Chapter 12
Automated Planning Logic Synthesis for Autonomous Unmanned Vehicles in Competitive Environments with Deceptive Adversaries

Petr Svec and Satyandra K. Gupta

Abstract. We developed a new approach for automated synthesis of a planning logic for autonomous unmanned vehicles. This new approach can be viewed as an automated iterative process during which an initial version of a logic is synthesized and then gradually improved by detecting and fixing its shortcomings. This is achieved by combining data mining for extraction of vehicle's states of failure and Genetic Programming (GP) technique for synthesis of corresponding navigation code. We verified the feasibility of the approach using unmanned surface vehicles (USVs) simulation. Our focus was specifically on the generation of a planning logic used for blocking the advancement of an intruder boat towards a valuable target. Developing autonomy logic for this behavior is challenging as the intruder's attacking logic is human-competitive with deceptive behavior so the USV is required to learn specific maneuvers for specific situations to do successful blocking. We compared the performance of the generated blocking logic to the performance of logic that was manually implemented. Our results show that the new approach was able to synthesize a blocking logic with performance closely approaching the performance of the logic coded by hand.

12.1 Introduction

Development of increased autonomy for unmanned vehicles to successfully fulfill ordinary mission tasks, such as navigation between two locations while avoiding

Petr Svec
Institute for Systems Research and Department of Mechanical Engineering,
University of Maryland, College Park, MD 20742, USA
Phone: 301-405-5577
e-mail: petrsvec@umd.edu

Satyandra K. Gupta
Institute for Systems Research and Department of Mechanical Engineering,
University of Maryland, College Park, MD 20742 USA
Phone: 301-405-5306
e-mail: skgupta@umd.edu

S. Doncieux et al. (Eds.): New Horizons in Evolutionary Robotics, SCI 341, pp. 171–193.
springerlink.com

dynamic obstacles, is still considered to be a challenge. Handling all the variations in the encountered environments requires writing and extensive tuning a large amount of lines of code by human programmers. Yet the real difficulty comes when the unmanned vehicle has to autonomously face human-competitive adversary utilizing deception. This truly challenging scenario is typical for a combat mission where even a single mistake in the decision of the vehicle can have serious consequences.

Computational synthesis (CS) [1] deals with the problem of how to automatically compose and parametrize a set of functional building blocks into increasing amount of modules that are further organized into a hierarchical solution structure with the desired functionality. This is in contrast to classical optimization in which the number and structure of modules and parameters being optimized is known in advance. The rapid growth of CS in the recent years can be accounted to increasingly affordable computational power and continuing advances in machine learning and simulation techniques.

For many years since its original inception, Genetic Programming (GP) [2, 3] has been emerged as an invention machine for automated synthesis of controllers that are simpler and more efficient than those engineered with other standard design methods. The random exploration feature of this technique allows generating creative human-readable solutions in contrast to human developers who are often limited by constraints of standard engineering methods. GP as one of the robust evolutionary techniques has been used for automatically generating computer programs in various domains. These programs usually have a tree structure and are generated using an algorithm similar to the traditional genetic algorithm (GA) [4]. In literature, there are many successful GP applications to numerous problems from different domains [5] including robotics, optimization, automatic programming, machine learning, etc. In robotics, GP is used as a methodology that uses evolutionary algorithms to automatically synthesize controllers and body configuration for autonomous robots. Most of the controllers were evolved for behaviors as obstacle avoidance [6, 7], wall-following [8], line following [9], light seeking, robot seeking [7], box pushing [10], vision-driven navigation [11], homing and circling [12], predator versus prey strategies [13], co-evolution of control and bodies morphologies [14], game playing [15-17] or group control for survey missions [18]. GP was also utilized for the automated synthesis of human-competitive strategies for robotic tanks run in a closed simulation area to fight other human-designed tanks in international leagues [19]. There was also some progress on development of limited machine intelligence for classical strategic games like backgammon or chess endgames [20].

Most of the evolved controllers, however, are purely reactive and contain a predefined number of modules. This substantially limits their usage for complex tasks that involve competitive adversaries, either machines or human operators themselves. We therefore propose a new approach for automated planning logic synthesis that can be viewed as an iterative learning process during which an initial version of the logic is automatically synthesized and then gradually improved by detecting and fixing its shortcomings. This is achieved by the interplay of data mining for state

extraction and classification and GP for navigation code generation. During the automated development, no external information on how the logic should be generated is therefore needed. The planning logic is represented as a composition of one main navigation controller and a set of navigation plans. The navigation controller is used to control the vehicle's behavior in all situations besides the situations for which specific maneuvers are needed to boost the performance of the logic. The symbolic representation of the planning logic greatly simplifies its functional analysis in contrast to the artificial neural network representation that is difficult to analyze. Moreover, the symbolic representation allows integrating human knowledge naturally so the analysis of the logic can provide the basis for hand-coding logic for real-life applications.

We tested our new approach in the context of a larger project aimed at the development of a mission planning system [21] for the automated generation of planning logic for unmanned surface vehicles (USVs) [22, 23]. In this part of the work, our focus is specifically on automated generation of logic used for blocking the advancement of an intruder boat towards a valuable target. This task requires the USV to utilize reactive planning complemented by short-term forward planning to generate local navigation 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 [24]. In this level, the unmanned vehicle automatically executes mission-related commands when response times are too short for operator intervention. The operator, however, may cancel or redirect the vehicle's intended actions. As far as the performance evaluation of the logic is concerned, we 1) manually implemented USV's logic for blocking the hand coded intruder, and 2) compared its performance to the automatically generated USV's logic by pitting it against the same intruder in a large amount of test scenarios.

An overview of the overall approach is shown in Figure 12.1. First, we developed a physics-based meta-model of a full blown USV dynamics model to be able to test the planning logic in a simulation environment in real-time [25]. Second, we developed a mission planning system whose main part is GP based evolutionary module for generating individual components of the planning logic (see section 12.3). In order to combine the elements of the project into a cohesive system, we designed a USV simulation environment [26]. The USV simulation environment integrates various components of the project into a complete simulation system and acts as a simulation platform for the evolutionary module. One of the components of the USV simulation environment is the virtual environment (VE) based simulator (see Figure 12.2) which serves as an emulator of the real USV environment and contains gaming logic which allows human players to play against each other or against the computer. Finally, we evaluated the performance of the automatically generated USV's logic against a hand coded human-competitive intruder exhibiting deceptive behavior (see section 12.4).

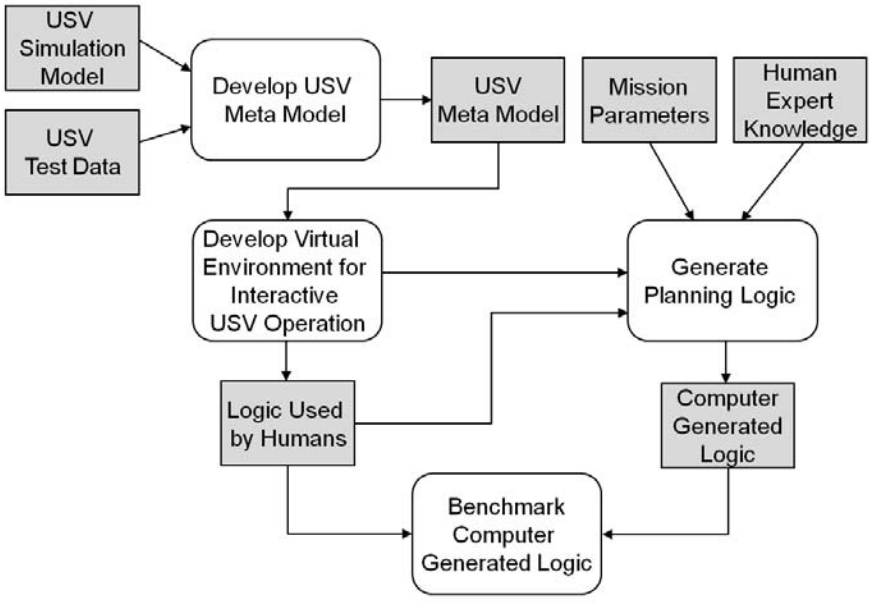

Fig. 12.1 Overview of the overall approach.

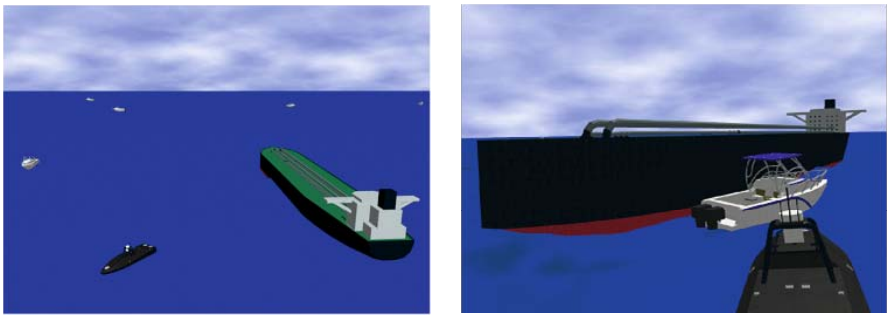

Fig. 12.2 Virtual environment.

12.2 USV System Architecture

The overall USV's system architecture consists of several modules that are responsible for different tasks, e.g. sensing, localization, navigation, planning, behavior control, communication, human interaction, or monitoring [22, 23]. The 6 degrees of freedom USV dynamics simulation model was implemented as described in [25]. This detailed model considers disturbances from the surroundings and is used for game playing inside the virtual environment. Due to its computational requirements, its simplified version that has 3 degrees of freedom is used for automated logic generation where evaluation speed is an issue.

12.2.1 USV Virtual Sensor Models

The role of sensing is to provide information about the current state of the vehicle in the environment. This information serves as a basis for decision making and control by taking other objects in the environment into account. A real USV is usually equipped by radar and multiple short-range visibility sensors to be able to capture the state of its surroundings. However, for the navigation system of the USV to be effective it only needs to process relevant key sensory information abstracted from the raw sensor data. The sensory information is thus represented as a vector containing only the features required for a successful fulfillment of the mission task. The values of the relevant features are computed using the data from the virtual sensors [27] that provide intermediate abstraction of the raw sensor data. Some of the features can also have one or more parameters using which their final values are computed.

The navigation system of the USV uses virtual visibility, relational, and velocity sensors. The virtual visibility sensor is a detection sensor with cone-shaped detection regions (see Figure 12.3). The dimension 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 to the USV or to each other. It provides boolean values for computation of the values of the relational features. The velocity virtual sensor returns velocities of other objects inside the environment.

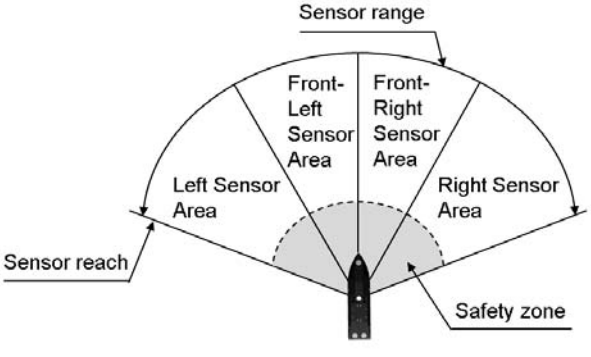

Fig. 12.3 Virtual visibility sensor model.

12.2.2 Planning Architecture

The complexity of interactions of a mobile robotic system suggests structured (non-monolithic) high-level planning architecture. The unmanned boats must behave based on the effect of several independent threads of reasoning. This is due to the

highly parallel nature of events and processes in an uncertain and often dynamic environment. The control architecture can meet this requirement if it is modular, and when the modules can act simultaneously in a coordinated cooperation in real time.

12.2.2.1 Planning Logic Representation

The planning logic allows the USV to make a decision from a set of allowable actions in a particular situation during its run. It consists of a main navigation controller represented as a decision tree and zero or more navigation plans (see Figure 12.4). For the operators to understand and trust the vehicle's actions is of a great importance. Hence, the representation of the planning logic is symbolic which simplifies its functional verification by human auditing.

The main navigation controller operates the USV unless the vehicle approaches a state for which a specific short-term navigation plan exists. The navigation plan thus represents a certain maneuver the USV executes in a certain situation to increase its performance. The main components of the navigation controller are high-level parameterized navigation commands $NC = \{$*go-intruder-front (front-left, left, front-right, right), turn-left (right), go-straight*$\}$, conditional variables $CV = \{$*intruder-on-the-left (right, front, at-the-back), 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, usv-left (right, front-left, front-right) visibility-sensor-area-activated, intruder-target-angle-between, intruder-velocity-le-than*$\}$, standard boolean values and operators $BVO = \{$*if, true, false, and, or, not*$\}$, *program blocks* $PB = \{$*seq2, seq3*$\}$, and system commands $SC = \{$*usv-sensor, usv-velocity, usv-match-intruder-velocity*$\}$. The main components of the navigation plan are high-level commands *NC* and program blocks *PB*. The leaves of the decision tree can be conditional variables or navigation commands. The inner nodes can be conditionals, navigation commands, or system commands. Each navigation command corresponds to a particular high-level controller, which is a parameterized composition of simple behaviors according to the behavior-based control architecture [28]. The next section describes this in detail.

The conditional variables, navigation, and system commands are parameterized. The parameters of a navigation command define its underlying property. The positional commands (e.g. *go-intruder-front*) are defined using 5 parameters. 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 planning logic, 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 navigation commands have two parameters that represent the turning rate and the length of the command execution. The translation velocity is explicitly controlled by the velocity commands. The *usv-sensor* system command effectively changes the parameters of the USV's sensors allowing it to explicitly control the obstacle avoidance behavior to be between very safe and very risky. Each

parameter of the command is propagated to the underlying primitive behaviors of each corresponding high-level controller.

The input into the planning logic is data from the virtual sensors. The values of all conditional variables are computed using this data. So for example, the boolean value of the *intruder-on-the-left* variable is directly provided by the virtual relation sensor, while the data for computation of the *intruder-velocity-le-than* parameterized variable is provided by the virtual velocity sensor.

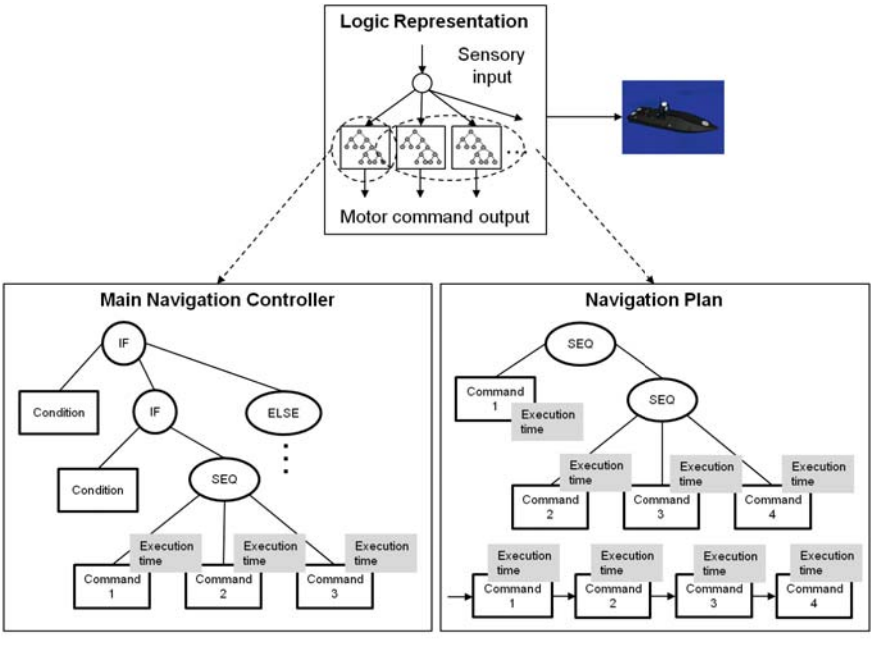

Fig. 12.4 Planning logic representation.

12.2.2.2 Hierarchical Control Architecture

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 planning logic executor decides whether the current USV's state is close enough to one of the states for which a corresponding navigation plan exists. If such a plan exists, the planning logic executor (see Figure 12.6) directly executes the plan, otherwise it executes the main navigation controller to generate a new plan. The decision whether to execute a specific navigation 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 navigation plan. The state space is thus divided into two regions where in the first region the USV generates and executes plans using the navigation controller, whereas in the other region the USV directly executes previously learned

plans. The distance between normalized states is computed using the standard Euclidean distance metric.

The full state of the USV (see Figure 12.5) is a vector $s = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, v_1, v_2, d\}$. The angle α_1 represents an angle between the USV's heading and the direction to the target, α_2 is an angle between the intruder's heading and the direction to the target, β_1 is the USV's steering angle, β_2 is the intruder's steering angle, $v1$ is the USV's translation velocity, $v2$ is the intruder's translation velocity, γ_1 is an angle between the USV's heading and the direction to the intruder, and d is the distance between the USV and the intruder.

By acquiring and processing sensory information in short-term cycles, and planning, the planning system determines a navigation command to be executed through the behavior-based control system to direct the vehicle inside the environment. The planning logic executor of the navigation system takes as inputs sensor data, mission parameters, USV meta model, and the planning logic. It decides which component of the logic to execute to generate a plan based on the current vehicle's state. The plan consists of a number of navigation commands, each being executed for a certain amount of time. The ultimate outputs of an activated command are way points that are directly translated by a low-level controller into motor commands for device drivers of a particular actuator.

The control architecture is hierarchical and follows the behavior-based paradigm [28]. It consists of planning, executive, and reactive layers (see Figure 12.6). The planning layer is responsible for interpreting stored planning logic and generating navigation plans that contain one or more navigation commands at a time. The commands are stored in a queue to be further dispatched for execution by the dispatcher in the executive layer. The commands are usually planned for short-term execution,

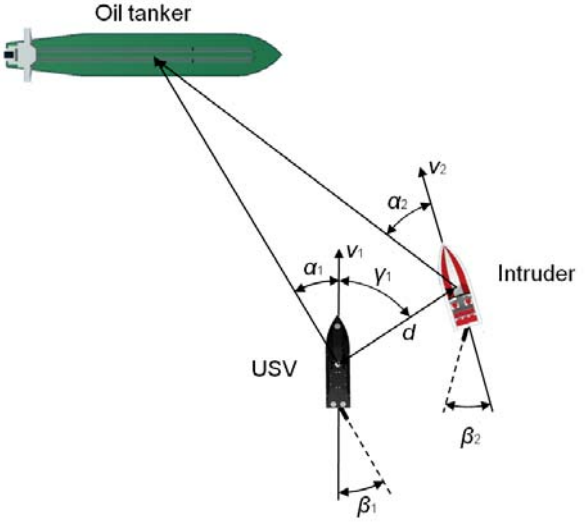

Fig. 12.5 USV's state in respect to the intruder and target.

such as planning of strategic maneuvers. The vehicle thus does not act purely reactively to its surroundings unless an exception occurs.

Each command corresponds to a high-level controller, which is a parameterized composition of simple behaviors organized into layers according to the behavior-based control architecture [28]. The executive layer is responsible for processing the commands in the queue and invoking the corresponding high-level controllers in a series for planned periods of time. The length of the execution is defined as a parameter of the command. The executive layer is also responsible for monitoring execution of the controllers and handling exceptions. An exception occurs if the current state of the vehicle substantially deviates from the predicted trajectory defined by the plan. The planning logic executor remains inactive until all the commands from the queue are processed in which case the dispatcher requests new commands from the planning logic executor and the control process continues.

The reactive layer implements the behavior-based subsumption architecture [28]. This architecture decomposes a complicated high-level controller into a set of simple behaviors (steer left / right, go straight, arrive) organized into layers. These primitive behaviors are finite state machines acting in response to sensor inputs and producing actuator action outputs. Multiple behaviors can be activated simultaneously producing different conflicting motor commands. This means that a certain amount of coordination is needed. Due to its robustness, we have chosen a priority-based arbitration mechanism, picking the actuator action output of the behavior with the highest priority as the overall action output of the currently activated high-level controller. This closely follows the behavior-competitive paradigm that imposes that only one behavior can have control over the robot's actuators while each of them can access all sensors. In this paradigm, the behavior in the highest layer has the highest priority (for example obstacle avoidance) while the behavior in the lowest layer represents the most abstract functionality. In the architecture, each high-level controller specifies a fixed priority ordering of behaviors as defined by [29].

The planning logic is executed in a perception-action high rate cycle. The input into the logic is processed sensor data and the ultimate output is an actuator action. The ability of the architecture to deal with highly dynamic and unpredictable scenarios is due to its underlying reactive behavior-based competitive component. This component triggers local obstacle avoidance mechanism at a high rate to immediately detect possible collision threats.

The primitive behaviors are of a great importance as they are able to quickly produce an action in a highly dynamic environment where fast response and responsibility are crucial. A behavior is a simple unit that produces an output of a pre-defined type, in our case a two-dimensional vector containing desired translation velocity and steering angle. Conditions for activating behaviors are preprogrammed. The architecture defines the following primitive behaviors: *obstacle avoidance, go to location, go straight, turn left*, and *turn right*.

The *obstacle avoidance* behavior implements a simple but efficient obstacle avoidance mechanism for dynamic environments and is a necessary part of all high-level controllers. It uses the virtual visibility sensor (see Figure 12.3) in order to identify the location of detectable obstacles. It directly produces desired translation

velocity and steering angle to safely steer the boat away from the closest identified obstacles. The desired steering angle increases with the proximity to the obstacles while the translation velocity decreases.

The manual design of a flexible reactive obstacle avoidance mechanism is not an easy task. The USV should be able to prevent a collision with its surroundings while effectively executing its mission task. This brings plenty of challenges since the behavior of many standard obstacle avoidance methods is driven by its carefully tuned parameters. These parameters control the behavior of the vehicle, particularly how much steering should be applied when a nearby obstacle is positioned at a certain distance and angle, and how fast the vehicle should be moving in that situation. Hence for our mission, the resulting behavior can be quite different with different parameters essentially controlling the vehicle's safe distance from the adversary and blocking efficiency at the same time. Insufficient avoidance steering can lead to collisions. On the other hand, too much steering will veer the vehicle away from the adversary leading to ineffective blocking. Moreover, as far as computational requirements are taken into account, the obstacle avoidance mechanism should be very fast so that it does not consume much of the precious simulation time.

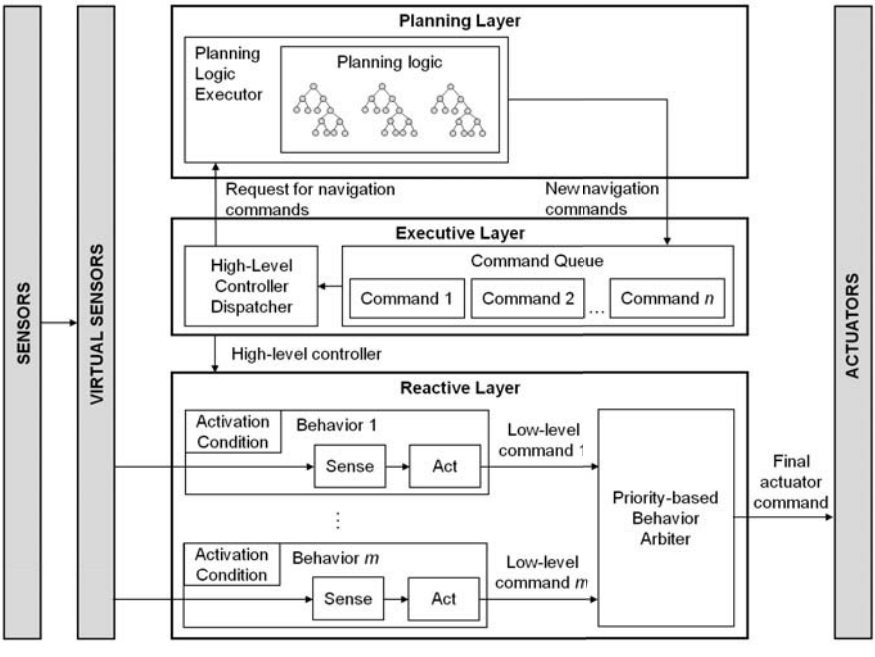

Fig. 12.6 Hierarchical behavior-based control architecture.

We have implemented a collision avoidance method that uses high-level sensory information, e.g. positions, orientations, and dimensions of obstacles, to directly decide which steering angle and translation velocity to request. The avoidance

mechanism uses a fixed set of control parameters. However, the behavior can be conveniently controlled by modification of the underlying parameters of the visibility sensor. This way, the USV can get closer to obstacles than it would be otherwise possible and thus effectively define a balance between safe and aggressive maneuvering. The command *usv-sensor* of the planning logic modifies the reach and range parameters of the virtual visibility sensor cones.

By default, the behaviors always choose such translation and steering velocities that maximize the USV's performance. So for example, *go-straight* behavior uses maximum translation velocity to get to the requested position in the shortest amount of time. The planning logic can override this by calling *usv-velocity* system command. This command switches the vehicle to its controlled velocity mode in which the translation velocity of the USV is controlled by the higher-level planning logic.

12.3 Planning Logic Synthesis

12.3.1 Test Mission

Our task was to automatically generate a planning logic for the USV to slow down an intruder boat moving toward the protected object. The USV's blocking logic is defined in the context of a test mission. During this mission, the USV must protect an oil tanker by patrolling around it while avoiding collisions with friendly boats and scanning the environment for a possible intruder. The environment around the oil tanker is divided into danger and buffer zones (see Figure 12.7). Once the intruder enters the buffer zone, the USV approaches the intruder boat and circles it for surveillance purposes. If the intruder enters the danger zone, the USV does its best to block the intruder, slowing the intruder's progress toward the tanker.

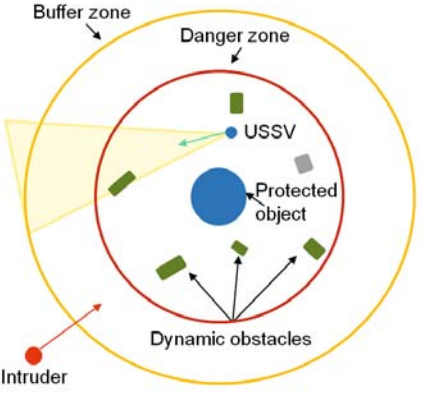

Fig. 12.7 Test mission.

12.3.2 Synthesis Scheme

The natural way of automated logic generation is to let the unmanned vehicle to autonomously learn its own logic for a given task. Learning the logic usually requires many simulations of different actions in different situations so that the vehicle can determine which actions are beneficial and which are not. Our approach for the automated logic synthesis can be viewed as a completely automated iterative process during which an initial version of the logic is automatically synthesized and then gradually improved by detecting and fixing its shortcomings.

The planning logic generation process consists of six main parts as shown in Figure 12.8. First, an initial version of the logic containing only a navigation controller is generated and consequently evaluated inside the simulator in m distinct evaluation runs. This evaluation returns a set of states of failure $SF = \{SOF_1, \ldots, SOF_n\}$, $n \leq m$ in which the vehicle fails its mission task. Given this set, a representative state of failure $SOF_{REP} \in SF$ is found for which the average of distances to its k nearest neighboring states is minimal. SOF_{REP} is thus a state located in the center of a cluster with the highest density of states of failure. The distance between the normalized states is computed using the Euclidean distance metric. Next, we compute a corresponding state of exception SOE_{REP} for the representative state of failure SOF_{REP}. SOE_{REP} defines a state in which proximity (given by the activation distance parameter δ) the vehicle can execute a specific navigation plan to decrease the probability of occurrence of the corresponding SOF_{REP} and all the failure states in its close neighborhood. Once the state of exception SOE_{REP} is computed, a new specific navigation plan is synthesized for this state. To prevent

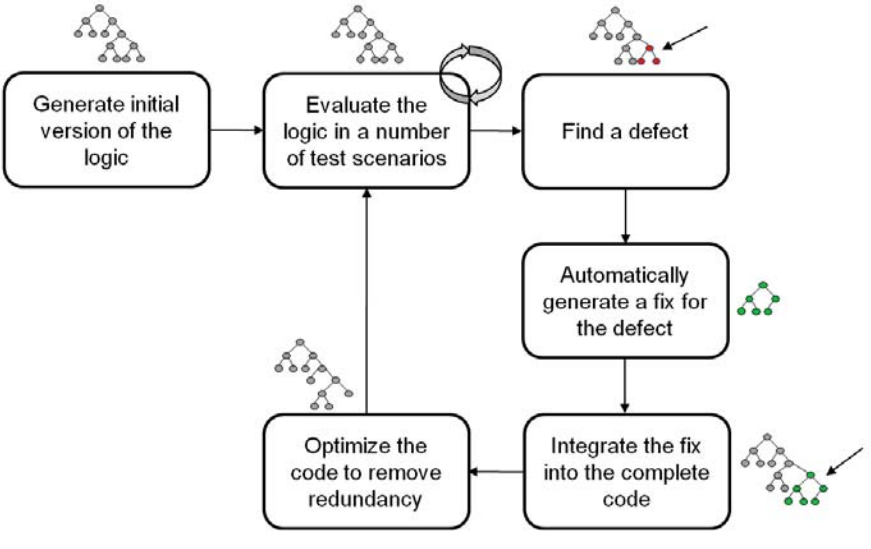

Fig. 12.8 Planning logic synthesis overview.

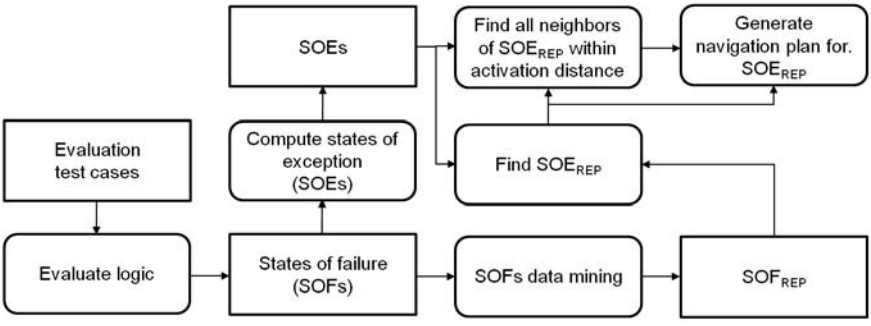

Fig. 12.9 Extraction of states of exception.

overspecialization of the new plan, we evaluate its performance using all states of exception found within SOE_{REP} activation distance δ during the overall logic evaluation. The new plan together with its corresponding SOE_{REP} is then integrated into the whole logic, the logic is optimized, and the process starts again.

In the context of our mission, *SOF* defines a situation in which the USV has high probability of losing its future maneuver to block the intruder. Its corresponding *SOE* is found by reverting back in time for a certain number of time steps τ to record a state from which a new specific navigation plan can be executed to prevent a possible future failure. The activation distance parameter δ defines a minimum distance between the current USV's state and any previously recorded SOE_{REP} from which a corresponding navigation plan can be executed.

12.3.3 Planning Logic Components Evolution

Both the navigation controller and navigation plans as components of the planning logic are automatically generated using separate simulated evolutionary processes. The specific evolutionary method we used is the strongly-typed Genetic Programming imposing type constraints on the generated Lisp trees [15, 16]. 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 (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 [42]. We seed the initial population by human-coded Lisp fragments to speed up the process. The initial values of parameters of all navigation commands and conditionals are either seeded or randomly generated. The navigation controller of the planning logic 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 evolved.

The terminal T and function sets F consists of navigation commands, system commands, conditional variables, boolean values and operators, and program blocks as defined in section 12.2.2. The sets are defined as

$T_{controller} = T_{plan} = NC \cup SC$

$F_{controller} = CV \cup BVO \cup PB; F_{plan} = PB$

Within the population, each individual has a different structure responsible for different way of how it responds to its environment. The individuals are evaluated in the context of the whole logic 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 in such a way 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 squared distances of the USV from the target over all time steps. This squared distance is normalized due to the different initial distances of the intruder from the target in the test scenarios. 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 logic component with low-safety guarantee. The fitness function is as follows:

$$F = \frac{1}{T} \sum_{i=1}^{T} \left(\frac{d_i}{d_0}\right)^2 \tag{12.1}$$

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

The navigation controller is evaluated using a 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 amount of time steps which equals to 300 seconds in real time. The maximum speed of the USV is set to be 10% higher than the speed of the intruder, other properties of the vehicles are the same. The navigation plan is evaluated using all states of exception found within the activation distance of its corresponding SOE_{REP}. The evaluation lasts for a maximum amount of time steps which equals to 10 seconds in real time.

The individuals in the initial population mostly exhibit 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 12.1). The following evolutionary operators are used:

- Reproduction – copies one individual directly to the next generation without any modification. We use a strong elitism to propagate the best individual directly into the next generation. This makes sure that the best individual is not lost during the evolutionary process as a consequence of recombination.
- Mutation – we use three types of mutation operators: structural mutation of a randomly selected sub-tree, preventing bloat [30] by shrinking a randomly chosen sub-tree to a single node, and Gaussian mutation of chosen parameters.
- Crossover – randomly selects sub-trees from two input trees and swaps them.

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

The optimization of the generated navigation controller removes all branches of the code that have not been executed during evaluation scenarios. Moreover, each generated navigation plan is truncated to contain only the navigation commands that do not exceed the execution time of the plan. This effectively prevents bloat and generates cleaner code.

A detailed description of the functionality of all the operators used can be found in [2]. The control parameters of the evolutionary process used for evolution of the navigation controller and plans are summarized in Table 1.

Table 12.1 GP Parameters

Population size / number of generations	500 / 100 (controller) 50 / 20 (plan)
Crossover probability	0.84
Tournament size	2
Structure mutation probability	0.05
Elite set size	1
Shrink structure mutation probability	0.01
Min. and max. initial GP tree depth	3 and 6 (controller) 2 and 4 (plan)
Mutation probability of parameters of navigation commands	0.5
Maximum GP tree depth	50 (controller) 10 (plan)
Crossover probability	0.84

12.4 Computational Experiments

12.4.1 General Setup

In the test mission domain, 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 time as possible. We set up an experiment to compare the performance of the automatically

generated USV's logic for blocking to the USV's logic 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 40 seconds 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 USV's logic is evaluated in 800 evaluation runs to account for the intruder's deterministic behavior interspersed with random actions. Each evaluation run lasts for a maximum amount of time steps which equals to 300 seconds in real time. The dimension of the scene is 800 x 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 random deviation of 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 maximum time for the evaluation run is set to 5 minutes. The USV's maximum velocity is 10 m/s, while the intruder's maximum velocity is 9 m/s.

12.4.1.1 Experimental Protocol

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

Second, we manually implemented the USV's blocking logic against the hand coded intruder. This involved implementation of its main navigation controller, manually finding 49 states of failure and their corresponding states of exception, hand coding the logic for each state of exception, and overall evaluation. The logic was improved iteratively in the span of 3 weeks. Its overall size reached 500 lines of Lisp code.

Third, we used the mission planning system to automatically generate the USV's blocking logic using the hand coded intruder as the competitor. The activation distance parameter δ was set to 0.2 for all navigation plans. The representative state of failure $SOF_{REP} \in SF$ was found as a state with the minimal average distance to its 7 nearest neighboring states. The number of time steps τ for computation SOE from SOF was set to 150. In the current version of the approach, SOF is determined to be a state in which the intruder is closer to the target than the USV.

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

12.4.1.2 Intruder's Planning Logic

The hand-coded intruder's logic is represented as a single decision tree that contains standard navigation commands as well as their randomized versions. The partial non-determinism of the logic allows the intruder to use different actions in the same situations. The intruder repeatedly deceives the USV's logic by randomly taking actions in some specific situations so that the USV is not able to find a motion pattern in intruder's behavior that can be easily exploited for blocking.

The intruder's logic can be 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 larger 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 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 Figure 12.10a). 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 handles also similar type of situations when the USV is on the front right of the intruder and the target is on the right.

The third section is very similar to the second section with the exception that the USV is directly on the left or right side of the intruder (see Figure 12.10b). In these cases, the intruder does its best to deceive the USV. The intruder would often slow down rapidly trying to get an advantageous position, randomly proceed with a complete turn, or execute 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 the situations during which the intruder is positioned behind the USV inside the rear grey area as shown in Figure 12.10c. The larger distance of the intruder from the USV gives it opportunity to exploit the USV's tendency to over shoot a little in the process of blocking. In this case, if the USV has high velocity, the intruder's logic would suddenly slow it down and turn it 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 Figure 12.10d). Again, the intruder can rapidly slow down with probability 0.3.

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

The intruder's logic can modify the reach and range parameters of its virtual visibility sensor to control the balance between a safe and aggressive obstacle avoidance mechanism. For example, if the intruder wants to make an aggressive turn in a 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 obstacle avoidance behavior sensitivity is reduced for a short period of time so that the intruder can easily pass

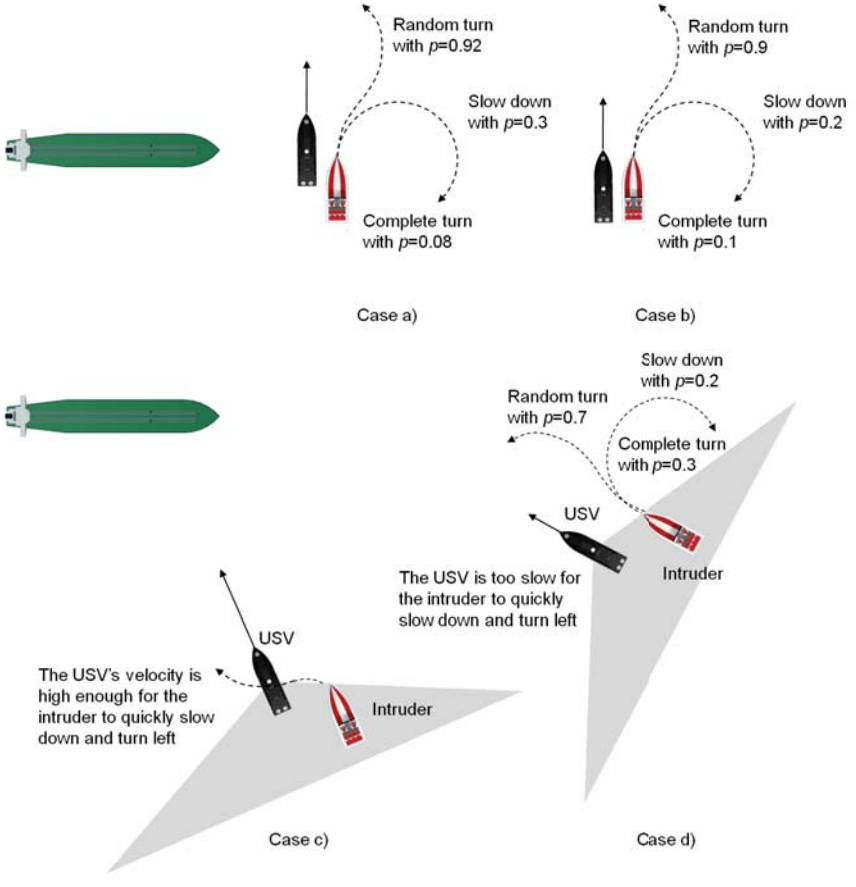

Fig. 12.10 Representative portions of intruder's planning logic.

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 pit against a human player.

12.4.2 Results

The experimental run that generated a blocking logic with the highest performance is shown in Figure 12.11. The horizontal axe of the graph shows different versions of the logic consisting of gradually increasing amount of navigation plans. The vertical axe shows the blocking performance in terms of the intruder's pure time delay for each version of the USV's logic. The best performance is reached by the version 30 of the logic and amounts to 42 seconds of pure delay in median. This can be

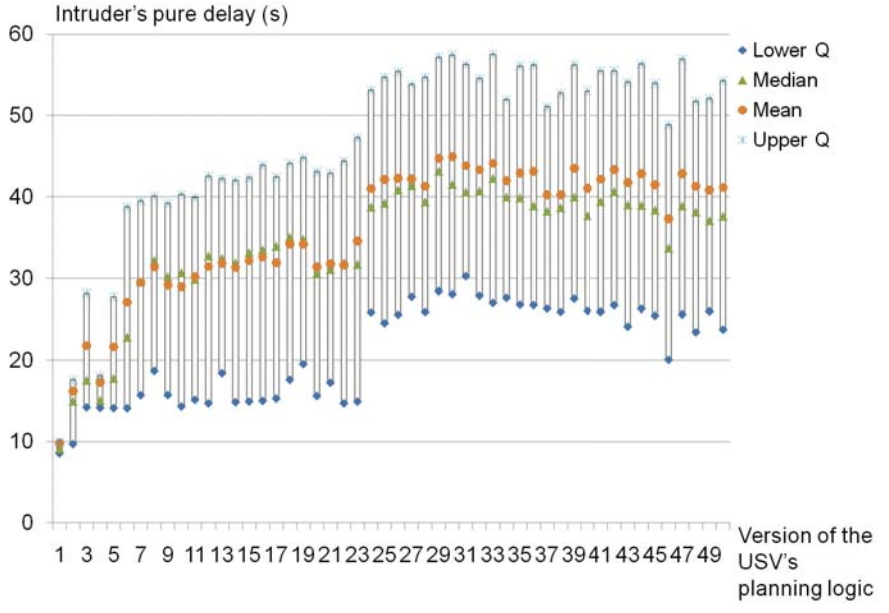

Fig. 12.11 Evaluation of the USV's blocking performance. The performance is expressed as a pure time delay applied on the intruder. Each version of the USV's logic was evaluated in 800 runs.

compared to the pure time delay of 46 seconds in median imposed by the hand coded USV on the same intruder. This result thus shows that the best performance of the automatically generated USV's logic closely approaches the blocking performance of the hand coded logic.

The automated generation of the logic took approximately 1 day to generate the main navigation controller and approximately 3 days on the average to generate navigation plans for 49 automatically defined states of exception. Its overall size reached 900 lines of code. From the set of 10 experimental runs, only 2 were able to find logic with the similar performance to the best one. The remaining 8 runs prematurely stagnated due to over-specialization of some of the evolved navigation plans. Even a single defective navigation plan synthesized for one of the key situations can significantly influence the performance of the whole logic. This shows that the learning of the USV's logic against the intruder utilizing attacking logic interspersed with randomized actions is a challenging task.

The first few versions of the logic have low performance as they contain only a few navigation plans describing specialized maneuvers for a small number of key situations. However, as the learning process progresses, more and more navigation plans handling new situations are added and the overall performance gradually improves. This continues until the version 30 of the logic after which the performance stagnates. This can be attributed to difficulty in solving new complex situations in which problems with the generalization of navigation plans arise.

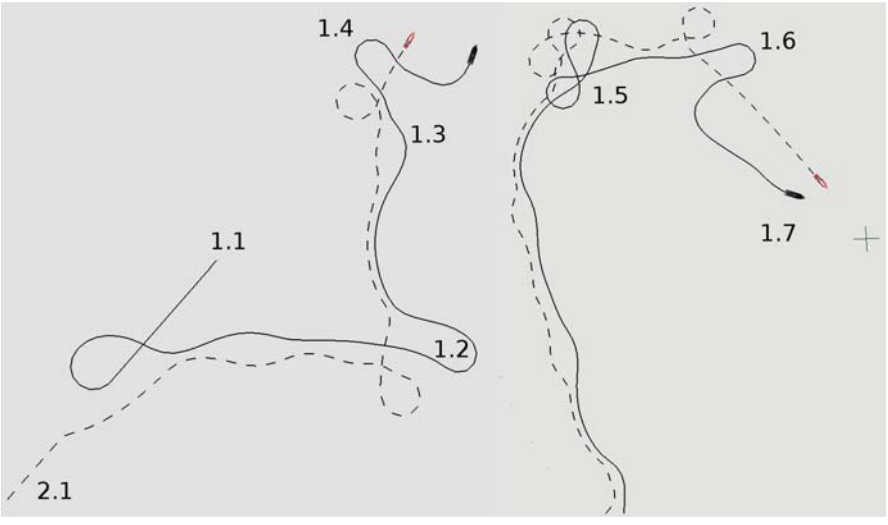

Fig. 12.12 Example of a run in which the USV managed to block the intruder for 45 seconds. The start position of the USV is marked as 1.1, while the start position of the intruder is marked as 2.1.

An example of a run in which the USV reached 45 seconds of pure time delay imposed on the intruder is shown in Figure 12.12. 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 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 to take another trial in intercepting 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 takes another attempt in intercepting the intruder but overshoots in 1.6 and the intruder finally reaches its goal 1.7.

12.5 Conclusions

We have presented a new approach for automated synthesis of a symbolic planning logic for an autonomous unmanned vehicle operating in an environment with a deceptive adversary. The idea behind this approach is to evolve an initial version of the logic first and then further improve its performance by evolving additional components that can reliably handle specific situations that can arise during the mission.

We used GP technique for automated generation of navigation plans for corresponding automatically extracted states of failure.

In the context of our test mission, we developed a mission planning system to automatically generate a planning logic for USV to block the advancement of an intruder boat toward a valuable target. The USV's logic consists of a navigation controller and multiple navigation plans describing specific maneuvers for specific situations. The intruder is human competitive and exhibits deceptive behavior so that the USV cannot exploit any regularity in its attacking tactic for blocking.

In our experiments, we compared the performance of the hand coded USV's blocking logic to the performance of the logic that was automatically generated. The results showed that the performance of the automatically generated USV's logic (42 seconds of pure delay in median) closely approaches the performance of the hand coded USV's logic (46 seconds). Both types of the USV's logic were evaluated against a human competitive intruder. Hence, the approach described in this chapter clearly demonstrates the viability of automatically synthesizing planning logic for autonomous unmanned vehicles in competitive environments with deceptive adversaries.

Acknowledgments

This research has been supported by Office of Naval Research and NSF Grants OCI-0636164 and CMMI-0727380. Opinions expressed here are those of the authors and do not necessarily reflect opinions of the sponsors.

References

[1] Lipson, H., Antonsson, E., Koza, J., Bentley, P., Michod, R., Alber, R., Rudolph, S., Andronache, V., Scheutz, M., Arzi-Gonczarowski, Z.: Computational synthesis: from basic building blocks to high level functionality, pp. 24–31 (2003)

[2] Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Dordrecht (2003)

[3] Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)

[4] Goldberg, D.: Genetic Algorithms in Search and Optimization. Addison-Wesley, Reading (1989)

[5] Floreano, D., Mattiussi, C.: Bio-inspired artificial intelligence: theories, methods, and technologies (2008)

[6] Barate, R., Manzanera, A.: Automatic Design of Vision-based Obstacle Avoidance Controllers using Genetic Programming. In: Monmarché, N., Talbi, E.-G., Collet, P., Schoenauer, M., Lutton, E. (eds.) EA 2007. LNCS, vol. 4926, pp. 25–36. Springer, Heidelberg (2008)

[7] Nehmzow, U.: Physically embedded genetic algorithm learning in multi-robot scenarios: The PEGA algorithm (2002)

[8] Dain, R.: Developing mobile robot wall-following algorithms using genetic programming. Applied Intelligence 8, 33–41 (1998)
[9] Dupuis, J., Parizeau, M.: Evolving a vision-based line-following robot controller, p. 75 (2006)
[10] Koza, J., Rice, J.: Automatic programming of robots using genetic programming, pp. 194–194 (1992)
[11] Gajda, P., Krawiec, K.: Evolving a vision-driven robot controller for real-world indoor navigation. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 184–193. Springer, Heidelberg (2008)
[12] Barlow, G., Oh, C.: Evolved Navigation Control for Unmanned Aerial Vehicles. In: Frontiers in Evolutionary Robotics, p. 596. I-Tech Education and Publishing, Vienna (2008)
[13] Haynes, T., Sen, S.: Evolving behavioral strategies in predators and prey. In: Weiss, G., Sen, S. (eds.) IJCAI-WS 1995. LNCS, vol. 1042, pp. 113–126. Springer, Heidelberg (1996)
[14] Buason, G., Bergfeldt, N., Ziemke, T.: Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments. Genetic Programming and Evolvable Machines 6, 25–51 (2005)
[15] Jaskowski, W., Krawiec, K., Wieloch, B.: Winning Ant Wars: Evolving a Human-Competitive Game Strategy Using Fitnessless Selection. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 13–24. Springer, Heidelberg (2008)
[16] Togelius, J., Burrow, P., Lucas, S.: Multi-Population Competitive Co-evolution of Car Racing Controllers, pp. 4043–4050 (2007)
[17] Doherty, D., O'Riordan, C.: Evolving Agent-Based Team Tactics for Combative Computer Games (2006)
[18] Richards, M., Whitley, D., Beveridge, J., Mytkowicz, T., Nguyen, D., Rome, D.: Evolving cooperative strategies for UAV teams, p. 1728 (2005)
[19] Shichel, Y., Ziserman, E., Sipper, M.: GP-Robocode: Using Genetic Programming to Evolve Robocode Players, pp. 143–154 (2005)
[20] Sipper, M., Azaria, Y., Hauptman, A., Shichel, Y.: Designing an Evolutionary Strategizing Machine for Game Playing and Beyond. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37, 583–593 (2007)
[21] Schwartz, M., Svec, P., Thakur, A., Gupta, S.K.: Evaluation of Automatically Generated Reactive Planning Logic for Unmanned Surface Vehicles. In: Performance Metrics for Intelligent Systems Workshop, Gaithersburg, MD (2009)
[22] Cornfield, S., Young, J.: Unmanned Surface Vehicles - Game Changing Technology for Naval Operations. In: Roberts, G.N., Sutton, R. (eds.) Advances in Unmanned Marine Vehicles, 69th edn., The Institution of Electrical Engineers, Stevenage, United Kingdom (2006)
[23] Finn, A., Scheding, S.: Developments and Challenges for Autonomous Unmanned Vehicles: A Compendium. Springer, Heidelberg (2010)
[24] Committee on Autonomous Vehicles in Support of Naval Operations, National Research Council. The National Academies Press, Autonomous Vehicles in Support of Naval Operations (2005)

[25] Thakur, A., Gupta, S.K.: 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 (2010)
[26] Svec, P., Schwartz, M., Thakur, A., Anand, D.K., Gupta, S.K.: A simulation based framework for discovering planning logic for Unmanned Surface Vehicles. In: ASME Engineering Systems Design and Analysis Conference, Istanbul, Turkey (2010)
[27] LaValle, S.: Filtering and Planning in Information Spaces (IROS tutorial notes) (2009)
[28] Brooks, R.A.: Intelligence Without Representation. Artificial Intelligence 47, 139–159 (1991)
[29] Brooks, R.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2, 14–23 (1986)
[30] Poli, R., Langdon, W., McPhee, N.: A Field Guide to Genetic Programming. Lulu Press (2008)