Developing Autonomy for Unmanned Surface Vehicles

Satyandra K. Gupta

Department of Mechanical Engineering and Institute for Systems Research

University of Maryland, College Park, MD 20742

Technical Report, Simulation Based System Design Laboratory,
University of Maryland, College Park, MD 20742, October 2013

Summary

Autonomous unmanned surface vehicles (USV) are expected to play an ever increasing role in a wide variety of tracking, search and rescue, surveillance, and defense missions. Recent work in artificial intelligence (AI) planning is focused on developing a basic computational understanding of these planning problems and developing corresponding algorithms for facilitating autonomous vehicle operations. However, the current work in AI planning makes highly restrictive assumptions about the domain characteristics. This limits the use of these approaches for solving real-world planning problems. On the other hand, the work in the field of robot system development is focused on developing a working solution for a specific robot platform and planning problem. Hence, the lessons learned from such implementations are difficult to apply to other problems in the same class.

Currently it takes a significant amount of time to develop the best-in-class planner for autonomous vehicle operation. Factors that contribute to this are the following:

- *Selection of an appropriate model for representing the world:* There are many different possibilities for representing obstacles, terrain, and passages in the world. Representative examples includes voxel based representations, boundary representations, and octree representations. In addition to selecting the representation, decisions need to be made regarding the level of model fidelity (i.e., degree of approximation). In addition, often Voronoi diagrams or medial axis need to be computed to facilitate faster reasoning. The representation of the world significantly affects the computational time and plan quality. Unfortunately, different representations prove to be useful in different scenario. So, currently it takes a long time to evaluate different representations and select the best one.

- *Selection of a representation of plans:* In real-time applications, plan representation significantly affects the overall performance. Plans that provide very detailed vehicle moves can be executed by the vehicle without major adaptation. But generating such detailed plans taxes the planning system computationally and hence compromises the plan quality due to time constraints. On the other hand, low-resolution plans require the plan execution system to perform significant adaptation and thus limit how quickly the execution system can take actions. Hence finding the appropriate representation is crucial to getting a good performance from the planning system.

- *Selection of a planning algorithm:* Many different planning algorithms have been proposed for autonomous vehicle operations. These algorithms have different computational performance and capabilities. Uncertainty in the world models and the possibility of catastrophic failures make these planning problems very challenging. The same action in different world-states can have radically different consequences. Hence the effect of an action needs to be modeled as non-deterministic state transitions. Actions that have the possibility of catastrophic

failure in a state need to be handled carefully. Selecting a planning framework that produces the best possible results and avoids catastrophic failures is a time consuming task.

A major issue in the development of increased autonomy for robotic vehicles such as USVs is the time and expense of developing the software necessary to handle a large variety of missions and all the variations in the encountered environments. This is a challenging task and requires writing a significant amount of code by human programmers and extensive parameter tuning.

We aim to find an improved approach for generating action selection policies for autonomous operations. In particular, we are interested in retaining complex characteristics of the real-world problems and developing a framework by combining elements of AI planning and motion planning.

Recent advances in simulation area enable us to use high fidelity simulations to capture the influence of real-world constraints/characteristics on the autonomous vehicle planning problem. It also allows us to carry out an accurate analysis of proposed solutions. The advent of low-cost, high-performance computing architectures enables us to explore a very large number of solutions in a short period of time. Advances in procedural representations enable us to automatically generate complex candidate solutions. Hence, our premise is that high fidelity simulations can now facilitate innovation and discovery process.

This report describes our efforts to develop planners for USVs. Chapter 1 describes an evolutionary computation-based approach for automated action selection policy synthesis for unmanned vehicles operating in adverse environments [1, 2, 3, 4, 5, 6, 7]. Chapter 2 describes GPU based algorithms for computation of state transition models for USVs using 6 degree of freedom dynamics simulations of vehicle-wave interaction [8, 9, 10, 11]. Chapter 3 describes a trajectory planning and tracking approach for following a differentially constrained target vehicle operating in an obstacle field [12, 13]. Finally, Chapter 4 describes a contract-based, decentralized planning approach for guarding a valuable asset by a team of autonomous USVs against hostile boats in an environment with civilian traffic [14, 15].

Chapter 1

Automated Synthesis of Action Selection Policies for Unmanned Vehicles Operating in Adverse Environments

Contributors: Petr Švec and Satyandra K. Gupta

1.1 Introduction

Manual development of a truly robust robotic system operating in an environment with an adversary exhibiting a deceptive behavior is a challenge. This scenario is typical for combat mission tasks where even a single mistake in the decision of the unmanned vehicle can have fatal consequences. In such scenarios, the vehicle has to be prepared to rapidly execute specialized maneuvers in addition to its default strategy in specific situations as defined by its control algorithm, or action selection policy to successfully accomplish its task. The difficulty in the development of such a policy consists in manual handling of the vehicle's failure states that arise in the encountered environments. This requires intensive repeatable testing of the overall vehicle's behavior using a large suite of different test scenarios, identifying the shortcomings, and implementing various contingency-handling behaviors [16].

In this article, we introduce a new approach for automated synthesis of an action selection policy for unmanned vehicles operating in a continuous state-action space. This approach can be viewed as an iterative synthesis process during which an initial version of the policy is automatically generated and then gradually improved by detecting and fixing those shortcomings that have a high potential of causing various task failures. The presented technique belongs to the class of evolutionary methods that directly search for the policy [17], as opposed to computing a value function [18]. In contrast to the majority of the direct policy search methods, our technique utilizes a dedicated local search procedure that finds specific additional macro-actions [19] (in the form of action plans or maneuvers) allowing the vehicle to preemptively avoid the

failure states that cause substantial decrease in the total performance of its default policy.

The iterative detection of failure states and refinement of the policy helped us handle the large, continuous, and in large part fragmented [20] state space of the considered reinforcement learning problem. The fracture of the state space presents a significant challenge for standard evolutionary algorithms [21] to evolve a well-performing policy, since the vehicle may be required to execute very different actions as it moves from one state to another. By explicitly detecting the failure states, we are able to identify the subspaces of the state space that possess the characteristics of high fracture that hampers the generalization of policy actions. Once the failure states are identified using multiple evaluation runs, they are traced back in time to find exception states from which new macro-actions can be executed. The use of macro-actions in contrast to using primitive actions simplifies and speeds up the synthesis, as there is no need to generate primitive actions for a greater number of states to successfully handle the failure states. The new action plans are readily incorporated into the policy, since they operate over a set of locally bounded and overlapping state space regions. The technique takes advantage of the fact that a large proportion of the state space is not encountered during the actual policy execution, so that the most critical failure states are always handled first. This is similar to the idea of the Prioritized Sweeping technique [22], using which the computation of the value function is focused on important states according to some criteria. We use Genetic Programming (GP) [23] as a discovery component of the macro-actions to be applied locally in the exception states. During the whole process, no external human input on how the policy should be synthesized is therefore needed.

The policy is internally represented as a composite of one default high-level controller and a set of specialized action plans. The default controller is used to control the vehicle's behavior in all states except the states for which specific macro-actions in the form of action plans are needed. The action space of the vehicle is represented by a set of primitive commands, each having continuous parameters. The commands are combined, parametrized, and composed into a structure by the synthesis process to perform the overall task. The inherent symbolic representation of the policy greatly simplifies the analysis of its behavior. In addition, the symbolic representation allows integrating human knowledge and the analysis of the policy can provide the basis for improving the code.

Our approach was tested in the context of a large project aimed at the development of a general mission planning system [24, 5] for automated synthesis of action selection policies for Unmanned Surface Vehicles (USVs) [25, 26]. In this chapter, we specifically focus on automated synthesis of a policy used for blocking the advancement of an intruder boat toward a valuable target. This task requires the USV to utilize reactive planning complemented by short-term forward planning to generate local action plans describing specific maneuvers for the USV. The intruder is human-competitive in the sense that its attacking efficiency approaches the attacking efficiency of deceptive strategies exhibited by human operators. Our aim is to reach the level 3 of autonomy as defined in [27]. In this level, the unmanned vehicle automatically executes mission-related commands when response times are too short for

operator intervention.

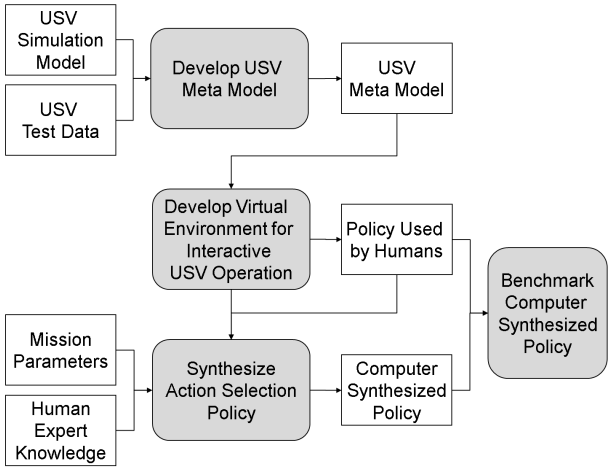


Figure 1.1: Overview of the overall approach for synthesis of an action selection policy for USVs

1.2 Problem Formulation

We are interested in synthesizing action selection policies suitable for solving sequential decision tasks that have highly fragmented state spaces [20] and require high reactivity in planning. The policies should thus allow the unmanned vehicles to quickly respond to the current state of the environment without any long-term reasoning about what actions to execute. In addition, the policies need to be represented symbolically in order to simplify their audit.

The task for the synthesis is to find a policy $\pi : S \rightarrow A$ that maps macro-actions A (also known as temporally extended actions that last for one or more time steps N) [28] to states S, such that the expected total discounted reward r for the task is maximized. The total reward r is computed as cumulative reward by taking into account rewards r_{s_t,a_t} of individual macro-actions $a_t \in A$ taken from states $s_t \in S$. The policy π consists of one default policy π_D and a set $\{\pi_i\}_{i=1}^{n}$ of n additional specialized policy components $\pi_i : S_i \rightarrow A$ that map macro-actions A to a specific set of overlapping regions $\Pi = \{S_1, S_2, ..., S_n\}$ of the state space S.

We define a failure state $s_F \in S$ as a state in which the vehicle acquires a large negative reward due to violation of one or more hard C_h or soft C_s task constraints during the execution of a given task. The violation of even a single hard task constraint $c_h \in C_h$ (e.g., hitting an obstacle) guarantees that the vehicle will not be able to recover from its failure, and thus will not successfully finish its task. On the other hand, the violation of a soft task constraint C_s (e.g., losing a chance to block the intruder for a small amount of time) also results in a negative reward but does not terminate the task execution, since the vehicle can compensate for this type of task constraint violation in further planning. In both cases, the vehicle can avoid

the violation of the constraints by executing appropriate pre-emptive macro-actions A from a special set of representative exception states $S_{E,REP} \subset S$.

More formally, there exist m sequences

$$P = \{p_i | p_i = (s_{i,1}, a_{i,1}, s_{i,2}, a_{i,2}, \dots s_{i,l-1}, a_{i,l-1}, s_{F,i,l})\}$$

of states and atomic actions (as special cases of macro-actions with the length of one time step) representing transition paths in the state-action space that gradually attract the vehicle to failure states

$$S_F = \{s_{F,1}, s_{F,2}, \dots, s_{F,m}\} \subset S$$

if no additional macro-actions A are executed. We define a state of exception $s_{E,i}$ for a path p_i as the most suitable state from which a corrective macro-action $a_c \in A$ can be executed to deviate p_i from that state towards the states that result in the highest achievable expected reward. The state of exception $s_{E,i}$ can be found by searching in reverse from its corresponding failure state $s_{F,i}$ in p_i, which is equivalent to going back in time. Further, we define S_E as a set of exception states for all paths P leading to failures, and $S_{E,REP}$ as a representative set of close-by exception states from which the greatest number of failure states can be avoided.

The above formulation presented a general policy synthesis problem. Now, we will define a special case for the blocking task. In this case, the task is to automatically synthesize an action selection policy $\pi_U : S \to A_U$ for a USV to block the advancement of an intruder boat toward a particular target. The policy π_U should maximize the expected total reward r expressed as the maximum pure time delay the USV can impose on the intruder. The intruder executes a policy $\pi_I : S \to A_I$ that exhibits a deceptive attacking behavior. This prevents the USV from exploiting regularity in the intruder's actions. For this task, the USV needs to utilize reactive planning to be able to immediately respond to the current pose of the intruder boat by executing appropriate blocking maneuvers. The blocking maneuvers implement macro-actions A_U as defined by our framework. In this particular case, we define a failure state s_F as a state in which the intruder is closer to the target than the USV. The defined failure state thus covers both the soft and hard task constraints violations.

1.3 USV System Architecture

The developed policy synthesis approach is closely coupled to the underlying system architecture of the USV. This architecture consists of several modules that are responsible for different tasks, e.g. sensing, localization, navigation, planning, behavior control, communication, human interaction, or monitoring [25, 26].

The USV utilizes a reactive controller with short-term forward planning for quick execution of maneuvers to be able to immediately respond to the state of the environment. This is in contrast to using a purely planning controller that deliberates about what sequence of actions to execute. The reasoning process of this type of controller would consume considerable amount of time since the planner would need to compute

a number of candidate sequences of actions, evaluate each of them, and choose the one with the highest utility. In this work, we use the term short-term forward planning to emphasize the fact that the vehicle's action selection policy produces a sequence of time-varying actions (i.e., in the form of macro-actions) without any deliberation (as it is in the case of the purely planning controller), as opposed to directly executing individual primitive actions in each vehicle's state. This speeds up and simplifies the policy synthesis (see Section 1.4) since there is no need to repeatedly generate multiple primitive actions for a single representative exception state to successfully handle the corresponding failure state. The macro-actions cover a larger subset of the state-action space and thus simplify and increase the speed of the policy synthesis process.

The high simulation speed of the USV's dynamics model is critical for policy synthesis and therefore we use its simplified version with 3 degrees of freedom. This simplified model has been adapted from the full-blown 6 degrees of freedom USV dynamics model as described in [10]. The full-blown model considers ocean disturbances and is used for high-fidelity physics-based real-time simulations inside the virtual environment.

1.3.1 Virtual Sensor Models

The planning system needs to utilize only relevant key sensory information abstracted from the raw sensor data. This information is represented as a vector of features of different types (e.g., numerical or boolean) required for a successful fulfillment of the mission task. The values of the features are computed by a predefined set of virtual sensors [29] that process raw sensory data. The features can have one or more parameters using which their values are computed (e.g., features of the boolean type).

The planning system uses virtual visibility, relational, and velocity sensors. The virtual visibility sensor is a detection sensor with cone-shaped detection regions (see Fig. 1.2a). The size of the overall sensor area is defined by its reach and range parameters. Each region returns a boolean value expressing the presence of other objects and a normalized distance to the closest object. The relational virtual sensor provides relevant information on how other objects are situated with respect to the USV or to each other. It computes boolean values to be stored as values of the relational features. The velocity virtual sensor returns the velocities of other objects inside the environment.

1.3.2 Planning Architecture

USV State Definition

The full state of the USV (see Fig. 1.2b) is represented by an 8-dimensional vector $s = [\alpha_1, \alpha_2, \phi_U, \phi_I, v_1, v_2, \gamma, d]$ of state variables that encode attributes of the environment for the task. The angle α_1 represents the angle between the USV's heading and the direction to the target, α_2 is the angle between the intruder's heading and the direction to the target, ϕ_U is the USV's steering angle, ϕ_I is the intruder's steering

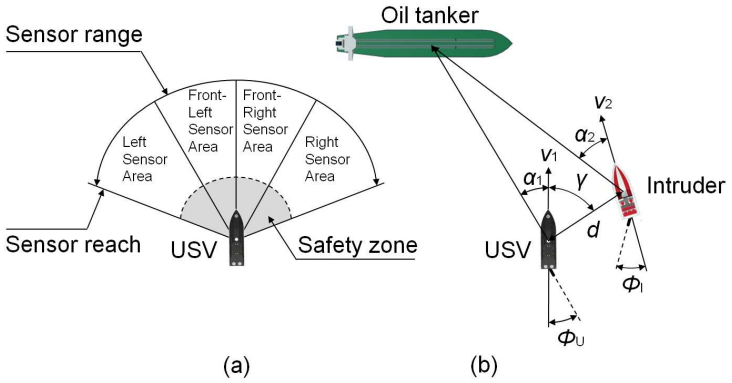


Figure 1.2: (a) USV visibility sensor model, (b) USV's state in respect to the intruder and target

angle, v_1 is the USV's translational velocity, v_2 is the intruder's translational velocity, γ is the angle between the USV's heading and the direction to the intruder, and d is the distance between the USV and the intruder.

Policy Representation

The policy π_U allows the USV to make a decision from a set of allowable actions based on the observed world state. The default policy π_D is represented as a high-level controller (see Fig. 1.3b left). This controller is made by a decision tree that consists of high-level parametrized navigation commands NC, conditional variables CV, standard boolean values and operators BVO, program blocks PB, and system commands SC (see Tab. 1.1). The leaves of the decision tree can be conditional variables, boolean values, navigation commands, or system commands. The inner nodes can be boolean operators or program blocks.

The additional policy components are represented as action plans that are sequences of parametrized navigation NC and system SC commands, and can be grouped into program blocks PB. The action plans are internally represented as trees so that the same synthesis mechanism used for generating the default controller can also be used for generating the plans. The leaves of the tree can be navigation or system commands.

The navigation and system commands have parameters, for example, the positional navigation commands (e.g., *go-intruder-front*) are defined using five parameters, where the first two parameters represent the USV's relative goal position (in polar coordinates) around the intruder. This effectively allows the vehicle to cover all feasible positions, as defined by its policy in a certain area around the intruder. The next two parameters represent a cone-shaped blocking area around the relative goal position. Once the vehicle gets inside the blocking area, it starts slowing down to limit the intruder's movement. The last parameter represents the length of the command execution. The turning left/right action commands have two parameters that represent the desired steering angle and the length of the command execution. The translational velocity of the vehicle is explicitly controlled by the ve-

NC	go-intruder-front (front-left, front-right, left, right) turn-left (right) go-straight
CV	intruder-on-the-left (right, front, front-left, front-right, at-the-back, at-the-back-left, at-the-back-right) intruder-has-target-on-the-left (right) usv-has-target-on-the-left (right) usv-intruder-distance-le-than usv-closer-to-target-than-intruder usv-facing-intruder
BVO	if, true, false, and, or, not
PB	seq2, seq3
SC	usv-sensor, usv-velocity, usv-match-intruder-velocity

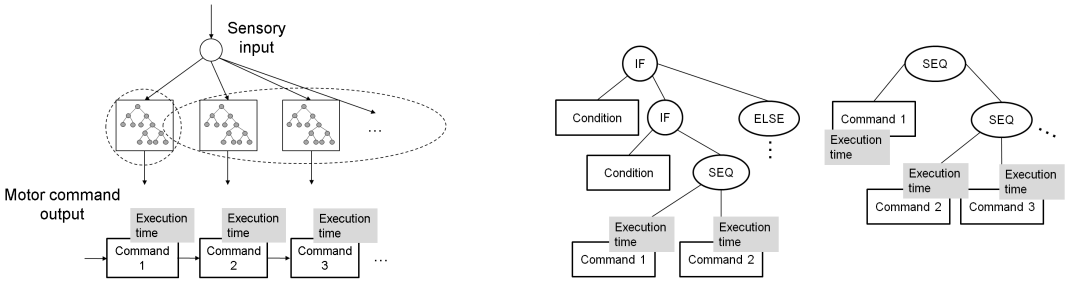
Table 1.1: Policy primitives

locity commands, where the *usv-velocity* command sets an absolute velocity given as a parameter, whereas the command *usv-match-intruder-velocity* sets a velocity given as a parameter relatively to the current velocity of the intruder. The *usv-sensor* system command changes the parameters of the virtual visibility sensor and thus allows it to explicitly control the risk level of the obstacle avoidance behavior.

Policy Execution

During the mission, the USV periodically senses its surroundings and classifies its current state with respect to the intruder and the target. The classification mechanism of the policy executor decides whether the current state is close enough to one of the states for which a corresponding action plan exists. If such a plan exists, the policy executor directly executes the plan, otherwise it executes the default controller to generate a new sequence of actions. The decision whether to execute a specific plan depends on the activation distance parameter δ. This parameter defines the minimal distance that has to be achieved between the current USV's state and any state in the predefined set to activate a specific plan. The state space $S = S_1 \cup S_2$ is thus divided into two regions where in the first region S_1 the USV generates and executes plans using the default controller, whereas in the other region S_2 the USV directly executes previously generated or manually defined plans. The distance between normalized states is computed using the standard Euclidean distance metric.

The input into the policy is data from the virtual sensors that compute the values of all conditional variables. So, for example, the boolean value of the *intruder-on-the-left* variable is directly provided by the virtual relation sensor while the value of the *intruder-velocity-le-than* parametrized variable is provided by the virtual velocity sensor.



(a) Overall policy that consists of one default controller and a set of action plans. The product of the execution is a sequence of action commands (bottom)

(b) Example of a default controller (left) and an action plan (right)

Figure 1.3: Policy representation

Planning and Control Architecture

By acquiring and processing sensor data in short-term cycles, and planning, the planning and control system (see Fig. 1.4) determines an action command to be executed to direct the vehicle inside the environment. The policy executor of the system takes as inputs sensor data, mission parameters, the USV meta model, and the action selection policy. It decides which component of the policy to execute to generate an action plan based on the current state of the vehicle. The plan consists of a number of action commands, each being executed for a certain amount of time. The ultimate output of an activated command is a sequence of motion goals $G = \{g_1, \ldots, g_n\}, g_i = [x, y, \theta, v, t]$, where $[x, y, \theta]$ is the desired pose of the vehicle, v is the velocity, and t is the time of arrival. The motion goals are directly processed by the trajectory planner. The computed trajectories are consequently executed by a low-level controller to generate corresponding motor commands for device drivers of a particular actuator.

The planning and control architecture consists of planning, behavior, trajectory planning, and low-level control layers (see Fig. 1.4). The planning layer is responsible for interpreting the stored policy, which results in a number of commands queued for execution. The command processor inputs a command from the queue and either activates a corresponding behavior that interprets the command (e.g., in case of navigation commands), or modifies the properties of the trajectory planner (e.g., in case of system commands) in order to change the velocity or sensitivity of the obstacle avoidance mechanism of the vehicle. The commands are usually planned for short-term execution, such as planning of strategic maneuvers. The length of the execution is defined as a parameter of the command. The policy executor remains inactive until all the commands from the queue are processed, in which case the command processor requests new commands from the policy executor.

Each navigation command corresponds to the behavior selected by the behavior selector in the behavior layer. The behaviors produce sequences of motion goals when executed by the behavior executor. In the current architecture, the behaviors

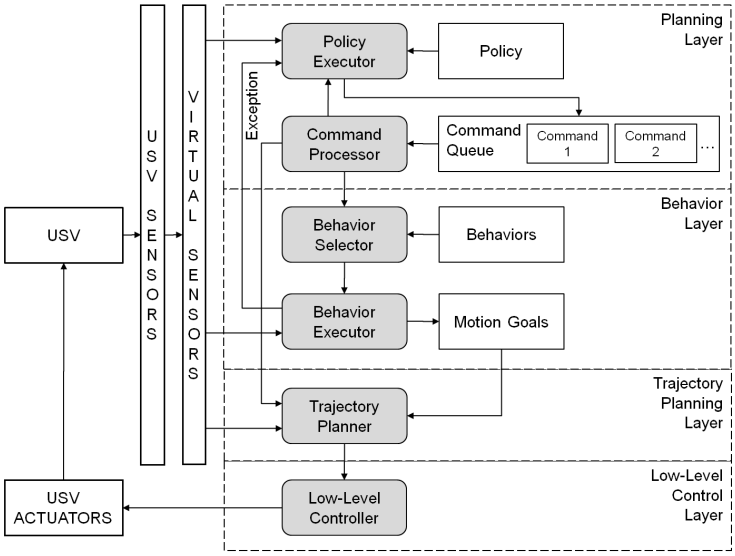


Figure 1.4: USV planning and control architecture

are *turn-left/right*, *go-straight*, and *go-to-intruder*. The trajectory planner receives a motion goal from the behavior executor and computes a feasible trajectory for the vehicle to reach the pose defined by the motion goal with a predefined velocity and within the given time.

The behavior executor is also responsible for monitoring execution of the trajectory and handling exceptions. An exception occurs if the vehicle loses its chance to successfully reach the motion goal as defined by the corresponding behavior. In that case, the command queue is emptied and new commands are supplied by executing the policy.

By default, the behaviors always choose a translational velocity that maximizes the USV's performance. So for example, the *go-straight* behavior uses maximum translational velocity to get to the requested position in the shortest amount of time. The policy can override this velocity by calling the *usv-velocity* or *usv-match-intruder-velocity* system commands.

1.4 Approach

In this section, we describe our approach to policy synthesis, which belongs to the class of evolutionary techniques [17] that directly search for the policy in contrast to computing a value function [18]. In addition to the evolution of a default policy, the presented iterative technique utilizes a specific local evolutionary search that finds additional macro-actions using which the vehicle can avoid frequent task execution failures.

The overall process is completely automated and starts by the synthesis of a default version of the policy π_D (see Fig. 1.5). The policy is then gradually refined

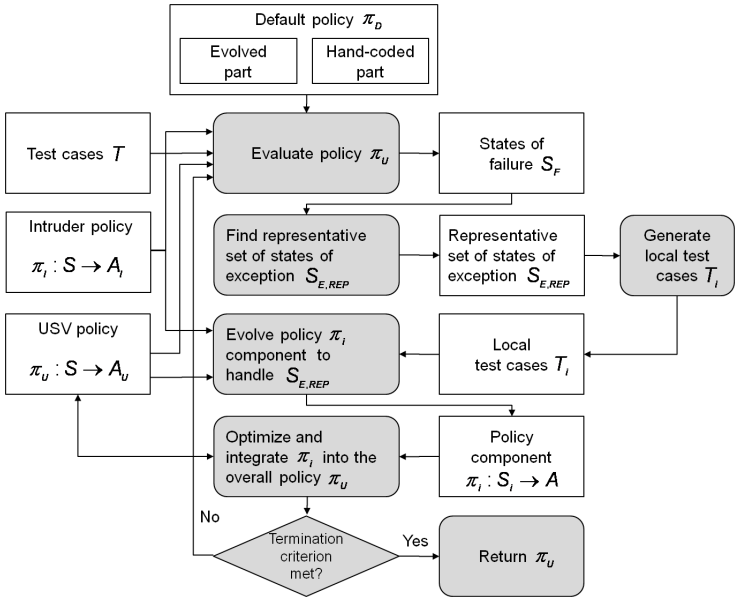


Figure 1.5: Policy synthesis overview

by detecting failure states in which the vehicle has a high probability of failing in its task, and generating new pre-emptive action plans incorporated into this policy to avoid the failure states in future planning.

The synthesis technique exploits the fact that a large proportion of the state space is not frequently encountered during the actual policy execution. This observation together with the use of the default policy makes it possible to partially handle the combinatorial state explosion [21] by evolving locally bounded optimal actions for only specific subsets of the continuous state space. This is further supported by generating macro-actions of variable size (defined by the number of commands and their execution time) in contrast to generating primitive actions, which would require identification of a substantial greater number of exception states to successfully handle the detected failure states. Hence, this results in faster synthesis as there is no need to generate primitive actions for every single state to handle the failure states.

The actual algorithm iteratively computes a stationary policy $\pi_U : S \rightarrow A_U$ that attempts to maximize the expected accumulated reward. The main steps of the algorithm are as follows (for a diagram depicting the whole process see Fig. 1.5):

1. Employ GP to evolve an initial version π_D of the policy π_U. The initial version may consists of a hand-coded portion to speed up the synthesis process.

2. Evaluate π_U inside the simulator using m distinct evaluation test cases T. The evaluation returns a set of failure states $S_F = \{s_{F,1}, s_{F,2}..., s_{F,n}\}, n \leq m$ in which the vehicle fails its mission task. Each test case $t \in T$ defines the initial states of the vehicles and determines the maximum number of time steps N for evaluation of the policy.

3. Given S_F, find a corresponding set of exception states $S_E = \{s_{E,1}, s_{E,2}..., s_{E,n}\}$ in whose proximity (given by the activation distance parameter δ) the vehicle has the potential to avoid the failure states S_F if it executes appropriate action plans.

4. Extract a representative set of exception states $S_{E,REP} \subseteq S_E$, as described in detail in Section 1.4.1, in which the vehicle has the potential to avoid the largest number of the detected failure states. In this way, the focus is always restricted to the most critical failure states first while leaving the rest for possible later processing.

5. Generate a set of local test cases T_i to be used during the evolution of a new action plan π_i as described in the next step. Each test case $t \in T_i$ encapsulates a corresponding exception state and determines the maximum number of time steps $N_i \leq N$ for evaluation of the plan.

6. Employ GP to evolve a new action plan $\pi_i : S_i \rightarrow A$ for the representative set of exception states $S_{E,REP}$. This prevents the over-specialization of the plan by evaluating its performance using a sample of states from this set (presented as the test cases in T_i).

7. Optimize the new plan π_i and integrate it into the policy π_U. If the termination condition is not satisfied, continue to step 2. The distance between the normalized states is computed using the Euclidean distance metric.

1.4.1 Extraction of a representative set of exception states

A diagram that describes the extraction process of representative exception states is shown in Fig. 1.6. First, the algorithm finds corresponding exception states S_E for given failure states S_F by reverting back in time for a predefined number of time steps N_τ.

Second, given S_E, the algorithm iterates over all exception states $s_E \in S_E$ and for each of them finds its neighboring states $S_{E,\delta}$ within the activation distance δ. Then, for s_E and the states from its immediate neighborhood $S_{E,\delta}$, the algorithm finds corresponding failure states $S_{F,E}$ together with all their neighbors $S_{F,E,\delta}$ within the distance δ. The algorithm terminates by returning the representative set of exception states $S_{E,REP}$ that is associated with the largest number of corresponding failure states $S_{F,E} \cup S_{F,E,\delta}$. The set $S_{E,REP}$ consists of a representative exception state $s_{E,REP}$ and its immediate neighboring states $S_{E,REP,\delta}$ within the distance δ.

Fig. 1.7 shows an example of a detected representative set of exception states $S_{E,REP}$ (marked by the indicated set of circles) and their corresponding failure states (marked as crosses) connected by relational links. The states are projected to 2D plane by multidimensional scaling that uses the Kruskal's normalized stress criterion [30].

In the context of our mission task, a failure state s_F defines a situation in which the intruder is closer to the target than the USV. Its corresponding exception state

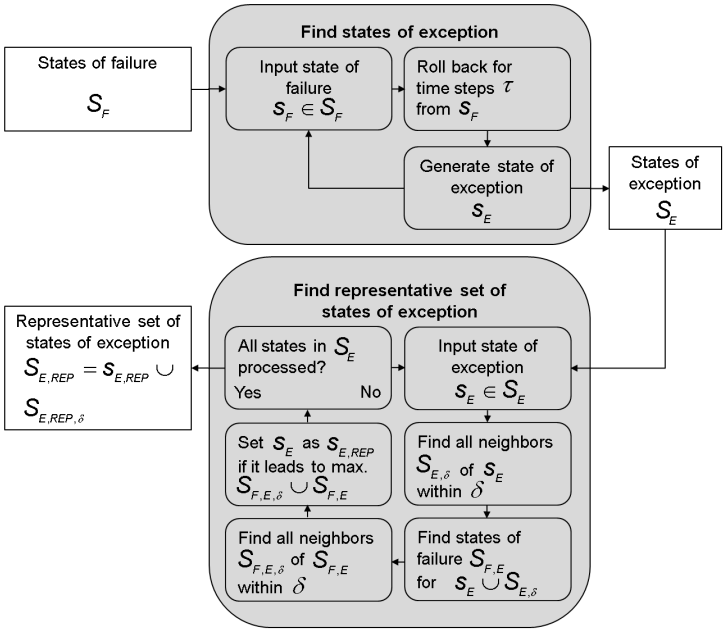


Figure 1.6: Extraction of a representative set of exception states $S_{E,REP}$

s_E is found by reverting back in time for a predefined number of time steps N_τ to record a state from which a new specific action plan can be executed to prevent a possible future failure. The simple way of determining s_E can be further improved by developing a special task-related heuristic that precisely determines a state from which the vehicle will have the highest chance of successfully avoiding the largest number of possible failure states.

1.4.2 Evolution

The evolution searches through the space of possible plan configurations to find the best possible action plan for a particular state. Both the default policy π_D (represented as a controller) and specialized action plans $\{\pi_i\}_{i=1}^n$ as components of the policy are automatically generated using separate evolutionary processes. The specific evolutionary method we used is the strongly-typed Genetic Programming imposing type constraints on the generated Lisp trees [31]. This is a robust stochastic optimization method that searches a large space of candidate program trees while looking for the one with the best performance (so-called the fitness value).

The evolutionary process starts by randomly generating an initial population of individuals represented as GP trees using the Ramped half-and-half method [31]. The initial values of parameters of all action commands and conditionals are either seeded or randomly generated. The default controller π_D of the policy π_U is generated using a human-written template for which GP supplies basic blocking logic. The first portion of the template encodes a maneuver using which the vehicle effectively approaches the intruder at the beginning of the run, as there is no need for it to be explicitly

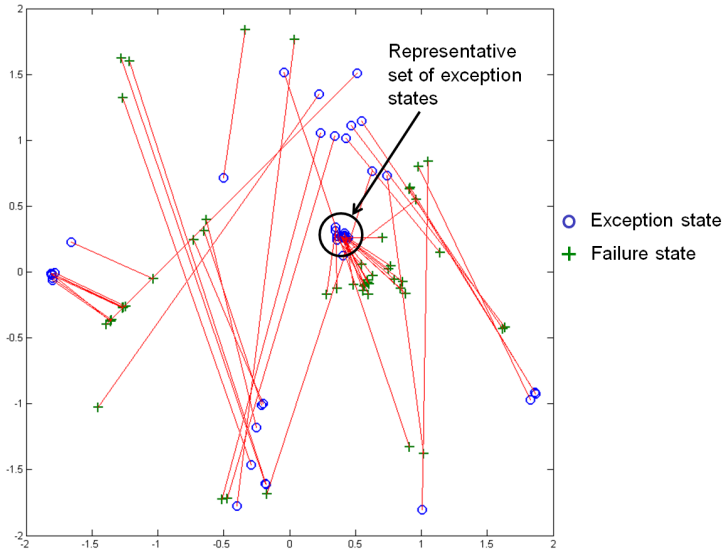


Figure 1.7: Representative set of exception states. The failure states (marked as crosses) and exception states (marked as circles) are projected to 2D plane using multidimensional scaling with the Kruskal's normalized stress criterion. The links between each pair of states express the failure-exception state bindings.

evolved.

The terminal and function sets T and F consist of action commands, system commands, conditional variables, boolean values and operators, and program blocks as defined in section 1.3.2. The sets are defined as

$$T_{default-controller} = T_{plan} = NC \cup SC$$
$$F_{default-controller} = CV \cup BVO \cup PB; F_{plan} = PB$$

Within the population, each individual has a different structure responsible for different ways of responding to its environment. The individual plans are evaluated in the context of the whole policy inside the simulator. The sensory-motor coupling of the individual influences the vehicle's behavior, resulting in a specific fitness value that represents how well the USV blocks the intruder.

We favor individuals which can rapidly establish basic blocking capabilities and optimize them to push the intruder away from the target over the entire trial duration. To do so, the fitness f is defined as the sum of normalized squared distances of the USV from the target over all time steps. If a collision occurs, either caused by the USV or the intruder, the zero fitness value is assigned to the individual, and the selection pressure eliminates the policy component with low-safety guarantee. The fitness function is as follows:

$$f = \frac{1}{N} \sum_{t=1}^{N} \left(\frac{d_i}{d_0} \right)^2 \tag{1.1}$$

Population size / number of generations	500 / 100 (controller) 50 / 30 (plan)
Tournament size	2
Elite set size	1
Min. and max. initial GP tree depth	3 and 6 (controller) 2 and 4 (plan)
Maximum GP tree depth	30 (controller) 10 (plan)
Reproduction prob.	0.1
Crossover prob.	0.84
Structure mutation prob.	0.05
Shrink structure mutation prob.	0.01
Mutation prob. of parameters of action commands	0.5

Table 1.2: GP parameters

where N is the total number of time steps, d_i is the distance of the intruder from the target at time step t, and d_0 is the initial distance of the intruder from the target in a particular test scenario. The total fitness value of the individual is maximized and computed as an average of the fitness values resulting from all scenarios.

The default controller π_D is evaluated using a hand-coded human-competitive intruder in 8 different scenarios. In each scenario, the intruder has a different initial orientation, and the USV always starts from an initial position closer to the target. The evaluation lasts for a maximum number of time steps equal to 300 seconds in real time. The maximum speed of the USV is set to be 10% higher than the speed of the intruder, but other properties of the vehicles are the same. The action plan is evaluated using a sample of states from $S_{E,REP}$ found within the activation distance δ of its corresponding $s_{E,REP}$. The evaluation lasts for a maximum number of time steps equal to 10 seconds in real time.

The individuals in the initial population mostly exhibit a random behavior. By selecting and further refining the individuals with high fitness, their quality gradually improves in subsequent generations. During this process, the individuals are randomly recombined, mutated, or directly propagated to the next generation. These operations are applied with the predefined probabilities (see Table 1.2).

During the policy synthesis, the USV learns the balance between safe and dangerous maneuvering by mutating the reach and range parameters of its virtual visibility sensor. The policy is thus co-evolved with the sensor parameters of the vehicle to

control the obstacle avoidance mechanism.

The optimization of the generated default controller π_D removes all branches of the code that have not been executed during evaluation scenarios. Moreover, each generated action plan π_i is truncated to contain only the action commands that do not exceed the maximum execution time of the plan.

A detailed description of the functionality of all the operators used can be found in [23]. The control parameters of the evolutionary process used for evolution of the default controller and plans are summarized in Table 1.2.

1.5 Computational Experiments

1.5.1 General Setup

We tested the developed approach in the context of a combat mission task during which the USV protects a valuable target against an intruder boat. In this task, the intruder boat has to reach the target as quickly as possible while the USV has to block and delay the intruder for as long a time as possible. We set up an experiment to compare the performance of the automatically generated USV's policy to the USV's policy coded by hand. We compare the performance in terms of pure time delay imposed by the USV on the intruder. To get a fair assessment of the USV performance, the time values being compared must be normalized by a 40-second baseline. This baseline represents the amount of time needed to reach the target if the intruder is completely unobstructed. Any additional time above this baseline thus represents the effective delay time of the intruder when being blocked by the USV.

The policy of the USV is evaluated in 800 runs to account for the intruder's nondeterministic behavior. Each evaluation run lasts for a maximum number of time steps equal to 300 seconds in real time. The dimension of the scene is 800×800 m with the target positioned in the center. At the beginning of each run, the USV and the intruder are oriented toward each other with a random deviation of up to 10 degrees and the USV is positioned on a straight line between the intruder and the target. The initial distance of the USV from the target is approximately 240 m while the intruder's initial distance is 360 m. The USV's maximum velocity is 10 m/s while the intruder's maximum velocity is 9 m/s.

1.5.2 Experimental Protocol

First, we manually implemented an initial version of the intruder's attacking policy and tested it against human players to evaluate its performance in the virtual environment. The policy was further improved in multiple iterations over a period of 6 weeks. Its overall size reached 485 lines of Lisp code. The outline of the policy functionality is described in the next section. We evaluated the performance of the policy by pitting human players against it playing as USVs. The human players achieved 90 seconds of pure time delay on average imposed on the intruder. This shows that the intruder's attacking policy is quite sophisticated as it exhibits a deceptive behavior. If

the intruder's behavior was not deceptive, the human players would have been able to quickly find a motion pattern in the behavior that could be exploited for "indefinite" blocking, and thus not useful for the synthesis.

Second, we implemented the USV's blocking policy against the hand-coded intruder. The policy was improved iteratively over a period of 3 weeks. Its overall size reached 500 lines of Lisp code. The main difficulty when implementing the policy by hand was the manual identification of the most critical cases (or exceptions) in which the policy had a low performance. This is generally non-trivial and requires substantial human effort.

Third, we used the mission planning system to automatically generate the USV's blocking policy using the hand-coded intruder as the competitor. The activation distance parameter δ was set to 0.2 for all action plans. In the current version of the approach, a failure state is determined to be the state in which the intruder is closer to the target than the USV. An exception state is computed by going back in time for 150 time steps from a given failure state.

Finally, we compared the performance of the automatically synthesized USV's policy to the hand-coded one.

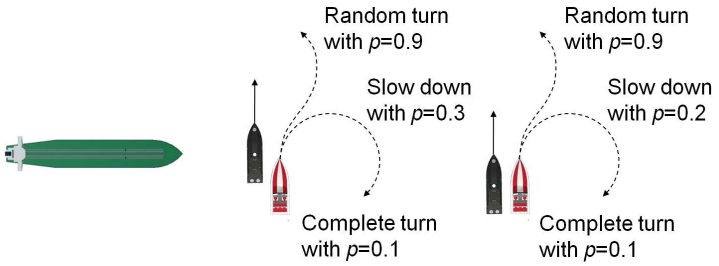
1.5.3 Intruder's Policy

The hand-coded intruder's policy is represented as a single decision tree that contains standard action commands as well as their randomized versions. The intruder's policy is divided into five main sections. Each of these sections handles a different group of situations that can arise during the combat. The first section handles situations in which the distance of the intruder from the target is greater than 130 m and the angle between its translation direction and the target is more than 80 degrees. In these situations, the intruder attempts to rapidly change its direction of movement toward the target by aggressively turning left or right, depending on the relative position of the USV.

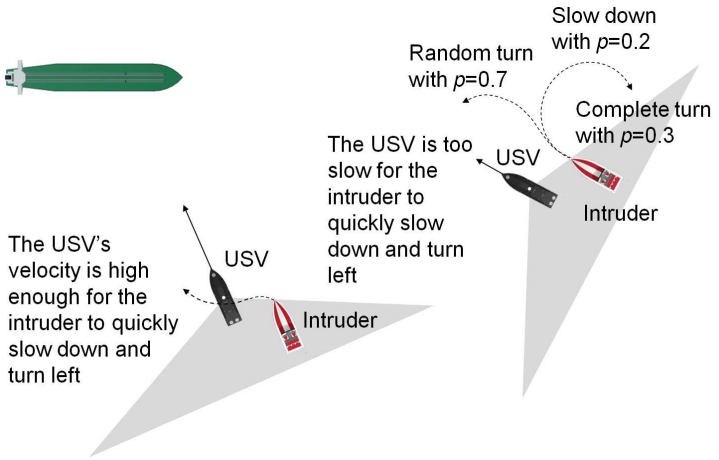
The second section handles situations in which the USV is very close to the intruder, positioned relatively to its front left, and the target is on the intruder's left side (see Fig. 1.8a left). In this case, the intruder has two options. Either it executes a random turn with probability 0.9 or it proceeds with a complete turn. In both cases, the intruder can slow down rapidly with probability 0.3 to further confuse the adversary. This section also handles a symmetric type of situations when the USV is on the front right of the intruder and the target is on the right.

The logic of the third section is very similar to the logic of the second section with the exception that it handles the situations when the USV is directly on the left or right side of the intruder (see Fig. 1.8a right). In these cases, the intruder deceives the USV by randomly slowing down to get an advantageous position, proceeding with a complete turn, or executing a partial turn. The probability of the complete turn is 0.1 and the probability of slowing down is 0.2.

The fourth section deals with situations in which the intruder is positioned behind the USV inside the rear grey area as shown in Fig. 1.8b left. The greater distance of the intruder from the USV gives it opportunity to exploit the USV's tendency to



(a)



(b)

Figure 1.8: Representative portions of intruder's policy

overshoot a little in the process of blocking. In this case, if the USV has high velocity, the intruder slows down and turns toward the stern of the blocking USV, passing the USV from behind. Otherwise, the intruder randomly turns with probability 0.7 or it proceeds with a complete turn (see Fig. 1.8b right). Again, the intruder can rapidly slow down with probability 0.3.

Finally, if the intruder is not in close proximity to the USV, it computes the best sequence of actions in order to get to the target as quickly as possible.

The intruder's policy modifies the reach and range parameters of its virtual visibility sensor to indirectly control the balance between a safe and aggressive obstacle avoidance mechanism. For example, if the intruder wants to make an aggressive turn in close proximity to the USV, it has to take risk by decreasing the reach of the sensor to be able to quickly proceed with the turn. In this case, the sensitivity of the obstacle avoidance behavior is reduced for a short period of time so that the intruder can easily pass the USV from behind. If the intruder always aimed to safely avoid the adversary, it would not get any chance to get to the target, especially if it competes against a human player.

1.5.4 Results and Discussion

The experimental run that generated the blocking policy with the highest performance is shown in Fig. 1.9. The horizontal axis of the graph shows different versions of the policy consisting of a gradually increasing number of action plans. The vertical axis shows the blocking performance in terms of the intruder's pure time delay for each version of the USV's policy. The best performance is reached by version 249 of the policy and amounts to 65 seconds of pure time delay on average and a median of 53 seconds. This can be compared to the pure time delay of 53 seconds on average and a median of 46 seconds imposed by the hand-coded USV on the same intruder. This result thus shows that the best performance of the automatically generated policy exceeds the blocking performance of the hand-coded policy.

The automated generation of the policy took approximately 1 day to generate the default controller and approximately 23 days on average to generate action plans for 300 automatically defined exception states on a machine with configuration Intel(R) Core(TM)2 Quad CPU, 2.83 GHz. From the set of 10 experimental runs, only 2 were able to find a policy with similar performance to the best one. The remaining 8 runs prematurely stagnated due to over-specialization of some of the evolved action plans and imprecise extraction of exception states. Even a single defective action plan synthesized for one of the key situations can significantly influence the performance of the whole policy. This shows that the synthesis of a policy for the USV to block the intruder utilizing a nondeterministic attacking policy is a challenging task.

The results show that the performance of the first few versions of the policy is low as they contain only a few action plans describing specialized maneuvers for a small number of key situations. However, as the synthesis process progresses, more action plans handling new situations are added and the overall performance gradually improves. This way the initial policy becomes sophisticated due to newly evolved action plans. The synthesis process continues until version 249 of the policy after which the performance stagnates. This can be attributed to difficulty in solving new complex situations in which problems with the generalization of action plans and correct detection of exception states arise.

The graph in Fig. 1.10 illustrates the average distance of failure states from the target during the synthesis process. It is shown that at the beginning of the synthesis (up to the version 47 of the policy), the failure states occur farther from the target, while most of them appear closer to the target at an average distance of 75 meters, where most intense combats happen.

Evolution of a single policy against an adversary exhibiting a nondeterministic behavior thus generates a highly suboptimal solution. To further improve the performance of the policy, distinctive states are automatically isolated for which short-term action plans are generated. As a result of that, the final policy demonstrates a clear performance advantage over the policy without the high performing substitutes.

An example of a run in which the USV reached 45 seconds of pure time delay imposed on the intruder is shown in Fig. 12. The USV starts at the location 1.1 while the intruder starts at the location 2.1. The first situation in which the USV executes a specific maneuver is marked as 1.2. In this situation, the USV steers sharply to the left

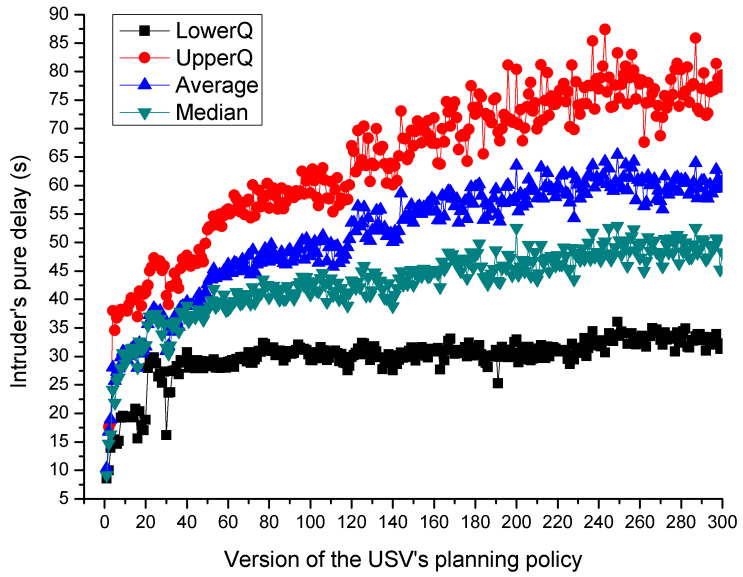


Figure 1.9: Evaluation of the USV's blocking performance. The performance is expressed as a pure time delay imposed on the intruder. Each version of the USV's policy was evaluated in 800 runs.

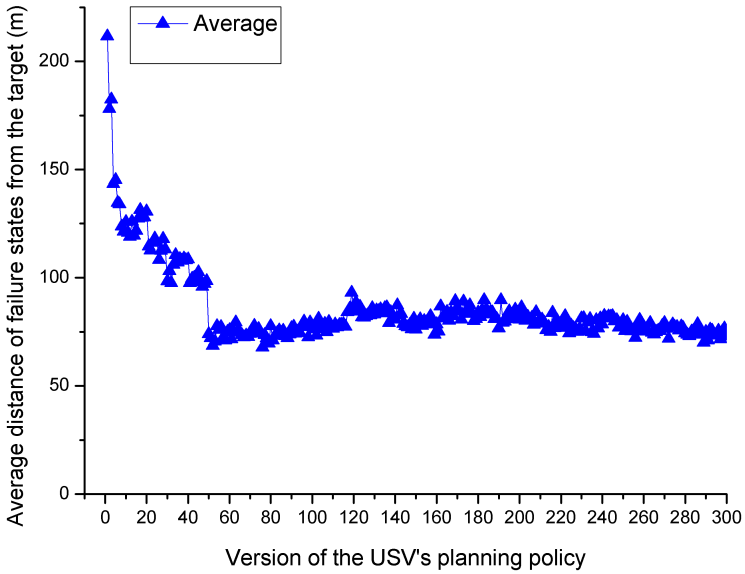


Figure 1.10: The average distance of failure states from the target during the synthesis of the USV's policy.

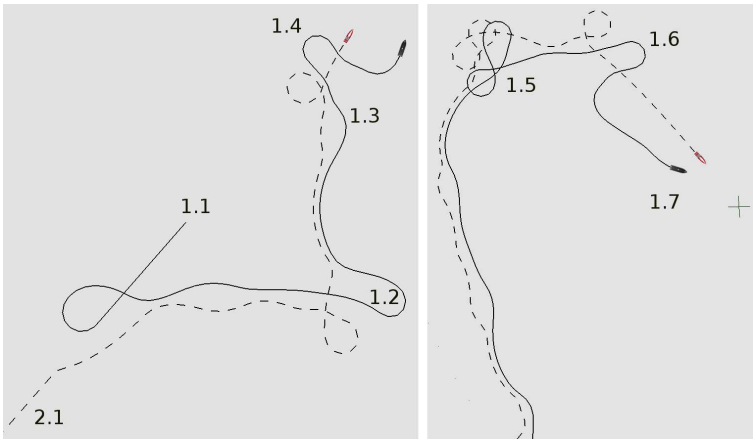


Figure 1.11: Example of a run in which the USV managed to block the intruder for additional 45 seconds. The start position of the USV is marked as 1.1 while the start position of the intruder is marked as 2.1.

in an attempt to intercept the intruder. The run continues until 1.3 where the USV attempts to deflect the intruder's heading by first carefully turning to the left and then aggressively blocking from the side. The intruder, however, instantly responds by executing a sharp left turn, which makes the USV attempt again to intercept him in the situation 1.4. Yet the USV overshoots in the process of blocking. The run continues for the next 23 seconds all the way up to the target. In the situation 1.5, the intruder executes a random sequence of two sharp turns to deceive the USV and thus to increase its chances for the attack. The USV, however, successfully follows and makes another attempt to intercept the intruder but overshoots in 1.6, and the intruder finally reaches its goal 1.7.

1.6 Summary

In this chapter, we have introduced a new approach for automated action selection policy synthesis for unmanned vehicles operating in adverse environments. The main idea behind this approach is to generate an initial version of the policy first and then gradually improve its performance by evolving additional policy components. These components are in the form of macro-actions that allow the vehicle to pre-emptively avoid frequent task execution failures. The combination of (1) explicitly detecting subspaces of the state space that lead to frequent failure states, (2) discovering macro-actions as variable sequences of primitive actions for avoiding these failure states, and (3) having elaborated control and planning mechanisms, allowed us to successfully solve a non-trivial reinforcement learning problem.

Our particular focus was on the synthesis of a symbolic policy for an autonomous USV operating in a continuous state-action space with a deceptive adversary. We have developed a mission planning system to automatically generate a policy for the USV to block the advancement of an intruder boat toward a valuable target. The

intruder is human-competitive and exhibits a deceptive behavior so that the USV cannot exploit regularity in its attacking action rules for blocking. The USV's policy consists of a high-level controller and multiple specialized action plans that allow the vehicle to rapidly execute sequences of high-level commands in the form of maneuvers in specific situations.

In our experiments, we have compared the performance of a hand-coded USV's blocking policy to the performance of the policy that was automatically generated. The results show that the performance of the synthesized policy (65 seconds of pure delay on average, a median of 53 seconds) exceeds the performance of the hand-coded policy (53 seconds on average, a median of 46 seconds). Hence, the approach described in this chapter is viable for automatically synthesizing a symbolic policy to be used by autonomous unmanned vehicles for various tasks.

Chapter 2

GPU Based Generation of State Transition Models Using Simulations for Unmanned Sea Surface Vehicle Trajectory Planning

Contributors: Atul Thakur, Petr Švec, and Satyandra K. Gupta

2.1 Introduction

USVs operate in ocean environment with disturbances caused by waves, currents, wake of other ships, etc. The disturbances impart significant uncertainty in vehicle's motion. Due to the presence of high motion uncertainty, an action of a USV may not lead to the exact desired motion despite of using a feedback controller. Vehicle dynamics and motion uncertainty together make the task of trajectory planning very challenging, particularly in highly cluttered environments. Consider Figure 2.1, which shows the influence of vehicle dynamics and motion uncertainty on the planned trajectories in a USV mission. Circles M, P, and Q denote three consecutive waypoints of a mission. When the USV reaches P, it needs to find the optimum trajectory (with minimum length and risk of collision) between P and Q. One way, to solve this problem, would be to find the optimum trajectory, without explicitly considering the ocean disturbances (shown as trajectory A). Disturbances may be insignificant in the case when the sea-state is calm (*e.g.*, sea-states 1 to 3) or the USV is heavy enough to get deviated. If sea-state is very rough (*e.g.*, sea-state 4 and higher) or the USV is lighter, then the ocean disturbances may not allow the vehicle to closely follow the intended trajectory (A) as the disturbances may lead to collisions with an obstacle in a narrow region. A trajectory, which is safer with respect to the ocean disturbances but longer in length is shown in Figure 2.1 as trajectory B. It is thus evident that the physics of interaction between USV and ocean waves greatly influences the choice

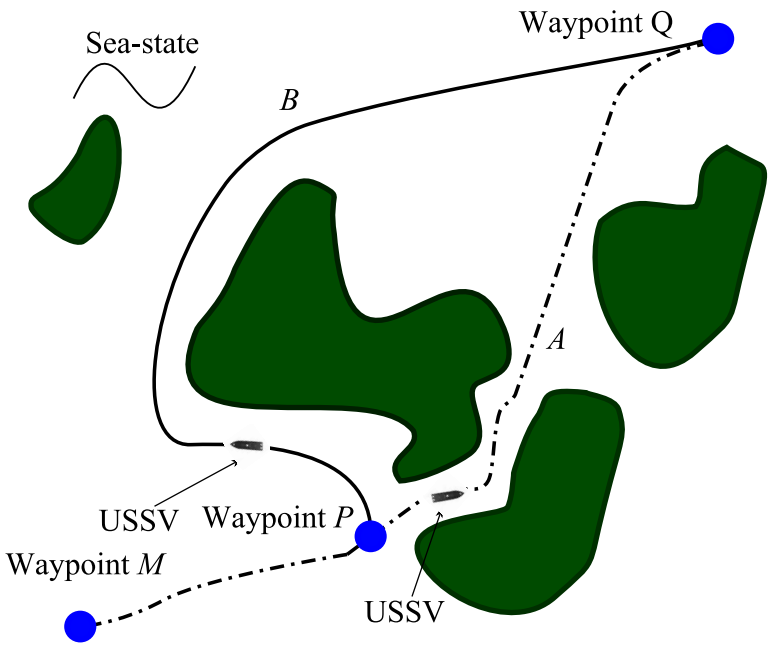


Figure 2.1: Trajectory plans for different ocean wave disturbance conditions. Trajectory A is shorter but riskier and may lead to collisions in the event of high sea-state whereas B is longer but more conservative to minimize the risk of collision. If sea-state is calm (*e.g.*, sea-states 1 to 3) or USV is heavy enough so that the disturbances are insignificant then trajectory A should be chosen. If sea-state is rough (*e.g.*, sea-state 4 and higher) or USV is light then trajectory B should be chosen.

of trajectory plan. The variations take place in dynamical parameters of USV-ocean interaction such as inertia tensor of USV due to fuel loss, damping parameters due to change in water density, etc. during mission. In addition to that sea-state changes during the execution of mission due to change in weather. Based upon dynamics based interaction between USV and ocean waves and changing sea-state, a suitable trajectory, which is safe and still not overly conservative among A and B, needs to be planned.

Above outlined physics-aware trajectory planning problem in highly uncertain ocean environments can be solved by combining MDP [32, 33, 34] framework and a dynamically feasible motion primitive based state space representation [35, 36]. MDP is a natural framework to formulate the problem of trajectory planning under motion uncertainties [32, 34]. The use of motion primitives in MDP based framework, thus, allows generating trajectories that explicitly consider the constraints imposed by the vehicle dynamics. This is unlike planning on a rectangular grid that might yield a dynamically infeasible trajectory. In this chapter we incorporate vehicle dynamics and motion uncertainty into motion-primitives using Monte Carlo runs of high-fidelity 6 DOF dynamics simulation of interaction between USV and ocean waves. We represent uncertainty in motion primitives by using a state transition function. This function

maps each possible discretized state of the vehicle and a given action to a list of possible resulting states with respective transition probabilities.

State transition function can be obtained, by either running field experiments or by using computer simulations. Performing field experiments is the most accurate method, but is expensive and may be infeasible when needed to be performed during a mission. Moreover, the number of experiments may be very large when multiple sea-states and vehicle dynamics parameters need to be taken into account, which is the case during long missions. Computer simulations are inexpensive and can be improved, by incorporating experimental data. The complexity of a mission and an environment require generating the state transition map on-line, based on the information gathered by the sensors during vehicle operation. It may be infeasible to run all possible simulations off-line (*i.e.* before the mission) to generate the state transition map. This is because, the sea-state is decided by several factors, namely, the amplitudes, frequencies, wave directions, and the wave numbers of the wave components forming the ocean wave [37, 38]. Each of these factors is continuous and has non-linear influence on the ocean wave. In addition to this, the ocean interacts with USVs in a non-linear fashion. The initial conditions for the simulations can thus, become combinatorially prohibiting for an off-line estimation of state transition map. A potential flow theory based high fidelity 6 DOF dynamics simulator can generate a fairly accurate state transition map and aid in generating both safe and low cost trajectories [39, 40, 41]. The major problem with using such a high fidelity simulator is the slow computational performance of the simulation. This is mainly because of the fluid to rigid body interaction computation. One way to accelerate the computations is by using parallelization. Performing parallel computations would require computing clusters to be placed on the base and communicating with the vehicle over a network, which might be generally unreliable due to possible communication disturbances. The alternative of placing clusters directly on-board might add to the weight of the payload, which may not be desired or generally not possible. Another way to perform on-board computing is by using GPU. Recently, expensive scientific computations in various robotics problems are performed using GPUs, which are very powerful and lightweight as well [42, 43, 44, 45]. In addition to using GPU computing, we also employ model simplification techniques [40] to further improve computational performance of the simulation. A faster computation of state transition map may be required for some parts of the mission, where computing speed-up available using GPU alone may not be enough. The accelerated dynamics simulation is then used to establish connectivity among the vehicle's discretized states to develop a state transition map represented using a connectivity graph. The trajectory planning problem is then solved using value iteration of the stochastic dynamic programming (SDP) [32].

The key contributions of this chapter are: (1) incorporation of 6 DOF dynamics simulation in state transition map estimation, (2) use of GPU based parallelization schemes to make the simulation faster for on-line computation of state transition map, (3) incorporation of temporal coherence based model simplification techniques with GPU acceleration for computing state transition map even faster, and (4) solution of trajectory planning problem using developed state transition map data structure, so that the computed trajectory plan satisfies the dynamics constraints as well as

handles motion uncertainties.

2.2 Problem Statement and Solution Approach

2.2.1 Problem Statement

Given,

(i.) a finite non-empty state space X,

(ii.) a finite non-empty action space $u(x)$ for each state $x \in X$,

(iii.) a dynamics motion model $\dot{x} = f(x, u, w)$ of the USV, where w is a nondeterministic noise term and fluid flow is based on potential flow theory,

(iv.) goal state x_G, and

(v.) obstacle map Ω such that,

$$\begin{aligned} \Omega(x) &= 1, \text{ if } x \text{ lies on obstacle} \\ &= 0, \text{ if } x \text{ is on free space,} \end{aligned}$$

compute following:

(i.) State transition map over X and u: The state transition map should represent the motion uncertainty exhibited by the given motion model under each given action in u in the form of associated probability of transition for the corresponding state transition probability $p(x_k|x_i, u_i)$, $\forall x_k$, $x_i \in X$, and $u_i \in u$. Perform GPU based computing acceleration and develop model simplification techniques for on-line estimation of state transition map.

(ii.) Trajectory plan: Using the state transition map computed in step (i), determine trajectory plan to generate dynamically feasible trajectory in each planning cycle to reach target location x_G from any given starting location of the USV. The computed trajectory plan ensures that the generated trajectory is updated in every planning cycle to recover from the pose errors introduced due to the influence of the ocean environment. This kind of trajectory plan is also referred to as *feedback plan* in Chapter 8 of Ref. [32]. We assume perfect state information is available at all times.

2.2.2 Approach

The approach is enumerated in the following steps.

(i.) Enhance the given USV motion model to suit the requirements for GPU implementation. Implement the motion model on GPU and develop simplification algorithms to enable faster simulation.

(ii.) Model the trajectory planning problem as MDP by representing state-action space in a lattice data structure and compute the state transition map for the discretized action space.

(iii.) Apply value iteration of stochastic dynamic programming to determine the trajectory plan. The generated trajectory plan enables the USV to find the optimal trajectory from each discretized state $x \in X$.

In the following sections, we discuss the above steps in detail.

2.3 Dynamics Simulation of USVs

We implemented the 6 DOF dynamics model for vehicles given by Fossen [38]. We extend the governing equations of the implemented dynamics model [40] to handle the arbitrary number of wave components and to incorporate uncertainty into the system. A detailed description that includes the computation of the fluid based on potential flow theory [46] can be found in [8].

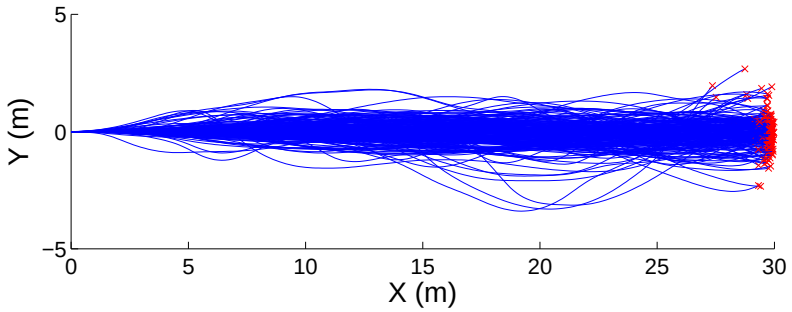
2.3.1 Uncertainty in the Motion Model

The ocean wave (the ocean wave height and the velocity field) was initialized using given wave amplitudes, frequencies, and directions. An ocean wave, once initialized evolves deterministically with time and can be predicted exactly using the solution of Laplace equation [46]. However, the ocean waves initialized with the same parameters might look different due to the presence of the uniformly random phase lags ψ_j between each wave component. This leads to prediction of slightly different trajectories in each simulation run of the USVs operating under the ocean waves with exactly identical ocean wave parameters (initialized with uniform random phase lags) and an action. This effect is shown in Figure a, in which the USV is acted upon by a PID controller to move along a straight line for 256 different simulation runs in ocean wave built up of identical wave components. The variation in the trajectories of the USV in each simulation run is due to the uncertainty introduced by random phase lags in the ocean wave components despite of the other ocean parameters and the PID control objective being exactly identical. Figure b shows the histogram of final positions reached by the USV when commanded to reach at $(30, 0)$ with an orientation of 0^0 due to the disturbances caused by the ocean waves. Figure c shows the variation in the final orientation while the commanded action was 0^0. It should be noted that the variation in the ending pose is one of the main cause of randomness in the motion model, which accumulates over the trajectory.

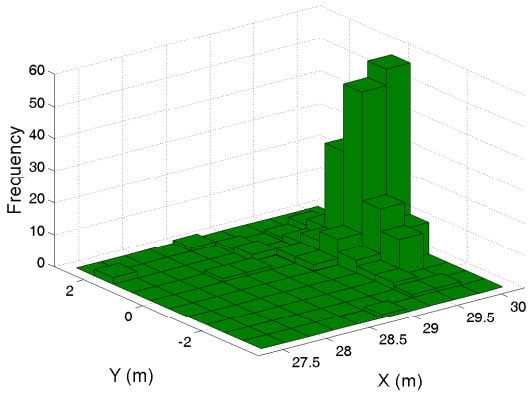
Formally, we can express the parametric form of the dynamics model (with uniformly random initial phase lag parameters) as follows:

$$\dot{x} = f(x, u, w) \tag{2.1}$$

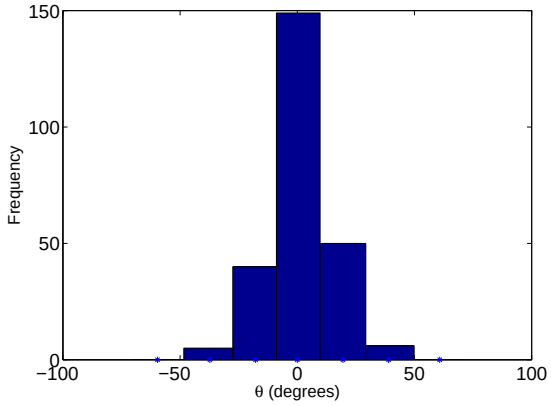
where w is the noise introduced due to the uniform random phase lags ψ_j.



(a) Sample trajectories.



(b) Frequency distribution of USV's ending positions for sample trajectories.



(c) Frequency distribution of USV's ending orientations for sample trajectories.

Figure 2.2: Uncertainty in USV's motion model for action along X axis generated using sample size of 256 (computed for sea-state 4 with average ocean wave height of $1.8m$ and boat moving at the velocity of $3ms^{-1}$).

2.3.2 Simulator Implementation on GPU

As described in Section 2.3.1, and shown in Figure 2.2, the USV ends up in different poses for exactly identical action objective and initial states for different phase lag initializations. This means that for sufficiently large number of simulation runs with uniform phase lag initializations for a given set of initial conditions and action goal, the distribution of final states will represent the influence of the ocean on the USV's

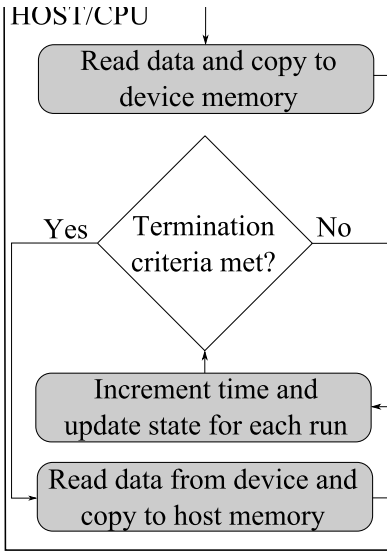


Figure 2.3: Implementation of USV simulator on GPU.

motion. In this section, we describe Monte Carlo simulation based approach to estimate the influence of nondeterministic effects of ocean on USV's motion for a given set of action goals. The Monte Carlo sampling based approach requires running numerous dynamics simulations with uniformly random initial phase lags among wave components and hence, real-time performance of the simulator may not be enough [40]. We describe the enhancements made in the potential flow theory based simulation model [39, 40, 41] for suitability of implementation on GPU.

We define state vector tuple $X = (x_1, x_2, .., x_M)$, where $x_k = [v_k^T x_k^T]^T$ is state vector of k^{th} simulation run and M is the number of Monte Carlo simulation runs.

Let $U = (u_1, u_2, ..., u_M)$ be the action vector tuple with each element as a vector specifying action for corresponding simulation run. We denote action set for the entire sample using the tuple $\Upsilon = (U_1, U_2, ..., U_P)$ where U_j's are action vector tuples and P is the number of action goals.

Also, let $W = (w_1, w_2, ..., w_M)$ be the phase lag vector tuple where $w_k = [\psi_{1,k}, \psi_{2,k}, ..., \psi_{Q,k}]^T$ is the phase lag vector for k^{th} simulation run and $\psi_{q,k}$ is phase lag of q^{th} wave component of k^{th} simulation run.

Thus, the augmented dynamics equation can be written by generalizing Equation 2.1 for M Monte Carlo simulation runs as follows:

$$\dot{X} = F(X, U, W) \tag{2.2}$$

where, F is the modified dynamics function representing simultaneous simulation runs.

Let $F_{WT} = (F_{W,1}^T, F_{W,2}^T, ..., F_{W,M}^T)$ be the ocean wave force tuple where $F_{W,k}$ is ocean wave force vector for k^{th} simulation run.

The computation steps of the simulation are enumerated below (see Figure 2.3).

Algorithm 1 - GPU based Monte Carlo Simulation of USV's dynamics

Input

(a.) Initial state vector tuple of USV X_0,

(b.) number of Monte Carlo runs M,

(c.) desired target state vector tuple X_t,

(d.) desired trajectory length l,

(e.) radius of acceptance r,

(f.) number of ocean wave components Q,

(g.) time step size Δt,

(h.) action set Υ,

(i.) polygonal geometry of USV, and

(j.) sets of amplitudes A_q, frequencies ω_q, directions θ_q corresponding to each wave component, where index q varies from 0 to $Q-1$,

Output Set of M trajectories

Steps

(i.) Initialize the state vector tuple of USV $X = X_0$, phase lag tuple W, time $t = 0$, and trajectory length vector $L = [0, 0, ..., 0]^T$. Copy all configuration variables such as ocean wave parameters, dynamics parameters and geometry parameters to constant memory cache of GPU so that data is not required to be transferred in each simulation time step.

(ii.) Transform the USV geometry to M states represented by X. Each transformation is performed by separate GPU thread. In this case, same instruction of transformation needs to operate on MN similar data, where N is the number of polygonal facets representing the USV's geometry. We perform computations of this step on GPU.

(iii.) Determine the instantaneous wet surfaces ($S_{B,j}$) of the USV by finding out the facets lying beneath and on the wave surface (computed by superimposing given Q ocean wave components corresponding to j^{th} phase lag vector w_j and compute the wave force tuple F_{WT}. In this case computation of intersection of each polygonal facet with instantaneous ocean wave and force computation is performed by separate GPU threads. The number of independent operations required is again MN. We perform computations of this step on GPU.

(iv.) Determine the required control force vector tuple corresponding to the action set Υ using Equation 3.4.

(v.) Determine the Coriolis matrix $C_k(v)$ and the damping matrix $D_k(v)$ corresponding to each Monte Carlo simulation run. The number of independent operations required in this step is M. We perform computations of this step on GPU.

(vi.) Use Euler integration to solve Equation 2.2 by using the wave force tuple, Coriolis matrix, and damping matrix. Update time t to $t + \Delta t$. The number of operations needed in this step is M. We perform computations of this step on GPU.

(vii.) Find Euclidean distance ΔX between state tuple obtained from step (vi) and X and update trajectory length vector L with $L+\Delta X$. Compare each element of L with the desired trajectory length l. The Monte Carlo runs, for which trajectory length exceeds the set trajectory length l, do not update corresponding elements of X whereas for other runs update the elements in X with the solution found in step (vi). Since this is a step with logical branching we perform it on CPU.

(viii.) If trajectory lengths for all the runs exceeds l or all simulated instances of USV are within the radius of acceptance r from the respective target positions then return M trajectories else go to step (ii).

2.3.3 Simulation Results of GPU Based Parallelization

We used NVIDIA's CUDA software development kit version 3.2 with Microsoft Visual Studio 2008 software development platform on Microsoft Windows 7 operating system for the implementation of Algorithm 1. The graphics hardware used was NVIDIA GeForce GT 540M mounted on Dell XPS with Intel(R) Core(TM) i7-2620M CPU with 2.7GHz speed and 4GB RAM. We chose the number of CUDA threads per block to be 256 for each kernel function. For the CUDA kernel function needed to work on J data members, we chose the number of computing blocks to be $\frac{MN+T-1}{T}$. The number of triangular facets in the USV model used in the simulations was 11158 and the bounding box dimensions of the model was $12 \times 4 \times 4$ m. We chose ocean wave composed of $Q = 20$ components with 6 components having amplitude of 0.2 m while rest 14 with amplitude of 0.1 m. 16 of the ocean wave component had frequency of 1 Hz while four of them had frequency of 2 Hz, and the direction θ_w's were evenly distributed in the range 0 to 2π rad . We chose simulation time step of size 0.05 s and ran the simulation for 200 time steps for 256 random phase lag initializations. Table 2.1 shows the comparison of the computational performance on GPU as compared to the CPU based computation. OpenMP based multi-threading enabled 85% average CPU usage, while running the baseline simulations. GPU based approach resulted into speed-up by factor ranging from 3.8 to 14.0, for the presented test case. Table 2.1 also shows that the speed-up factor increases with increase in the number of Monte Carlo runs, because of the highly data parallel nature of the computations. For larger number of simulation runs, the cost of memory transfer is appropriated and hence, speed-up is larger compared to smaller number of runs.

2.3.4 Model Simplification on GPU

More than 99% of the computation time in the USV simulation is spent in computing the forces acting on the USV due to the ocean waves [40]. The ocean is represented

Table 2.1: Comparison of the computation gain due to GPU over the baseline computations performed on CPU

M	Baseline computation time on CPU (s)	GPU Computation time (s)	Speed-up
1	13.7	3.6	3.8
2	27.1	5.3	5.2
4	54.2	8.0	6.8
8	107.4	13.0	8.3
16	214.4	22.0	9.8
32	425.6	36.9	11.5
64	846.6	65.4	12.9
128	1691.2	123.8	13.7
256	3386.3	241.9	14.0

as a spatio-temporally varying heightfield in this chapter. One of the major factors influencing wave forces is the variation in the wave heightfield in addition to the fluid velocity around the USV. The ocean wave heightfield does not change significantly with each simulation time step. For example, for a simulation time step of length 0.05 s, the possibility of ocean wave heightfield around the USV changing significantly is very low. In such a situation, one can utilize the force computed in the previous time step in the current time step of the simulation to save some computational effort. This is the underlying idea behind temporal coherence. In order to explain the idea of temporal coherence in a more concrete way, we define *instantaneous ocean heightfield* and *heightfield distance vector* as follows.

Definition 1 Let the ocean wave be specified by Q components of given amplitudes, frequencies, and directions. Let state vector tuple X denote the instantaneous states of the USV for each Monte Carlo simulation run, B_j be the bounding boxes of the USV located at the poses given by X and rectangles $R_{B,j}$ be the projections of B_j on the XY plane. Let Λ_j denote uniform grid of size $m \times n$ on $R_{B,j}$.

We define instantaneous ocean wave height-field G as a $(Q \times mn)$ sized matrix G, such that the rows of G are the vectors made up of ordered elevations of the ocean wave at the mn grid points on Λ_j.

Definition 2 For a pair of ocean wave height-fields G_1 and G_2, we define the *heightfield distance vector* $\mathbf{h}_d$ between G_1 and G_2 as the following row-wise second order norm.

$$\mathbf{h}_d = \|G_{1,j} - G_{2,j}\| \tag{2.3}$$

where $G_{1,j}$ is the j^{th} row of G_1, and index j denotes Monte Carlo run from 1 to M.

The force need not be computed in a simulation time step, if the ocean wave heightfield distance around the USV from the previous simulation time step is not

significant. The temporal coherence test is performed as an additional operation in step (ii) of Algorithm 1 (described in Section 2.3.2). If it is found that the ocean wave heightfield distance corresponding to at least one Monte Carlo run has changed significantly then step (iii) of Algorithm 1 is performed, else step (iii) is skipped and step (iv) is directly executed. By this, the execution of step (iii) in Algorithm 1 is avoided some times, which introduces some simplification error, but reduces computation time.

The steps for performing temporal coherence based model simplification on the GPU are described below.

Algorithm 2 - Temporal coherence based Model Simplification

Input

(a.) State vector tuple X,

(b.) number of Monte Carlo runs M,

(c.) number of rows (m) and columns (n) of grid,

(d.) simulation time t and time step size Δt,

(e.) threshold τ for heightfield, and

(f.) threshold $d\tau$ for differential of heightfield.

Output Decision about whether to perform force computation in the next time step or reuse force computed in the earlier time step

Steps

(i.) If $t = 0$ return decision to perform force computation.

(ii.) If $t = \Delta t$ then initialize heightfield G_p and differential heightfield dG_p to a null matrix of size $M \times mn$ and store in global memory.

(ii.) Compute ocean wave heightfield G at time t and then compute differential heightfield $d_G = G - G_p$.

(iii.) Compute heightfield distance $\mathbf{h}_d$ vector between G and G_p.

(iv.) Compute differential heightfield distance $d\mathbf{h}_d$ between dG and dG_p.

(v.) If all the elements of $\mathbf{h}_d$ are less than τ and if all the elements of $d\mathbf{h}_d$ are less than $d\tau$, return the decision to reuse the previous value of force else update $G_p = G$ and $dG_p = dG$ and return the decision to recompute force.

2.3.5 Results of Model Simplification on GPU

We chose $m = 2$, $n = 5$, $M = 256$, and $d\tau = 0.1$, and performed the simulations under identical ocean and USV dynamics parameter settings and varied τ from 0.00 to 0.10 in the increments of 0.025. The computation speed-up factor over the GPU baseline computation time (when $\tau = 0.00$) varies from 1.04 to 3.61 depending on the set threshold τ as shown in Figure a. The mean square error in force computation introduced due to increasing threshold τ is shown in Figure b. The temporal coherence based simplification introduces errors in the computation of the final pose of the USV in Monte Carlo runs. Figure c shows the variation in the Euclidean distance of final positions of USV from the nominal position and the difference of each final USV orientations from the nominal orientation obtained by the Monte Carlo simulation runs. The nominal pose is the commanded pose to which USV should reach if there are no disturbances. In the test case, the nominal position is $(30, 0)$ and the nominal orientation is 0 rad. The variation in distance and orientation difference increases with increasing threshold. This is because, greater the threshold, fewer number of times force is computed and hence, more will be the inaccuracy.

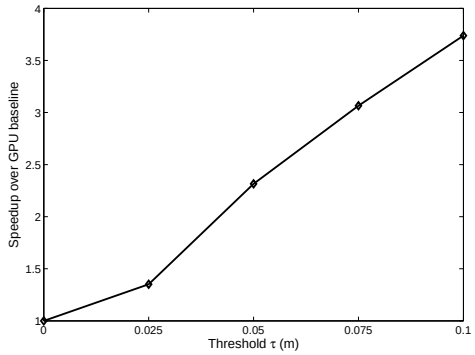
We chose fixed randomization of the ocean wave for evaluating the plots, in order to prevent the influence of randomization on the computing time and the variation of the pose errors.

It should be noted in Figure 2.4, that the computing gains increase slowly in the range $0 < \tau < 0.025$, because, for smaller threshold, algorithm is unable to reuse force values computed in the previous steps and owing to the same reason, the variation in the final pose and nominal pose is also comparatively less pronounced. For larger values of threshold $\tau > 0.075$, the variation in the distance and orientation increases rapidly. It can thus be concluded that τ can be varied in the window of $0.025 < \tau < 0.075$, for obtaining computational gain at the expense of acceptable errors.

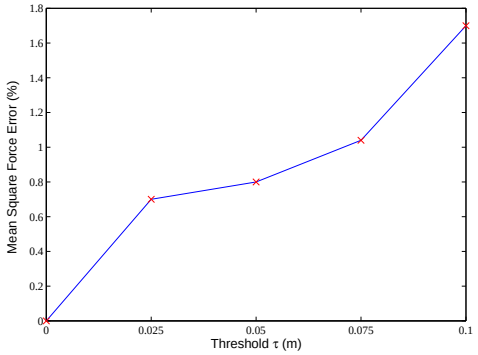
Figure 2.5 compares the computational gains obtained using temporal coherence on the GPU and CPU based simplification [40] approaches with the baseline computed on CPU (see Table 2.1). The main observations and respective analysis from the Figure 2.5 are explained below.

(i.) The computational speed-up factor obtained using temporal coherence on GPU is in the range of 4.7 to 43.1 and increases with the number of Monte Carlo runs. The increase in the speed-up can be attributed to the fact that the main computational cost of GPU operations is the data transfer between GPU and CPU. In the USV simulation, the data of state variables and the system matrices need to be transferred from GPU to CPU which is a constant time operation. When the number of runs is less, the appropriated compute time is larger whereas for large number of Monte Carlo runs the cost of memory transfer reduces and hence, the speed-up factor increases.

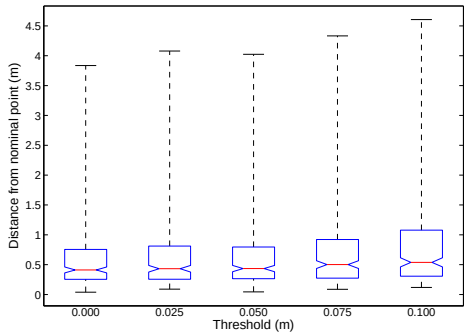
(ii.) The computational speed-up factor due to temporal coherence over GPU baseline ranges from 1.2 to 3.1.



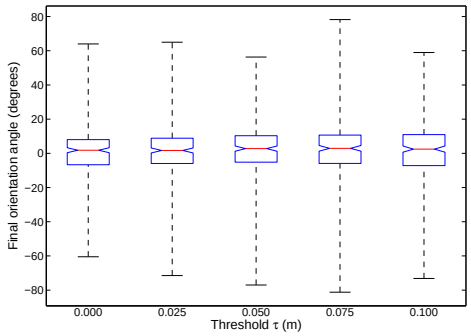
(a) Computing speed-up over GPU baseline vs threshold.



(b) Variation of mean square force error with threshold.



(c) Variation of distance from nominal point vs threshold.



(d) Variation of orientation from nominal point vs threshold.

Figure 2.4: Model simplification results of GPU based temporal coherence.

(iii.) The error associated with the model simplification performed on GPU and CPU is computed by taking the mean squared percentage error between the time series of the computed forces using the simplified method and the baseline method [40]. The model simplification based on temporal coherence implemented on GPU led to an error of 1.04% for the threshold $\tau = 0.075$ and $d\tau = 0.1$. Also, the figure shows variation of computational speed-up using model simplification algorithms based on clustering and temporal coherence on CPU [40] for simplification parameters $C = 60$, $\tau = 0.070$, and $d\tau = 0.1$. The approximation parameters C, τ, and $d\tau$ are chosen such that the errors due to GPU and CPU based simplification is similar, for fair comparison of associated speed-up in each case. Model simplification performed on CPU lead to an average factor of speed-up of 6.4 and average force error of 1.25% over the CPU baseline. It can be seen in Figure 2.5 that for single run the CPU based simplification approach outperforms the purely GPU based approach by a factor of 1.5 and for two runs the CPU based simplification approach is better than purely GPU based approach by a factor of 1.1. Again the reason is the appropriation of computing time spent on the constant time data transfer operations over larger number

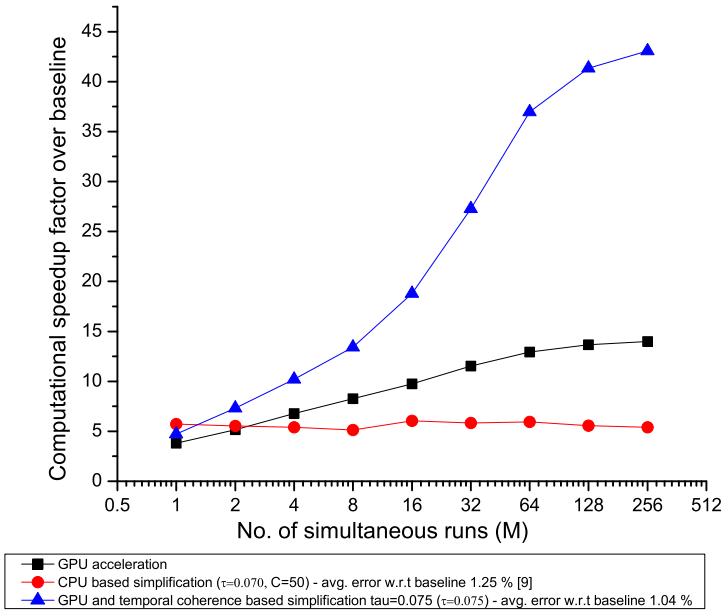


Figure 2.5: Results of GPU acceleration.

of runs. It is thus evident that using purely GPU based approach may not be enough in applications in which a single USV needs to be run in a VE, as model can become more complex, stretching the GPU to its limits. In applications, where some error is tolerable, model simplification can significantly speed-up the application at the cost of small errors.

(iv.) For larger number of runs, which are pertinent to applications such as transition probability estimation, GPU based approach gives very high speed-up factor (in case of Figure 2.5, about 43.1 with average mean square force error of 1.04%).

(v.) Figure 2.5 also shows that the computing speed-up due to temporal coherence, gradually saturates as the number of simultaneous runs increases. The reason is that the possibility of many heighfields being less than the threshold simultaneously reduces as the number of heightfields increase.

2.4 Application of Generated State Transition Map in Trajectory Planning of USVs

In this section we present the application of state transition probabilities obtained from model simplification techniques, developed in Section 2.3 in the area of trajectory planning of USVs.

2.4.1 MDP Formulation

In this section we present the MDP formulation for the trajectory planning problem and the algorithms to compute various components of the MDP.

State-Action Space Representation

For the dynamics computations, the state of the USV is defined as an augmented vector of pose and velocity according to Equation 2.1. The sizes of the vectors representing pose and velocity are 6 each, making the size of the state vector as 12. In addition, the USV state-action space is continuous and it is very difficult to search an optimal policy, in such a high dimensional and continuous space. In this section we present the state-action space dimension reduction and a suitable discretization to pose the problem of trajectory planning of USVs as a MDP.

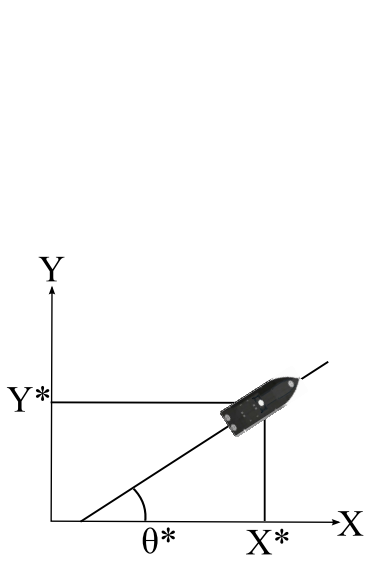
The motion goals for the USVs are usually specified in terms of the target pose $[x,\, y,\, \theta]^T$) and the target velocity $[v_x,\, v_y,\, \omega_z]^T$ on the ocean's nominal water plane. This means that the state space for the MDP can be reduced to $[x,\, y,\, \theta,\, v_x,\, v_y\, \omega_z]^T$. During operation the velocity of USVs do not change significantly during the operations and that justifies the choice of the constant desired velocity of the boat. Also we tuned the PID controller such that the angular velocity is kept within a set bound. The sway velocity and the angular velocities in the roll and the pitch directions are generally very small and can be ignored in the MDP state space. Nevertheless, the transition model computation is performed using all the 12 DOFs. The state space for the planning purposes in this chapter is thus, reduced to a 3-tuple given by Equation 2.4.

$$s = [x\ y\ \theta]^T \tag{2.4}$$

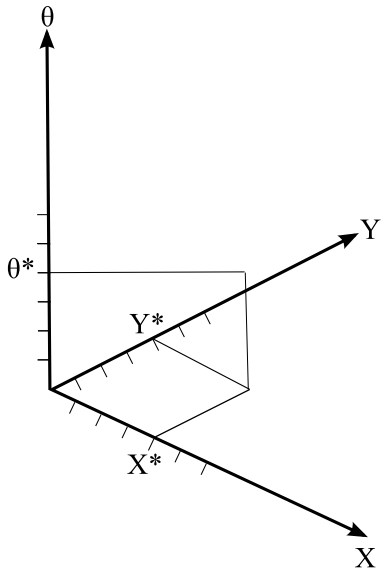
where (x, y) are the coordinates of the USV's CG in the XY plane, and θ is the orientation of the boat in the XY plane.

We chose the grid dimension to be 15.0 m (USV length is nearly equal to 12 m). The orientation discretization is chosen to be 0.524 rad. The state space is depicted in Figure 2.6, in which the XY plane contains the location of the CG and the θ axis denotes the orientation of the boat. Pose of the boat in the XY plane is shown in Figure a and the corresponding location in the 3-D state space is shown in Figure b.

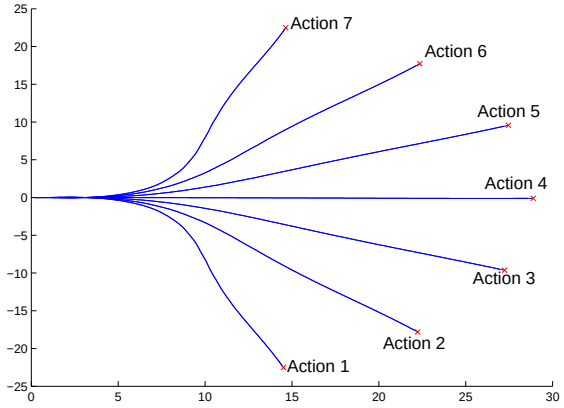
The action space is discretized as a set of relative pose commands from an initial state. We chose the set of relative final pose as 7 radial pose vectors having radial distance of 30.0 m with final desired steering angles varying from -2.142 to 2.142 rad in the increments of 0.428 rad. We chose the desired path lengths and the steering angles by first running the boat for 10 s along polar directions. Using this technique we generated a set of 7 waypoints which sufficiently cover the space around the boat and are dynamically reachable in 10 s.



(a) Typical pose of USV in XY plane.



(b) Location of USV in state space.



(c) Action discretization.

Figure 2.6: State-action space of MDP.

State Transition Map Computation

The state transition map for the continuous space was expressed in the form of Equation 2.1. In this section we shall describe the computation scheme to determine the state transition map for the given USV in the discrete state space. We first present the definition of the state transition probability.

Definition 3 Given an initial state x_t and an action u_t at time t, the probability $p(x_{t+\Delta t}|x_t, u_t)$ of ending up in the state $x_{t+\Delta t}$ is called the state transition probability. We assume that the time taken to execute the action u_t is Δt.

Figure 2.7 shows a USV situated initially in the state x_t. When an action u_t is applied to it for 256 sample runs, the trajectories traced by the USVs are shown in

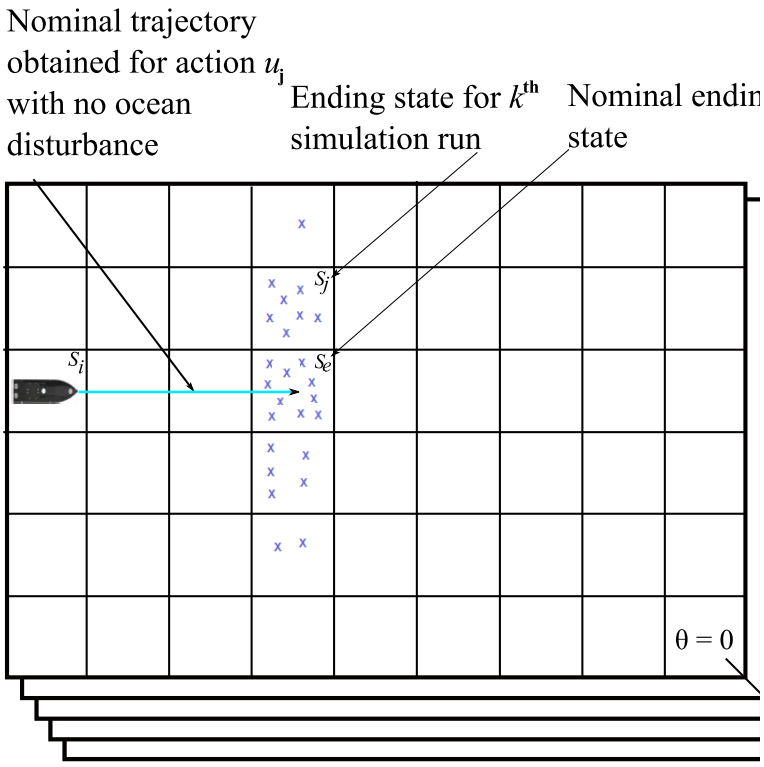


Figure 2.7: State transition map computation.

the figure. The USVs trace a slightly different trajectory for each sample run due to the ocean wave and USVs interaction force. The variation in the resulting states for a given initial state and an action yields a probability distribution over the resulting states.

We use a data structure similar to state lattice [35] to consider the dynamics but enhance it by embedding the information of the state transition probabilities in the arcs of the state lattice and then make use of the stochastic dynamic programming to solve the problem of trajectory planning under the motion uncertainty. The steps to compute the state transition map are described below.

Algorithm 3 - State transition map computation

Input

(a.) Set of trajectories $T = (T_1, T_2, ..., T_P)$ for a given action set $\Upsilon = (U_1, U_2, ..., U_P)$ using Algorithm 1 and 2,

(b.) List of discretized states $S = [s_1, s_2, ..., s_L]^T$ representing the region of USV operation.

Output State transition map.

Steps

(i.) Perform geometric transformation of each trajectory in T. Figure 2.7 shows a portion of the state space with the rectangular grids representing region on the nominal water plane, while the layers representing the orientation of the vehicle that vary from 0 to 2π *rad*. For each simulation run (beginning from s_i and taking action u_j), the USV ends up in different states, shown by crosses in Figure 2.7. State s_e represents the nominal ending state under the action u_j when there is no environmental disturbance.

(ii.) Construct graph by connecting the states (nodes) reachable from another states (nodes) using the sample actions (arcs) in T.

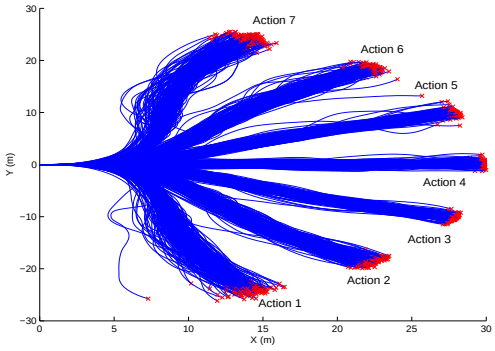
(iii.) Determine the transition probabilities by finding out the ratio of the number of connections between the two connected states and the total sample count of the actions taken. In Figure 2.7, let the state s_j be connected to the state s_i for $n(s_j)$ times out of N sample runs. Compute the probability p_{ij} of transition from state s_i to state s_j using $p_{ij} = \frac{n(s_j)}{N}$.

(iv.) Return the state transition map.

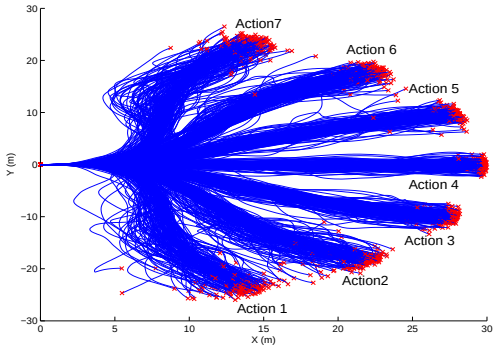
In this way, the state transition map is obtained, whose nodes are the states and the arcs are the dynamics constraints. Each connection has a probability associated with it, due to the system dynamics and the presence of uncertainty due to ocean waves. It should be noted that the maximum number of children nodes of a given parent node, for a state space with L nodes can be up to L based on the variations in the samples of the action and level of uncertainty in the environment. This makes the data structure very flexible in the sense of capturing the dynamics constraints and the environmental uncertainty for extreme situations such as rough sea-state.

Reward Function

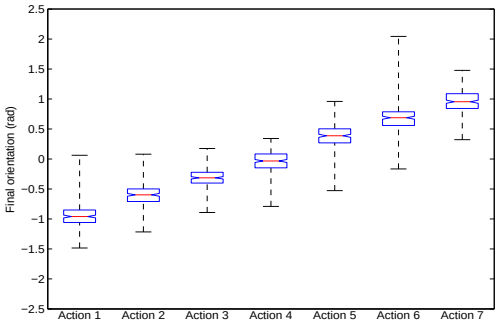
The final element of MDP is the immediate reward for transitioning from a given state to another state by taking an action. The time spent by the USV to perform an action, can be determined by the length of the trajectory traversed by it. This entails that larger the length of the trajectory, smaller should be the reward for the action, generating the trajectory, and thus, we should consider the negative value of the trajectory length as the reward. We chose the negative of the average length of the S trajectories traversed by the USV for each action in the control set to determine the reward.



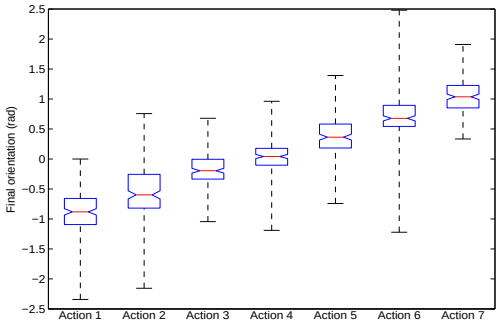
(a) Trajectories for each action computed at sea-state 3.



(b) Trajectories for each action computed at sea-state 4.



(c) Final orientations for each action computed at sea-state 3.



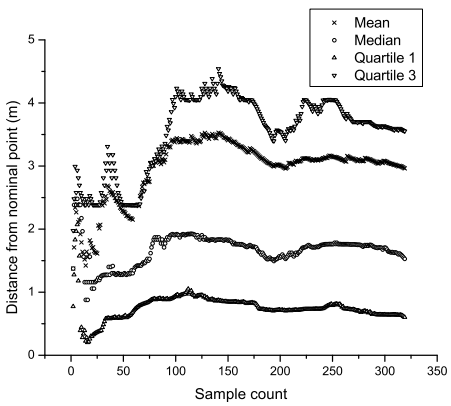
(d) Final orientations for each action computed at sea-state 4.

Figure 2.8: Control set computed for different sea-states.

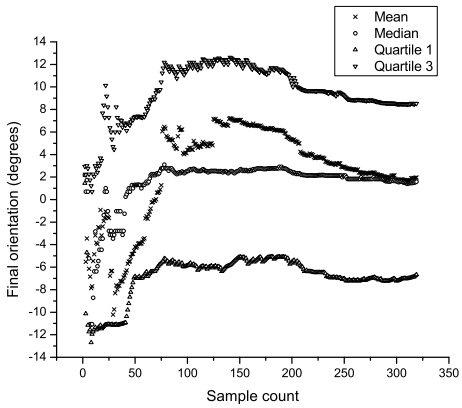
2.4.2 Results of Trajectory Planning

For the given action set Υ described in Section 2.4.1, we determine the set of trajectories T, using Algorithm 1 and 2, for the sea-states 3 and 4. The average wave height, for the sea-state 3, ranges from 0.5 m to 1.25 m, whereas, for the sea-state 4, the average wave height ranges from 1.25 m to 2.5 m [38]. The resulting set of trajectories, for 256 Monte Carlo simulation runs is shown in Figure 2.8. The variation in the resulting trajectories due to different actions in the action set is shown in Figures a and b for sea-state 3 and 4 respectively. Variations in the final orientation in each Monte Carlo run due to ocean uncertainty for various actions are shown in Figures c and d. The variations in the trajectories for the sea-state 4 is larger compared to that for the sea-state 3 due to the higher average ocean wave height. We used $\tau = 0.075$ and $d\tau = 0.1$ to obtain Figure 2.8. The time taken to compute the set of trajectories for each sea-state was 9.1 minutes. We chose sample size of 256 because the variability of the pose data can be captured using a sample size of around 256. This is illustrated in Figure 2.9.

The MDP formulated in the Section 2.4.1 can be solved using numerous algorithms



(a) Variation in sample median and quartile distance from nominal position.

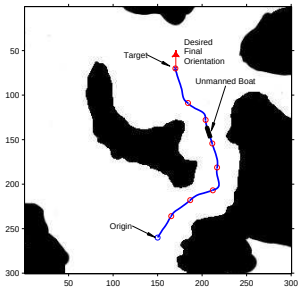


(b) Variation in sample median and quartile orientation from nominal orientation.

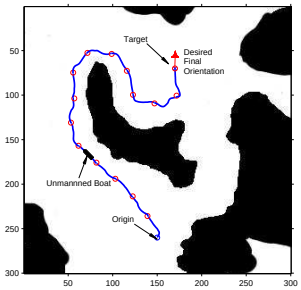
Figure 2.9: Variation in sample distance and orientation from nominal pose vs sample size.

including the value iteration, policy iteration, hierarchical techniques, approximate techniques, etc. [47]. We chose the value iteration for computation of the solution. The details of the value iteration algorithm could be found in many references including [32, 34, 47]. The policies for each sea-state are computed using the value iteration over the obtained respective state transition maps. The optimal policies are then used for determining the trajectory to the target. Even if, the USV is deviated from planned trajectory (mainly, the orientation is changed significantly in high sea-state) due to forces caused by ocean waves, the computed policy can be used to regenerate an optimal trajectory to the target state. The obtained trajectory is optimal given the uncertainties (caused by ocean waves and unmodeled effects such as the variations in the angular velocities and the linear velocity due to the implemented PID controller limitations). The dynamics based motion model developed in this chapter includes these variations in the transition probability. The transition probability is then used to compute the optimal policy in the MDP framework, which has been proved by the researchers to yield global optima [33, 47]. It should be noted that the optimality is achieved in the resolution sense, meaning that the chosen resolution of the state-action space influences the achieved optimality. Finer the resolution better will be the optimal solution, but greater will be the computational complexity.

The resulting trajectories for sea-states 3 and 4 are shown in Figure 2.10. The origin and target orientation of the vehicle is $\frac{\pi}{2}$. In case of sea-state 3, a shorter and riskier trajectory (through a narrow passage in the midst of obstacles) is computed by the approach discussed in this chapter as shown in Figure a. In case of sea-state 4 a longer but safer path is computed as shown in Figure b. It should be noted that because of the dynamics constraints built into the search graph through the motion primitives discussed in this chapter, the generated trajectories are always dynamically feasible. This is the reason that the trajectory in case of Figure b bends near the target, as the vehicle needs to go down in order to turn to the desired orientation



(a) Trajectory computed between origin and target states at sea-state 3.



(b) Trajectory computed between origin and target states at sea-state 4.

Figure 2.10: Executed trajectories obtained by using feedback plan for sea-states 3 and 4. The feedback plans are computed using the algorithms developed in this chapter

of $\frac{\pi}{2}$. The disturbances due to the ocean waves are computed during the simulation, which causes unpredictable deviations in the trajectory. The computed policy makes it possible for the USV to take best action to safely reach the target, even after it gets deviated due to the unpredictable ocean waves. The data structure and the algorithms presented in this chapter, enables incorporating the effects of dynamics and uncertainty in ocean waves, in the computation of state transition map, helping in performing physics-aware trajectory planning.

The comparison of computational performance of GPU and CPU based model simplification techniques are shown in Table 2.2. The time taken to perform Monte Carlo simulations of the USV dynamics to compute state transition map using the algorithm developed in this chapter was 9.1 minutes. It should be noted that the CPU baseline was computed by implementing the parallel version (using OpenMP) of the simulator code reported in Ref. [40]. The overall speed-up using GPU in conjunction with temporal coherence based model simplification technique was by a factor of 43.4 by introducing an error of 1.04%. The number of states were chosen as 4800 ($20 \times 20 \times 12$) and the number of actions as 7. The value iteration took 28 s to converge when computed on the computed state transition map.

2.5 Conclusions

In this chapter, we presented GPU based algorithms to compute state-transition probabilities for USVs using 6 DOF dynamics simulations of interactions between ocean-wave and vehicle. We used the obtained state transition model in standard MDP based planning framework to generate physics-aware trajectory plans. We extended the dynamics model for USV simulation, reported in Ref. [40], to incorporate multiple wave components and random phase lags in ocean waves. The approach described in this chapter is flexible and is capable of handling any USV geometry, dynamics parameters, and sea-state. We developed GPU based algorithm to perform

Table 2.2: Summary of computational performance obtained using GPU and temporal coherence based simplification over CPU computations. Number of actions in action set $P = 7$ and number of simulation runs $M = 256$.

	CPU Baseline	GPU Baseline	CPU with clustering ($C = 50$) and temporal coherence ($\tau = 0.070$)	GPU with temporal coherence ($\tau = 0.075$)
Computation time (min)	395.0	28.2	80.2	9.1
Speed-up factor over CPU baseline	1.0	14.1	4.9	43.4
Error introduced (%)	0.0	0.0	1.25	1.04

fast Monte Carlo simulations to estimate state transition probabilities of USVs operating in high sea-states. We further improved the computational performance by developing model simplification algorithm based on temporal coherence. The overall computational speed-up obtained is by a factor of 43.4 by introducing a simulation error of 1.04% over the CPU baseline. Transition probabilities can be computed within 9.1 minutes by using the algorithms described in this chapter. The chapter also presents a case study to demonstrate the application of the developed state transition map to generate physics-aware trajectory plans for sea-states 3 and 4 in MDP based planning framework.

Chapter 3

Target Following with Motion Prediction for Unmanned Surface Vehicle Operating in Cluttered Environments

Contributors: Petr Švec, Atul Thakur, Eric Raboin, Brual C. Shah, and Satyandra K. Gupta

3.1 Introduction

Autonomous unmanned surface vehicles (USVs) [25, 48, 49, 50] can increase the capability of other surface or underwater vehicles by continuously following them in marine applications. These include sea-bed mapping and ocean sampling tasks [51, 52, 53] in environmental monitoring [54, 55, 56], cooperative surveillance by means of a network of heterogeneous vehicles (e.g., air, ground, and marine unmanned platforms) to provide situational awareness [57, 58, 59], search and rescue [60, 61], or harbor patrolling and protecting vulnerable areas [62, 63, 64, 1, 15].

Consider the case of a cooperative team of USVs guarding an asset against hostile boats in naval missions [15]. In these missions, the vehicles are required to approach passing boats, recognize adversaries, and possibly employ active blocking [1] to prevent the adversaries from reaching the asset. In order to maximize the guarding performance of the entire team, it is necessary for each USV to consider its own dynamics when approaching or following the boats. Moreover, each USV should be able to reliably compute its desired position $[x, y]^T$, orientation ψ, and surge speed u values, jointly defined as a motion goal, by estimating the future poses of the boats.

Performing autonomous follow task in an unfamiliar, unstructured marine environment (see Figure 4.1) with obstacles of variable dimensions, shapes, and motion dynamics such as other unmanned surface vehicles, civilian boats, adversaries, shorelines, or docks poses numerous planning challenges. The follow capability of the USV is inherently influenced by its own maneuverability constraints, the specific motion

characteristics of the target boat, the amount of knowledge about the intended motion of the target boat, sensing limitations, and the complexity of the marine environment. Due to differences in motion capabilities with respect to the target boat, the USV may not be able to track the same trajectory as the target boat. Instead, it may need to determine a different trajectory to follow, while keeping itself in proximity to the target boat and still avoid collisions. In addition, the USV may need to handle sharp turns while tracking the trajectory. Therefore, the mere utilization of manually developed and tuned control rules would not lead to sufficiently safe and task efficient follow strategies in environments with obstacles. In order to cope with the above outlined challenges, the USV needs to have a) the capability to estimate the future motion of the target boat based on its current state, dynamic characteristics, as well as obstacles in the operating environment, b) the capability to determine an advantageous and safe pose, i.e., in the form of a motion goal, in the close vicinity to the target boat, and c) fast, dynamically feasible trajectory planning and reliable trajectory tracking to guarantee physics-aware obstacle avoidance when approaching the motion goal.

We have developed a planning and tracking approach that incorporates a novel algorithm for motion goal computation, and tightly integrates it with trajectory planning and tracking components of the entire system to perform integrated planning and control. The motion goal is computed based on differential constraints of the USV and the target boat, expected motion of the target that is computed using a probability distribution over its possible control actions, and spatial constraints imposed by the environment. In particular, it forward-projects the control actions of the target boat to estimate a probability distribution of its future pose, computes candidate motion goals, selects the motion goal that minimizes the difference between the arrival time of the USV and the target boat to that goal, and attempts to minimize the length of the USV's trajectory.

The developed trajectory planner incorporates A* based heuristic search [65] to efficiently find a collision-free, dynamically feasible trajectory to a motion goal in a discretized state-action space, forming a lattice [35]. The trajectory is computed by sequencing predefined control actions (i.e., maneuvers or motion primitives) generated using the dynamics model of the USV [10, 66]. The trajectory is executed by a trajectory tracking controller to efficiently follow waypoints that make up the nominal trajectory. The controller computes the desired speed for each segment of the trajectory given the maximum allowable surge speed of the segment. This allows the USV to arrive to the motion goal at the required time. We have carried out experimental tests to derive the required acceleration and deceleration distance for the vehicle to match the desired speed.

In general, one can assume that the USV is capable of rejecting ocean disturbances in low sea states due to its mechanical design and the use of feedback based position control including roll stabilization [66]. However, this assumption may not be valid in higher sea states. Therefore, we determine collision zones around obstacles by computing the region of inevitable collision [67] and replan the trajectory with a high frequency to account for the uncertainty in the vehicle's motion. In addition, we assume that the combined set of sensors (e.g., lidar, stereo cameras, and radar),

digital nautical charts, and Kalman filtering [34] will provide us reasonable state estimation of obstacles as well as the USV.

Since the target boat may be moving with a high speed, the trajectory needs to be computed sufficiently fast and still preserve its dynamical feasibility in the close vicinity to the USV. The quality of the computed trajectory as well as the computational performance of the planner depends on the number and type of control actions and dimensionality of the underlying state space in different planning stages. A high-quality trajectory may be close to optimum in terms of its execution time but may take a long time to be computed on a given machine. On the other hand, a low-quality trajectory will be computed very quickly but may be longer with unnecessary detours. Both the cases would thus lead to a poor overall follow capability. Hence, it was necessary to sacrifice allowable level of optimality by superimposing high and low dimensional state spaces and utilizing a multi-resolution control action set. In particular, the dimensionality of the state space as well as the number of control actions is reduced with the increase of the distance from the USV towards the target. We have conducted a detailed empirical analysis to find a trade-off between fast computation and trajectory length.

The outline of the chapter is as follows. First, we present the definition of the problem in Section 3.2, followed by an overview of the overall approach in Section 3.3. This overview includes a description of the developed USV system architecture (see Figure 3.1) that integrates all planning modules. Second, we describe the state-action space representation that is used in both motion goal prediction in Section 3.5 as well as in nominal trajectory planning in Section 3.6.1. This is followed by a description of the developed trajectory tracking technique in Section 3.6.2. Finally, we present simulation as well as experimental results in Section 4.4.

3.2 Problem Formulation

Given,

(i.) a continuous, bounded, non-empty state space $X \subset \mathbb{R}^3 \times \mathbb{S}^1$ in which each state $\mathbf{x} = [x, y, u, \psi]^T$ consists of the position coordinates x and y, the surge speed u, and the orientation ψ (i.e., the heading angle) about the z axis of the Cartesian coordinate system (i.e., the heave, roll, and pitch motion components of the vehicle can be neglected for our application);

(ii.) the current state of the USV $\mathbf{x_U} = [x_U, y_U, u_U, \psi_U]^T$ and the moving target $\mathbf{x_T} = [x_T, y_T, u_T, \psi_T]^T$;

(iii.) an obstacle map Ω such that $\Omega(\mathbf{x}) = 1$ if $\mathbf{x} \in X_{obs} \subset X$, where X_{obs} is the subspace of X that is occupied by obstacles, otherwise $\Omega(\mathbf{x}) = 0$;

(iv.) a continuous, bounded, state-dependent, control action space $U_U(\mathbf{x}) \subset \mathbb{R}^2 \times \mathbb{S}^1$ of the USV in which each control action $\mathbf{u_U} = [u_d, \delta t, \phi_d]^T$ consists of the desired surge speed u_d, rudder angle ϕ_d, and execution time δt. Similarly, we are given a control action space $U_T \subset \mathbb{R}^2 \times \mathbb{S}^1$ for the target boat;

(v.) an action selection model $\pi_T : X \times U_T \rightarrow [0, 1]$ for the target boat defining a probability distribution over its control actions U_T;

(vi.) 3 degrees of freedom parametric model of the USV $\dot{\mathbf{x}}_\mathbf{U} = f_U(\mathbf{x}_\mathbf{U}, \mathbf{u}_{\mathbf{U,c}})$ [38], where the thrust and moment are generated by the vehicle's actuators. The actuators take $\mathbf{u}_{\mathbf{U,c}} = [u_c, \phi_c]^T$ as the control input, where u_c is the speed of the propeller in rpm, and ϕ_c is the rudder angle. Similarly, we are given the target boat dynamics $\dot{\mathbf{x}}_\mathbf{T} = f_T(\mathbf{x}_\mathbf{T}, \mathbf{u}_{\mathbf{T,c}})$;

Compute,

(i.) a motion goal $\mathbf{x_G} = [x_G, y_G, u_G, \psi_G]^T \in X_{free}$ together with the desired arrival time t_G to reach $\mathbf{x_G}$ from $\mathbf{x_U}$, where $X_{free} = X \backslash X_{obs}$; and

(ii.) a collision-free, dynamically feasible trajectory $\tau : [0, t] \rightarrow X_{free}$ between $\mathbf{x_U}$ and $\mathbf{x_G}$. The desired surge speed u_d of each trajectory segment $\mathbf{u_{U,k}}$ needs to be updated such that $\|\mathbf{x}_{\mathbf{U},\mathbf{t_G}} - \mathbf{x}_{\mathbf{G},\mathbf{t_G}}\| = 0$, where $\mathbf{x}_{\mathbf{U},\mathbf{t_G}}$ and $\mathbf{x}_{\mathbf{G},\mathbf{t_G}}$ represent the states of the USV at the time t_G.

The motion goal $\mathbf{x_G}$ and the trajectory τ are required to be recomputed in each planning cycle to keep track of the moving target, handle dynamic obstacles, and pose errors introduced due to motion uncertainty caused by the ocean environment. The state estimation is assumed to be provided by the extended Kalman filter [34].

3.3 Overview of Approach

There are three main components of the approach presented in this chapter, namely, motion goal prediction (see Section 3.5), trajectory planning (see Section 3.6), and trajectory tracking (see Section 3.6.2). The developed navigation, guidance, and control system architecture depicting the individual components is shown in Figure 3.1.

We first generate a discrete, lower-dimensional representation $X_d \subset \mathbb{R}^2 \times \mathbb{S}^1$ of the continuous state space $X \subset \mathbb{R}^3 \times \mathbb{S}^1$ in which each discrete state $\mathbf{x_{d,j}} = [x_j, y_j, \psi_j]^T$ omits the surge speed component u_j (see Section 3.4). In addition, we generate a discrete control action space representation $U_{U,d}$ of the continuous control action space U_U. In order to make the trajectory planning feasible, we decouple the search in the velocity space from the search for the trajectory in the pose space. We utilize the trajectory tracking component that determines the required speed for each segment of the trajectory to be able to reach a motion goal $\mathbf{x_G}$ at the required time t_G.

The discrete state and action space representations are used for constructing a 3D lattice structure [35] representing the discretized planning space to be used by the developed motion goal prediction and trajectory planning algorithms. The lattice structure allows us to reduce the planning complexity by adaptive discretization (see Section 3.6.1 and 3.6.1) but at the same time is capable of capturing vehicle's dynamic constraints in a resolution sense.

The motion goal prediction algorithm estimates the state of the target boat within future planning steps by forward-projecting its control actions based on the specified

probabilistic action selection model $\pi_{T,d}$. The algorithm computes a suitable location around the moving target in the form of a motion goal $\mathbf{x_G} \in X_{free}$. The motion goal is computed based on the current states of the boats $\mathbf{x_{U,d}}$ and $\mathbf{x_{T,d}}$, the obstacle map Ω, and $\pi_{T,d}$. The details of this layer are described in Section 3.5.

The trajectory planning component receives the motion goal $\mathbf{x_G}$ and computes a collision-free trajectory τ represented as a sequence of waypoints from the current state $\mathbf{x_{U,d}}$ of the USV to the motion goal. The planner searches the 3D lattice space [35, 8] (see Figure 3.3) to generate the nominal trajectory. The details of this component are described in Section 3.6.

Finally, the trajectory tracking component computes the desired surge speed for each segment of the trajectory, and sends speed and yaw control commands (i.e., required throttle and rudder positions) to the low-level controllers of the USV so that it can reliably navigate through the waypoints and arrive to the motion goal at the required time t_G. The details of this layer are described in Section 3.6.2.

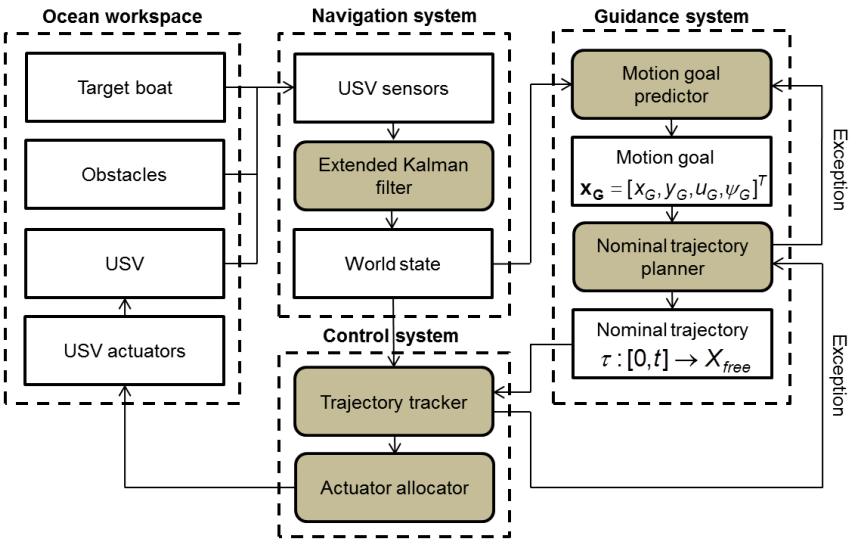


Figure 3.1: Developed USV system architecture for following a moving target in a marine environment with obstacles.

3.4 State-Action Space Representation

We discretize the continuous state space $X \subset \mathbb{R}^3 \times \mathbb{S}^1$ into a finite discrete space X_d. A state $\mathbf{x_{d,j}} = [x_j, y_j, \psi_j]^T \in X_d$ represents a position (x_j, y_j) and orientation ψ_j of the USV in a global inertial frame of reference. Depending upon the resolution of discretization, $\mathbf{x_{d,j}}$ also represents a polygonal (i.e., a rectangle in case of rectangular discretization) region in X_d in which the pose $[x_j, y_j, \psi_j]^T$ of the USV is assumed to be fixed. This discrete state space representation includes both position and orientation of the vehicle and hence is useful for incorporating constraints based on the dynamics of the USV. Choosing a suitable discretization of the lattice grid required trading

off the speed and resolution completeness [68] of the planner. In our approach, the discretization level was determined empirically by attempting coarse discretization first and then increasing the resolution of the grid until it satisfied our task. The two particular factors that we considered were the dimension of the vehicle and its control action set.

Similarly, we discretize the continuous control action space U_U into a discrete control action set $U_{U,d}$ and map each control action $\mathbf{u_{U,d,k}} \in U_{U,d}$ to a reachable pose $[\Delta x_k, \Delta y_k, \Delta \psi_k]^T$ in the body coordinate system of the vehicle. Each control action is compliant with the vehicle's dynamics f_U. We pre-compute the control actions in $U_{U,d}$ using a hand-tuned PID controller. However, other control techniques such as the sliding mode [69], backstepping [70], etc. can be used [71]. An example of a discrete control action set $U_{U,d} = \{\mathbf{u_{U,d,1}}, \ldots, \mathbf{u_{U,d,5}}\}$ is shown in Figure 3.2a) for the vehicle used in the experiments in Section 4.4. The choice of the number of control actions required trading off the computational performance, resolution completeness of the planner, and maneuverability of the vehicle. We have determined the number of control actions empirically by starting with three actions and then increasing their count until it satisfied this trade-off. However, there have been few techniques published recently that directly address the design of control action sets [72, 73, 74, 75, 76] and as such this problem is still an open research area.

We define a lattice L as a structure that maps $\mathbf{x_{d,j}}$ to $\mathbf{x_{d,j,k}}$ using a control action $\mathbf{u_{U,d,k}}$ for $j = 1, 2, \ldots, |S|$ and $k = 1, 2, \ldots |U_{U,d}|$. The lattice thus represents an instance of a graph with discretized states $\mathbf{x_{d,j}}$ as nodes and actions $\mathbf{u_{U,d,k}}$ as connecting arcs. Each action $\mathbf{u_{U,d,k}} \in U_{U,d}$ is used to establish a connection between dynamically reachable nodes in L. For a node $\mathbf{x_{d,j}} = [x_j, y_j, \psi_j]^T \in X_d$, the neighbor corresponding to an action $\mathbf{u_{U,d,k}} = [\Delta x_k, \Delta y_k, \Delta \psi_k]^T \in U_{U,d}$ is given by Equation 3.1. The lattice structure captures dynamics constraints in the form of dynamically feasible connecting edges between the nodes unlike orthogonal grids [68, 35]. The lattice can also be geometrically viewed as multiple X-Y planning layers representing 2D planning spaces with a fixed orientation ψ of the USV as shown in Figure 3.3.

$$\mathbf{x_{d,j,k}} = \begin{bmatrix} x_j \\ y_j \\ \psi_j \end{bmatrix} + \begin{bmatrix} cos\Delta\psi_j & -sin\Delta\psi_j & 0 \\ sin\Delta\psi_j & cos\Delta\psi_j & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta y_k \\ \Delta \psi_k \end{bmatrix} \tag{3.1}$$

It is necessary to appropriately design the control action set $U_{U,d}$ (i.e., the number, type, and duration of control actions) such that the required efficiency as well as the quality of the nominal trajectory is achieved. In our approach, we manually designed the control action set using a 3 degrees of freedom simulation model of the USV [8, 10, 3]. To generate more complex actions, gradient-based optimization techniques [77] may be used. The control action set can also be generated automatically using special techniques [72, 73, 74, 75, 76] that attempt to balance between the computational efficiency and quality of the generated trajectories.

The simulator is based upon a simplified 3-DOF model (see Equation 3.2 adapted from Fossen [38]).

$$
\begin{aligned}
m_{11}\dot{u} + d_{11}u &= T \\
m_{22}\dot{v} + d_{22}v &= 0 \\
m_{33}\dot{r} + d_{33}r &= M \\
\dot{x} &= ucos\psi - vsin\psi \\
\dot{y} &= usin\psi + vcos\psi \\
\dot{\psi} &= r
\end{aligned}
\tag{3.2}
$$

where d_{ii} represents the hydrodynamic damping, m_{ii} represents added mass terms, T represents thrust due to propeller actuation, and M represents the moment due to rudder actuation. The added mass terms m_{11}, m_{22} and m_{33} are computed using Equation 3.3 [71]. In this equation, L, W, and B are the length, width, and draft of the boat hull, respectively.

$$
\begin{aligned}
m_{11} &= 1.05m \\
m_{22} &= m + 0.5\rho\pi D^2 L \\
m_{33} &= \frac{m(L^2 + W^2) + 0.05mB^2 + 0.5\rho\pi D^2 L^3}{12}
\end{aligned}
\tag{3.3}
$$

There are many actuator models available for the USVs for computing actuation thrust T and moment M and can be substituted into Equation 3.2 [38, 39]. We used an actuation model [39] given in Equation 3.4.

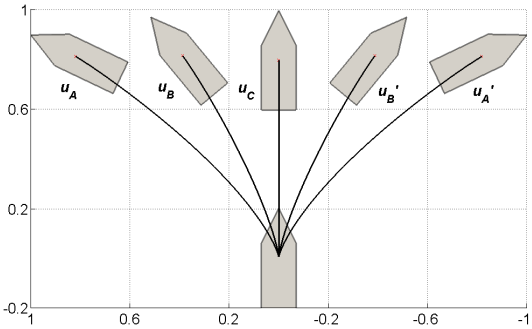
$$
\begin{aligned}
T &= K_1 \parallel u_c \parallel u_c \\
M &= K_2 K_1 \parallel u_c \parallel u_c \phi_c
\end{aligned}
\tag{3.4}
$$

where K_1 and K_2 are actuator constants, u_c is the propeller's rpm, and ϕ_c is the rudder angle.

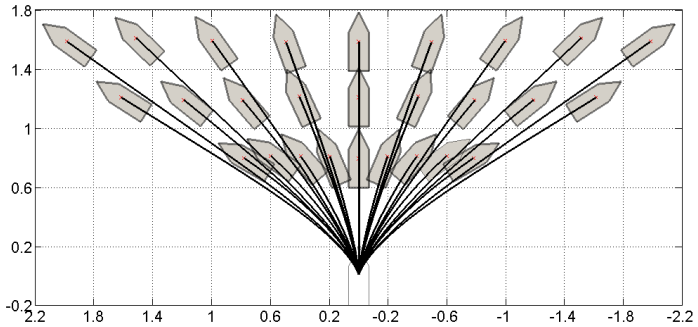
The target boat may have different motion characteristics than the USV. Hence, similarly as we discretize the state-action space of the USV, we discretize the state-action space for the target boat. This results in a discrete state space $X_{T,d}$ and action space $U_{T,d}$. This discretization is mostly needed for motion prediction of the target boat and motion goal computation as described in Section 3.5.

3.5 Motion Goal Prediction

To maintain a suitable distance between the USV and the target boat, the motion goal predictor selects a desired motion goal $\mathbf{x_G}$ and arrival time t_G. This is done by estimating the future poses of the target boat and then evaluating several candidate motion goals for the USV. Since the motion goal must be computed relatively quickly, we search for a sub-optimal solution by combining Monte-carlo sampling and heuristic evaluation techniques. The overall process is described in Algorithm 1.



(a) A sparse control action set containing 5 actions.



(b) A dense control action set containing 27 actions.

Figure 3.2: Action sets for building the lattice data structure. The dense action set discretizes the action space into 27 control actions. This set is particularly useful when a higher number of obstacles are present or demand on optimality is stricter. However, the dense action set makes the planning speed slower due to more number of connected nodes in the planning space, i.e., in the lattice.

The action selection model $\pi_{T,d} : X_{T,d} \times U_{T,d} \rightarrow [0, 1]$ defines a probability distribution over the target's discretized control actions at each pose $\mathbf{x_{T,d}} \in X_{T,d}$. To explore the future poses of the target, we sample N random trajectories $\tau_{T,j} \in R_T$ starting from the target's current pose and forward-projecting the target's actions up to some finite time horizon t_k. During the sampling process, each control action $\mathbf{u_{T,d,k}}$ is selected with probability $\pi_{T,d}(\mathbf{x_{T,d,i}}, \mathbf{u_{T,d,k}})$. Sampled trajectories that lead to a collision with an obstacle are discarded from R_T.

By recording the poses reached by the target boat along each sampled trajectory, it is possible to estimate the probability $P_{T,i}(x_j, y_j)$ that the target will be at position (x_j, y_j) at time t_i. A two-dimensional Gaussian kernel is used to smooth out the probability distribution represented by $P_{T,i}$, as illustrated in Figure 3.4. We also compute $\psi_{T,i}(x_j, y_j)$, the mean orientation at (x_j, y_j) across all samples at time t_i.

For each time point $t_i \in \{t_0, t_1, \ldots t_k\}$, the future position of the target boat can be approximated by

$$(x_i^*, y_i^*) = \arg \max_{(x_j, y_j)} P_{T,i}(x_j, y_j). \tag{3.5}$$

Similarly, let $\psi_i^* = \psi_{T,i}(x_i^*, y_i^*)$ approximate of the future orientation of the target boat. We select the candidate motion goal $\mathbf{x_{G,i}} \in X_{U,d} = [x_i, y_i, u_T, \psi_i]^T$ that is

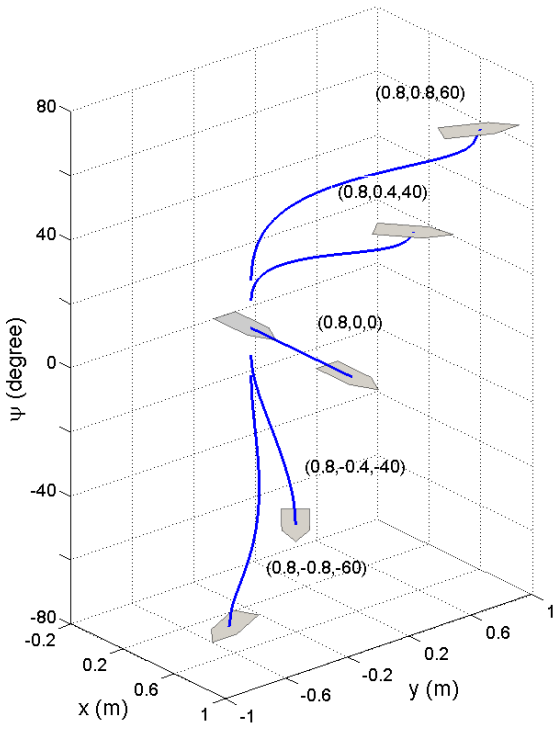


Figure 3.3: Planning space representation.

closest to the projected target pose at time t_i, such that

$$\mathbf{x_{G,i}} = \arg \min_{\mathbf{x_{G,j}} \in \mathbf{X_{U,d}}} |(x_i^*, y_i^*) - (x_j, y_j)| + \alpha \Delta(\psi_i^*, \psi_j) \tag{3.6}$$

where $\alpha\Delta(\psi_i^*, \psi_j)$ is the interior angle between ψ_i^* and ψ_j weighted by some parameter α.

Let $g(\mathbf{x_{G,i}})$ be the amount of time it takes the USV to travel from its current location $\mathbf{x_{U,d}}$ to the candidate goal $\mathbf{x_{G,i}}$. This can be computed by the A* heuristic search process described in Section 3.6.

We define the cost function for motion goal $\mathbf{x_{G,i}}$ as,

$$c(\mathbf{x_{G,i}}) = \begin{cases} \delta^{-i}(g(\mathbf{x_{G,i}}) - t_i + 1), & \text{if } g(\mathbf{x_{G,i}}) > t_i \\ \delta^{-i}, & \text{otherwise,} \end{cases} \tag{3.7}$$

where $g(\mathbf{x_{G,i}}) - t_i$ is the estimated difference in arrival time between the USV and target boat, and $\delta \in [0, 1]$ is a discount factor. The smaller that δ is, the stronger the bias towards earlier goals. From the set of candidate motion goals X_G, a final motion goal $\mathbf{x_G}$ is selected such that the cost function is minimized,

$$\mathbf{x_G} = \arg \min_{x_{G,i} \in X_G} c(\mathbf{x_{G,i}}). \tag{3.8}$$

Given the final motion goal $\mathbf{x_G}$, the desired arrival time t_G is determined by,

$$t_G(\mathbf{x_G}) = \begin{cases} g(\mathbf{x_G}), & \text{if } g(\mathbf{x_G}) > t_i + t_{lag} \\ t_i + t_{lag}, & \text{otherwise,} \end{cases} \tag{3.9}$$

Algorithm 1 ComputeMotionGoal()

Input: The current poses $\mathbf{x_{T,d}}$ and $\mathbf{x_{U,d}}$ of the target boat and USV, a probabilistic model of the target boat $\pi_{T,d}$, and a map of obstacles Ω.

Output: A desired motion goal $\mathbf{x_G}$ and arrival time t_G.

Let R_T be a set of N randomly generated trajectories for the target boat, where each trajectory $\tau_{T,i} \in R_T$ begins at state $\mathbf{x_{T,d}}$ and time t_0 and continues until time t_k, such that each sampled action $\mathbf{u_{T,d,j}}$ is selected with probability $\pi_{T,d}(\mathbf{x_{T,d,i}}, \mathbf{u_{T,d,j}})$.
for each time point $t_i \in \{t_1, t_2, \dots t_k\}$ **do**
  for each trajectory $\tau_{T,j} \in R_T$ **do**
    Increment $P_{T,i}(x_{i,j}, y_{i,j})$ by $1/N$, where $(x_{i,j}, y_{i,j})$ is the location of $\tau_{T,j}$ at time t_i.
  end for
  Smooth $P_{T,i}$ by applying an $m \times m$ Gaussian kernel.
  Let (x_i^*, y_i^*) be the location that maximizes $P_{T,i}(x_i^*, y_i^*)$ and let ψ_i^* be the mean orientation at (x_i^*, y_i^*) across all trajectories in R_T at time t_i.
  Let $\mathbf{x_{G,i}} \in X_{U,d} = [x_j, y_j, u_U, \psi_j]$ be a candidate motion goal for the USV which minimizes the distance to the projected state, $|(x_i^*, y_i^*) - (x_j, y_j)| + \alpha\Delta(\psi_i^*, \psi_j)$
end for
Let $\mathbf{x_G}$ equal the candidate motion goal $\mathbf{x_{G,i}} \in X_G$ that minimizes the cost function $c(\mathbf{x_G})$ and ensures a collision-free path up to time T_{ric}
Compute a desired arrival time $t_G(\mathbf{x_G})$ such that the USV will arrive at $\mathbf{x_G}$ shortly after the target boat
return $(\mathbf{x_G}, t_G(\mathbf{x_G}))$.

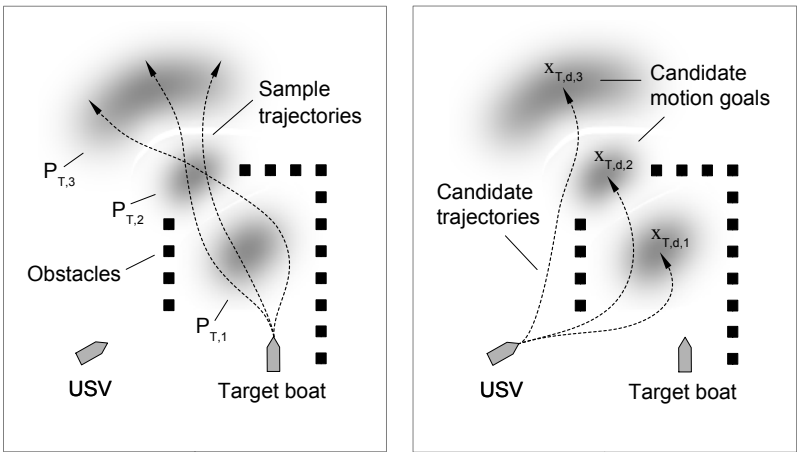
where t_{lag} is the desired amount of time that the USV should follow behind the target boat. This ensures that the USV will reduce its speed when it can afford to do so, such as when it is already close the target boat.

Due to the cost function $c(\mathbf{x_G})$, the algorithm only considers motion goals which guarantee a collision-free trajectory for the USV up to some time T_{ric}. States which lead to an inevitable collision are given an infinite cost, as described in detail in Section 3.6.1. For simplicity, we may use sampled poses from R_T if the dynamics of the USV and target boat are equivalent, since the trajectories in R_T are guaranteed to be collision free. If no collision-free motion goals are found, the algorithm will default to the currently assigned motion goal.

3.6 Trajectory Planning and Tracking

3.6.1 Nominal Trajectory Planning

The nominal trajectory planner generates a collision free trajectory $\tau : [0, t] \to X_{free}$ between the current state $\mathbf{x_U}$ of the USV and the motion goal $\mathbf{x_G}$ in the obstacle field Ω. The trajectory τ is computed by concatenating control actions in $U_{U,d}$ which are



(a) (b)

Figure 3.4: Motion goal prediction steps (a) Probability distributions $P_{T,1}$, $P_{T,2}$ and $P_{T,3}$, generated by sampling future target trajectories at several time points. (b) Candidate motion goals generated by selecting the most probable target pose.

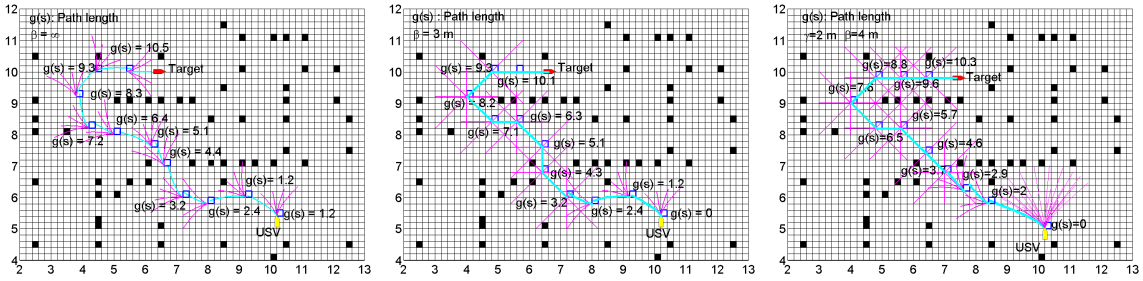
designed a-priori. The planner is based on A* heuristic search [65] over the lattice. The planner associates $\mathbf{x_G}$ and $\mathbf{x_U}$ with their closest corresponding states $\mathbf{x_{G,d}}$ and $\mathbf{x_{U,d}}$ in the lattice L. Similarly, the planner maps the position of every obstacle $\mathbf{x_O} \in X$ to its closest corresponding state $\mathbf{x_{O,d}} \in X_d$.

We also extract the region of inevitable collision [67] in X_d for the given obstacle field Ω. In general, the region of inevitable collision is defined as a set of states that lead to a collision regardless of a control action the vehicle executes. We avoid searching for the trajectory in this region (i.e., during the search, we do not expand candidate nodes that fall into this region) by assigning an infinite cost to all its states.

In this chapter, we approximate the region of inevitable collision by determining a set of lattice nodes from which the vehicle will collide with an obstacle after a specified time horizon. In the computation, we assume that the vehicle will continue in its course by moving straight and with maximum speed. This is generally a valid assumption as the length scale of the control action is much smaller and possibility of sudden steering is less in such a small distance. In addition, our 3D pose based representation takes orientation of the USV into account. A USV that is very close to an obstacle can be said to be inside the region of inevitable collision if it is approaching the obstacle. On the other hand, if the USV is moving away from the obstacle from the same pose then it can be said to be out of the region of inevitable collision.

More formally, for each lattice node $\mathbf{x_{d,j}} \in X_d$ that represents a pose $[x_j, y_j, \psi_j]^T$ of the USV, we use the dynamic model of the vehicle to determine a lattice node $\mathbf{x'_{d,j}}$ that represents the vehicle's pose after the time T_{ric}. If the forward projected lattice node $\mathbf{x'_{d,j}}$ lies on an obstacle, i.e., $\Omega(\mathbf{x'_{d,j}}) = 1$, then we label $\mathbf{x_{d,j}}$ to be lying inside the region of inevitable collision (see Equation 3.10).

$$RIC(\mathbf{x_{d,j}}, u_{max}, T_{ric}) = \begin{cases} 1, \text{ if } \mathbf{x'_{d,j}} \in \Omega \\ 0, \text{ otherwise,} \end{cases} \tag{3.10}$$



(a) (b) (c)

Figure 3.5: (a) A trajectory computed by searching entirely in 3D pose state space ($\beta = \infty$). (b) A trajectory computed by searching partially in 3D pose state space ($\beta = 3$ m) and 2D position state space. (c) A trajectory computed using a dense action set with 27 control actions up to the trajectory length of $\gamma = 2$ m with $\beta = 4$ m. The boat length is 0.6 m.

In Equation 3.10, $\mathbf{x'_{d,j}}$ is the lattice node closest to the actual state $\mathbf{x'_j} = \mathbf{x_j} + T_{ric}u_{max}[cos\psi_j, sin\psi_j, 0, 0]^T$, and u_{max} is the maximum surge speed of the vehicle.

The time horizon T_{ric} for determining the region of inevitable collision can be set by the user of the planning algorithm depending upon the preferred allowable risk. In particular, in this chapter, we set T_{ric} experimentally based on a state transition model of the USV (i.e., a probabilistic state transition function) which was developed in our previous work [8]. The state transition model gives us a probability distribution over future states of the vehicle given its initial state, a control action set, and sea state. We developed this model using Monte Carlo runs of a high-fidelity 6 DOF dynamics simulation of interaction between the USV and ocean waves. Our simulator can handle any USV geometry, dynamics parameters, and sea state. This approach for computing the region of inevitable collision allows us to more precisely define the size of the collision zones and thus make the planner less conservative when planning in narrow regions.

During the search for the trajectory, lattice nodes are incrementally expanded towards $\mathbf{x_{G,d}}$ in the least-cost fashion according to the trajectory cost $f(\mathbf{x_d}) = g(\mathbf{x_d}) + h(\mathbf{x_d})$, where $g(\mathbf{x_d})$ is the optimal cost-to-come from $\mathbf{x_{d,U}}$ to $\mathbf{x_d}$, and $h(\mathbf{x_d})$ is the heuristic cost-to-go between $\mathbf{x_d}$ and $\mathbf{x_{G,d}}$. The admissible heuristic cost function $h(\mathbf{x_d})$ (i.e., not allowed to overestimate the actual cost of the trajectory) reduces the total number of expanded nodes in the lattice as per the A* graph search algorithm that guarantees optimality of the computed plan.

The cost function was designed in terms of trajectory length. The cost-to-come $g(\mathbf{x_d})$ represents the total length of a trajectory between the current state $\mathbf{x_{U,d}}$ of the vehicle and $\mathbf{x_d}$ and is computed as $g(\mathbf{x_d}) = \sum_{k=1}^{K} l(\mathbf{u_{U,d,k}})$ over K planning stages, where $l(\mathbf{u_{U,d,k}})$ is the length of the projected control action $\mathbf{u_{U,d,k}} \in U_{U,d}$. If the control action $\mathbf{u_{U,d}}$ takes the vehicle to a collision state $\mathbf{x_{col,d}}$ (i.e., the state for which $\Omega(\mathbf{x_{col,d}}) = 1$ or $RIC(\mathbf{x_{col,d}}, u_{max}, T_{ric}) = 1$), then $l(\mathbf{u_{U,d}})$ is set to ∞. Each

control action is sampled using intermediate waypoints. These waypoints are then used for collision checking. The heuristic cost $h(\mathbf{x_d})$ is represented as the Euclidean distance between $\mathbf{x_d}$ and $\mathbf{x_{G,d}}$. Alternatively, according to [35], one can utilize a precomputed look-up table containing heuristic costs of trajectories between different pairs of states in an environment without obstacles.

The dynamically feasible trajectory τ is a sequence $\{\mathbf{u_{U,d,1}}, \mathbf{u_{U,d,2}}, \ldots, \mathbf{u_{U,d,K}}\}$ of precomputed atomic control actions, where $\mathbf{u_{U,d,k}} \in U_{U,d}$ for $k = 1, \ldots, K$. This sequence of control actions is then translated into a sequence of $K+1$ waypoints $\{\mathbf{w_{d,1}}, \ldots, \mathbf{w_{d,K+1}}\}$ leading to the motion goal $\mathbf{x_{G,d}}$. Thus, the generated nominal trajectory is guaranteed to be dynamically feasible and ensures that the USV will be able to progressively reach the waypoints in a sequence without substantial deviations, which otherwise may lead to collisions.

In case of the described *follow* task, the motion goal keeps changing dynamically and hence the trajectory planner needs to compute the plans repeatedly. For this reason, the planner must be very fast and efficient. For the very requirement of following the target boat in a cluttered space, the planning space must be 3D pose based. A higher dimensional planning space may significantly slow down the trajectory planner. For this reason, we enhanced the computational performance of the planner by (a) superimposing higher and reduced dimension state spaces, and (b) utilizing a control action set with multiple levels of resolution. These two extensions are explained in detail in the following two subsections.

Superimposition of Higher and Reduced Dimension State Spaces

Superimposing pose based 3D lattice and position based 2D grid state space representations can improve planner performance by sacrificing allowable level of optimality. At closer distances (i.e., $g(\mathbf{x_d}) < \beta$) from the initial state $\mathbf{x_{d,U}}$, the planning space is chosen to be a 3D lattice, whereas at a farther distance (i.e., $g(\mathbf{x_d}) \geq \beta$) the planning space is chosen to be a 2D grid. This scheme is useful as trajectory waypoints in a closer vicinity to the USV must be dynamically feasible and hence should be on a 3D lattice. However, farther waypoints can be tolerated to be on a rectangular grid. This simplification may, however, lead to the computation of a sub-optimal trajectory plan.

The threshold distance β can be set by the user of the planning system depending upon the need of performance improvement and tolerance to trajectory sub-optimality. In order to superimpose the 2D and 3D representations, we have designed a neighbor function (see Equation 3.11). The 2D control action set $U^{2D}_{U,d}$ unlike the 3D control action set $U_{U,d}$ has only 2D actions with their terminal orientations along the x axis.

$$neighbor(\mathbf{x_d}) = \begin{cases} \{\mathbf{x_d} + \mathbf{u_{U,d,k}} | \mathbf{u_{U,d,k}} \in U_{U,d}\}, \text{ if } g(\mathbf{x_d}) < \beta \\ \{\mathbf{x_d} + \mathbf{u^{2D}_{U,d,k}} | \mathbf{u_{U,d,k}} \in U^{2D}_{U,d}\}, \text{ otherwise.} \end{cases} \tag{3.11}$$

The above neighbor function restricts the search to 2D space when the trajectory length $g(\mathbf{x_d})$ increases beyond β. Figure 3.5a) shows a trajectory being searched completely in 3D pose space, where the parameter β is set to infinity. The trajectory

being searched for the case when $\beta = 7$ m is depicted in Figure 3.5b). As soon as the length of the trajectory gets greater than this threshold (i.e., $\beta > 7$), the search continues in 2D position space.

Multi-resolution Control Action Set

In the present problem of following a moving target boat, the generated trajectory is useful only up to a limited time horizon. This is because as the USV follows the trajectory, the target boat also moves and thus a large portion of the original trajectory can quickly become obsolete. This necessitates recomputation of the trajectory by the USV, which implies that the trajectory can be of smaller accuracy at a farther distance from the USV. By using a dense action set (see Figure 3.2b) in close proximity (i.e., $g(\mathbf{x_d}) \leq \gamma$) to the USV and a sparse action set (see Figure 3.2a) at a farther distance (i.e., $g(\mathbf{x_d}) > \gamma$), the computation time can be significantly reduced. This, however, may result in a highly suboptimal trajectory. So the user of the system needs to carefully tune γ in order to get the best balance between the speed of computation and quality of the generated trajectory.

The combined influence of the parameters β (introduced in Section 3.6.1) and γ is shown in Figure 3.5c. In this case, the planner searches through a graph connected using a dense, dynamically feasible control action set up to the trajectory length γ, then it switches to using a sparse, dynamically feasible action set for $max(0, \beta - \gamma)$ distance, and finishes the search through 2D position space for the rest of the trajectory length.

3.6.2 Nominal Trajectory Tracking

We have developed a controller to track the generated trajectory. The controller is based on the classic Line-Of-Sight (LOS) algorithm [38, 78] for driving the USV between consecutive waypoints. According to the LOS algorithm, the vehicle always heads towards the next waypoint when it gets within a *circle-of-acceptance* of the current waypoint. During following the waypoints, the controller is not responsible for considering obstacle avoidance, as the set of waypoints generated by the planner is guaranteed to be in X_{free} as described in Section 3.6.1.

The boat dynamics considered in this chapter is underactuated and hence its speed in the sway direction cannot be controlled directly. The control required to follow waypoints must be divided between surge speed and heading controllers. We accomplish this by using a PID based cascaded surge speed u and yaw ψ rate controllers. The heading controller produces corresponding rudder commands. We assume that the roll and pitch is maintained to zero.

The desired heading angle at i^{th} time instance is calculated using $\psi_{U,d} = atan2(y_i - y, x_i - x)$, where $[x, y]^T$ is the current position of the USV and $\mathbf{w_{d,i}} = [x_i, y_i, u_i]^T$ is the current active waypoint, where u_i is the desired speed for reaching this waypoint. After this, the yaw error is computed using $e_{\psi_U} = \psi_{U,d} - \psi_U$, where ψ_U is the current orientation of the USV. The yaw error is then used by the PID controller to obtain the control action, i.e., the rudder angle ϕ_i.

Algorithm 2 ComputeDesiredSpeed(): Compute desired surge speed for all segments of the trajectory τ so that the USV can reach its motion goal x_G at the required time t_G.

Input: A trajectory $\tau = \{\mathbf{u_{U,d,k}}\}_{k=1}^{K}$ that defines the maximum allowable speed $u_{d,k,max}$ for each of its segments $\mathbf{u_{U,d,k}}$, and the required arrival time t_G to the motion goal x_G.

Output: Desired surge speeds $V_d = \{u_{d,k}\}_{k=1}^{K}$ for all trajectory segments $\{\mathbf{u_{U,d,k}}\}_{k=1}^{K}$.

1: Compute initial desired surge speeds $V_d = \{u_{d,k}\}_{k=1}^{K}$ for all segments of the trajectory τ. The initial speed is computed as $u_{d,k} = L/t_G$, where $L = \sum_{k=1}^{K} l(\mathbf{u_{U,d,k}})$ is the total length of the trajectory and $l(\mathbf{u_{U,d,k}})$ is the length of the trajectory segment $\mathbf{u_{U,d,k}}$.
2: Let Q be the priority queue containing the trajectory segments $\{\mathbf{u_{U,d,k}}\}_{k=1}^{K}$ in ascending order according to their maximum allowable surge speed $u_{d,k,max}$.
3: **while** Q not empty **do**
4: Remove a segment $\mathbf{u_{U,d,k}} \leftarrow Q$.First() from Q with the minimum allowable surge speed $u_{d,k,max}$.
5: Compute the time lost $t_{lost} = t_{d,k,max} - t_{d,k}$ when traversing $\mathbf{u_{U,d,k}}$ with $u_{d,k,max}$. The time needed to traverse the segment $\mathbf{u_{U,d,k}}$ under the constraint $u_{d,k,max}$ is $t_{d,k,max} = l(\mathbf{u_{U,d,k}})/u_{d,k,max}$, whereas $t_{d,k} = l(\mathbf{u_{U,d,k}})/u_{d,k}$ is the time it takes to traverse the segment without any speed constraints.
6: **if** $t_{lost} > 0$ **then**
7: Set $u_{d,k} \leftarrow u_{d,k,max}$ in V_d.
8: Set $u_{d,l} = 1/(t_{d,l} - t_{lost}/|Q|)$ for all remaining segments $\mathbf{u_{U,d,l}} \neq \mathbf{u_{U,d,k}}$ in V_d, where $t_{d,l} = l(\mathbf{u_{U,d,l}})/u_{d,l}$.
9: **end if**
10: **end while**
11: **return** V_d.

$$\phi_i = P_{yaw} e_{\psi_U} + I_{yaw} \int e_{\psi_U} dt + D_{yaw} \frac{de_\psi}{dt} \tag{3.12}$$

In Equation 3.12, P_{yaw}, I_{yaw}, and D_{yaw} are proportional, integral, and derivative parameters of the PID controller, respectively.

Due to the discretization of the state and control action spaces, there may occur small gaps between two consecutive control actions in the computed trajectory (see Fig. 3.5a). In order to ensure reliable tracking of such trajectory, we define a region of acceptance around each waypoint and increase the size of the collision zones around the obstacles to minimize the probability of a collision. This also resolves other possible tracking problems that may arise because of the use of an estimated USV's dynamics when computing the control action set for the trajectory planner. Moreover, this also prevents creation of loops when the vehicle is not able to precisely reach a given waypoint. It is possible to design control actions such that they connect seamlessly. However, this requires substantial design effort. Lastly, since the trajectory is recomputed with a high frequency and the controller always follows the first

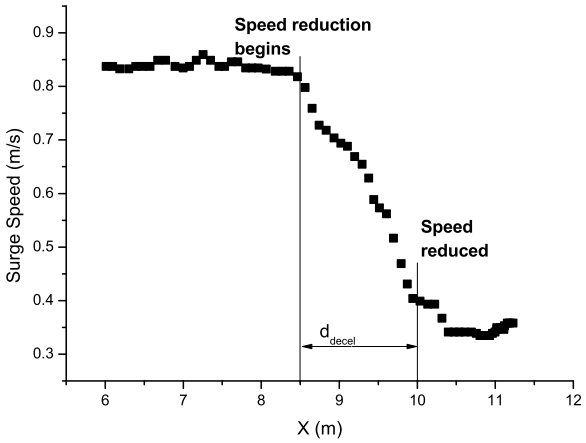


Figure 3.6: The boat is commanded to go along x axis up to the distance of 8.5 m with the surge speed of 0.8 ms^{-1}, after which the surge speed is lowered to 0.3 ms^{-1} resulting in the deceleration distance d_{decel} of 1.5 m.

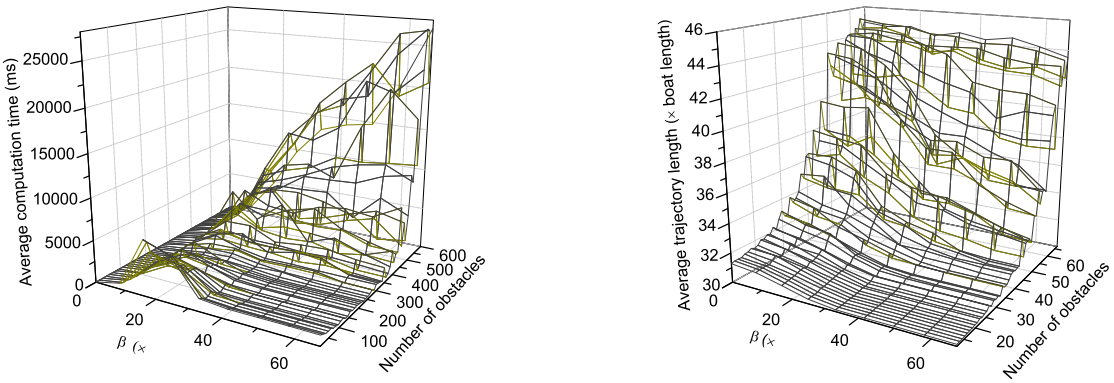
few, dynamically-feasible waypoints along the trajectory, the USV will not experience the discontinuous transition between 3D and 2D lattices (see Section 3.6.1).

In the follow task, the controller must be able to handle variations in the surge speed during steep turns. Each segment $\mathbf{u_{d,k}}$ of the trajectory τ thus defines the maximum allowable surge speed $u_{d,k,max}$ of the vehicle. In addition, the USV cannot immediately decrease its surge speed before turning, rather it should start decelerating in advance to attain the desired speed near the turn.

In our experiments, we selected a higher speed of 0.8 ms^{-1} for straight segments of the trajectory while a smaller speed of 0.3 ms^{-1} for turns. We determine the distance d_{decel} required for the USV to decelerate to a given surge speed using physical experiments. Figure 3.6 shows the result of an experimental test to determine d_{decel}. In this test, we controlled the boat to go in a straight line along the x axis with a surge speed of 0.8 ms^{-1}. At a distance of 8.5 m, the boat was commanded to reduce its surge speed to 0.3 ms^{-1}. We used PID controller for controlling the surge speed. We determined that the boat needs to run for about $d_{decel} = 1.5$ m in order to decelerate to the desired speed of 0.3 ms^{-1}.

We adjust the desired surge speed $u_{d,k}$ of each trajectory segment $\mathbf{u_{U,d,k}}$ (i.e., the desired speed for reaching each waypoint $\mathbf{w_{d,k}}$) so that the USV arrives to $\mathbf{x_G}$ at the required time t_G. We compute the desired speed iteratively (see Algorithm 2) given the maximum allowable surge speed $u_{d,k,max}$ constraint defined for each segment of the trajectory. We start with the segment with the minimum allowable speed, compute the time lost while traversing this segment, and uniformly increase the speed of other segments to compensate for the time loss. The algorithm continues until no segment is left which requires adjusting its speed.

The algorithm is not optimal since it does not account for the speed transitions between the segments. However, since the trajectory is supposed to be recomputed with a high frequency, the desired speed of the segments is gradually adjusted with each newly computed trajectory and the state of the vehicle.



(a) (b)

Figure 3.7: Variation of the average trajectory computation time (see plot a)) and trajectory length (see plot b)) with respect to the trajectory length threshold β for switching between 3D and 2D representations of the state space.

Once the desired surge speed $u_{d,k}$ is computed for all segments of the trajectory τ, then the error for the speed controller is computed using $e_u = u_{d,k} - u$, where u is the current surge speed of the vehicle. The speed error e_u is then used in the surge PID controller to obtain the control action, i.e., the throttle control voltage $V_{throttle}$.

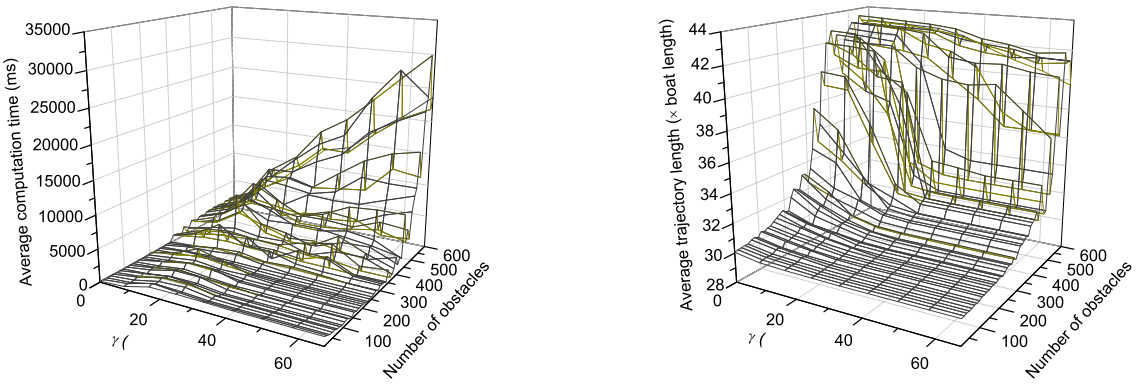
$$V_{throttle} = P_v e_v + I_v \int e_v dt + D_v \frac{de_v}{dt} \tag{3.13}$$

Similarly as in Equation 3.12, the parameters P_v, I_v, and D_v in Equation 3.13 represent proportional, integral, and derivative terms of the PID controller, respectively.

3.7 Simulation and Experimental Results

3.7.1 Numerical Experimental Setup for Evaluation of Trajectory Planner Parameters

We conducted simulation experiments in order to test the influence of β and γ parameters on the trajectory length and planner performance. For these experiments, we generated 50 obstacle densities with the increasing number of obstacles from 12 to 600 in the increments of 12. For each obstacle density, we created 25 randomly generated cases. So we acquired $50 \times 25 = 1250$ obstacle scenes with 50 different obstacle densities. In addition, we generated 50 instances of initial and goal locations of the USV for each obstacle scene, resulting in the total of $50 \times 25 \times 50 = 62500$ test cases. For each test case, we computed a trajectory and recorded its length and computation time using the developed trajectory planner (see Section 3.6.1). In the presented results, the parameters β and γ, trajectory length, and the dimension of



(a) (b)

Figure 3.8: Variation of the average trajectory computation time (see plot a)) and trajectory length (see plot b)) with respect to the trajectory length threshold γ for switching between a control action set consisting of 27 and 5 control actions.

the environment are expressed as multiples of boat length that was set to 0.6 m. The dimension of the environment was set to 33.3×33.3 boat lengths (i.e., 20×20 m).

3.7.2 Evaluation of β Parameter

In order to test the influence of the parameter β on the computational time and trajectory length, we varied its value from 0 to 67 boat lengths (i.e., 0 to 40 m given the boat length of 0.6 m) in the increment of 8 (i.e., 5 m). Corresponding to each value of β, we generated a trajectory for each of the 62500 cases as described above. This led to $9 \times 62500 = 562500$ evaluations in total. This included computation of $25 \times 50 = 1250$ trajectories for each value of β and a specific obstacle density. The plot in Figure 3.7a) shows the variation of the computation time averaged over 1250 samples for each value of the β parameter and obstacle density. Similarly as in the previous subsection, the plot in Figure 3.7b) shows the variation of average trajectory length as multiples of boat length.

A few observations can be made as follows:

1. As the value of the parameter β increases, the computation time also increases up to a certain value of β. The reason is that as β increases, the length of the trajectory for which the search is performed in 3D space is increased in proportion to the trajectory length for which the search is carried out in 2D space. However, the computation time starts reducing beyond a particular value of β. This occurs because if the trajectory is searched in 3D up to a larger distance, much shorter trajectory may be found. A shorter trajectory entails a smaller number of nodes being expanded, which decreases the required computational time. For further higher values of β, the computation time saturates.

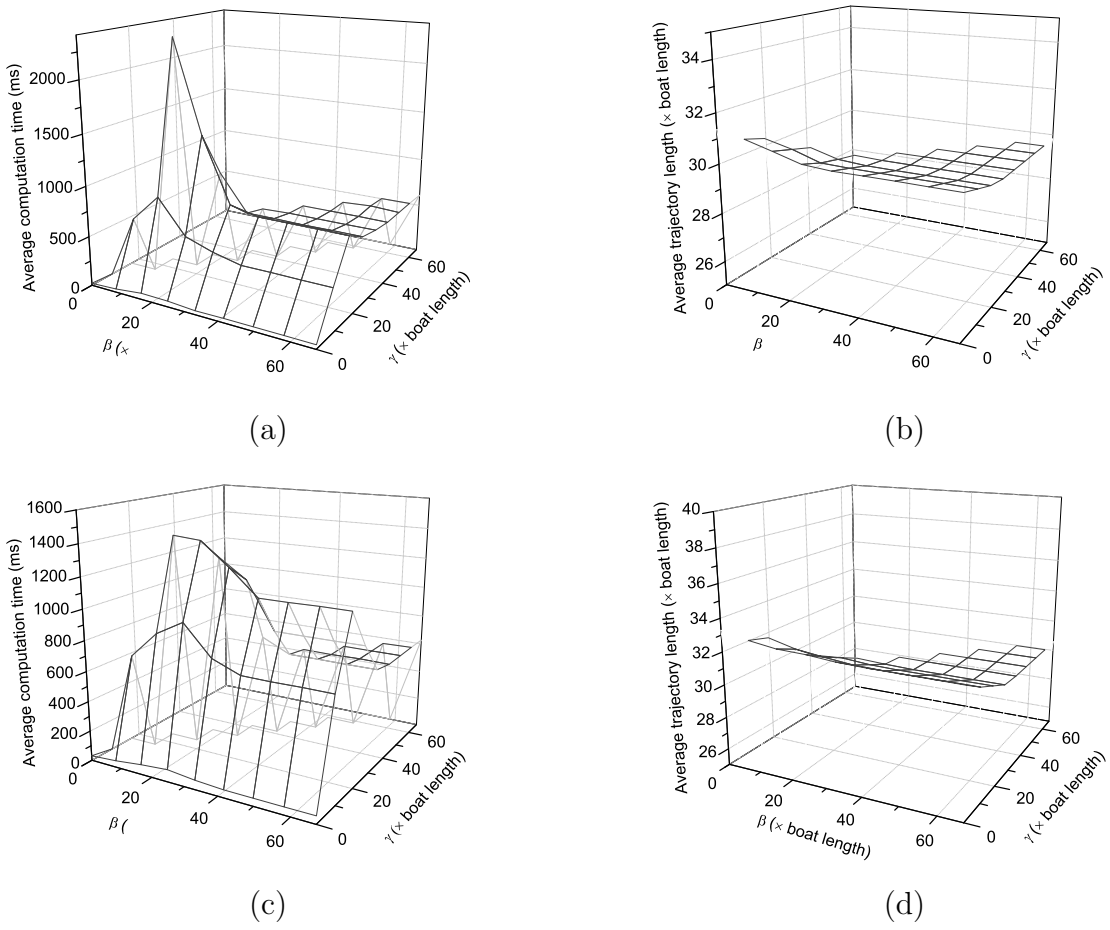


Figure 3.9: Variation of the average computation time with threshold parameters γ and β for 120 (see plot a)) and 300 obstacles (see plot c)) in the scene. Variation of the average trajectory length with threshold parameters γ and *beta* for 120 (see plot b)) and 300 obstacles (see plot d)) in the scene.

This is because, beyond the optimum trajectory length determined in 3D space, increasing β has no effect.

2. The saturation value of computation time occurring for large values of β is greater than the computation time of the pure 2D search.

3. The trajectory length is large when β is small as 2D search is not able to exploit smooth turns that are possible using 3D lattice based search. This however is not true in general for all cases. For example, a lattice based structure will prevent transitions to adjoining states due to inbuilt vehicle constraints. However, we would like to report here that in general, the average case trajectory length can be smaller for a lattice search as compared to pure 2D search.

4. The value of β should be chosen such that it does not lie in the peaked region. From the experimental test cases we can determine the prohibited regions of β in which computation performance becomes worse for a given obstacle density. In other words, there is a certain range of β for a given obstacle density for which the planner performance gets deteriorated, however, any value of β less than that will result in performance improvement. This improvement in the performance, however, will have an adverse effect on the optimality of the generated trajectories. This can be observed from the plot of trajectory length variation versus the parameter β (see Figure 3.7b)), which shows that for smaller β, the lengths of trajectories are greater. A balance can be chosen by the user of the algorithm to get performance gain by incurring allowable loss of optimality of the computed trajectory.

3.7.3 Evaluation of γ Parameter

In order to test the influence of the parameter γ on the computational time and trajectory length, we varied its value from 0 to 67 boat lengths in the increment of 8 as described above for the parameter β. This lead to $9 \times 62500 = 562500$ evaluations in total. For each γ and obstacle scene we generated 1250 trajectories. The plot in Figure 3.8a) shows the variation of the computation time averaged over 1250 samples for each value of the γ parameter and obstacle density. Similarly, the plot in Figure 3.8b) shows the variation of average trajectory length.

A few observations can be made as follows:

1. As the parameter γ increases, the average computation time also increases. This is because the trajectory is searched in a densely connected lattice up to γ distance of trajectory length, after which the sparsely connected lattice is used. However, beyond a certain limit of γ, the trajectory length gets smaller, causing lesser number of nodes being expanded and hence, the computation time reduces again. This effect is similar to the effect observed with the parameter β. The computation time saturates at values greater than optimal trajectory length searched in the densely connected lattice.

2. The saturation value of the computational time occurring for large values of γ (searched purely in densely connected lattice) is higher than that of very small values of γ (searched purely in sparsely connected lattice).

3. The average trajectory length reduces as γ increases because trajectory searched in a dense lattice (i.e., due to a large value of γ) is more optimal compared to the one searched in a sparsely connected lattice.

4. The value of γ should be chosen such that it does not lie in the peaked region. From the experimental test cases, we can determine the prohibited regions of γ in which computation performance becomes worse for a given obstacle density. This is again similar to the behavior of β parameter.

3.7.4 Evaluation of Combined Effect of β and γ Parameters

In order to study the combined influence of β and γ parameters, we have designed a test case, with β and γ varying from 0 to 67 of boat lengths. It should be noted that $\beta \geq \gamma$, as the influence of γ (i.e., computing in densely connected lattice) takes priority over the influence of β. For each combination of the parameters γ and β we ran tests in two obstacle densities (namely with 120 and 300 obstacles). For each obstacle density we ran 25 randomly generated cases as described before. This resulted in running 62500 cases and computing a trajectory for each case. The variation of average trajectory lengths and computation time is illustrated in Figures 3.9a-d). The plots show the prohibited regions for each combination of β and γ. In these regions, average computation time increases.

3.7.5 Simulation Results of Overall Planning Approach

To evaluate the motion goal prediction algorithm described in Section 3.5 we conducted a series of simulation experiments using randomly generated test cases. Each test case consisted of a fixed trajectory for the target boat and a set of 48, 72, 96, 120 or 144 randomly placed obstacles in an environment with the dimension of 33×33 boat lengths (i.e., $20 \times 20\ m$). We generated 200 different trajectories for a total of 1000 different test cases. An example test case with the trajectories for both the USV and target boat is shown in Figure 3.10c.

The USV and target boat were given a maximum velocity of 0.7 boat lengths/s (i.e., $0.4\ m/s$). Each second, the simulator invoked calls to compute a new motion goal for the USV, either using the *heuristic* motion goal predictor described in Section 3.5, or by directly using the current pose of the target boat, specified by $\mathbf{x_{T,d}}$. The purpose of the experiments was to demonstrate the effectiveness of the heuristic predictor of the motion goal when compared to the simple target boat tracking. The comparison was made in terms of the distance traveled by the USV and the amount of time spent inside a circular proximity region X_P around the target, specified by minimum radius r_{min} and maximum radius r_{max}. Figure 3.10b illustrates the reduction in trajectory length, with a trajectory approximately 3.3 boat lengths shorter than the trajectory in Figure 3.10a generated by the USV using the simple tracking.

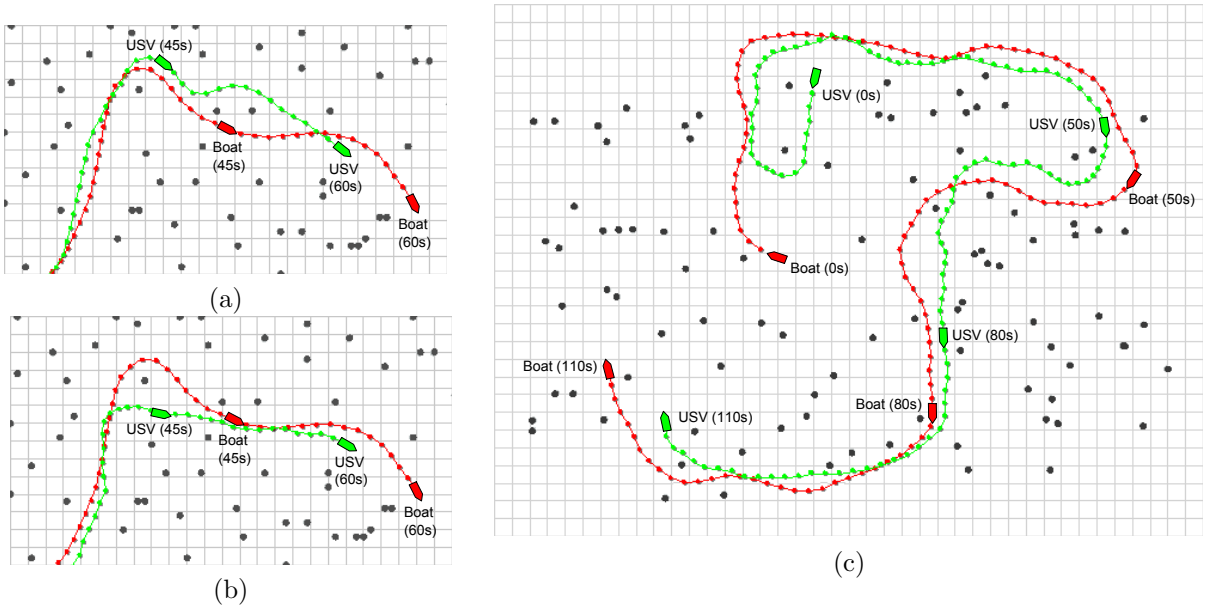


Figure 3.10: (a) Trajectory of the USV using simple motion goal prediction. (b) Trajectory of the USV using heuristic motion goal prediction. (c) Trajectory of the USV using heuristic motion goal prediction in a randomly generated scenario with 96 obstacles.

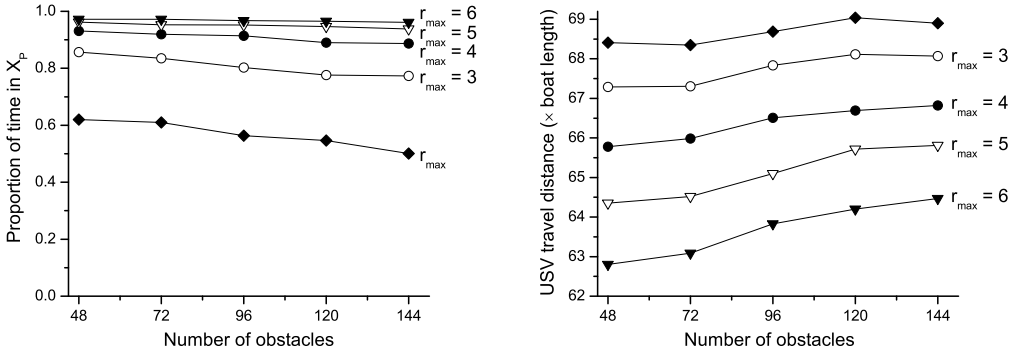
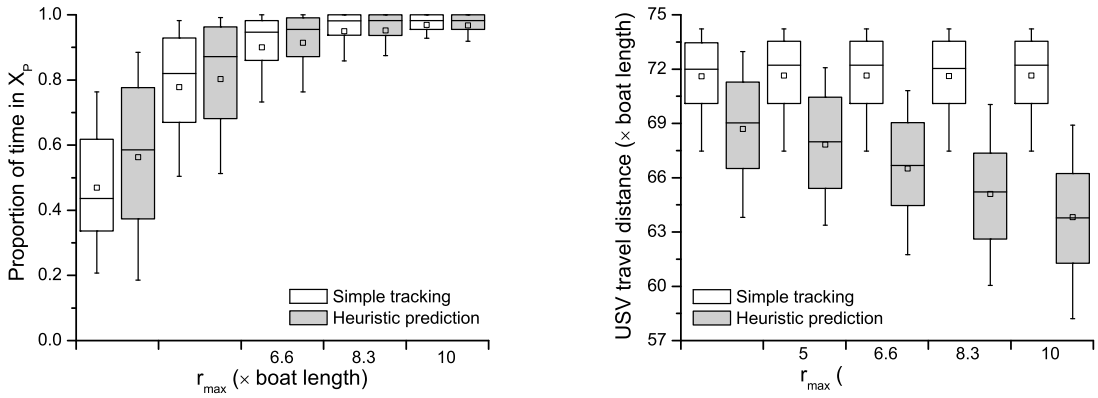


Figure 3.11: USV performance while using heuristic motion goal prediction for various obstacle densities and r_{max} values. (a) Proportion of time the USV spends in the proximity region. (b) Average distance traveled by the USV.



(a) (b)

Figure 3.12: Comparison of simple and heuristic motion goal prediction for a set of 200 randomly generated test cases with 96 obstacles.(a) Portion of time the USV spends in the proximity region. (b) Average distance traveled by the USV. For each figure, the box plot shows the median, upper and lower quartile. The whiskers show the upper and lower decile. The mean value is represented by a small square.

Throughout the experiments, the value of r_{min} was fixed to 1.6 boat lengths (i.e., 1 m), while r_{max} was varied between 3.3 (i.e., 2 m) and 10 boat lengths (i.e., 6 m). The lag time required by the heuristic motion goal predictor was set to $t_{lag} = (r_{min} + r_{max})/(2u_U)$, encouraging the USV to position itself half way between r_{min} and r_{max}. The number of time points used by the heuristic, k, was set to 5 and the time horizon, t_k, was set to 10 s, and the discount factor, δ, was fixed at 0.9. The trajectory planner used by the motion goal predictor was configured to terminate after 5000 iterations to reduce its running time. If no viable trajectory was found to any of the candidate motion goals, then the simple motion goal was used instead.

At the start of each test case, the USV was positioned within 8.3 boat lengths (i.e., 5 m) of the initial location of the target boat, and the USV was oriented in a direction facing the target. The target boat then followed its pre-defined trajectory for 2 minutes, while the simulator recorded the amount of time that the USV was within the proximity region, and the length of the trajectory followed by the USV.

Results for each of the test cases are presented in Figure 3.11, with a more detailed breakdown in Figure 3.12 comparing the heuristic motion goal prediction with simple tracking. As expected, increasing r_{max} reduces the total traveled distance of the USV using the heuristic motion goal prediction, while it has no effect on the USV using the simple tracking. When r_{max} is equal to 10, there is a nearly 10% reduction in trajectory length. This is achieved with no apparent decrease in the average amount of time spent in the proximity region, and in some cases (e.g., when $r_{max} = 3.3$) the amount of time spent in the proximity region noticeably increases when compared to the simple tracking. As the obstacle density increases, the average time spent in the proximity region decreases for both the motion goal computed using the developed

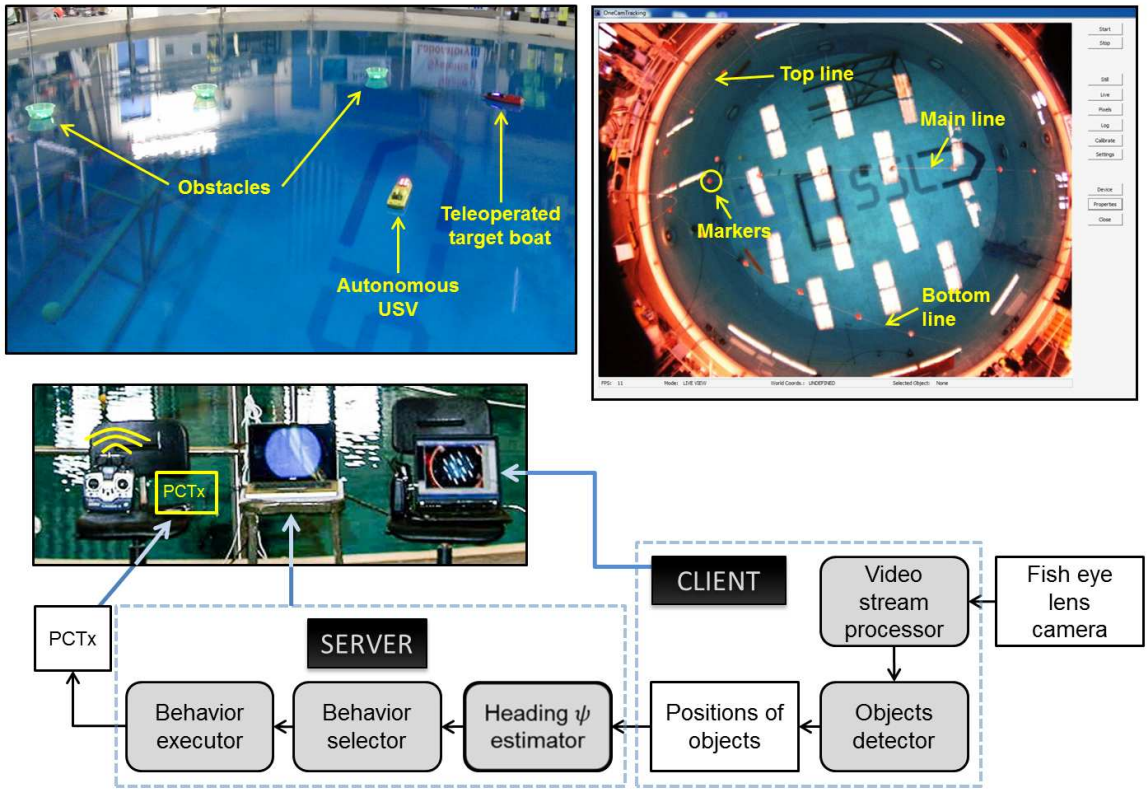


Figure 3.13: Developed physical setup for testing autonomous behaviors in the Neutral Buoyancy Research Facility (NBRF) at the University of Maryland. The developed planner for following the target boat is represented as a follow behavior executed by the behavior selection and execution components of the entire system.

heuristic and the motion goal determined as the current pose of the target boat, but the heuristic motion goal appears to retain its advantage. On the other hand, as the obstacle density increases, the average travelled distance by the USV increases or remains approximately to be the same for different values of r_{max} (see Figure 3.11b)).

Although the simulation was not performed in real-time, the heuristic motion goal predictor had an average running time of 1.78 s, with a standard deviation of 1.07 s, just slightly slower than the allotted time of 1 s to perform the computation. The computation in real-time would be achieved by utilizing the dimensionality reduction for the trajectory planner as described in Sections 3.6.1 and 3.6.1 and optimization of the planner software.

3.7.6 Experimental Setup

We have developed an experimental setup (see Figure 3.13) for physical evaluation of the planning approach for target boat following. The evaluation was conducted using two radio controlled boats in a 15 m wide tank within the Neutral Buoyancy Research Facility (NBRF) at the University of Maryland. The boats are powered

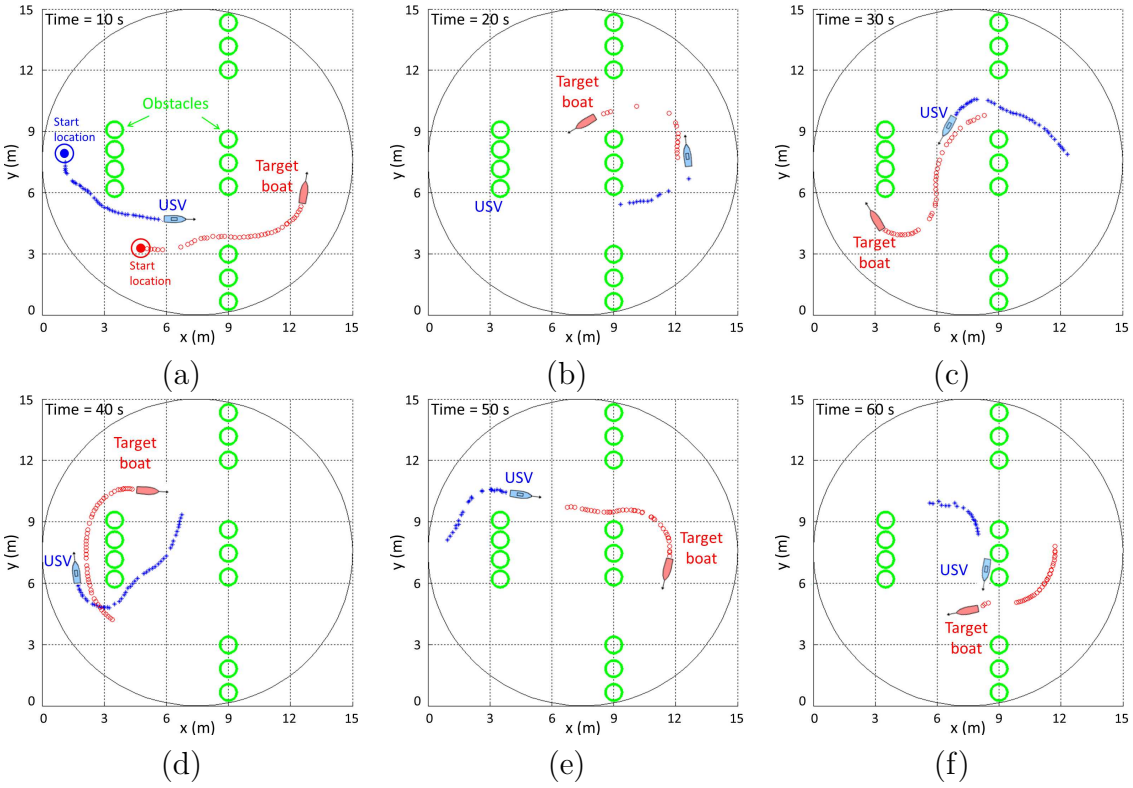


Figure 3.14: Experimental result of the developed planning approach for following a target boat by an autonomous unmanned surface vehicle (USV) in an environment with static obstacles. The case f) illustrates a situation in which the USV takes a shortcut while following the target boat.

with a single 9.6 volt battery and consist of one DC motor to control the forward and reverse speed of the propeller and a servo motor to control the angular motion of rudder. We use Hitec transmitter and receiver pair which operates on 2.4 GHz bandwidth. We utilize Endurance PCTx interface between the laptop and the RC transmitter to allow radio transmission of actuator commands to control the throttle and rudder positions of the autonomous USV. The USV is remotely controlled by the developed nominal trajectory planner and trajectory tracking controller running on a laptop. The setup also allows us to control the target boat using a remote controller interface.

The positions of the boats are perceived using a vision system that includes a single fish eye lens camera mounted 8 m above the level of the pool. We have developed a color blob detection image processing algorithm to detect the boats. The USV as well as the target boat use SMD LEDs with high intensity to be easily recognizable by the blob detection algorithm.

We have also developed a camera calibration module to relate the measured entire pool area to a local coordinate frame of the USV planning system running on a laptop. The calibration of the camera provides us a model of its geometry as well as distortion model of the lens. The calibration system uses three reference lines with the same color markers separated by a fixed distance of 1.5 m (see Figure 3.13). The input to the algorithm are the markers on each reference line starting from the origin of the coordinate frame and relative angles of the two of the lines (labeled as the top and bottom lines in Figure 3.13) with the main line. The algorithm then maps the entire physical pool to the planning workspace with the main line as the positive direction of the x axis. The algorithm is based on the calibration procedure developed by Sturm and Maybank [79] and the Matlab camera calibration toolbox [80]. The algorithm represents a simplified version of the referred algorithms since the boats move approximately in the same plane.

The sensor uncertainty is handled through the use of the extended Kalman filter [34]. Due to the difficulty of reliable perception of all obstacles in the pool, we let the user to mark their positions using the interface of the planning system. In this way, the user has the option to define a set of virtual obstacles that may reflect the physical obstacles in the scene.

For physical experiments, the trajectory planner utilizes the control action set for the USV as shown in Figure 3.2a). It consists of 5 control actions u_A, u_B, u_C (and their symmetric counterparts u'_B and u'_C) with different final orientations $u_{A,\theta} = 0.785\ rad$, $u_{B,\theta} = 0.4636\ rad$,$u_{C,\theta} = 0\ rad$, and positions $u_{A,f} = [0.8, 0.8]^T\ m$,$u_{B,f} = [0.8, 0.4]^T\ m$, $u_{C,f} = [0.8, 0]^T\ m$. The control actions were determined based on the dynamic characteristics of the boat as well as its dimension ($0.6 \times 0.15 \times 0.13\ m$) and the obstacle density of the experimental environment. We specified the final orientation as well as position for each control action and employed a PID controller to validate the dynamical feasibility of the control action in the simulator. In addition, we set the maximum surge speed for the USV to be constant of 0.8 ms^{-1} for straight segments of the trajectory and 0.3 ms^{-1} for turns.

3.7.7 Experimental Results of Overall Planning Approach

In the designed physical experiment, an autonomous USV was supposed to follow a human-teleoperated target boat within the tank. The USV had a complete knowledge of the operating scene. The developed planner was utilized to find the shortest possible, dynamically feasible trajectory among obstacles to approach the target with a given orientation. Due to the limited dimension of the scene, we did not evaluate the motion goal prediction algorithm in these experiments. The developed trajectory tracking controller allowed the USV to follow the trajectory without overshooting and taking unnecessary corrective maneuvers.

Figure 3.14 shows a sequence of planning situations that arose during the execution of the follow task. In all these situations, the planner computes the shortest possible trajectory by taking the current positions of both the USV and the teleoperated boat, and estimating their heading. In most of the situations, the USV takes the same trajectory as the teleoperated boat. However, as can be seen in the situation illustrated in Fig. 3.14f), the USV chooses a shorter, different trajectory to approach the teleoperated boat. The experiment also shows that the developed system is able to compute and reliably execute the nominal trajectory.

3.8 Summary

In this chapter, we presented an approach for target boat following using an autonomous unmanned surface vehicle. The developed approach consists of the motion goal prediction, trajectory planning, and trajectory tracking algorithms. The motion goal prediction algorithm is used for estimation of the future pose of the target boat and computation of the desired pose for the USV to efficiently follow the target. A fast trajectory planner was developed for computation of a trajectory that complies with differential constraints of the USV. A trajectory tracking controller was developed for computation of desired velocities along the trajectory.

We have demonstrated the capabilities of the planner using both simulation and physical experiments. We have shown the efficiency of the motion goal prediction component in terms of the time spent in a proximity region around the target boat and travelled distance savings while following the target boat in environments with varying obstacle densities. The follow task imposes strict planning time constraints. Hence, it was necessary to satisfy these constraints by searching through a hybrid, pose-position state space with a multi-resolution control action set. We have carried out a detailed study of the developed techniques to speed up the search by finding a balance between computational time requirements and trajectory length. For the evaluation of the trajectory planner and tracker on a physical boat, we have developed a physical setup that encompasses vision, image processing, control, and software components.

Chapter 4

Predictive Asset Defense by Team of Autonomous Surface Vehicles in Environment with Civilian Boats

Contributors: Eric Raboin, Petr Švec, Dana S. Nau, and Satyandra K. Gupta

4.1 Introduction

Teams of cooperative, highly-maneuverable unmanned surface vehicles (USVs) [25] can be utilized for guarding designated regions or assets (e.g. oil tanker, commercial cargo ship, etc.) against hostile boats in naval missions. The use of autonomous robotic systems brings several advantages which include reducing the risk of human fatalities and significantly decreasing the cost of missions, while preserving the expected level of security. This, however, imposes multiple challenging requirements on the decision-making capability of these vehicles.

Guarding an asset by a team of USVs requires cooperative patrolling of the surrounding area, approaching and observing passing boats, recognizing hostile boats, and delaying their progress towards the asset by active blocking (see Fig. 4.1). Intelligent, balanced decisions about which tasks to perform must be made by the vehicles to keep adversaries away from the asset for as long as possible and to prevent them from reaching the asset without being recognized. This presents a non-trivial challenge for the USVs since the identity of the hostile boats may not be known at the time they enter visibility range. In addition, the possibility of intermittent communication interruptions and the differential constraints of the vehicles impose additional complications. The vehicles also have to consider the time dependencies of the problem since selecting current tasks requires knowledge of what future tasks are possible in order to maximize the expected performance of the entire team. Finally, the task allocation must be done efficiently despite the very large state-action space since multiple tasks can be assigned to multiple vehicles simultaneously, and the developed approach should be general enough to be usable for a range of scenarios and missions.

This chapter presents a decentralized, contract-based planning approach for a

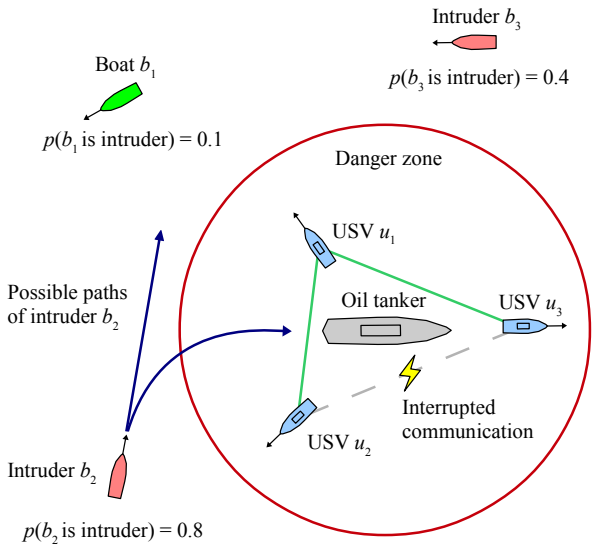


Figure 4.1: A team of unmanned surface vehicles (USVs) is guarding an oil tanker against hostile boats in a region with civilian traffic. During the operation, each boat is assigned a probability of being hostile based on observations made by the USVs.

team of USVs to guard a valuable asset against hostile boats in an environment with civilian traffic. The work is mostly concerned with high-level task allocation and behavior optimization to allow safe, high-performance, autonomous operation. The developed planner computes an approximate solution to an instance of the MT-MR-TA (i.e., multi-task robots, multi-robot tasks, time-extended allocation) variant of the task allocation problems in real-time [81].

According to the developed planning approach, the individual vehicles use two-side share and offer contracts to incrementally agree on the observe, guard, and delay tasks. The contracts allow the vehicles to establish a specific communication and task allocation protocol. Model-predictive simulations are leveraged for evaluation of candidate dynamic task allocations, i.e. by looking-ahead and estimating the utility of the allocation in order to optimize the assignment of future tasks based on the current state of the boats in the scene.

The tasks are realized by their corresponding parameterized behaviors that are optimized for a specific mission (e.g., defined by the number of available USVs, an estimated number of intruders, spatial distribution of the incoming boats in respect to the target, expected distribution of civilian traffic, etc.). The behaviors implement a local, reactive obstacle avoidance and interception strategy that respects the differential constraints of the vehicles. The weightings of the behaviors and simulation parametrization of the planner are optimized concurrently to account for their individual contributions to the overall guarding strategy.

The developed planner is capable of computing a task allocation plan in real-time and is scalable to a large number of vehicles. We demonstrate its performance in

simulation using a team of autonomous unmanned surface vehicles (USVs) guarding a valuable asset. We compare it to a baseline approach, which does not carry out additional information-gathering tasks, and a heuristic strategy that does not use predictive simulation to aid in the task assignment. We also evaluate how the performance of the planner is affected by varying the frequency of communication interruptions and the accuracy of the predictive simulation.

The outline of the chapter is as follows: in Section 4.2, we provide a formal definition of our multi-agent planning problem, followed by a detailed description of our approach in Section 4.3. In Section 4.4 we describe an experimental setup and discuss our results, then provide a brief discussion of future work in Section 4.5.

4.2 Problem Formulation

We define a multi-agent planning problem where a team of USVs must defend a stationary target from a team of hostile intruders. The USV team's objective is to delay the hostile boats' arrival at the target. More formally, given,

(i.) a continuous, non-empty state space $X \subseteq \mathbb{R}^3 \times \mathbb{S}$, where each state $\mathbf{x} = \{x, y, \psi, v\}$ defines the coordinates x and y, heading angle ψ, and surge speed v of a single boat or USV

(ii.) a control action space $Q(\mathbf{x}) \in \mathbb{R}^2$ for each $\mathbf{x} \in X$, where each control action $q = \{\Delta v, \Delta\psi\}$ defines a change in surge speed Δv and heading angle $\Delta\psi$

(iii.) a team of USVs $U = \{u_1, u_2, \dots u_m\}$ protecting a target positioned at location l_{target} where v_{max,u_i} is the maximum surge speed of USV u_i

(iv.) a set of passing boats $B = \{b_1, b_2, \dots b_n\}$ including intruder boats $I \subseteq B$ where v_{max,b_i} is the maximum surge speed of boat b_i

(v.) the state of the world $s = \{\mathbf{x}_{u_1}, \dots \mathbf{x}_{u_m}, \mathbf{x}_{b_1}, \dots \mathbf{x}_{b_n}\}$ defining the state $\mathbf{x}_{u_i}$ of every USV u_i and state $\mathbf{x}_{b_i}$ of every boat b_i,

(vi.) observation history $O_{u_i,t_k} = \{o_{t_1}, o_{t_2}, \dots o_{t_k}\}$ for u_i at time t_k, where $o_{t_j} = \langle s_{t_j}, F_{t_j} \rangle$ is the state of world s_{t_j} at time t_j and $F_{t_j} = \{f_{b_1}, \dots f_{b_n}\}$ is a set of observed features f_{b_i} (e.g. size, color) for each boat b_i,

(vii.) joint observation history $\mathcal{O}_{u_i} = \{O_{u_1,t_j}, \dots O_{u_m,t_k}\}$ representing the combined observation histories of the USV team based on the most recent information available to USV u_i

(viii.) a communication reliability function $C_{t_k}(u_i, u_j)$ which returns *true* if USV u_i is able to communicate with USV u_j at time t_k, and *false* otherwise

(ix.) a set of *observe* tasks, H_o, where USVs are responsible for gathering information about the passing boats; a set of *guard* tasks, H_g, where USVs must position themselves in vulnerable areas around the target; and a set of *delay* tasks, H_d, where USVs must intercept and then block a hostile USV,

(x.) an observation classification function $P(b_i \in I|\mathcal{O}_{u_j})$ that returns the probability that boat b_i is an intruder given observation history $\mathcal{O}_{u_j}$

(xi.) a non-deterministic opponent model $\pi_{b_i}(\mathcal{O}_{u_j})$ that returns a velocity vector $\mathbf{v}$ for boat b_i, defining the behavior of boat b_i given observation history $\mathcal{O}_{u_j}$

(xii.) a response team probability threshold p_{alert}, indicating at what probability $P(b_i \in I|\mathcal{O}_{u_j})$ an alert should be triggered for boat b_i

Compute,

(i.) a joint task allocation $\mathcal{A}_{u_i} = \{H_{u_1}, H_{u_2}, \dots H_{u_m}\}$ for the USV team from USV u_i's perspective, where $H_{u_j} \subseteq H_o \cup H_g \cup H_d$ is the set of tasks assigned to USV u_j. Each guard or observe task may only be assigned to one USV at a time, while delay tasks may be assigned to multiple USVs.

(ii.) a policy $\pi_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$ that returns a control action $q \in Q$ for USV u_i given the current observation history $\mathcal{O}_{u_i}$ and task allocation $\mathcal{A}_{u_i}$.

The USV team does not know *a priori* which boats are hostile, but can determine whether a boat is hostile through observation. The features $f_{b_i} \in F_{t_j}$ in the USV team's observation history are used by the classification function $P(b_i \in I|\mathcal{O}_{u_i})$ to determine the probability that boat b_i is an intruder. We assume that this function is given, and that its exact nature will vary depending on the scenario.

If at any time the probability $P(b_i \in I|\mathcal{O}_{u_i})$ exceeds p_{alert} for any boat b_i, then an alert is triggered. The *delay time*, t_{delay}, is the difference between the time t_{alert} that an alert was triggered, and the time $t_{arrival}$ when an intruder arrives at the target. The objective of each USV is to find a task allocation $\mathcal{A}^*_{u_i}$ and policy $\pi^*_{u_i}$ that maximize the expected t_{delay} for the first boat b_j that reaches the target,

$$\langle \mathcal{A}^*_{u_i}, \pi^*_{u_i} \rangle = \arg \max_{\mathcal{A}_{u_i}, \pi_{u_i}} E[t_{delay}|\pi_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})]. \tag{4.1}$$

4.2.1 Communication

Communication between USVs is modeled as a network of pairwise connections, shown in Fig. 4.2. For a given time step, the communication link between any two USVs u_i, u_j may be interrupted, meaning they are not able to directly exchange information about task assignments or observations. As a result, the joint observation history $\mathcal{O}_{u_i}$ and task allocation $\mathcal{A}_{u_i}$ may differ for each USV depending on the severity of the interruptions.

At each time step t_k, an attempt is made to synchronize the information held by each USV. The most recent observation history O_{u_j,t_k} and task assignment H_{u_j} of USV u_j are added to USV u_j's joint observation history $\mathcal{O}_{u_i}$ and task allocation $\mathcal{A}_{u_i}$ if communication between USVs is possible, as determined by $C_{t_k}(u_i, u_j)$. If USV u_i cannot directly communicate with USV u_j, it may still learn of u_j's observations and task assignment through a third USV u_k that can communicate with both agents, but it will take two time steps for this information to propagate. Additional considerations when trying to coordinate task exchanges between USVs are discussed at the end of Section 4.3.3.

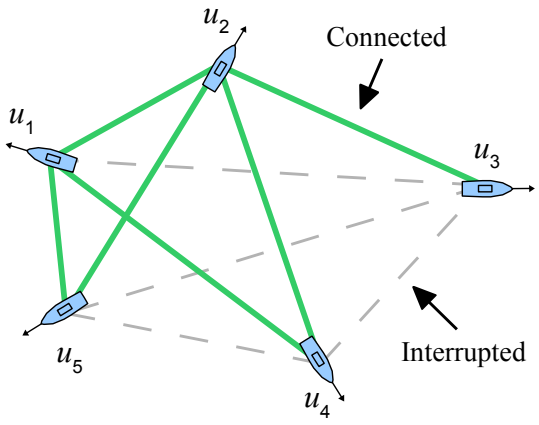


Figure 4.2: An example of a communication network.

4.3 Approach

The joint task allocation, $\mathcal{A}_{u_i}$, is computed online and updated during each planning time step. Several candidate allocations are evaluated, each resulting from incremental changes to the current task allocation where at most two USVs must coordinate by sharing or offering tasks. Our algorithm uses both heuristics and model-predictive simulation to determine which candidate task allocation should be selected.

The policy for each USV, $\pi_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$, accepts the task allocation and observation history as input and generates an appropriate control action $q \in Q$. The policy is implemented using parameterized behaviors that have been tuned offline by a genetic algorithm. These low-level behaviors are defined below, followed by a description of our algorithm for high-level task allocation, a description of the model-predictive simulation process, and an overview of how we utilize the genetic algorithm for behavior optimization.

4.3.1 Behaviors

The policy $\pi_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$ is computed by finding a motion goal that satisfies the lower-level policies for the guard, observe, and delay tasks in u_i's task assignment $H_{u_i} \in \mathcal{A}_{u_i}$, then finding an action $q \in Q$ to steer the USV towards that goal while performing local obstacle avoidance.

The motion goal for each individual task $h_j \in H_{u_i}$ is given by $M_{h_j}(\mathcal{A}_{u_i})$ as defined below

$$M_{h_j}(\mathcal{A}_{u_i}) = \begin{cases} \text{a boat's location, } l_{b_j}, & \text{if } h_j \in H_o, \\ \text{a guard location, } l_{g_j}, & \text{if } h_j \in H_g, \\ l_{int}(u_i, b_j, \mathcal{A}_{u_i}), & \text{if } h_j \in H_d, \end{cases} \tag{4.2}$$

where $l_{int}(u_i, b_j, \mathcal{A}_{u_i})$ is a heuristically calculated intercept point for USV u_i to block boat b_j, given the set of USVs that are assigned to delay b_j in $\mathcal{A}_{u_i}$.

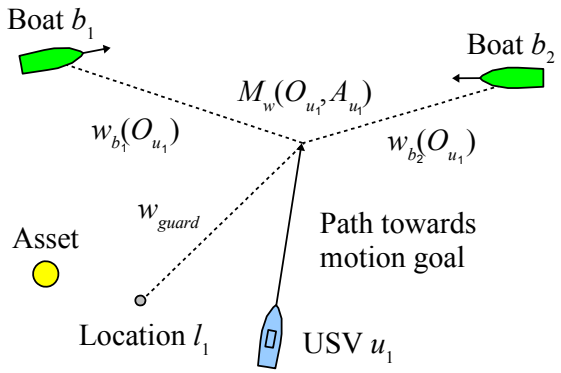


Figure 4.3: USV u_1 approaches the weighted motion goal $M_w(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$ corresponding to two observe tasks for boats b_1 and b_2 and a guard task for location l_1.

The actual motion goal for USV u_i is defined as

$$M_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i}) = \begin{cases} M_{h_j}(\mathcal{A}_{u_i}), & \text{if } \exists h_j \in H_{u_i} \cap H_d, \\ M_w(\mathcal{O}_{u_i}, \mathcal{A}_{u_i}), & \text{otherwise,} \end{cases} \tag{4.3}$$

which returns $M_{h_j}(\mathcal{A}_{u_i}) = l_{int}(u_i, b_j, \mathcal{A}_{u_i})$ if H_{u_i} contains some delay task h_j. Otherwise, it returns a weighted motion goal (see Fig. 4.3) based on the USV u_i's currently assigned guard and observe tasks,

$$M_w(\mathcal{O}_{u_i}, \mathcal{A}_{u_i}) = \frac{\sum_{h_j \in H_{u_i}} w_{h_j}(\mathcal{O}_{u_i}) M_{h_j}(\mathcal{A}_{u_i})}{\sum_{h_j \in H_{u_i}} w_{h_j}(\mathcal{O}_{u_i})}, \tag{4.4}$$

where $w_{h_j}(\mathcal{O}_{u_i})$ is the weight of task h_j, equal to w_{guard} if h_j is a guard task, and equal to $w_{b_j}(\mathcal{O}_{u_i})$ if h_j is an observe task for boat b_j,

$$w_{b_j}(\mathcal{O}_{u_i}) = w_{intr} P(b_j \in I | \mathcal{O}_{u_i})(1 + \frac{w_{dist}}{|l_{target} - l_{b_j}|}). \tag{4.5}$$

The parameters w_{guard}, w_{intr}, and w_{dist} are tuned for each mission using the method described in Sec. 4.3.5.

The calculation of $l_{int}(u_i, b_j, \mathcal{A}_{u_i})$ is based on a simplification of the movement of USVs and intruders. Without considering differential constraints, we calculate a straight line path from the intruder to the guarded asset, and then calculate the optimal intercept point for each USV to the intruder. An example of this heuristic calculation is shown in Fig. 4.4.

During the calculation of l_{int}, the speed of the intruder is estimated to be $v_{b_j}(k) = v_{max,b_j} * (b_{block})^k$ after being intercepted by k USVs, adjusting the optimal intercept point for all subsequent USVs.

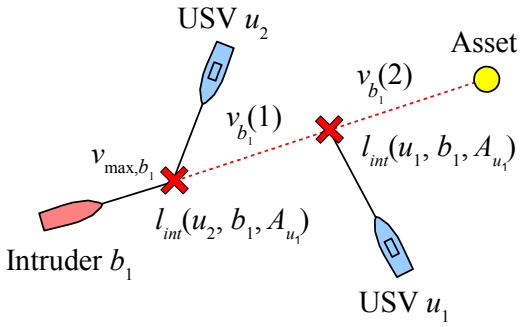


Figure 4.4: Heuristic model of USVs u_1 and u_2 intercepting intruder b_1 in a simplified version of the problem. The intercept points serve as a motion goal for the delay behavior of the USVs in the actual game.

4.3.2 Control Action Selection

Given motion goal $M_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$, an appropriate control action $q \in Q(\mathbf{x}_{u_i})$ must be selected to direct the USV towards its goal while avoiding collisions with other boats or static obstacles. The set of feasible control actions $Q(\mathbf{x}) = A(\mathbf{x}) \times \Theta(\mathbf{x})$ is subject to physical constraints, where

$$A(\mathbf{x}) = \{\Delta v : a_{min}(\mathbf{x}) \leq \Delta v \leq a_{max}(\mathbf{x})\} \tag{4.6}$$

$$\Theta(\mathbf{x}) = \{\Delta \psi : \theta_{min}(\mathbf{x}) \leq \Delta \psi \leq \theta_{max}(\mathbf{x})\} \tag{4.7}$$

define the minimum and maximum change in surge speed Δv and turning angle $\Delta \psi$ given state $\mathbf{x}$. We assume that the maximum turning radius decreases as the surge speed v increases. We also assume that boats cannot travel in reverse, so $a_{min}(\mathbf{x})$ is zero when the surge speed v is zero.

We use a simple steering model for the motions of boats and USVs, where control action $q = \{\Delta v, \Delta \psi\}$ determines the change in surge speed v and orientation ψ, while $\Delta x = v\cos(\psi)$ and $\Delta y = v\sin(\psi)$ determine the change in coordinates (x, y). To estimate the movement of the USV after time step Δt, we use the formula

$$\mathbf{x}' = \mathbf{x} + \Delta \mathbf{x}(q)\Delta t \tag{4.8}$$

where $\Delta \mathbf{x}(q) = \{\Delta x, \Delta y, \Delta \psi, \Delta v\}$ represents the change in $\mathbf{x}$ given control action q.

Given state $\mathbf{x}_{u_i}$ and motion goal M_{u_i}, let ψ_{u_i} be u_i's current heading angle and let $\phi_{M_{u_i}}$ be the desired heading angle in the direction of M_{u_i}. The steering angle to achieve this new heading is determined by,

$$\Delta \psi_{M_{u_i}}(\mathbf{x}_{u_i}) = \arg \min_{\Delta \psi \in \Theta(\mathbf{x}_{u_i})} |d(\phi_{M_{u_i}}, \psi_{u_i} + \Delta \psi)|. \tag{4.9}$$

where $d(\phi_j, \phi_k)$ is the minimum difference between any two angles ϕ_j and ϕ_k. Similarly, the change in surge speed is determined by,

$$\Delta v_{M_{u_i}}(\mathbf{x}_{u_i}) = \arg \min_{\Delta v \in A(\mathbf{x}_{u_i})} |\gamma_{M_{u_i}} - (v_{u_i} + \Delta v)| \tag{4.10}$$

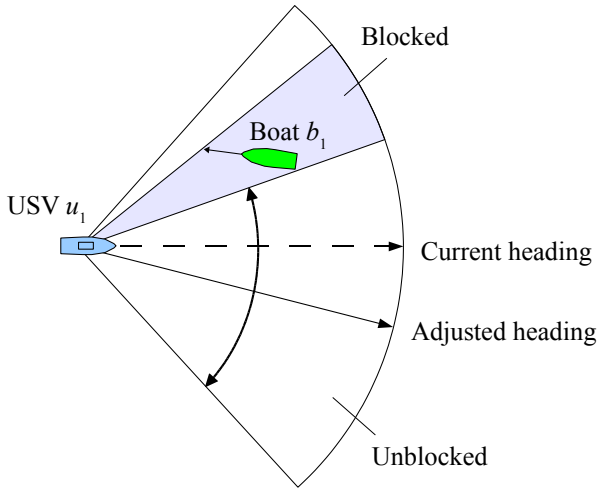


Figure 4.5: USV u_1 adjusts its heading to steer away from the region blocked by boat b_1 based on the depicted obstacle avoidance fan.

where $\gamma_{M_{u_i}}$ is the desired surge speed of USV u_i as it approaches M_{u_i}. The resulting control action is simply,

$$q_{M_{u_i}}(\mathbf{x}_{u_i}) = \{\Delta v_{M_{u_i}}(\mathbf{x}_{u_i}), \Delta\psi_{M_{u_i}}(\mathbf{x}_{u_i})\} \tag{4.11}$$

however this control action may lead to a collision with obstacles such as other boats or rocks. To reduce the chance of collision, the desired heading $\phi_{M_{u_i}}$ and velocity $v_{M_{u_i}}$ are adjusted using reactive obstacle avoidance.

As depicted in Fig. 4.5, each USV has an obstacle avoidance fan with radius r_{avoid} and angular span s_{avoid} to identify which obstacles pose a risk of collision. Headings within the obstacle avoidance fan are considered blocked if they are occupied by an obstacle or will become occupied by an obstacle within some time t_{lead} based on the obstacles' current velocities. Obstacles are assumed to have a non-zero radius.

Let $Z = \{z_1, z_2, \ldots, z_n\}$ be a set of unblocked sectors inside the obstacle avoidance fan, where each $z_j = [\phi_{j,a}, \phi_{j,b}]$ is a range of angles that are not blocked and where the ordering constraint $\phi_{j,b} \leq \phi_{j+1,a}$ holds for all $j < n$. Let $\phi_{j,mid}$ be a midpoint between $\phi_{j,a}$ and $\phi_{j,b}$. Heading ϕ is considered safe if it is within the obstacle avoidance fan, and $\phi \in [\phi_{1,a}, \phi_{1,mid}]$ or $\phi \in [\phi_{n,mid}, \phi_{n,b}]$, which is trivially true if $n = 1$.

If $\phi^*_{M_{u_i}}$ is the most direct heading to the motion goal, the adjusted heading after reactive obstacle avoidance is,

$$\phi_{M_{u_i}} = \begin{cases} \phi^*_{M_{u_i}}, & \text{if } \phi^*_{M_{u_i}} \text{ is safe,} \\ \phi_{k,mid} \text{ s.t. } z_k = z_{max}, & \text{otherwise,} \end{cases} \tag{4.12}$$

where z_{max} is the widest unblocked sector,

$$z_{max} = \arg\max_{z_j \in Z} |d(\phi_{j,b}, \phi_{j,a})| \tag{4.13}$$

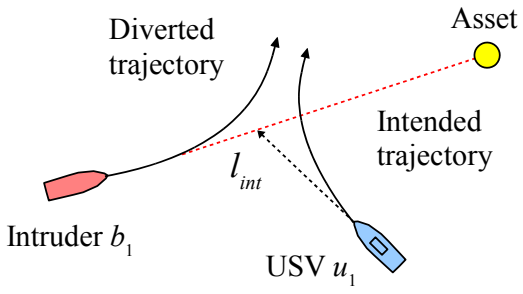


Figure 4.6: USV u_1 intercepts intruder b_1 diverting it from its intended path to the asset.

The surge speed of the USV is not affected by obstacle avoidance unless $Z = \emptyset$, at which point the USV will slow to a stop. Thus, the desired surge speed is,

$$\gamma_{M_{u_i}} = \begin{cases} 0, & \text{if } Z = \emptyset, \\ v_{max,u_i} \frac{|M_{u_i} - l_{u_i}|}{r_{slow}}, & \text{if } |M_{u_i} - l_{u_i}| < r_{slow}, \\ v_{max,u_i}, & \text{otherwise,} \end{cases} \tag{4.14}$$

where l_{u_i} is u_i's current location, and r_{slow} determines at what distance the USV should start to slow down.

To avoid needlessly complex trajectories, we define non-zero radius r_{goal} such that the USV is considered at its destination if it is within r_{goal} of its motion goal M_{u_i}. The resulting policy for USV u_i is simply,

$$\pi_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i}) = \begin{cases} q_{M_{u_i}}(\mathbf{x}_{u_i}), & \text{if } |M_{u_i} - l_{u_i}| > r_{goal}, \\ q_{stop}(\mathbf{x}_{u_i}), & \text{otherwise,} \end{cases} \tag{4.15}$$

where $q_{stop}(\mathbf{x}_{u_i}) = \{a_{min}(\mathbf{x}_{u_i}), 0\}$ is a control action that quickly halts the movement of the USV.

The control action selection for passing boats is identical to USVs, only a different motion goal M_{b_j} and different set of parameters r_{avoid}, s_{avoid} and r_{slow} are selected. For our experiments detailed in Section 4.4, the parameters for the passing boats including intruders are predefined, while the parameters for the USVs are learned using a genetic algorithm, described in Section 4.3.5.

If one or more USVs move within the obstacle avoidance fan of an another boat, the boat will be forced to adjust its trajectory to avoid a collision. This is illustrated in Fig. 4.6, where a USV intercepts an intruder, diverting its trajectory away from the target.

4.3.3 Task Allocation

At the start of each scenario, an initial allocation $\mathcal{A}_{u_i,0}$ assigns a set of guard locations to each USV, distributed uniformly at radius r_{guard} around the target. As new boats

enter the scene, observation tasks for each boat are assigned to the nearest USV. At regular time intervals, a reallocation step occurs, in which each USV u_i computes a revised allocation $\mathcal{A}'_{u_i}$, defined as

$$\mathcal{A}'_{u_i} = \arg \max_{\mathcal{A}_{u_i,j} \in C} eval(\mathcal{O}_{u_i}, \mathcal{A}_{u_i,j}) \tag{4.16}$$

where C is a set of candidate task allocations determined by Alg. 3, and $eval(\mathcal{O}_{u_i}, \mathcal{A}_{u_i,j})$ computes an estimated *delay time* for a candidate $\mathcal{A}_{u_i,j}$ using the model-predictive simulation as described in Sec. 4.3.4. Each candidate $\mathcal{A}_{u_i,j} \in C$ differs from $\mathcal{A}_{u_i}$ by re-assigning a single task from H_{u_i} to another USV, sharing a task from H_{u_i} with another USV, or both. These represent incremental changes, useful for gradually improving the joint task allocation without the need to evaluate all possible allocations.

Algorithm 3 GENERATECANDIDATES($\mathcal{O}_{u_i}, \mathcal{A}_{u_i}, u_i$): Generate a set of candidate task allocations.

1: $C \leftarrow \{\mathcal{A}_{u_i}\}$ **if** $\exists h_j \in H_{u_j} \cap H_d$ **then**
2: $C \leftarrow C \cup$ SHARETASKS($\mathcal{O}_{u_i}, \mathcal{A}_{u_i}, u_i$) **for** each $\mathcal{A}_{u_i,j} \in C$ **do**
3: $C \leftarrow C \cup$ OFFERTASKS($\mathcal{O}_{u_i}, \mathcal{A}_{u_i,j}, u_i$)
4: **return** C

The function OFFERTASKS($\mathcal{O}_{u_i}, \mathcal{A}_{u_i}, u_i$) returns a set of task allocations $C_O = \{\mathcal{A}_{u_i,0}, \mathcal{A}_{u_i,1}, \ldots \mathcal{A}_{u_i,m}\}$ where each $\mathcal{A}_{u_i,j} \in C_O$ is the same as the input allocation $\mathcal{A}_{u_i}$, except that a task h_f has been removed from H_{u_i} and given to another USV $u_j \in U$ instead. The task $h_f \in H_{u_i}$ is the task whose individual motion goal $M_{h_f}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$ is furthest from u_i's current motion goal. Intuitively, this means we are dropping a task that the USV u_i is least able to fulfill, and creating candidate allocations for the other USVs who may be better suited for that task.

The definition of SHARETASKS($\mathcal{O}_{u_i}, \mathcal{A}_{u_i}, u_i$) is very similar to OFFERTASKS, only the original task h_f is never dropped and delay tasks are the only tasks considered. If a delay task already has d_{max} USVs assigned to it, the task will not be shared. If at any point $P(b_i \in I|\mathcal{O}_{u_i})$ exceeds the alert threshold p_{alert}, the observe task for the boat b_i will be automatically converted into a delay task for b_i.

For USV u_i to either offer or share a task with another USV u_j, the two USVs must explicitly communicate. If $C_{t_k}(u_i, u_j)$ is false, indicating no communication is possible between u_i and u_j, then exchanges between these agents cannot be performed until communication is re-established. Since communication interruptions are dynamic, we evaluate all candidate exchanges even if they are currently infeasible, then determine whether they are still infeasible once the evaluation is finished. If the best performing candidate allocation is still infeasible due to interrupted communication, then the next best candidate is selected, unless there are no feasible candidates to choose from.

4.3.4 Predictive Simulation

Each USV u_i generates a set C of candidate task allocations using the method in Alg. 3. To evaluate each $\mathcal{A}_{u_i,j} \in C$, we generate a set of k possible worlds $W = \{w_i\}_{i=1}^{k}$

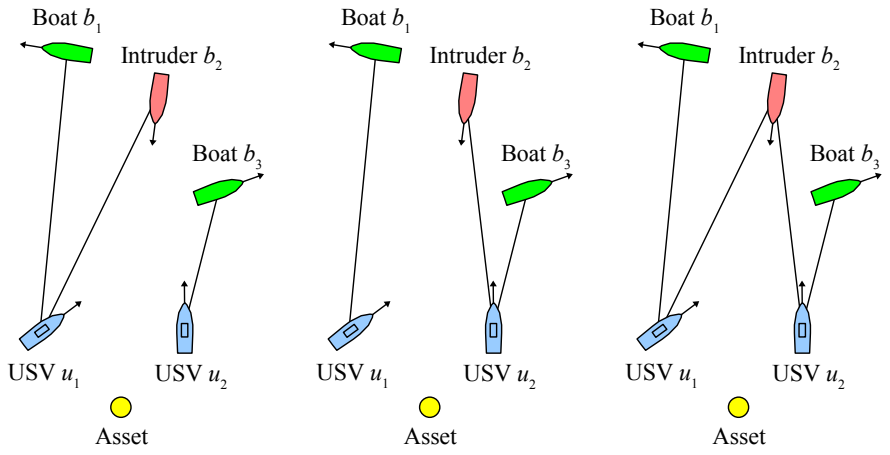


Figure 4.7: Candidate task allocations a) the current task allocation $\mathcal{A}$ without modification, b) modification of $\mathcal{A}$ with a delay task offered to USV u_2 by USV u_1, c) modification of $\mathcal{A}$ with a delay task shared to USV u_2.

consistent with the current u_i's observation history $\mathcal{O}_{u_i}$. Each possible world w_i is generated by selecting a set of possible intruder boats $I_i \subseteq B$ based on the observations made by the USV team, i.e. different scenarios consisting of the encountered boats that are assumed to be intruders and boats that are not intruders. The number of possible sets I_i is determined by the minimum x_{min} and maximum x_{max} number of intruders specified for the problem domain, where $k = \sum_{i=x_{min}}^{x_{max}} \binom{n}{i}$ given n total boats in the scene. If some boats have already been positively identified as intruders, this is reflected in the set of possible worlds considered. For each set of possible intruders I_i we estimate the joint probability,

$$P_{I_i} = \left(\prod_{b_j \in I_i} P(b_j \in I | \mathcal{O}_{u_i}) \right) \left(\prod_{b_j \in B \setminus I_i} 1 - P(b_j \in I | \mathcal{O}_{u_i}) \right)$$

where P_{I_i} is an approximation of $P(I_i = I | \mathcal{O}_{u_i})$ where I is the true set of intruders that may be unknown to the USV u_i. This approximation is computed by assuming that the appearance of each intruder is statistically independent from the appearance of other intruders.

For performance reasons, we only simulate the m most likely possible worlds and compute a weighted average to produce a heuristic estimate of the expected utility, $E[\mathcal{U}(\Pi_u)]$ where $\Pi_u = \{\pi_{u_i}(\mathcal{O}_{u_i}, \mathcal{A}_{u_i}) : u_i \in U\}$ is the joint policy of the entire USV team. As before, utility is defined as $t_{delay} = t_{arrival} - t_{alert}$. During the simulation, the policies π_{u_i} and π_{b_j} of USVs and boats are integrated to produce successor states.

During the predictive simulation, the task re-allocation step (see Sec. 4.3.3) is performed at less than the normal frequency, and a fast heuristic evaluation method is used to select the best USV, described below. This is to prevent the predictive

simulation from recursively calling itself, which would otherwise lead to an exponential blow up as the simulation searches deeper. Each trial is also given a maximum lookahead time, after which the *delay time* is estimated based on the current state.

When determining which USV should receive a guard or observe tasks, the heuristic selects the USV u_j that has the least distance between its current motion goal and the new task. If USV u_j is already assigned a delay task, the distance between its current motion goal and the motion goal corresponding to the new task is multiplied by $w_{occupied}$. When assigning delay tasks, the USV u_i is selected that maximizes the estimated delay time, given the trajectory provided by the heuristically calculated intercept point $l_{int}(u_i, b_j, \mathcal{O}_{u_i}, \mathcal{A}_{u_i})$.

In the worst case, if $x_{min} = 0$ and $x_{max} = n$, the number of possible worlds k evaluated by the predictive simulator is bounded by $O(2^n)$. For small problems it may be acceptable to evaluate all possible worlds, but for larger problems it is computationally infeasible, which is why we only select the m most likely possible worlds. Alternatively, one might sample from the possible worlds to form a statistical estimate, but we do not explore that approach in this chapter.

4.3.5 Optimization of Behaviors

We used a genetic algorithm (GA) [82] to optimize the 11 underlying parameters w_{guard}, w_{intr}, w_{dist}, b_{block}, d_{max}, r_{guard}, $w_{occupied}$, r_{slow}, r_{goal}, r_{avoid} and s_{avoid}, of the behavior and control action selection policies to further improve the expected utility of the USV policy. The optimization of these parameters allows the USVs to make balanced decisions between guarding a certain location, observing incoming boats, and intercepting and thus delaying the movement of identified intruders.

The genetic algorithm was run for 150 generations using a population size of 100 chromosomes, where each chromosome represented a complete set of parameters. The parameters for the initial population were assigned at random, while subsequent populations were bred based on the fitness values of the previous generation. Each chromosome's fitness was measured using the median *delay time* from 250 random simulation runs. We utilized roulette wheel selection to determine the breeding population, and applied genetic operators with a crossover rate of 0.35 and mutation rate of 0.08. Each chromosome in the subsequent population might have some or all of its parameters modified by these operators.

4.4 Results

4.4.1 Experimental Setup

We have evaluated our planning approach using two distinct scenarios. In scenario 1, shown in Fig. 4.8, the target is positioned within a circular region without any static obstacles. In scenario 2, shown in Fig. 4.9, the target is positioned above static terrain restricting the direction of incoming boats. In scenario 1 there are a total of 5 USVs and 3 intruders, while in scenario 2 there are 3 USVs and 2 intruders. Not

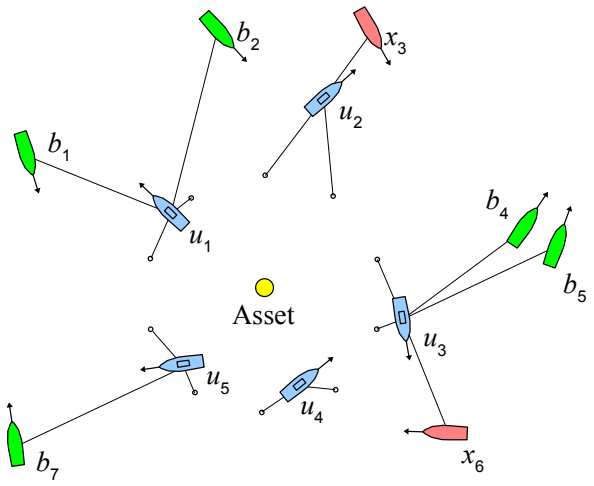


Figure 4.8: Scenario 1, five USVs defend a target in an open ocean with several passing boats. Boat x_{19}, identified as an intruder, is pursued by USVs u_2 and u_4. The task assignments for each USV are shown as connecting lines.

counting USVs, both scenarios maintain a total of 8 boats in the scene at any given time.

At the beginning of each trial, the positions of passing boats are initialized at random locations around the target. During the run, new boats appear at random locations along the boundary of the operating space, which is defined as a ring in the scenario 1, (with an inner and outer radius of 80 and 100 m), or as two rectangles on the left and right sides of the target in the scenario 2 (with a distance of 80 m from the target and a width of 20 m). The number of boats in the scene is held at a fixed amount, so new boats will appear as other boats leave the operating space.

Each boat's initial trajectory is a path tangent to a randomly sized circle (or semi-circle in scenario 2) surrounding the target with minimum and maximum radius of 30 and 60 m. The intruders turn towards the target when they pass within 60 m, but if an intruder passes within 5 m of a USV, it will start approaching the target immediately. The maximum speed is 10 m/s for both USVs and all other boats. Whether a boat will be an intruder or not is determined by the elapsed time of the simulation. Only non-intruder boats will appear during the first 30 seconds of the simulation, immediately followed by 2 or 3 intruders depending on the scenario. Thus, a group of intruders will always appear at or around the same time in the simulation.

Both intruders and non-intruder boats use the same reactive obstacle avoidance strategy described in Section 4.3.2. For intruders, the parameters r_{avoid} and s_{avoid} are set to 10 m and 120° respectively, while for non-intruders they are set to 15 m and 180°. The intruder is given a more aggressive set of parameters allowing it to approach other boats more closely before adjusting its trajectory. For USVs, these parameters are optimized using the genetic algorithm described in Section 4.3.5.

To make the interactions between USVs and intruders more varied, an intruder will periodically reverse direction if it senses it is being diverted away from the target for more than 2 seconds. This prevents the USV and intruder from becoming locked in

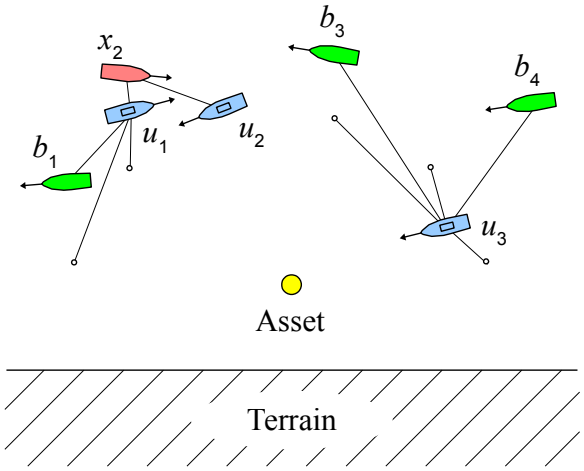


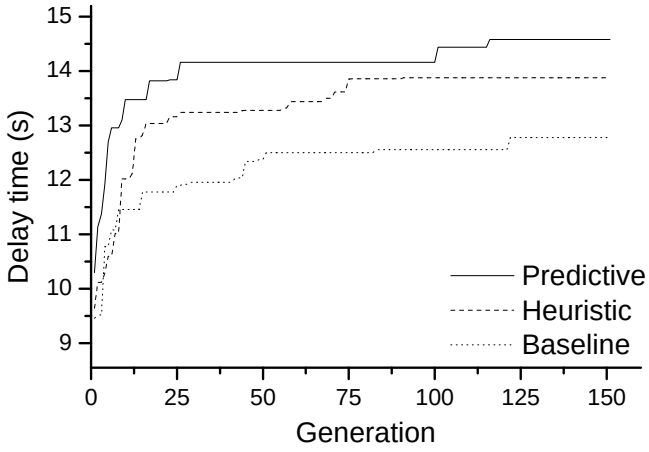
Figure 4.9: Scenario 2, three USVs defend a target that is protected by terrain to the south. USVs u_2 and u_3 are actively blocking intruder x_{13}, reducing the speed at which it approaches the target.

a predictable trajectory and makes for more realistic looking blocking behavior. This behavior, combined with aggressive parameters for obstacle avoidance, ensure that the intruder will eventually reach the target. However, this model is not guaranteed to be a best-response to USV team's strategy, and therefore cannot be used to determine the worst-case performance against any theoretical opponent.

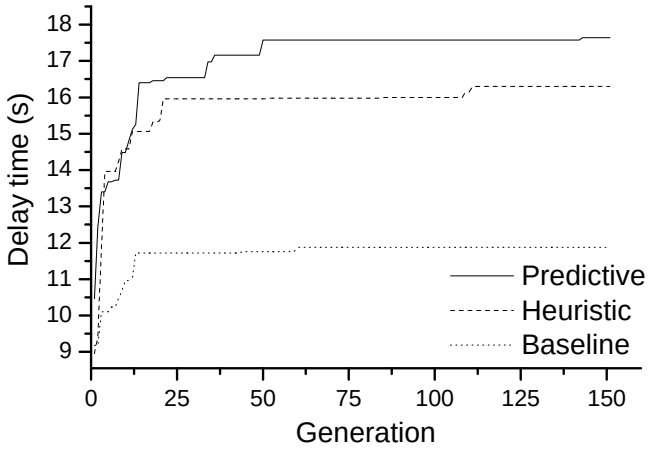
The observation classification function provides a simulated probability that each boat is an intruder based on the quality of the observations made by the USV team. The observation quality $\alpha_{b_j} \in [0, 1]$ for boat b_j is initially set to 0 and increases monotonically while USV u_i is within 50 m of boat b_j. If d represents the distance between u_i and b_j, then α_{b_j} increases at a rate of $0.5(1 - d/50)$ per second. This means it takes at most 5 seconds to obtain an observation quality of $\alpha_{b_j} = 1$ when observing boat b_j from a distance of 30 meters.

The function $P(b_j \in I|\mathcal{O}_{u_i})$ returns a prior probability of 0.05 when $\alpha_{b_j} = 0$, indicating that no observations have occurred. The choice of 0.05 is arbitrary, but is meant to represent a small non-zero chance that each boat could be an intruder. As α_{b_j} increases, the probability $P(b_j \in I|\mathcal{O}_{u_i})$ converges linearly to 1 or 0 depending on whether or not b_i is actually an intruder. Gaussian noise with standard deviation $0.1(1-\alpha_{b_j})$ is added to the probability function so that the change is non-monotonic. For all simulations, the value p_{alert} for determining whether a boat should be classified as a threat was set to 0.6.

During the predictive simulation described in Section 4.3.4 the task allocation step is performed at 1/5th the normal frequency (every 5 s instead of every 1 s), the lookahead time is set to 10 s, and only the most likely possible world is simulated ($m = 1$). These parameters are useful for ensuring the predictive simulation can be executed in real time.

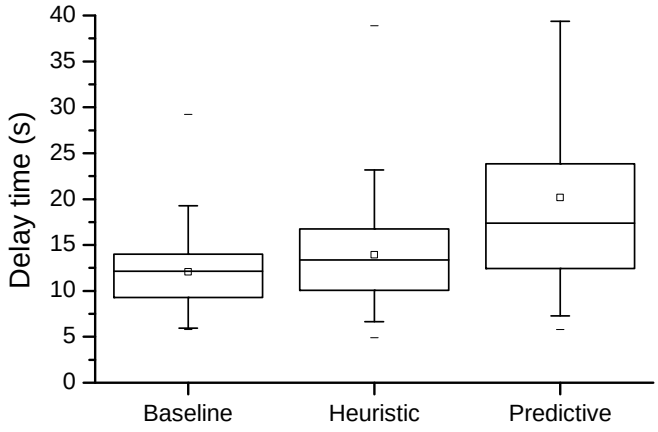


(a) Scenario 1

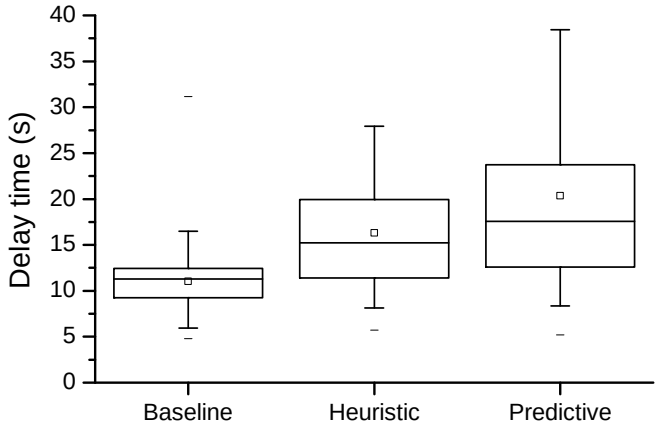


(b) Scenario 2

Figure 4.10: Median *delay time* for each generation of the genetic algorithm when optimizing strategies for scenarios 1 and 2. Results shown for the best performing chromosome in the population.



(a) Scenario 1



(b) Scenario 2

Figure 4.11: Average *delay time* across 1000 randomly seeded trials for USV teams using *predictive*, *heuristic* or *baseline* strategies. For both scenarios, the predictive strategy was better at delaying the intruders.

4.4.2 Result of Policy Evaluation

The genetic algorithm described in Sec. 4.3.5 was performed six separate times to generate optimized parameters sets for three strategies in each of the two scenarios. The change in performance across 150 generations is shown in Fig. 4.10a and Fig. 4.10b for the scenario 1 and 2, respectively.

The *predictive* strategy is the complete strategy described in Sec. 4.3. The *heuristic* strategy does not perform any predictive simulation and makes choices based only on the heuristic evaluation method described in Sec. 4.3.4. The *baseline* strategy does not assign observe tasks at all; each USV waits at its default guard location until an intruder is identified, at which point a delay task is assigned to the closest d_{max} USVs. Each strategy was optimized independently using the genetic algorithm.

Fig. 4.11 a) and b) show the average *delay time* across 1000 different trials for each of three different strategies. The box plots show the median, upper and lower quartile of the data set, while the whiskers mark the 5th and 95th percentiles. The mean value is marked with a small square in each figure.

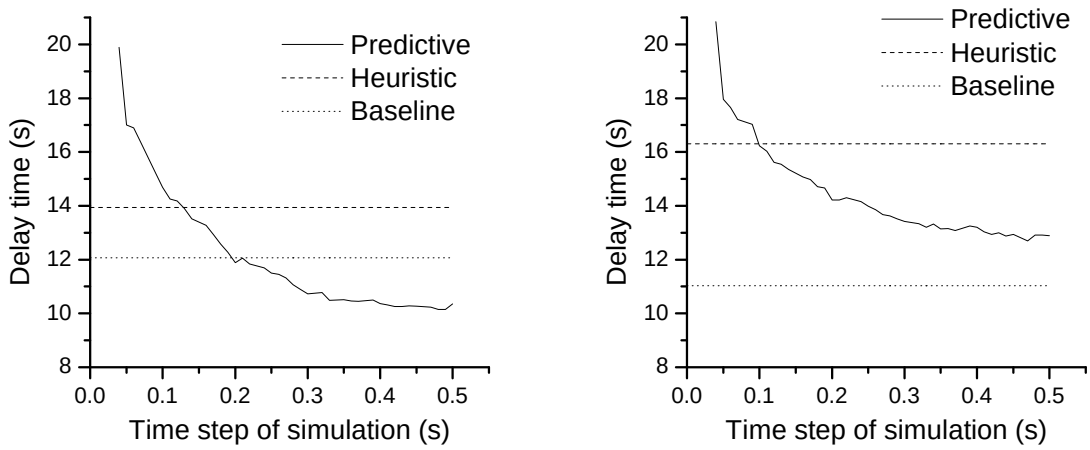
As expected, the predictive strategy performed best, followed by the heuristic strategy, and the baseline strategy performed worst. Compared to the baseline strategy, the predictive strategy increased the *delay time* by 67% in scenario 1 and and 84% in scenario 2. Compared to the heuristic strategy, the predictive strategy increased the delay time by 45% in scenario 1 and 25% in scenario 2. The difference in performance between the heuristic and predictive strategies is less apparent in scenario 2, likely due to the smaller number of choices during the task allocation step, increasing the chance of the heuristic making the right decision without any simulation.

4.4.3 Simulation Accuracy

The reliability of the predictive strategy depends on how accurately the predictive simulation is able to estimate the expected value of a candidate task allocation $\mathcal{A}_{u_i,j}$. To determine the effect of inaccuracies in the predictive simulation, we tried two ways to make the simulation less reliable, 1) by increasing the time step used by the predictive simulation (i.e., reducing its fidelity) and 2) by adding normally distributed error to the utility value returned by the predictive simulation (i.e., reducing its accuracy). The results of these experiments are shown in Fig. 4.12 and Fig. 4.13. In both the cases, there is a significant drop in performance as the simulation becomes less reliable, until its performance converges to some low level value.

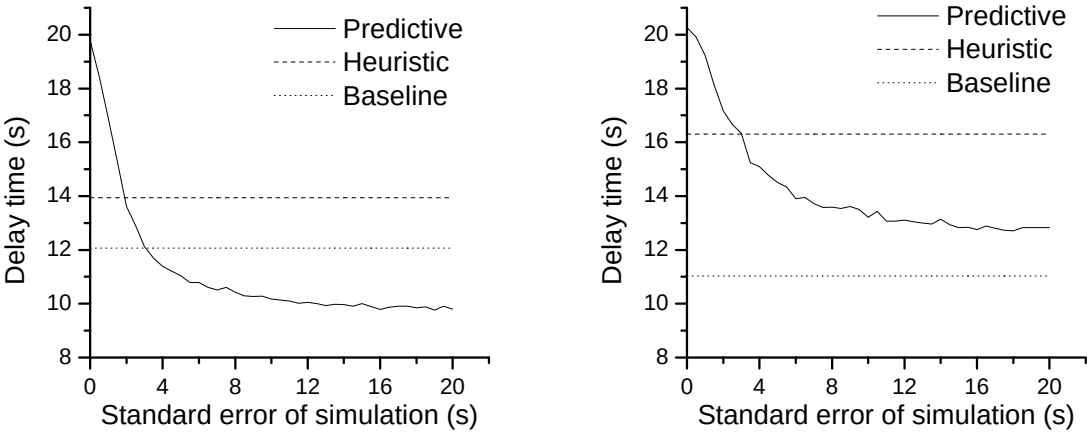
For scenario 1, the performance of the predictive strategy drops below that of the purely heuristic strategy when the simulation time step exceeds 0.14 seconds or the standard error exceeds 2 seconds. For scenario 2, this occurs when the simulation time step exceeds 0.1 seconds or the standard error exceeds 3 seconds. For reference, the time step of the actual simulator used for the experiments is 0.04 seconds. These results suggest that the quality of the simulation is very important to the feasibility of this approach.

In both scenarios, the predictive strategy was suitable for real-time computation, even at the smallest possible time step of 0.04 s. In scenario 1, the average running



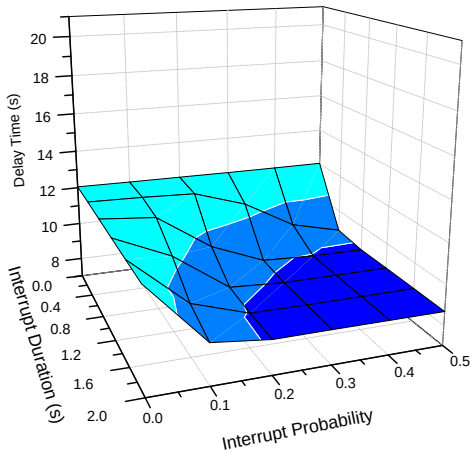
(a) Scenario 1 (b) Scenario 2

Figure 4.12: Mean *delay time* across 1000 randomly seeded trials for USV teams using the *predictive* strategy using time steps of various durations for the predictive simulation.

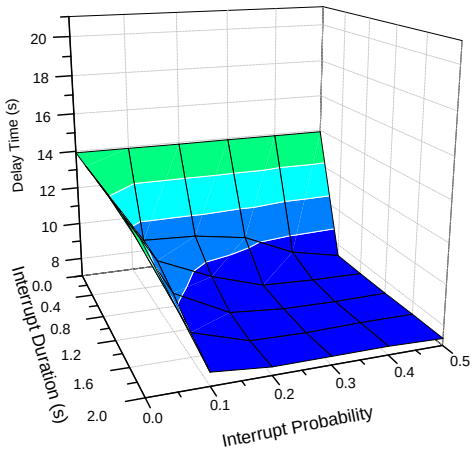


(a) Scenario 1 (b) Scenario 2

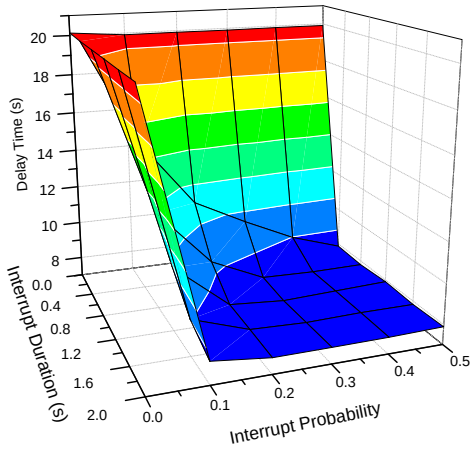
Figure 4.13: Mean *delay time* across 1000 randomly seeded trials for USV teams using the *predictive* strategy when normally distributed error is added to the result of the predictive simulation.



(a) Baseline, scenario 1

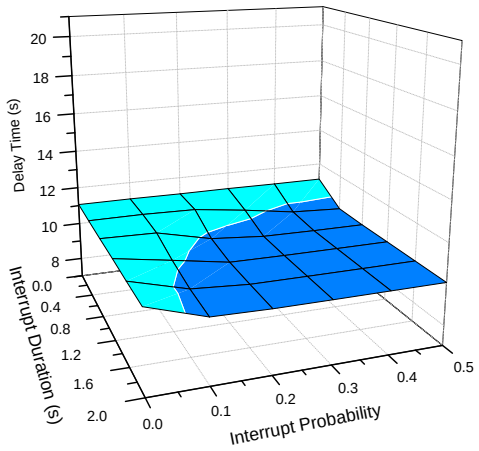


(b) Heuristic, scenario 1

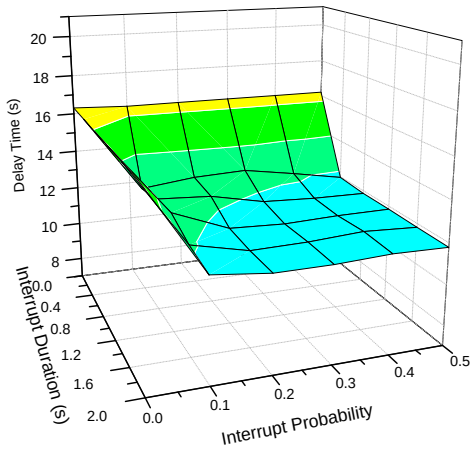


(c) Predictive, scenario 1

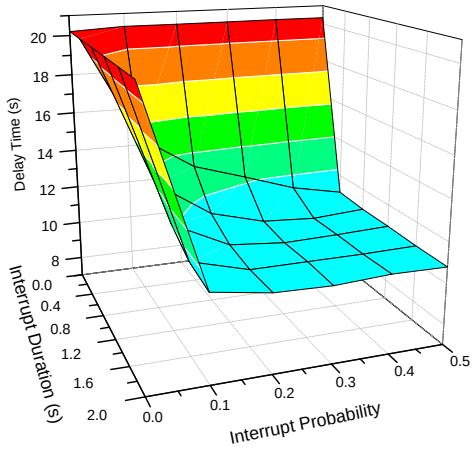
Figure 4.14: Mean *delay time* across 1000 randomly seeded trials for USV teams using the *predictive*, *heuristic*, and *baseline* strategies in the scenario 1 when communication between USVs is interrupted.



(a) Baseline, scenario 2



(b) Heuristic, scenario 2



(c) Predictive, scenario 2

Figure 4.15: Mean *delay time* across 1000 randomly seeded trials for USV teams using the *predictive*, *heuristic*, and *baseline* strategies in the scenario 2 when communication between USVs is interrupted.

time for a single USV to complete one task allocation step was 62 ms, with a worst-case time of 179 ms. For scenario 2, the average running time was 32 ms, with a worst-case time of 111 ms. The running time in the worst case was not significantly worse than the average-case, because in both cases we only evaluated one possible world. The allotted time between task allocation steps was fixed at 1000 ms, leaving room for more complex simulations to be used in the future.

4.4.4 Interrupted Communication

To determine the effect of interruptions in communication, we performed simulation runs where the communication link between each pair of USVs had a random chance of being interrupted. Fig. 4.14 and 4.15 show the average delay time for each of the three strategies when the probability and duration of interruption events are varied. The *interrupt probability* p_x is the chance that an interruption event will occur for a given pair of USVs at each 0.04 s time step, while the *interrupt duration* d_x is how long that interruption event will persist. Each interruption event is modeled as statistically independent and may overlap with other interruptions, blocking communication for a potentially indefinite period of time.

As the probability and duration of interruption events increase, it negatively impacts the performance of the USV team for all strategies. Going from no interruption (i.e. $p_x = 0$ and $d_x = 0$), to high interruption (i.e. $p_x = 0.5$ and $d_x = 2$ s), the mean *delay time* of the predictive strategy drops from 20.1 s to 7.9 s in scenario 1, and from 20.2 s to 10.9 s in scenario 2, a difference of 12.2 s and 9.3 s respectively. The impact on the heuristic strategy is less significant, dropping from from 13.9 s to 7.4 s in scenario 1, and from 16.3 s to 11.9 s in scenario 2, a difference of 6.5 s and 4.4 s respectively.

For both scenarios, the difference in performance between the heuristic and predictive strategies during high interruption is less than one second. Since task exchanges are not possible without communication between agents, high interruption renders the additional predictive simulation relatively useless. The baseline strategy also performs slightly better than both the heuristic and predictive strategies in scenario 1, with a delay time of 8.7 s, greater than the 7.9 s and 7.4 s of the other two. This may be due to the fact that the baseline strategy uses very few task exchanges to begin with and is therefore better optimized for situations with low communication.

4.5 Summary

In this chapter, we have developed a decentralized, contract-based planning approach for protecting an asset by a team of USVs operating in an environment with civilian traffic. The developed planner is able to deal with uncertainty about which boats are actual intruders and accounts for complex interactions of the USVs with the intruders when allocating tasks. The planner is capable of computing the task allocation in real-time, is scalable to large teams of USVs, and can be optimized for a specific mission.

We have demonstrated the performance of the planner in two different simulation scenarios. The performance was defined in terms of the expected time an intruder takes to reach the target after being detected by the team of USVs. In both scenarios, the developed model-predictive planner had a significant performance advantage compared to the baseline strategy as well as the heuristic strategy.

In future work, we will study the effect that blending multiple tasks with variable priorities has on guarding performance and learn an action-selection policy to produce state dependent, optimized parameters rather than utilizing a static set. We will enhance the planner to explicitly deal with the communication uncertainty by gradually adapting its task allocation strategy through blending the contract-based task sharing and offering with a purely local, greedy task assignment strategy. We will also enhance the model by realistic sensor noise, static obstacles with complex shapes, and apply the algorithm in the ground and aerial vehicles domains.

Bibliography

[1] P. Švec and S.K. Gupta. Automated synthesis of action selection policies for unmanned vehicles operating in adverse environments. *Autonomous Robots*, 32(2):149–164, 2012.

[2] M. Schwartz, P. Švec, A. Thakur, and S.K. Gupta. Simulation based synthesis of planning logic for autonomous unmanned sea surface vehicles. In *Simulation Driven Innovation and Discovery, Energetics Applications*. CALCE EPSC Press, College Park, 2011.

[3] P. Švec, M. Schwartz, A. Thakur, and S. K. Gupta. Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'11)*, San Francisco, CA, USA, September 25-30 2011.

[4] P. Švec and S.K. Gupta. Automated planning logic synthesis for autonomous unmanned vehicles in competitive environments with deceptive adversaries. In *New Horizons in Evolutionary Robotics: Post-Proceedings of the 2009 EvoDeRob Workshop*, pages 171–193. Springer, 2011.

[5] P. Švec, M. Schwartz, A. Thakur, D.K. Anand, and S.K. Gupta. A simulation based framework for discovering planning logic for Unmanned Surface Vehicles. In *ASME Engineering Systems Design and Analysis Conference (ESDA '10)*, Istanbul, Turkey, July 12-14 2010.

[6] P. Švec and S. K. Gupta. Competitive co-evolution of high-level blocking controllers for unmanned surface vehicles. In *Exploring New Horizons in Evo- lutionary Design of Robots Workshop, International Conference on Intelligent Robots and Systems (IROS'09)*, St. Louis, USA, October 11-15 2009.

[7] M. Schwartz, P. Švec, A. Thakur, and S. K. Gupta. Evaluation of automatically generated reactive planning logic for unmanned surface vehicles. In *Performance Metrics for Intelligent Systems Workhop (PERMIS '09)*, Gaithersburg, MD, USA, September 21-23 2009.

[8] A. Thakur, P. Švec, and S.K. Gupta. GPU based generation of state transition models using simulations for unmanned surface vehicle trajectory planning. *Robotics and Autonomous Systems*, 60(12):14571471, 2012.

[9] A. Thakur and S. K. Gupta. Generation of state transition models using simulations for unmanned sea surface vehicle trajectory planning. In *ASME 2011 International Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE)*, Washington D.C., USA, August 28-31 2011.

[10] A. Thakur and S.K. Gupta. Real-time dynamics simulation of unmanned sea surface vehicle for virtual environments. *Journal of Computing and Information Science in Engineering*, 11(3):031005, 2011.

[11] A. Thakur and S.K. Gupta. A Computational Framework for Real-Time Unmanned Sea Surface Vehicle Motion Simulation. In *ASME 2010 International Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE)*, Montreal, Canada, July 12-14 2010.

[12] P. Švec, A. Thakur, B. C. Shah, and S.K. Gupta. Target following with motion prediction for unmanned surface vehicle operating in cluttered environments. *Autonomous Robots*, 2013. Accepted for publication.

[13] P. Švec, A. Thakur, B. C. Shah, and S. K. Gupta. USV trajectory planning for time varying motion goal in an environment with obstacles. In *ASME Mechanisms and Robotics Conference*, Chicago, USA, August 12-15 2012.

[14] E. Raboin, P. Švec, D.S. Nau, and S.K. Gupta. Model-predictive asset defense by team of autonomous surface vehicles in environment with civilian boats. *Autonomous Robots*, 2014. In review.

[15] E. Raboin, P. Švec, D. S. Nau, and S. K. Gupta. Model-predictive target defense by team of unmanned surface vehicles operating in uncertain environments. In *IEEE International Conference on Robotics and Automation (ICRA'13)*, Karlsruhe, Germany, May 6-10 2013.

[16] C.R. Baker, D.I. Ferguson, and J.M. Dolan. Robust Mission Execution for Autonomous Urban Driving. *Intelligent Autonomous Systems 10: IAS-10*, page 155, 2008.

[17] S. Whiteson. *Adaptive representations for reinforcement learning*, volume 291. Springer Verlag, 2010.

[18] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[19] G. Theocharous and L.P. Kaelbling. Approximate planning in pomdps with macro-actions. *Advances in Neural Information Processing Systems*, 16, 2004.

[20] N. Kohl and R. Miikkulainen. Evolving neural networks for fractured domains. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1405–1412. ACM, 2008.

[21] D. Floreano and C. Mattiussi. *Bio-inspired artificial intelligence: theories, methods, and technologies*. 2008.

[22] D. Andre, N. Friedman, and R. Parr. Generalized prioritized sweeping. *Advances in Neural Information Processing Systems*, pages 1001–1007, 1998.

[23] J.R. Koza. *Genetic programming IV: Routine human-competitive machine intelligence*. Kluwer Academic Pub, 2003.

[24] M. Schwartz, P. Švec, A. Thakur, and S.K. Gupta. Evaluation of Automatically Generated Reactive Planning Logic for Unmanned Surface Vehicles. In *Performance Metrics for Intelligent Systems Workshop (PERMIS'09)*, Gaithersburg, MD, USA, September 21-23 2009.

[25] S.J. Corfield and J.M. Young. Unmanned surface vehicles–game changing technology for naval operations. *Advances in unmanned marine vehicles*, pages 311–328, 2006.

[26] A. Finn and S. Scheding. *Developments and Challenges for Autonomous Unmanned Vehicles: A Compendium*. Springer, 2010.

[27] N.S. Board. Autonomous vehicles in support of naval operations. *National Research Council, Washington DC*, 2005.

[28] C. Sammut and G.I. Webb. *Encyclopedia of machine learning*. Springer-Verlag New York Inc, 2011.

[29] S.M. LaValle. Filtering and planning in information spaces. *IROS tutorial notes*, 2009.

[30] M.A.A. Cox and T.F. Cox. Multidimensional scaling. *Handbook of data visualization*, pages 315–347, 2008.

[31] R. Poli, W.B. Langdon, and N.F. McPhee. *A field guide to genetic programming*. Lulu Enterprises Uk Ltd, 2008.

[32] S. M. LaValle. *Planning algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[33] M. L. Puterman. *Markov decision processes: Discrete stochastic dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[34] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT Press, 2005.

[35] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[36] T. Schouwenaars, B. Mettler, E. Feron, and J. P. How. Robust motion planning using a maneuver automation with built-in uncertainties. In *American Control Conference, 2003. Proceedings of the 2003*, volume 3, pages 2211 – 2216 vol.3, june 2003.

[37] O. M. Faltinsen. *Sea loads on ships and offshore structures.* Cambridge University Press, Cambridge, New York., 1990.

[38] T. I. Fossen. *Guidance and control of ocean vehicles.* Wiley, Chicester, England, 1994.

[39] P. Krishnamurthy, F. Khorrami, and S. Fujikawa. A modeling framework for six degree-of-freedom control of unmanned sea surface vehicles. In *Proc. and 2005 European Control Conference Decision and Control CDC-ECC '05. 44th IEEE Conference on*, pages 2676–2681, December 12–15, 2005.

[40] A. Thakur and S. K. Gupta. Real-time dynamics simulation of unmanned sea surface vehicle for virtual environments. *Journal of Computing and Information Science in Engineering*, 11(3):031005, 2011.

[41] S. P. Singh and D. Sen. A comparative linear and nonlinear ship motion study using 3-D time domain methods. *Ocean Engineering*, 34(13):1863 – 1881, 2007.

[42] J. Betz. Solving rigid multibody physics dynamics using proximal point functions on the GPU. Master's thesis, Rensselaer Polytechnic Institute, Troy, New York, 2011.

[43] J. Pan and D. Manocha. GPU-based parallel collision detection for real-time motion planning. In David Hsu, Volkan Isler, Jean-Claude Latombe, and Ming Lin, editors, *Algorithmic Foundations of Robotics IX*, volume 68 of *Springer Tracts in Advanced Robotics*, pages 211–228. Springer Berlin / Heidelberg, 2011.

[44] W. L. D. Lui and R. Jarvis. Eye-Full tower: A GPU-based variable multibaseline omnidirectional stereovision system with automatic baseline selection for outdoor mobile robot navigation. *Robotics and Autonomous Systems*, 58(6):747 – 761, 2010.

[45] J.T. Kider, M. Henderson, M. Likhachev, and A. Safonova. High-dimensional planning on the GPU. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2515 –2522, may 2010.

[46] J. N. Newman. *Marine Hydrodynamics.* MIT Press, Cambridge, MA, 1977.

[47] W. Powell. *Approximate dynamic programming: Solving the curses of dimensionality.* Wiley, 2007.

[48] V. Bertram. Unmanned surface vehicles–a survey. *Skibsteknisk Selskab, Copenhagen, Denmark*, 2008.

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

[50] M.M. Graham. Unmanned surface vehicles: An operational commander's tool for maritime security. Technical report, DTIC Document, 2008.

[51] A. Aguiar, J. Almeida, M. Bayat, B. Cardeira, R. Cunha, A. Häusler, P. Maurya, A. Oliveira, A. Pascoal, and A. Pereira. Cooperative control of multiple marine vehicles. In *Proc. of 8th IFAC International Conference on Manoeuvering and Control of Marine Craft*, pages 16–18, 2009.

[52] D.P. Eickstedt, M.R. Benjamin, and J. Curcio. Behavior based adaptive control for autonomous oceanographic sampling. In *IEEE International Conference on Robotics and Automation*, pages 4245–4250, 2007.

[53] J.P. Ryan, S.B. Johnson, A. Sherman, K. Rajan, F. Py, H. Thomas, J.B.J. Harvey, L. Bird, J.D. Paduan, and R.C. Vrijenhoek. Mobile autonomous process sampling within coastal ocean observing systems. *Limnol. Oceanogr.: Methods*, 8:394–402, 2010.

[54] W. Naeem, R. Sutton, and J. Chudley. Modelling and control of an unmanned surface vehicle for environmental monitoring. In *UKACC International Control Conference, August, Glasgow, Scotland ?*, 2006.

[55] E.T. Steimle and M.L. Hall. Unmanned surface vehicles as environmental monitoring and assessment tools. In *OCEANS 2006*, pages 1–5. IEEE, 2006.

[56] W. Naeem, T. Xu, R. Sutton, and A. Tiano. The design of a navigation, guidance, and control system for an unmanned surface vehicle for environmental monitoring. *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, 222(2):67, 2008.

[57] R. Yan, S. Pang, H. Sun, and Y. Pang. Development and missions of unmanned surface vehicle. *Journal of Marine Science and Application*, 9(4):451–457, 2010.

[58] J. Majohr and T. Buch. Modelling, simulation and control of an autonomous surface marine vehicle for surveying applications measuring dolphin messin. *IEE Control Engineering Series*, 69:329, 2006.

[59] 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, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*, pages 1–6, 2009.

[60] R.R. Murphy, E. Steimle, C. Griffin, C. Cullins, M. Hall, and K. Pratt. Cooperative use of unmanned sea surface and micro aerial vehicles at hurricane wilma. *Journal of Field Robotics*, 25(3):164–180, 2008.

[61] A.J. Shafer, M.R. Benjamin, J.J. Leonard, and J. Curcio. Autonomous cooperation of heterogeneous platforms for sea-based search tasks. In *Proceedings of MTS/IEEE OCEANS*, pages 1–10. IEEE, 2008.

[62] R.M. Kilgore, K.A. Harper, C. Nehme, and ML Cummings. Mission planning and monitoring for heterogeneous unmanned vehicle teams: A human-centered perspective. In *AIAA Infotech@ Aerospace Conference in Sonoma, CA*, 2007.

[63] E. Simetti, A. Turetta, G. Casalino, E. Storti, and M. Cresta. Towards the use of a team of USVs for civilian harbour protection: Real time path planning with avoidance of multiple moving obstacles. In *IEEE 3rd Workshop on Planning, Perception and Navigation for Intelligent Vehicles (IROS'09), St. Louis, MO, US*, 2009.

[64] Enrico Simetti, Alessio Turetta, Giuseppe Casalino, Enrico Storti, and Matteo Cresta. Protecting assets within a civilian harbour through the use of a team of usvs: Interception of possible menaces. In *IARP Workshop on Robots for Risky Interventions and Environmental Surveillance-Maintenance (RISE'10), Sheffield, UK*, 2010.

[65] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[66] T.I. Fossen. *Handbook of marine craft hydrodynamics and motion control*. Wiley, 2011.

[67] Thierry Fraichard and Hajime Asama. Inevitable collision states a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[68] S. M. LaValle. *Planning algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu.

[69] R. A. Soltan, H. Ashrafiuon, and K. R. Muske. State-dependent trajectory planning and tracking control of unmannedsurface vessels. In *American Control Conference, 2009. ACC '09.*, pages 3597 –3602, 2009.

[70] I. Zohar, A. Ailon, and R. Rabinovici. Mobile robot characterized by dynamic and kinematic equations and actuator dynamics: Trajectory tracking and related application. *Robotics and autonomous systems*, 59(6):343–353, JUN 2011.

[71] H. Ashrafiuon, K. R. Muske, and L. C. McNinch. Review of nonlinear tracking and setpoint control approaches for autonomous underactuated marine vehicles. In *American Control Conference (ACC), 2010*, pages 5203 –5211, 302010-july2 2010.

[72] M. Pivtoraiko and A. Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, pages 3231–3237. IEEE, 2005.

[73] Colin Green and Alonzo Kelly. Toward optimal sampling in the space of paths. In *13th International Symposium of Robotics Research*, November 2007.

[74] Mihail Pivtoraiko, Issa AD Nesnas, and Alonzo Kelly. Autonomous robot navigation using advanced motion primitives. In *IEEE Aerospace conference*, pages 1–7. IEEE, 2009.

[75] Lawrence H Erickson and Steven M LaValle. Survivability: Measuring and ensuring path diversity. In *IEEE International Conference on Robotics and Automation (ICRA'09)*, pages 2068–2073. IEEE, 2009.

[76] Mikhail Pivtoraiko. *Differentially Constrained Motion Planning with State Lattice Motion Primitives*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2012.

[77] S. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Prentice Hall, 2009.

[78] A.J. Healey and D. Lienard. Multivariable sliding mode control for autonomous diving and steering of unmanned underwater vehicles. *IEEE Journal of Oceanic Engineering*, 18(3):327 –339, July 1993.

[79] Peter F Sturm and Stephen J Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1. IEEE, 1999.

[80] Jean-Yves Bouguet. *Camera Calibration Toolbox for Matlab*. 2010. Available at http://www.vision.caltech.edu/bouguetj/calib_doc.

[81] B.P. Gerkey and M.J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[82] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.