robotics middleware

  openssh server ssh server for accessing Roboat

  terminator powerful multi-terminal emulator terminals

  vim powerful text editor that runs in terminal

- Roboat ROS Packages[1]

  roboat_acado enerates MPC controllers for roboat_core with ACADO code generation

  roboat_autonomy] filters LiDAR data based on the boat environment and task. Also includes, visual odometry launch files for rtabmap.

  roboat_core manages sensors, low-level serial communication and actuator control

  roboat_launch coordinates the execution of a task setup amongst by calling other packages with the relevant configuration

  roboat_localization localizes the boat with LiDAR, IMU and Camera data. It manages Autoware's NDT for LiDAR scan matching and employs robot localization for EKF filtering and rtabmap visual odometry

  roboat_utils contains utility scripts for commanding the boat with a joypad from a laptop, configuration files and an installer script for the Roboat setup

- Additional ROS Dependencies

  apriltags2 QR tag-based pose estimation

  autoware open-source autonomous car platform

  joy joypad drivers

  point cloud lib (PCL) pointcloud data formats and processing tools

  microstrain_3dm_gx5_45 IMU drivers. Includes 3dm_gx5_25 drivers.

  realsense Intel realsense RGB-D camera drivers for ROS

  rosserial ROS message communication over serial

  rtabmap visual odometry algorithms interface

  serial serial communication

[1]these packages may or may not be publicly available

velodyne velodyne LiDAR drivers and ROS messages

wstool ros workspace dependency manager

A.2 Trajectory Optimization

- System Dependencies

Docker virtualization tool for managing container-based software environments

Drake C++ Robotics Toolbox with python bindings. Includes flexible optimization solver interfaces.

Mosek solvers for many kinds of optimization, including MIQP

SNOPT nonlinear optimization solver

- Python Packages

graphtool constructs and computes on large graphs

icp icp implementation. Hosted by ClayFlannigan on Github.

ipython browser-based interface for notebook programming

matplotlib plotting and visualization tool

numpy matrix-based numerical tools and algorithms

pypolypart python interface for polypart, a C++, based polygon partitioning library. hosted by chozabu on Github

scipy scientific computing tools

shapely manipulates and analyzes planar geometric objects

tabulate printing table-formatted data

- Developed Packages

multiboatTrajectoryOptimization implements methods for formation changing described in chapter 5. Release planner on Github `https://github.com/bgheneti/MultiboatTrajectoryOptimization`

shapeshifting implements methods for shapeshifting described in chapter 5.

Release planned on Github

`https://github.com/bgheneti/shapeshifting`

Bibliography

[1] BestPractices - ROS wiki. `http://wiki.ros.org/BestPractices`. Accessed: 2018-12-3.

[2] roslaunch/Tutorials/Roslaunch tips for larger projects - ROS wiki. `http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects`. Accessed: 2018-12-3.

[3] M R Benjamin, J A Curcio, J J Leonard, and P M Newman. Navigation of unmanned marine vehicles in accordance with the rules of the road. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3581–3587, May 2006.

[4] Paul J Besl and Neil D McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, February 1992.

[5] D Brugali and A Shakhimardanov. Component-Based robotic engineering (part II). *IEEE Robot. Autom. Mag.*, 17(1):100–112, March 2010.

[6] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1476–1483, May 2016.

[7] Y Chen, M Cutler, and J P How. Decoupled multiagent path planning via incremental sequential convex programming. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5954–5961, May 2015.

[8] Ugo Conti and Mark Gundersen. Second generation design of wave adaptive modular vessels (WAM-V®): A technical discussion of design improvements. In *Proceedings of 11th International Conference on Fast Sea Transportation, Honolulu, Hawaii, USA*, pages 772–776, 2011.

[9] R Deits and R Tedrake. Efficient mixed-integer planning for UAVs in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 42–49, May 2015.

[10] Peter R (peter Raymond) Florence. Integrated perception and control at high speed. 2017.

[11] M E Flores and M B Milam. Trajectory generation for differentially flat systems via NURBS basis functions with obstacle avoidance. In *2006 American Control Conference*, pages 7 pp.–, June 2006.

[12] Thor I Fossen. *A Guidance and Control of Ocean Vehicles.* August 1994.

[13] Duncan Frost and Jules-Raymond Tapamo. Detection and tracking of moving objects in a maritime environment using level set with shape priors. *EURASIP Journal on Image and Video Processing*, 2013(1):42, July 2013.

[14] T C Furfaro, J E Dusek, and K D von Ellenrieder. Design, construction, and initial testing of an autonomous surface vehicle for riverine and coastal reconnaissance. In *OCEANS 2009*, pages 1–6, October 2009.

[15] Philip E Gill, Walter Murray, and Michael A Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 12(4):979–1006, April 2002.

[16] Sean Gillies and Others. Shapely: manipulation and analysis of geometric objects.

[17] Rohit Girdhar, Georgia Gkioxari, Lorenzo Torresani, Manohar Paluri, and Du Tran. Detect-and-Track: Efficient pose estimation in videos. December 2017.

[18] B J Guerreiro, C Silvestre, R Cunha, and A Pascoal. Trajectory tracking nonlinear model predictive control for autonomous surface craft. *IEEE Trans. Control Syst. Technol.*, 22(6):2160–2175, November 2014.

[19] E Hawkes, B An, N M Benbernou, H Tanaka, S Kim, E D Demaine, D Rus, and R J Wood. Programmable matter by folding. *Proc. Natl. Acad. Sci. U. S. A.*, 107(28):12441–12445, July 2010.

[20] H K Heidarsson and G S Sukhatme. Obstacle detection and avoidance for an autonomous surface vehicle using a profiling sonar. In *2011 IEEE International Conference on Robotics and Automation*, pages 731–736, May 2011.

[21] Stefan Hertel and Kurt Mehlhorn. Fast triangulation of the plane with respect to simple polygons. *Information and Control*, 64(1):52–76, January 1985.

[22] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit—an open-source framework for automatic control and dynamic optimization. *Optim. Control Appl. Methods*, 32(3):298–312, 2011.

[23] V Indelman, E Nelson, J Dong, N Michael, and F Dellaert. Incremental distributed inference from arbitrary poses and unknown data association: Using collaborating robots to establish a common reference. *IEEE Control Syst. Mag.*, 36(2):41–74, April 2016.

[24] Pablo Iñigo-Blasco, Fernando Diaz-del Rio, M Carmen Romero-Ternero, Daniel Cagigas-Muñiz, and Saturnino Vicente-Diaz. Robotics software frameworks for multi-agent robotic systems development. *Rob. Auton. Syst.*, 60(6):803–821, June 2012.

[25] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. May 2011.

[26] S Kato, E Takeuchi, Y Ishiguro, Y Ninomiya, K Takeda, and T Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, November 2015.

[27] KDE Group, University of Kassel, DMIR Group, University of Würzburg, L3s Research Center, and Hannover (Germany). Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems | BibSonomy. `https://www.bibsonomy.org/bibtex/2fa5ba14bf8f8ea50db52df64bac6f184/achakraborty`, 2016. Accessed: 2018-5-18.

[28] Monroe Kennedy, Dinesh Thakur, Vijay Kumar, M Ani Hsieh, and Subhrajit Bhattacharya. Optimal paths for polygonal robots in SE(2). In *ASME 2017 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V05BT08A075–V05BT08A075. American Society of Mechanical Engineers, August 2017.

[29] Jacques C Leedekerken, Maurice F Fallon, and John J Leonard. Mapping complex marine environments with autonomous surface craft. In Oussama Khatib, Vijay Kumar, and Gaurav Sukhatme, editors, *Experimental Robotics: The 12th International Symposium on Experimental Robotics*, pages 525–539. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[30] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, Olivier Koch, Yoshiaki Kuwata, David Moore, Edwin Olson, Steve Peters, Justin Teo, Robert Truax, Matthew Walter, David Barrett, Alexander Epstein, Keoni Maheloni, Katy Moyer, Troy Jones, Ryan Buckley, Matthew Antone, Robert Galejs, Siddhartha Krishnamurthy, and Jonathan Williams. A perception-driven autonomous urban vehicle. *J. Field Robotics*, 25(10):727–774, October 2008.

[31] Markus List, Peter Ebert, and Felipe Albrecht. Ten simple rules for developing usable software in computational biology. *PLoS Comput. Biol.*, 13(1):e1005265, January 2017.

[32] Sikang Liu, M Watterson, S Tang, and V Kumar. High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1484–1491, May 2016.

[33] Zhixiang Liu, Youmin Zhang, Xiang Yu, and Chi Yuan. Unmanned surface vehicles: An overview of developments and challenges. *Annu. Rev. Control*, 41:71–93, January 2016.

[34] J E Manley. Unmanned surface vehicles, 15 years of development. In *OCEANS 2008*, pages 1–4, September 2008.

[35] M H Mat Idris, M I Sahalan, M A Abdullah, and Z Z Abidin. Development and initial testing of an autonomous surface vehicle for shallow water mapping. 10(16):7113–7118, January 2015.

[36] Mateos L., Wang W., Gheneti B., Duarte F., Ratti C., and Rus D. Autonomous latching system for robotic boats. *IEEE International Conference on Robotics and Automation (accepted)*, 2019.

[37] D Mellinger, A Kushleyev, and V Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE International Conference on Robotics and Automation*, pages 477–483, May 2012.

[38] Wasif Naeem, Robert Sutton, and Tao Xu. An integrated multi-sensor data fusion algorithm and autopilot implementation in an uninhabited surface craft. *Ocean Eng.*, 39:43–52, January 2012.

[39] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. A survey of multi-agent formation control. *Automatica*, 53(C):424–440, March 2015.

[40] I O'Hara, J Paulos, J Davey, N Eckenstein, N Doshi, T Tosun, J Greco, J Seo, M Turpin, V Kumar, and M Yim. Self-assembly of a swarm of autonomous boats into floating structures. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1234–1240, May 2014.

[41] E Olson. AprilTag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407, May 2011.

[42] Georgios Papadopoulos, Hanna Kurniawati, Ahmed Shafeeq Bin Mohd Shariff, Liang Jie Wong, and Nicholas M Patrikalakis. Experiments on surface reconstruction for partially submerged marine structures. *J. Field Robotics*, 31(2):225–244, March 2014.

[43] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin Murphy. PersonLab: Person pose estimation and instance segmentation with a Bottom-Up, Part-Based, geometric embedding model. March 2018.

[44] Tiago Peixoto. The graph-tool python library. *figshare*, 2014.

[45] J W Romanishin, K Gilpin, and D Rus. M-blocks: Momentum-driven, magnetic modular robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4288–4295, November 2013.

[46] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Robotics. programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, August 2014.

[47] R B Rusu and S Cousins. 3D is here: Point cloud library (PCL). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4. ieeexplore.ieee.org, May 2011.

[48] Radu Bogdan Rusu. Semantic 3D object maps for everyday manipulation in human living environments. *KI - Künstliche Intelligenz*, 24(4):345–348, November 2010.

[49] S. Park, E. Kayacan, C. Ratti, and D. Rus. Coordinated control of a reconfigurable multi-vessel platform: Robust control approach. *IEEE Int. Conf. Robot. Autom.*, 2019.

[50] D Saldaña, B Gabrich, M Whitzer, A Prorok, M F M Campos, M Yim, and V Kumar. A decentralized algorithm for assembling structures with modular robots. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2736–2743, 2017.

[51] Tanner Schmidt, Richard Newcombe, and Dieter Fox. DART: dense articulated real-time tracking with consumer depth cameras. *Auton. Robots*, 39(3):239–258, October 2015.

[52] T Schouwenaars, B De Moor, E Feron, and J How. Mixed integer programming for multi-vehicle path planning. In *2001 European Control Conference (ECC)*, pages 2603–2608, 2001.

[53] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and Decision-Making for autonomous vehicles. *Annu. Rev. Control Robot. Auton. Syst.*, 1(1):187–210, May 2018.

[54] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In C Cortes, N D Lawrence, D D Lee, M Sugiyama, and R Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2503–2511. Curran Associates, Inc., 2015.

[55] Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant. Conflict-Based search for optimal Multi-Agent path finding.

[56] Nam P Suh. *Axiomatic design : advances and applications*. Oxford University Press, New York, 2001.

[57] Cheekuang Tam and Richard Bucknall. Cooperative path planning algorithm for marine surface vessels. *Ocean Eng.*, 57:25–33, January 2013.

[58] Russ Tedrake. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems. *Avaliable at: http://drake. mit. edu (accessed March 2014)*, 2014.

[59] David John Thompson. *Maritime Object Detection, Tracking, and Classification Using Lidar and Vision-Based Sensor Fusion.* PhD thesis.

[60] Matthew Turpin, Nathan Michael, and Vijay Kumar. Capt: Concurrent assignment and planning of trajectories for multiple robots. *Int. J. Rob. Res.*, 33(1):98–112, January 2014.

[61] Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Auton. Robots*, 37(4):401–415, December 2014.

[62] Vladyslav Usenko, Lukas von Stumberg, Andrej Pangercic, and Daniel Cremers. Real-Time trajectory replanning for MAVs using uniform b-splines and a 3D circular buffer. March 2017.

[63] Wei Wang, Luis A Mateos, Shinkyu Park, Pietro Leoni, Banti Gheneti, Fabio Duarte, Carlo Ratti, and Daniela Rus. Design, modeling, and nonlinear model predictive tracking control of a novel autonomous surface vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[64] Wang W., Gheneti B., Mateos L, Duarte F., Ratti C., and Rus D. Roboat: An autonomous surface vehicle for urban waterways. *IEEE Robotics and Automation Letters (submitted)*, 2019.

[65] Alicja WaÌğsik, Rodrigo Ventura, José N Pereira, Pedro U Lima, and Alcherio Martinoli. Lidar-Based relative position estimation and tracking for multi-robot systems. In *Robot 2015: Second Iberian Robotics Conference*, pages 3–16. Springer International Publishing, 2016.

[66] Michael T Wolf, Christopher Assad, Yoshiaki Kuwata, Andrew Howard, Hrand Aghazarian, David Zhu, Thomas Lu, Ashitey Trebi-Ollennu, and Terry Huntsberger. 360-degree visual detection and target tracking on an autonomous surface vehicle. *J. Field Robotics*, 27(6):819–833, November 2010.

[67] Jingjin Yu and Steven M LaValle. Multi-agent path planning and network flow. April 2012.

[68] M Zhao, T Anzai, F Shi, X Chen, K Okada, and M Inaba. Design, modeling, and control of an aerial robot DRAGON: A Dual-Rotor-Embedded multilink robot with the ability of Multi-Degree-of-Freedom aerial transformation. *IEEE Robotics and Automation Letters*, 3(2):1176–1183, April 2018.