NTNU – Trondheim
Norwegian University of
Science and Technology

A Decentralized Architecture Using the Null-Space-Based Behavioral Control For Multi-Agent Systems with Application to Border Patrolling, Search and Retrieval

Håvard Schei

Master of Science in Computer Science
Submission date: June 2013
Supervisor: Pauline Haddow, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Håvard Schei

A Decentralized Architecture Using the Null-Space-Based Behavioral Control For Multi-Agent Systems with Application to Border Patrolling, Search and Retrieval

Master thesis, spring 2013

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

This report proposes a new method for searching and retrieving a dynamically changing border using robotic multi-agent systems as an extension to the patrolling task - the problem of coordinating autonomous agents patrolling a border. The proposed method is a decentralized and self-organizing system built in the framework of the Null-Space-Based behavioral control and is controlled by a Finite State supervisor. The system encourages cooperation among the agents without the use of explicit and direct communication and from the simple interactions between agents the system hopes to emerge a global behavior in order to help fulfill the patrolling task.

Sammendrag

Denne rapporten foreslår en ny metode for søk og gjenoppretting av et grenseområdet som dynamisk endrer seg ved hjelp av multiagent system for roboter, som en viderefring av patruljeringsproblemet - problemet ved koordinere selvgående, grensepatruljerende agenter. Den foreslltte metoden er et desentralisert og selvorganiserende system bygget i rammeverket av en NULL-rombasert atferdskontroll og er styrt av en Finite State-veileder. Systemet oppmuntrer til samarbeid mellom agentene uten å bruke eksplisitt og direkte kommunikasjon, og gjennom enkle interaksjoner mellom agentene så håper systemet å fremskaffe en global atferd for å hjelpe med å fullføre patruljeringsoppgaven.

Preface

This master thesis is written at the Department of Computer and Information Science at the Norwegian University of Science and Technology during the spring of 2013. The IDI faculty and professors at NTNU provide students with several different project options, which can be interpreted and built upon by students own creativity. This master thesis builds upon the specialization project of the fall of 2012 which involved building a robot capable for participating in multi-agent systems and swarms.

The specialization project required research on multi-agent agent systems and swarm intelligence, and it was there I found an interest in the patrolling problem for multi-agent system. This master thesis aims to build upon the previous work done in this field.

Finally, I want to thank:

- Professor Pauline C. Haddow for supervising this project, giving valuable feedback during the process.
- Jean-Marc Montanier for attending the supervising meetings and giving valuable feedback.

Håvard Schei

Trondheim, June 17, 2013

Contents

List of Figures

Chapter 1

Introduction

This chapter presents a basic overview of this thesis, starting with the motivation behind it, the research questions asked, and the research methodology used. Finally, an overview over the rest of the thesis is presented.

1.1 Background and Motivation

This thesis deals with the development of a behavioral control for swarm-inspired multi-agent systems tasked with the patrolling problem - the task of coordinating autonomous agents in patrolling a certain area, and is directly based on the works of Marino, Parker, Antonelli, *et al.* [1] which presents a behavioral control for decentralized multi-agent systems formulated in the Null-Space-Based (NSB) framework. Their particular work proposes a specific approach to the patrolling problem where certain assumptions are made about the environment the agents operate in, such as a strong knowledge of where the border is located and that this border is static in terms of shape, position and size. The motivation behind this thesis is to research the possibilities of creating a system that removes these assumptions, using the same methodology and techniques from the original work of Marino, Parker, Antonelli, *et al.* in order to further develop the proposed behavioral control.

1.2 Goal

As described in section 1.1 Background and Motivation, the work of this thesis is trying to extend an incomplete approach to the flocking problem as the original work lacks the capability of searching and retrieving a dynamic border. This extension should be developed using the same mechanisms and techniques the original is using in order to create an overall coherent system. This can be summed up in a single sentence:

Goal: To develop a behavioral control for multi-agent systems capable of *locating* and patrolling a *dynamic* border capable of executing the *patrolling task*.

To expand on this goal two research questions are provided to support the overall goal:

Research question 1: Is cooperation among agents beneficial in order to find or retrieve the border?

A key thing to verify when using cooperation is to investigate if it actually contribute to a more efficient or robust system.

Research question 2: The original solution uses no communication, is this trait possible to maintain when using cooperation?

1.3 Thesis Structure

Chapter 2 presents the state of the art. It briefly explores the field of multi-agent systems and further presents the patrolling task. Chapter 3 contains related background information researched with focus on the null-space-based technique. Chapter 4 presents the architecture implementation used in this thesis as well as its assumptions and requirements. Chapter 5 presents the experimentation setups and results as well as an analysis of each experiment result and a overall evaluation of the system. Chapter 6 concludes the thesis and presents some notes on future work.

Chapter 2

State of the Art

In the following section the key concepts of multi-agent systems are described and set in context with the patrolling task problem, exemplified with two recently proposed approaches for solving it.

2.1 Multi-Agent Systems

A multi-agent system (MAS) is a system consisting of interacting agents that are able to in some extent autonomously make decisions to achieve their goals. More importantly, they are able to interact with each other, enabling coordination, cooperation, communication and so forth. Such systems are often deployed in environments to accomplish goals that would not be achievable for a single agent alone. MASs are used in many different areas of AI research such as distributed problem solving, coordination and collaboration, organization, communication, and belief-desire-intension systems [2].

Agents in MASs possesses the following important characteristics [2]:

- Autonomy: Each agent makes their decisions independently.
- Awareness: The agents usually does not have a full view of the environment they are operating in, or the environment is too complex for the agent to make practical use of it.
- Decentralization: There is no central decision making entity controlling or suggesting actions to the agents in the system.

In this thesis the term multi-agent system will refer to robotic multi-agent systems, although agents in these systems has no explicit definition, any agent that holds the characteristics described above can be defined as an agent in a MAS. This means that a MAS can comprise of computers on the Internet, humans computer programs, robots, et cetera.

2.1.1 Cooperation

Task performed using MASs can either be cooperative or non-cooperative tasks. Cooperative tasks are tasks that can not be achieved by a single agent and thus needs multiple individuals interacting in order to complete the task. Examples of such tasks are formation, aggregation and heavy foraging [2]. Bonabeau, Dorigo, and Theraulaz [3] states that non-cooperative tasks are tasks that could be achieved single-handedly by an agent given unlimited time, but would be more efficient if multiple agent cooperated executing the task. Examples of such task are searching, retrieving and sorting. Cooperation is an essential trait of most multi-agent systems.

2.1.2 Communication

In order to have cooperation in MASs there must exist some form of communication between the agents. Communication be both direct and indirect. Direction communication consist of explicitly exchanging information between agents simultaneously when all participants are present. Indirect communication on the other hand, does not conform to these assumptions. Such communication can be anything from a specific behavior, force, symbolic elements or usage of time [4]. The communication type used in this thesis is the indirect one, based on the perceived behavior between agents in the system. This will be described in detail in chapter 4.

2.1.3 Self-Organized Systems

Self-organized systems are systems where large number of agents cooperate in order to produce a global behavior that emerges from the simple behavior between the cooperating agents. Agents perform actions based on input from the local environment, with no notion of its contribution to the overall global behavior. Such emergent behavior renders no need for any controlling entity, and adheres to any requirements of decentralized systems.

2.2 The Patrolling Task

The patrolling task is defined as the act of traversing around an area in order to supervise or protect it [5]. This task has in the later years gained much attention due to the wide range of potential areas of application in the real world. The traditional view of the patrolling task is one or more agents physically traversing the perimeter of a given area, with applications ranging from surveillance of forest fires [6] and oil spill monitoring [7][8] to the more classical task of border security [9][10]. Autonomous unmanned robotic applications have been developed for this task and deployed in the real world, ranging from surface vehicles [11], ground vehicles [12] and aerial vehicles [13].

It is easy to see that the patrolling task can be handled by multi-agents systems, and Machado, Ramalho, Zucker, *et al.* [5] as well as Almeida, Ramalho, Santana, *et al.* [14] presents an extensive analysis of common issues and different solutions to the patrolling task seen from an multi-agent perspective. Traits such as communication, sensors capabilities, coordination and decision-making schemes are evaluated under different criteria with respect to the different issues in the patrolling task. These issues involves the various objectives, conditions and constraints stemming from the different patrolling missions. Such conditions can be the kind and size of the border, the number of robots, sensors and communication capabilities. This makes it difficult to give an exact definition of the multi-agent border patrolling task. More analytical approaches are given in [15] [16] and [17], but such implementations are not always practical, as [16] points out in their work where a hostile entity are tasked with entering a multi-agent patrolled area with full knowledge of the algorithm used by the agents patrolling the area, making their actions predictable in the eyes of the hostile entity. Seen from another view, in [15] shows that approaches using purely random movements for patrolling multi-agent system are seldom effective.

2.2.1 No Communication Approach

In [1] a framework for handling the patrolling task for multi-agent systems is presented. This approach is concerned with the distribution of agents around a border and how they behave in order to patrol it using a decentralized architecture for a multi-agent system. The architecture is based on the *null-space-based* behavior control approach, explained in section 3. Since this approach is tasked with only patrolling the border, and not finding it, every agent in the experiment is assumed to know the position of the border in order to move towards it, although they know nothing off how the border is formed or the size of it. This

assumption is made since they are only concerned with how the agents behave when a border is located, and not with *how* they find it.

The approach uses a supervisor to select suitable *actions* which are composed of basic behaviors combined using the NSB-framework. The details of this architecture is presented in section 4.3. With regards to patrolling the border, the supervisor can select between three main modes of operation, each with a limited number of behaviors to be used. These modes are

- Border not visible. An agent cannot see the border, move closer to it. This mode uses the assumption of the existence of a border, and will move the agent closer to it.
- Border visible, but not in patrolling range. This mode handles the behaviors concerned of moving closer to a *visible* border.
- Border in patrolling range. This mode handles the different behaviors used when an agent is patrolling the border.

In addition to these three modes, a mode with the highest priority exists with the sole purpose of maintaining a minimum distance to certain other agents named *friend agents*, which are agents that are not tasked with anything but to walk independently around in the environment. The agents using this approach are fully reactive and uses no form of planning or communication. They are aware of other agents in the environment and uses this to only to avoid colliding with each other. The environment besides the agents is static so the border is non-changing and there are no other static obstacles.

The main contribution from this paper are the usage of the NSB framework to combine possible conflicting behaviors into actions selectable by the supervisor, allowing the developer to focus on the more emergent behavior rather than the low level mechanics of combining the different behaviors.

2.2.2 Communication Approach

In [1] the patrolling task is tackled using a decentralized, hybrid system of finite states for a multi-agent system. Unlike the approach discussed in the previous section, this approach is also concerned with finding and detecting the border as well as patrolling it. The agents are controlled individually by a hierarchical control using three behaviors, namely *searching*, *pursuing* and *tracking*. Each behavior handles their task implied by their names in addition to simultaneously handle obstacle avoidance. The state transition scheme can be seen in figure 2.1. To control the movement of an agent a potential field is generated out from the

positions and linear speeds of other agents in a given distance threshold leading or trailing the agent. An agent and its neighboring agents can be seen as an connected, undirected graph, denoted as communication graph, where they share information such as border positions to create their respective potential fields.

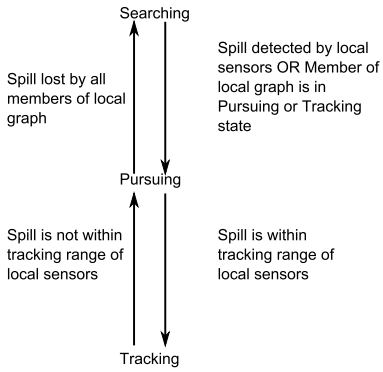


Figure 2.1: State transition diagram for the communication approach

State diagram showing the conditions for switching between the three states in the patrolling approach proposed in [1].

- Searching mode: When an agent is in the searching state and has detected other agents, a potential field is created based on the position of other agents in order to avoid collisions. If no agents are spotted the agent wanders randomly.
- Pursuing mode: When an agent detects a border, or one of the agents in the communication graph it switches to the pursuing state. A potential field is generated for the agent moving towards the position of the border. This potential field is added to a avoidance potential field, if any.
- Tracking mode: When an agent is in pursuing mode and the distance from the border is below a set threshold the agent switches to tracking mode and will move alongside with the border. If the communication graph of an agent is not empty, it will coordinate with its neighbors to maintain a speed so it does not collide or moves past other agents tracking the border.

As opposed to the architecture described in the section 2.2.1, the environment in which this framework is tested is much more dynamic where the experiments done shows that a set of agents are able to search, detect and track multiple borders whose position, shape and size as well as number change over time. The usage of communication graphs among the agents to propagate information is

intriguing, especially in the search phase.

Chapter 3

Background Theory

In this chapter the null-space-based (NSB) behavioral control is presented. The first section will provide a brief summary of existing behavioral approaches in MAS and how they compare to the NSB approach. Then in section 3.2.1 a mathematical background to the NSB approach is given and lastly then the NSB behavioral control is described in 3.2.2 and finally in section 3.2.3 an example with comparison to other behavioral controls is given.

3.1 Agent Behavioral Controls

For an agent acting in a highly uncertain and convoluted environment, it is a real challenge performing multiple missions simultaneously while adhering to the requirements for an autonomous agent [1], defined as:

- Responsiveness towards the environment. The agent should react to any changes that affects the agent directly or the goals of the agent within reasonable time, while adhering to the task at hand.
- Robustness. The agent must be able to handle unpredictable and unforeseen events and inaccurate sensor data.
- Reliability. Agents should operate with the same level of expected performance over time and should not degrade its capabilities.
- Intelligent behavior. An agent should show some degree of intelligence. An agent considering two paths towards a goal should choose the shortest one,

for example.

- Resolve multiple goals. The agent is expected to solve multiple goals. An agent meeting an obstacle in its way towards a goal position should maneuverer around the obstacle while at the same time moving towards the goal position.

Creating a static, predefined step-by-step plan would be a computational nightmare and result in a highly complex scheme, taking into account all the unpredictable events from the environment, for example collision avoidance. A solution is to abandon the idea of completely modeling the environment before deployment, and instead divide and group the missions tasks and the imposing tasks from the environment into sub-problems, where each sub-problem uses the sensors and the state of the agent to model the environment on-line, as suggested by the works of [18], [19] and [20]. The model should minimize the number of assumptions made about the environment and adapt to challenges from the environment.

These sub-problems - the decomposed problem - denoted as *behaviors*, then needs to be composed into one single command output to the agent, or in other words, the creation of a behavior best suited to the desire of the agent. Mechanism aimed at composing such behaviors are called *action selection mechanisms* (ASM). The challenge is how to generate an single command output from the different, and often conflicting behaviors, in a rational and coherent fashion. This is known as the *action selection problem*.

The action selection mechanisms are very different in how they combine the different behaviors into a single and final command output. Each behavior is designed independently from the others, with a single goal in mind. The output space of an action defines how many variables a behavior can manipulate in order to affect it. If different behavior tries to manipulate the same variable then there is a conflict between the behaviors.

Action selection mechanisms has traditionally been divided into two main categories [21], the arbitration approach and the command fusion approach. The next two subsection will elaborate on each category and given an example of an architecture originating from each category.

3.1.1 Arbitration Action Selection Mechanisms

Arbitration action selection mechanism are called competitive mechanisms because the different behaviors compete with each other to become the single behavior response output. In each selection cycle, arbitration ASMs selects *one*

behavior from a set of behaviors, and this behavior is given control of generating output for the system until the next selection cycle occurs. Arbitration mechanisms guarantees that there are no conflicting behaviors in the output command and produces a predictable output, but is prone to become an underused system in terms of not exploiting all degrees-of-freedom in the task output space. A good example representing the numerous different arbitration mechanisms is the Subsumption architecture:

Priority-Based Arbitration - The Subsumption Architecture

An example of an priority-based approach is the subsumption architecture [18] where the action selection process is a series of higher-level behaviors *subsuming* the lower-level behaviors. See figure 3.1 for an illustration of the architecture. The higher-level behaviors become increasingly more abstract as they are added, and sees the lower-level behaviors as preexisting competences available to utilize. Each layer of behavior works independently from the others, but higher-level behaviors can *subsume* lower-level behaviors, meaning they can suppress or inhibit the output from them, as well as use their output in calculating new outputs for themselves. For example, an object-following behavior can take into consideration the output from a lower-level obstacle-avoidance behavior. In this approach no supervisor is needed, since each layer can subsume the lower ones, and a layer is not subsumed it has the final and only say in controlling the actuators.

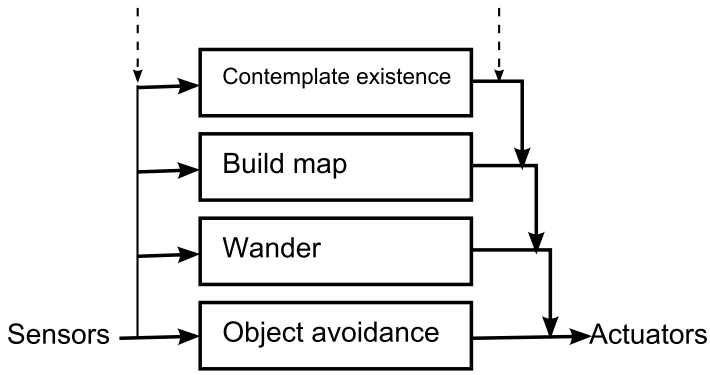


Figure 3.1: The subsumption architecture

3.1.2 Command Fusion Action Selection Mechanism

As opposed to *competitive* selection mechanisms described above, where only one behavior is selected, command fusion selection mechanism tries to combine the output of multiple behaviors into a single output. This approach is more capable to utilize all the available degrees-of-freedom in the task-output space, but is also very prone to produce an output that conflicts with one or more of the behavior it is produced from.

[21] identifies three steps that command fusion approaches use:

1. Action recommendations: Based on some behavioral criteria, some actions are more preferred than others.
2. Behavior aggregation: The recommended actions are combined based on some rule or approach.
3. Action selection: A fitting action is selected based on the recommendations from the aggregated behaviors.

A good example representing the numerous different command fusion mechanisms is the Subsumption architecture:

Superposition Command Fusion - The Motor Schema Architecture

The superposition approach is best described in the *motor schema* approach proposed by [19]. A set of behaviors is defined, where each behavior uses a *potential field* approach as proposed by [22] and [23]. In a potential field the goal objectives create attracting forces while obstacles create repulsive forces. So each behavior, based on their defined goals and obstacles, produces a field to traverse to to minimize the total potential. Traditionally, the whole environment is mapped to a potential field, but in the motor schema approach the potential field is only calculated where the agent is situated on-line. The output vector $\boldsymbol{r}_i$ from each of the N behaviors is then multiplied by a gain factor λ_i and then summed up to produce the final desired output command $\boldsymbol{v}_d = \sum_{i=1}^{N} \lambda_i \boldsymbol{r}_i$. The gain factors can be changed by some form of a supervisor to better adapt to a changing environment. The overall architecture is illustrated in figure 3.2

3.2 The Null-Space-Based Behavioral Approach

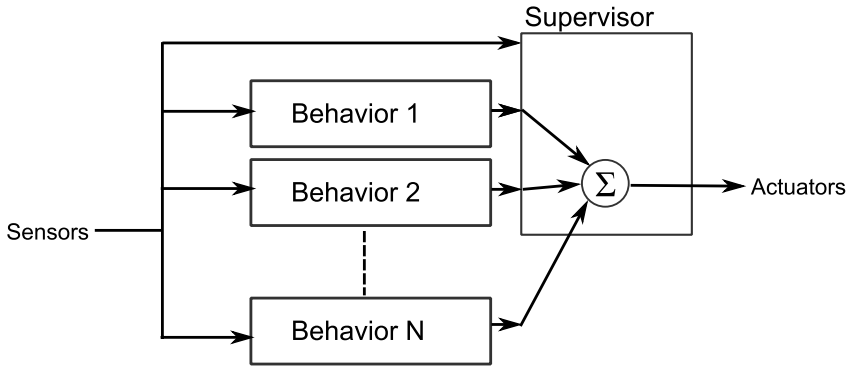


Figure 3.2: The motor schema architecture

The null-space-based behavioral approach can be seen as a compromise between the competitive (arbitration) and the cooperative (command fusion) approaches. Similar to the competitive approach, the different behaviors used by the NSB approach are prioritized, and as with the cooperative approach the prioritized behaviors are combined to create a final output. The key in this approach is *how* these prioritized behaviors are combined. To avoid conflicting behaviors, the prioritized behavior cuts off the conflicting components of the rest of the next prioritized behavior output. Details of this process will be elaborated in section 3.2.2.

3.2.1 Null-Space-Based Mathematical Background

In this section a mathematical background for the NSB calculations are presented in a general case.

Given configuration vector of a system:

$$\boldsymbol{p} \in \Re^{n\times 1}, \tag{3.1}$$

the task vector used to represent the task to be achieved by the system is:

$$\boldsymbol{\sigma} = \boldsymbol{f}(\boldsymbol{p}) \in \Re^{m\times 1}, \tag{3.2}$$

where $\boldsymbol{f}$ is a differentiable vector function with the following differential relationship:

$$\dot{\boldsymbol{\sigma}} = \frac{\delta \boldsymbol{f}(\boldsymbol{p})}{\delta \boldsymbol{p}} \dot{\boldsymbol{p}} = \boldsymbol{J}(\boldsymbol{p}) \dot{\boldsymbol{p}}, \tag{3.3}$$

where $\boldsymbol{J} \in \Re^{n \times m}$ is the configuration dependent Jacobian matrix. In order to produce new, desired configuration references $\dot{\boldsymbol{p}}_d$ for the system, the mapping of 3.3 is inverted:

$$\dot{\boldsymbol{p}}_d = \boldsymbol{K}(\boldsymbol{p_d}) \dot{\boldsymbol{\sigma}}_d, \tag{3.4}$$

where $\boldsymbol{K}$ is a control matrix dependent on $\boldsymbol{J}$, and the same size as $\boldsymbol{J}, (m \times n)$. Here, $\boldsymbol{K}$ is chosen so that 3.4 generates the minimum norm $\dot{\boldsymbol{p}}$ using the Moore-Penrose *pseudo-inverse* of the $\boldsymbol{J}$ as proposed by [24], which is defined as:

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^T (\boldsymbol{J}\boldsymbol{J}^T)^{-1}, \tag{3.5}$$

so that $\boldsymbol{K} = \boldsymbol{J}^{\dagger}$ and $\dot{\boldsymbol{p}}_d$ becomes $\dot{\boldsymbol{p}}_d = \boldsymbol{J}^{\dagger}(\boldsymbol{p_d}) \dot{\boldsymbol{\sigma}}_d$. Equation 3.4 is *open loop*, meaning that the configuration is continuously integrated over time to reach the new configuration. In order to counteract the numerical drift that occurs when doing this integration discreetly, a *closed loop* equation is used known as the *Closed Loop Inverse Kinematics* (CLICK) algorithm:

$$\dot{\boldsymbol{p}}_d = \boldsymbol{J}^{\dagger}(\dot{\boldsymbol{\sigma}}_{\boldsymbol{d}} + \boldsymbol{\Lambda} \tilde{\boldsymbol{\sigma}}), \tag{3.6}$$

where $\tilde{\boldsymbol{\sigma}} = \boldsymbol{\sigma_d} - \boldsymbol{\sigma}$ denotes the task error between the desired task and the actual task, and $\boldsymbol{\Lambda}$ is a positive constant diagonal *gain* matrix. Using discrete time the system looks like this:

$$\begin{cases} \dot{\boldsymbol{p}}_d(t_i) = \boldsymbol{J}^{\dagger}(\boldsymbol{p}_d(t_{i-1}))[\dot{\boldsymbol{\sigma}}_d(t_i) + \boldsymbol{\Lambda} \tilde{\boldsymbol{\sigma}}(t_{i-1})] \\ \boldsymbol{p}_d(t_i) = \boldsymbol{p}_d(t_{i-1}) + \dot{\boldsymbol{p}}_d(t_i) \Delta t, \end{cases} \tag{3.7}$$

If $n > m$ the system contains redundancy and the general solution $\dot{\boldsymbol{p}}_d$ contains a null-space projector matrix:

$$\boldsymbol{N} = (\boldsymbol{I_n} - \boldsymbol{J}^{\dagger}\boldsymbol{J}), \tag{3.8}$$

where $\boldsymbol{I}_n$ is the $(n \times n)$ Identity matrix. So $\dot{\boldsymbol{p}}_d$ now becomes

$$\dot{\boldsymbol{p}}_d = \boldsymbol{J}^{\dagger}(\dot{\boldsymbol{\sigma}}_{\boldsymbol{d}} + \boldsymbol{\Lambda} \tilde{\boldsymbol{\sigma}}) + \boldsymbol{N} \boldsymbol{v}_{null}, \tag{3.9}$$

where $\boldsymbol{v}_{null} \in \Re^n$ is an arbitrary vector. This means that the null-space projector matrix $\boldsymbol{N}$ removes any conflicting components from $\boldsymbol{v}_{null}$ and the remaining components are added to the redundant space of equation 3.6.

3.2.2 Null-Space-Based Behavioral Control

Considering a system where multiple different tasks are to be carried out on a configuration $\boldsymbol{p}$, it has to find a way to carry out most of them in a successful fashion. [25] proposes arranging them in a prioritized matter $[\boldsymbol{\sigma_1}, \boldsymbol{\sigma_2}, \ldots, \boldsymbol{\sigma_n}]$, and for each task project the next prioritized task onto the current task. Equation 3.9 shows that this can be done by replacing the $\boldsymbol{v}_{null,i}$ of a given, prioritized task i with the $\dot{\boldsymbol{p}}_{d,i+1}$ of the next prioritized task $i+1$. This means that the configuration reference for task $i+1$ are only added to the redundant space of the configuration reference of task i. In more general form:

$$\dot{\boldsymbol{p}}^{(i)} = \dot{\boldsymbol{p}}_{d,i} + \boldsymbol{N}_i \dot{\boldsymbol{p}}^{(i+1)}, \quad i = 1, 2, \ldots, N, \tag{3.10}$$

where $\dot{\boldsymbol{p}}_d = \dot{\boldsymbol{p}}^{(i)}$ and $\dot{\boldsymbol{p}}^{(n+1)} = 0$.

The number of task using in a null-space-based approach should not exceed the degrees of freedom for the configuration since the lowest prioritized task would not contribute to the final output. Considering a system configuration with n degrees-of-freedom and a task with a Jacobian that has a rank r. The available degrees-of-freedom for the next task is then $n - r$. Assuming that all task contribute to the final reference configuration, the maximum number of tasks would then be the sum of the ranks of the task Jacobians up to n:

$$\sum_{i=1}^{N} r_i \leq n, \tag{3.11}$$

A general stability analysis of the null-space-based behavior control is done in [26], and concludes with the following:

- **Two tasks:** The system is stable when the two tasks are at least independent.
- **Three tasks:** Such a system must have an orthogonality condition between two following tasks, while the remaining task needs to be independent.
- N **tasks:** For the general case of N tasks, no simple solution exists.

3.2.3 Null-Space-Based Example and Comparison

The NSB behavioral control approach can be shown in the context of a robot moving in a 3-dimensional space with all three degrees-of-freedom available. The configuration of the system would be the position of the robot:

$$\boldsymbol{p} = [x \quad y \quad z]^T \in \Re^{3\times 1}, \tag{3.12}$$

and the velocity of the robot is then:

$$\boldsymbol{v} = \dot{\boldsymbol{p}} = [\dot{x} \quad \dot{y} \quad \dot{z}]^T \in \Re^{3\times 1}, \tag{3.13}$$

Consider three tasks the robot should perform, and their corresponding velocities $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, $\boldsymbol{v_3}$. For the sake of simplicity, the task velocities contains at least one degree of redundancy each and none of them are equal:

$$\begin{aligned} \boldsymbol{v}_1 &= [0 \quad 0 \quad k_{1z}]^T \in \Re^{3\times 1} \\ \boldsymbol{v}_2 &= [0 \quad k_{2y} \quad k_{2z}]^T \in \Re^{3\times 1} \\ \boldsymbol{v}_3 &= [k_{3x} \quad 0 \quad k_{3z}]^T \in \Re^{3\times 1} \end{aligned} \tag{3.14}$$

In case of controls schemes such as the NSB control or other controls schemes where the tasks are evaluated in a prioritized manner, the tasks in this example are prioritized after their number, where the lowest number is the highest prioritized task. The task velocities are shown in figure 3.3

Using the NSB scheme approach

Using the null-space-based approach to calculate the desired velocity $\boldsymbol{v}_d$ from the tree tasks, equation 3.10 is used:

$$\boldsymbol{v}_d = \boldsymbol{v}_1 + \boldsymbol{N}_1(\boldsymbol{v}_2 + \boldsymbol{N}_2(\boldsymbol{v}_3 + \boldsymbol{N}_3 0)), \tag{3.15}$$

Here, $\boldsymbol{v}_3$ is projected onto the null-space of $\boldsymbol{v}_2$ removing any conflicting terms from $\boldsymbol{v}_3$ and adds it to $\boldsymbol{v}_2$ and is denoted as $\boldsymbol{v}^2$. This is then again projected onto the null-space of $\boldsymbol{v}_1$, again removing conflicting terms from $\boldsymbol{v}^2$ and added to $\boldsymbol{v}_1$ which is denoted as $\boldsymbol{v}^1 = \boldsymbol{v}_d$. The gains of the task here are for the sake of simplicity set to the identity matrix $\boldsymbol{I}_3$ These steps are illustrated in figure 3.4.

In comparison to the NSB approach, a layered-control approach, exemplified in the subsumption architecture described in section 3.1.1 is shown in figure 3.5

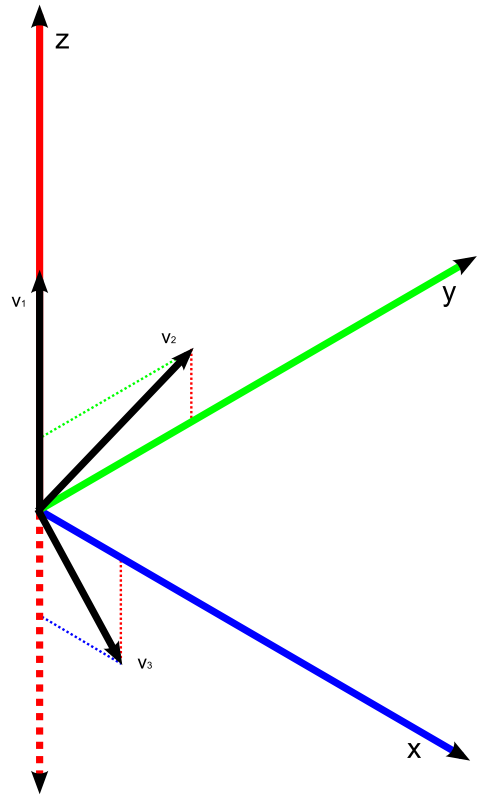


Figure 3.3: Three task velocities
The three task velocities numbered after their priorities where 1 is the highest prioritized task

where the final output is the task with the highest priority: $\boldsymbol{v}_d = \boldsymbol{v}_1$. This is a simple form of a competitive approach.

A second comparison of the NSB approach is made against a weighted control scheme, exemplified in the motor schema approach described in section **??**, which is a cooperative approach. All tasks are given a weight of $\alpha = 1$ so the final output velocity becomes $\boldsymbol{v}_d = \sum_{i=1}^{3} \alpha \boldsymbol{v}_i$. This is illustrated in figure3.6.

A more comprehensive case study between the null-space-based control system, the layered control system and a weighted control system can be found in [27].

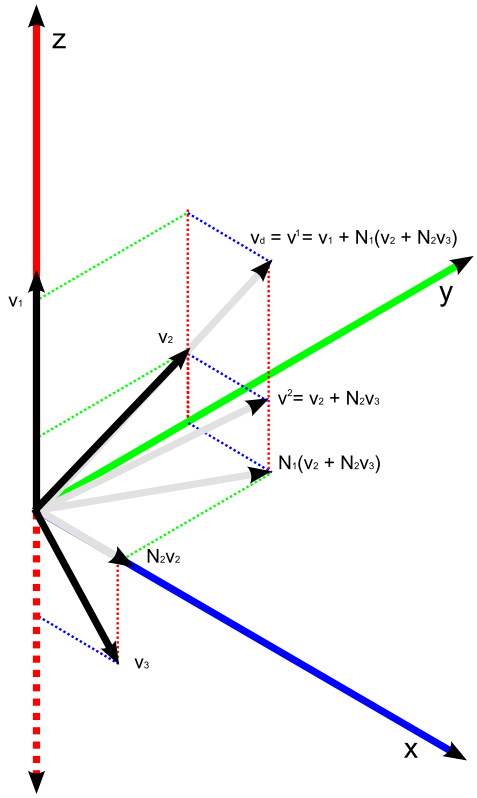


Figure 3.4: NSB three task decomposition

The task velocities are projected onto the null-spaces of their higher prioritized task velocities in an iterative manner. These projections are shown as gray arrows and labeled accordingly. $\boldsymbol{v}_3$ is projected onto the null-space of $\boldsymbol{v}_2$, denoted $\boldsymbol{N}_2$ and is added to $\boldsymbol{v}_2$ and results in $\boldsymbol{v}^2$, which is then projected onto the null-space of $\boldsymbol{v}_1$, denoted $\boldsymbol{N}_1$ and added to $\boldsymbol{v}_1$ which is denoted $\boldsymbol{v}^1$ and is the last task projection so $\boldsymbol{v}^1 = \boldsymbol{v}_d$

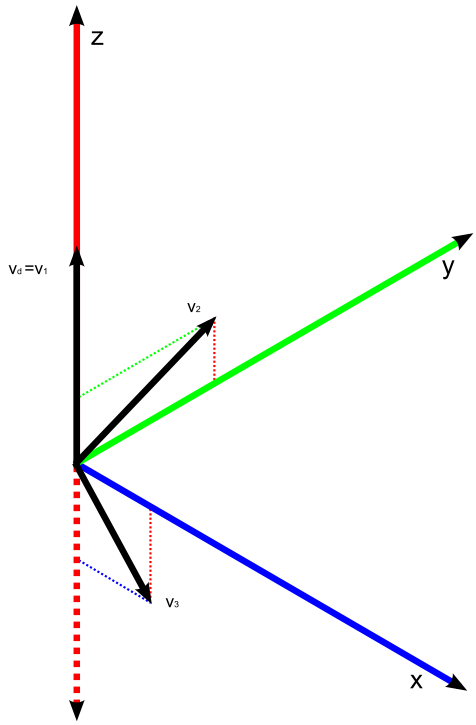


Figure 3.5: Layered task decomposition

In a simple layered control scheme the task with the highest priority is selected and here the final output becomes $\boldsymbol{v}_d = \boldsymbol{v}_1$.

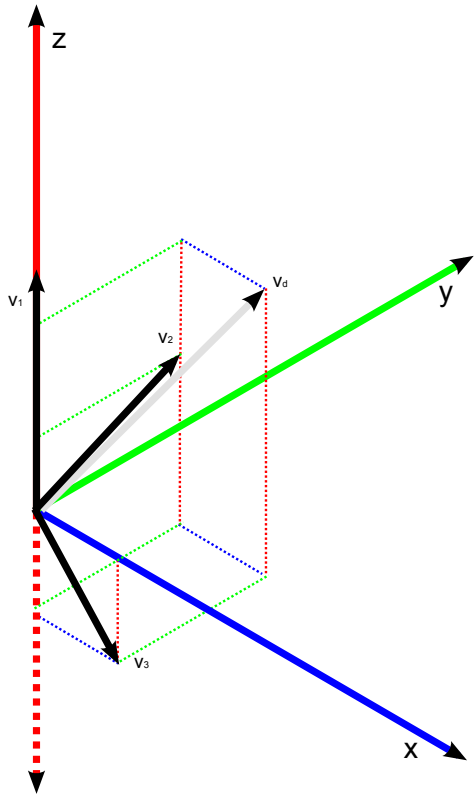


Figure 3.6: Weighted control scheme decomposition

In a weighted control scheme the outputs of the tasks are weighted and combined to produce the final velocity $\boldsymbol{v}_d$. In this particularly case, the output from the NSB approach is similar to the output from the weighted approach since there are few conflicting terms in the different velocities in the tasks.

Chapter 4

Model and Implementation

This chapter presents the proposed module and architecture for extending an existing approach that partially performs the patrolling problem described in section 2.2. Some background information and techniques use are also presented in order to present a coherent design overview. Lastly, in section 4.7 the simulator developed is presented.

4.1 Extended No Communication Approach

The main focus of this thesis is focused on how the architecture described in 2.2.1 can be extended to support a more dynamic environment, i.e. where the border to be patrolled is changing in size, shape and number over time, not unlike how borders the borders of oil spills and forest fires changes in nature. This means that the architecture needs to support a new search module for locating borders, and it should be defined in the same manner as the existing modules in order to create a coherent system. This is needed since the agents in that architecture always has a notion of where the border is in the environment relatively to themselves. Changing the border to be dynamic also imposes new challenges for the system. Cases where the border is changing to rapid for patrolling agents to keep up should be handled. This new module will be outlined in this section, as well as the new modifications made to the environment.

4.1.1 Requirements

The architecture this new module should be added to is deeply anchored to the null-space-based framework where each *action* is composed of prioritized behaviors combined in a non-conflicting way, as described in section 3.2.2. This new search mode should be constructed in the same fashion. This imposes some requirements for the new module:

- Behaviors should be properly mathematically defined in the null-space-based framework.
- Behaviors should be relatively simple and utilize the same set of sensors and actuators as the existing modules in the architecture.
- Actions should be able to be combined from simple behaviors.
- There should emerge a cooperative search behavior for agents using the actions.
- There should be no communication between the agents in the modules, as there are no communication in the existing modules.
- The new module should not interfere with the existing modules, i.e. it should only replace the *border not visible* mode. The rest of the architecture should be left intact.

4.1.2 Assumptions

The following assumptions are used by each agent when developing the framework for solving the dynamic border patrol task. In order to recreate the environment used in the *border patrol* task used in 2.2.1, the assumptions made here are based on those made in that work and modified to reflect the changes made to the environment in terms of the dynamic border.

Decentralized

No central decision making entity handing out commands to the agents exists. Each agent is autonomous, meaning that it only relies on the on-board computing capabilities to make independent decisions on its own, leading to a self-organized system.

Emergent behavior

The patrolling, localization and searching for the border should arise from relative simple actions originating from each agent. These group behaviors are emergent from the independent decisions from the collective of agents acting in the environment and not explicitly described in the behaviors of the agents.

Communication

No direct communication is used between the agents to reason about the task or to coordinate. This is a typical trait by self-organized swarm systems found in nature, such as bee swarms and ant colonies. Instead, indirect communication is more suited since it caters better for systems with volatile numbers of individuals trying to simultaneously perform tasks. In this approach, stigmergy is used to stimulate different behaviors among the agents. Using direct communication would also mean a more complex interaction procedure between the agents, resulting in more failure prone scheme and thus a less robust system.

Awareness

An agent can distinguish other agents from the border or other obstacles, but this notion is only of a discreet matter. I.e. an agent can ask *Is that* ***a*** *teammate?* but not *is that* ***the*** *teammate?* Further, it does not know if there exists other agents, obstacles or even a border to patrol, but the assumptions is that they *could* exist.

Localization

An agent has no notion of where in the environment it is situated. It only evaluates the positions of other visible objects, such as other agents, obstacles and the border, relative to itself. Since the system is assumed to be decentralized, each agent is then only reliable on its own sensors.

Visibility Range

In order to model the capabilities of real robot sensors, a threshold is used to cull objects that are within a distance threshold to an agent, to be visible. The visibility area for an agent here defined as a circle around an agent with using the threshold values as radius.

Safety Area

Each agent has a area defined around itself where no other agents or obstacles are allowed to enter. This area is used to avoid collision among the agents and obstacles in the environment. As with the visibility range, this area is defined as a circle around the agent.

Border Representation

The border is assumed to be a connected subspace in $\Re^2$ visible to the sensors of the agents in the environment. The representation of it is a set of vertices connected by two edges, forming an undirected graph. This way the agents can measure the distance from the border measuring the distance from a point down to a line. The border can also modify itself, changing in shape, size and numbers.

4.2 Search Module

When an agent is situated in the environment and has detect no border, it should clearly start searching for it. Since the agents uses no form of memory or has any notion of how the environment is designed, a random wander behavior is then best suited for searching in arbitrary locations. In the approach described in section 2.2.2 agents close to each other exchange information about where the borders are if any of them has detected one, and uses this to collectively move towards it. This way the agents does not necessarily need to locate the border themselves, it is enough that other teammates nearby find it and communicates it to their neighboring agents. Inspired by this approach where the agents cooperate to find the border, a non-communicative approach is proposed to help agents close to each other to collectively find the border.

This new approach is based on the work presented in [28] where they approach the flocking problem for multi-agent systems, consisting of grouping a set of agents together. This approach is all ready defined in the null-space-based framework, and will be briefly presented in section 4.2.1 in order to establish the groundwork the search module proposed in this thesis is built on. The new, proposed approach based on that paper will presented in section 4.2.2.

4.2.1 Flocking Using the NSB Approach

The flocking problem is here referred to the problem of making agents, using only local information, reach a particular configuration structure. Here this structure is a *lattice* structure, which can be seen as a connected, undirected graph where the nodes are equally distanced from their neighbors. In this scenario the agents represents the nodes, and to produce the lattice structure they should position themselves accordingly so that their distances to their neighboring is equal to each other. This structure should emerge only from the local interactions between the agents, and following the requirements and assumptions made in the previous section, it is clear that the agents should manage this task without communication. Using only their sensors, they can perceive other agents relatively to themselves and this is enough to generate fitting motion outputs to create the lattice graph.

Formation Structure Definition

The lattice graph can be defined using graph basic graph theory, following the notions used in [28]. A graph G consist of a set of vertices $\nu = \{1, 2, \ldots, n\}$ and a set of edges $\varepsilon \subseteq \{(i,j) : i,j \in \varepsilon, j \neq i\}$. The graph G is undirected so $(i,j) \in \varepsilon \implies (j,i) \in \varepsilon$. The adjacency matrix $\boldsymbol{A} \in \Re^{n \times m}$ holds the information between edges, so that $a_{i,j} \neq 0 \iff (i,j) \in \varepsilon$, and since the graph is undirected: $\boldsymbol{A} = \boldsymbol{A}^T$. The neighbors for a generic node $i \in \nu$ can then be defined as:

$$N_i = \{j \in \nu : a_{i,j} \neq 0\} = \{j \in \nu : (i,j) \in \varepsilon\}, \tag{4.1}$$

The nodes of the graph is here represented by the positions of the agents, and can be expressed by the vector $\boldsymbol{p} = [\boldsymbol{p}_1^T \quad \boldsymbol{p}_2^T \quad \ldots \boldsymbol{p}_n^T]^T \in \Re^{2n}$. The lattice structure can be defined as a pair $(G, \boldsymbol{p})$, where an edge between two agents ($a_{i,j} \neq 0$) is defined if their Euclidean distance is smaller than a threshold value d which is defined as the *lattice scale*. The definition for the neighbors N_i for a generic position $i \in \nu$ can now be described as:

$$N_i = \{j \in \nu :\| \boldsymbol{p}_i - \boldsymbol{p}_j \| < d, j \neq i\} \tag{4.2}$$

So for a structure to be considered as a lattice configuration it need to satisfy:

$$-\delta \leq \| \boldsymbol{p}_i - \boldsymbol{p}_j \| - d \leq \delta, \quad \forall_{i,j} \in \varepsilon(\boldsymbol{p}), \tag{4.3}$$

where $\delta \in \Re^+$ is a tolerance value. A figure depicting a arbitrary lattice structure can be seen in figure 4.1. The flocking problem is then solved by finding a velocity $\boldsymbol{v}$ for each agent so that they form a lattice structure with a desired scale d within a given tolerance δ. This motion command for a generic agent i against another agent is defined in 4.4.

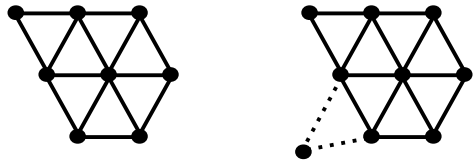


Figure 4.1: The lattice structure

A lattice configuration between 8 agents (left), and 9 agents are in a lattice formation where one agent is considered in the structured with a distance value $d_i \neq d$ but $d_i - d \leq \delta$.

Flocking Task Management

In addition to just flock around each other, the agents in [28] are tasked with moving towards a given point (rendezvous behavior) and generic obstacle avoidance. These behavior are defined in section 4.4. Using the null-space-based framework it is possible to combine the lattice flocking behavior in addition to another task. The lattice behavior is only defined as moving in one dimension, so it leaves on degree-of-freedom for another task to be projected onto it without creating a conflict. The same condition holds for obstacle avoidance, while the rendezvous behavior can utilize both degrees-of-freedom. The proper actions for combining the flocking behavior and the rendezvous and avoidance behaviors can be found in section 4.5.2.

The *Lattice Formation Actions* are of course tasked with positioning the agents accordingly towards one or more agent. The *Move to Rendezvous Point Action* is combined using the lattice flocking behavior as top priority since the rendezvous point behavior can take up both degrees-of-freedom and potentially override the lattice structure. This way the rendezvous behavior is potentially only partial cut off using the null-space projection from the lattice formation behavior. With the *Avoid Obstacle Action* the avoidance behavior is prioritized to ensure that no collision occurs, the lattice behavior is less likely to be cut off since both behaviors in this action only use one degree-of-freedom each.

In a scenario where agents should move towards a given point and at the same

time keep the lattice formation structure and avoid colliding with obstacles in the environment the control structure for an agent would change between the three actions described above in a hierarchical way where the avoidance action is top prioritized and the rendezvous behavior is number two. The selection process can be seen in figure 4.2.

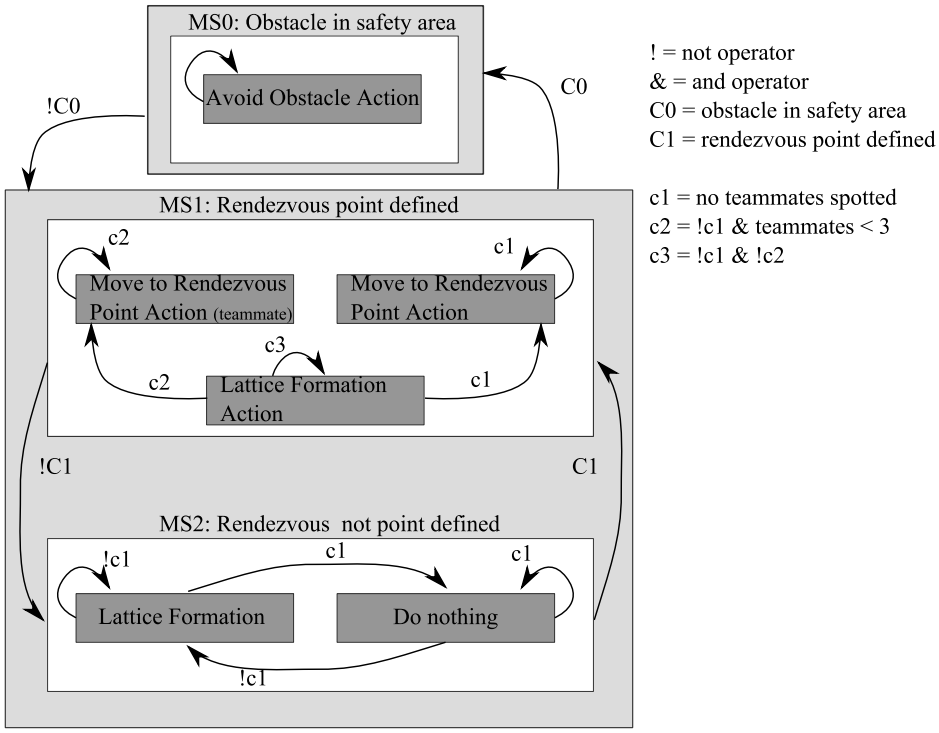


Figure 4.2: Flocking mission supervisor
The supervisor for group of flocking agents moving towards a target and avoiding obstacles.

4.2.2 Search Using Flocking

In the experiments presented in the flocking approach in [28], the agents moving towards a rendezvous point while maintaining the lattice structure and avoiding objects, has the rendezvous point predefined in the system, i.e. they know the position of the point relatively to their own position in the system, and can move towards it even if it is out of the range of their sensors. This assumptions is made

since they want to verify the flocking behavior for agents moving towards a point. In the approach used in this thesis, presented in section 4.1, this assumption would not hold since the agents does not have any notion of where they are, and it would also break the reactive nature of the architecture.

Border Aggregation

The approach proposed and used here involves that flocking agents should react to movements in the lattice structure. When a group of agents has created a lattice structure between themselves, the structure is a *stable state* if there is no movement in the structure when no border or obstacles are in the interaction range of any of them. If an obstacle is discovered the agents should react as proposed in the original flocking approach - avoid the obstacle while maintaining the formation. If one or more agents in the lattice structure discovers the border they disregard the formation behavior and move towards the border to start patrolling it. The agents that are neighbors to the leaving agents, denoted leader agents, and that has not detected the border themselves, notices this change and will start to move closer to the leader agents in order to try and maintain the lattice formation structure. At one point the distance between the leader agents and the neighboring agents will increase out of a set formation distance threshold, and the neighboring agents will stop trying to maintain the formation structure with the leader agents and instead directly follow them while trying to maintain the formation with the other agents still in the lattice structure. This behavior will then propagate throughout the lattice structure since the agents following the leader agents will themselves come out of the formation distance threshold and will be followed by the remaining agents in the formation. The consequence of this is that the whole lattice structure will follow any movement changes in the original structure, and in the case when an agents in such a formation detects a border it will move towards it and the rest of the structure will try to follow it, leading to more agents detecting the border, see figure 4.3. This way a *non-communicative* approach allowing a group of agents to discover a border emerges, not unlike how the *communicative* approach described in section 2.2.2 results in a collective of agents locating the border.

Exploring

In the case where a set of agents in a lattice structure is in a stable state, as discussed above, and none of them is in range of detecting a border, they should have some notion of moving the formation structure somehow in order to explore the environment in search of the border. Using the same mechanism discussed

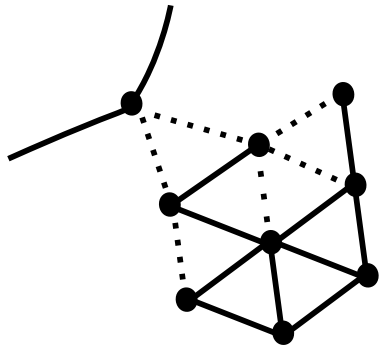


Figure 4.3: Lattice border aggregation

An agent belonging to a lattice structure has detected the border and starts patrolling it. The dotted lines indicates that the agents are out of range in terms of flocking towards the lattice structure and are being followed instead.

in the section above, where the whole lattice structure follows any change to the structure, an agent is selected by some probability to abandon the flocking behavior and instead start using a wander behavior. This results in the rest of the flock trying to follow this agent. The criteria for an agent becoming a leader is as follows:

- The lattice structure has to be stable, i.e. no motion in the neighboring agents is detected.
- No teammates or obstacles can be in the *safety area* of the agent. The safety area is a predefined distance that an agent has to any object in the environment. If something is in this area, the agent will move away from it.
- The agent can not be following any other agents. If an agent is following another agent that is a leader, it means that parts of the lattice structure is in motion.
- If the conditions above holds then there is a probability of p_{leader} that the agent is selected as leader.

If the agent is selected as leader at time $t(i)$ the conditions for it being leader at time $t(i+1)$ is:

- No border has been detected. If an border is detected, the agent is still seen as a leader by his neighbors since he is now in a patrolling state.

- The number of neighboring agents at time $t(i+1)$ must be the same as the previous times step $t(i)$. If an agent detects fewer neighbors it means that it either is moving to fast or there are other agents in the lattice structure pulling in another direction. The latter case could either be that another agent is chosen as leader or that the border has been detected. In any case, the leader agent should stop wandering about and go back to the formation. Multiple leader in a structure can be selected as leaders in the same period of time when they are not direct neighbors with each other. When an agent at one side of the structure starts moving, the propagation of the other agents takes some time to reach all the agents, so it is possible for other agents to meet the criteria for becoming a leader. This effect is highly dependent on the value of p_{leader}. If it detects more neighbors it means that it either has moved into another group of agents and should prioritize flocking with them, or that he has wandered into his own lattice structure.

Border recovery

Since the border is dynamic, the shape and size of may change at any time. Patrolling agents should be able to recover a lost border in an effective way. This scenario occurs when the border is changing too rapidly for the agents to track and patrol. Following the previous assumptions about the environment and the agents, they have no notion of where the border has moved and should start searching for it. Instead of each agent searching independently, they should start go into the exploring phase and start flocking. This behavior can be seen in figure 4.4.

4.3 Architecture

The architecture described in section 2.2.1 is a decentralized approach used by each agent to select a proper output for the actuators on the agents. It is a light-weight and straightforward architecture which promotes emergent behavior from combinations of simple behaviors combined using the null-space-based approach.

The architecture can be divided into three layers, see figure 4.5, *the agent layer*, *the action layer* and *the supervisor layer* starting from bottom to the top. The agent layer comprises of the sensors and the actuators of the agent, where the control loop starts and ends respectively. Starting from the sensors in the agent layer, their data are sent to the supervisor layer. Based on the input data the

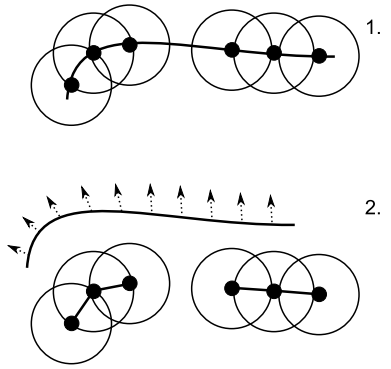


Figure 4.4: Lattice border lost

At 1 a set of agents are patrolling the border. At 2 the border is changing in a pace to fast for the agents to follow and they lose track of it and starts flocking in stead.

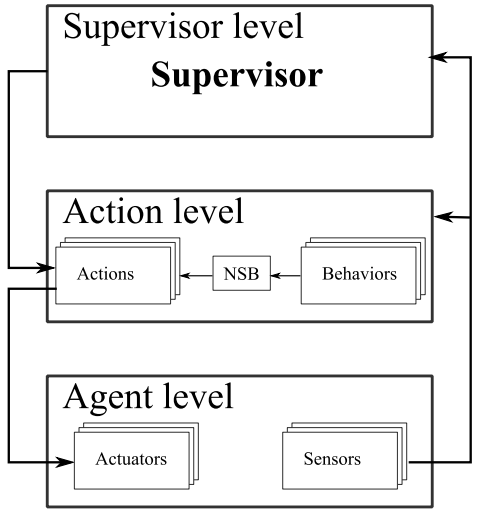


Figure 4.5: The overall architecture

The three layered architecture: The supervisor selects a proper action based on sensor data. The output generated by the action is then sent to the actuators of the agent. An action is combined by a hierarchical set of behaviors using the null-space-based approach.

supervisor selects an appropriate action from the action level. The selected action level then acts on the actuator in the agent layer. Clearly, the action and supervisor layer are yet to be defined, and can be done so in different ways. For example, the action layer can use one of the approaches described in 3.1, while the supervisor can, among other, use a fuzzy logic approach as described in [29] or as in [1] use a finite state automata approach. The latter approach will be used in this thesis.

4.4 Behaviors Definition

In this section the elementary behaviors used by the *actions* in the architecture is mathematically defined in the framework of the null-space-based approach, unless noted otherwise. Only four basic, elementary behaviors are defined, namely *maintain distance, reach position, move perpendicular* and *wander*.

Maintain distance behavior

This first behavior describes a critical part of the agents behavioral repertoire which is maintaining a certain distance from a given point. In this context it is use extensively for avoiding getting too close to obstacles, *friend agents* or other agents, as well as formation creation such as maintaining a constant distance to other agents. All these scenarios uses the same mathematical definition, but uses different values of task gains Λ, task values σ, and desired task values σ_d. The avoidance behavior is consequently only activated when the task value σ is below the σ_d, otherwise the behavior would produce an output moving the agent *closer* to the object it tries to avoid. The formal and generic definition is, given a point to avoid $\boldsymbol{p}$ given in the *local space* of the agent:

$$\text{Generic behavior} \begin{cases} \sigma = \|\boldsymbol{p}\|, \sigma_d = d, \\ \boldsymbol{J} = \boldsymbol{r}^T, \boldsymbol{J}^\dagger = \boldsymbol{r}, \boldsymbol{N} = \boldsymbol{I_2} - \boldsymbol{r}\boldsymbol{r}^T, \\ \boldsymbol{v_d} = \Lambda \boldsymbol{r}(d - \sigma), \end{cases} \tag{4.4}$$

and

$$\boldsymbol{r} = \frac{-\boldsymbol{p}}{\|\boldsymbol{p}\|} \tag{4.5}$$

Where d is the desired task value of the behavior function, and $\boldsymbol{J}$ is the task Jacobian,$\boldsymbol{I}_2$ is the identity matrix $\boldsymbol{N}$ is the null-space projection matrix and λ is

a scalar gain that is always positive. $\boldsymbol{v_d}$ is the desired velocity of the agent from this generic task given the value $\boldsymbol{p}$. See figure 4.7 for an illustration.

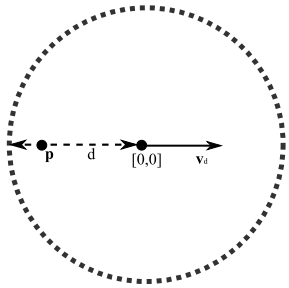


Figure 4.6: Generic avoidance behavior

The point $\boldsymbol{p}$ is the point a given agent should maintain a given distance d to. If the distance is above the value d it should move towards it, if it is under it should move away, as seen in the illustration. The d effectively acts as the radius of a no-entry zone when the behavior is used for avoidance. $\boldsymbol{v}_d$ is the generated output from the behavior.

It is now possible to define the avoidance behaviors for the two specific avoid task used in this mission, namely avoid *friend agent* and avoid collisions in general. Avoiding teammate has its own behavior defined since the agents in this mission has a lower threshold for avoiding *friend agents* than the threshold for avoiding collisions. So the definition for *friend* avoidance behavior becomes:

$$\text{Avoid friend agent} \begin{cases} \sigma_{af} = \|\boldsymbol{p_f}\|, \sigma_{af,d} = d_{af}, \\ \boldsymbol{J}_{af} = \boldsymbol{r}_{af}^T, \boldsymbol{J}_{af}^\dagger = \boldsymbol{r}_{af}, \boldsymbol{N}_{af} = \boldsymbol{I_2} - \boldsymbol{r}_{af}\boldsymbol{r}_{af}^T, \\ \boldsymbol{v}_{d,af} = \Lambda_{af}\boldsymbol{r}_{af}(d_{af} - \sigma_{af}), \end{cases} \tag{4.6}$$

where d_{af} is the desired distance an agent should keep against a *friend agent*, and $\boldsymbol{r}_{af}$ is found replacing $\boldsymbol{p}$ in Eq. 4.5 with $\boldsymbol{p}_f$ which is the position of the *friend agent* in the local coordinate system of the current agent. Similar for generic object avoidance, the definition becomes:

$$\text{Avoid object} \begin{cases} \sigma_{ac} = \|\boldsymbol{p_o}\|, \sigma_{ac,d} = d_{ac}, \\ \boldsymbol{J}_{ac} = \boldsymbol{r}_{ac}^T, \boldsymbol{J}_{ac}^\dagger = \boldsymbol{r}_{ac}, \boldsymbol{N}_{ac} = \boldsymbol{I_2} - \boldsymbol{r}_{ac}\boldsymbol{r}_{ac}^T, \\ \boldsymbol{v}_{d,ac} = \Lambda_{ac}\boldsymbol{r}_{ac}(d_{ac} - \sigma_{ac}), \end{cases} \tag{4.7}$$

where d_{ac} is the minimum distance an agent should keep against any obstacles in the environment, including other agents, and $\boldsymbol{r}_{ac}$ is found replacing $\boldsymbol{p}$ in Eq. 4.5 with $\boldsymbol{p}_o$ which is the position of the obstacle in the local coordinate system of the current agent. Both of these two behavior allows the agent to keep *friend agents*, other agents and obstacles on the border of their respective thresholds d_{af} and d_{ac}. These two behavior are only activated when their task values are below the desired task values.

The lattice formation behavior is tasked at keeping other visible agents at a constant distance d_l from itself. If a generic agent is too far away, i.e. the task value $\sigma_{f.d} < \sigma f$, the behavior generates an output that moves the given agent closer to the generic agent, otherwise the output moves the agent away from the point. The definition then becomes:

$$\text{Lattice formation} \begin{cases} \sigma_{lf} = \|\boldsymbol{p_a}\|, \sigma_{lf,d} = d_{lf}, \\ \boldsymbol{J}_{lf} = \boldsymbol{r}_{lf}^T, \boldsymbol{J}_{lf}^\dagger = \boldsymbol{r}_{lf}, \boldsymbol{N}_{lf} = \boldsymbol{I_2} - \boldsymbol{r}_{lf}\boldsymbol{r}_{lf}^T, \\ \boldsymbol{v}_{d,lf} = \Lambda_{lf}\boldsymbol{r}_{lf}(d_{lf} - \sigma_{lf}), \end{cases} \tag{4.8}$$

where d_{lf} is the *lattice scale*, r_{lf} is found replacing $\boldsymbol{p}$ in Eq. 4.5 with $\boldsymbol{p}_a$ which is the position of a generic agent in the local coordinate system of the current agent.

Reach Position

The reach position behavior enables the agent to move closer to a given point $\boldsymbol{p}_b$. In this scenario the task is used for moving closer to the border or to follow other agents. The definition is straight forward:

$$\text{Reach position} \begin{cases} \sigma_{rp} = \|\boldsymbol{p}_p\|, \sigma_{rp,d} = 0, \\ \boldsymbol{J}_{rp} = \boldsymbol{r}_{rp}^T, \boldsymbol{J}_{rp}^\dagger = \boldsymbol{r}_{rp}, \boldsymbol{N}_{rp} = \boldsymbol{I_2} - \boldsymbol{r}_{rp}\boldsymbol{r}_{rp}^T, \\ \boldsymbol{v}_{d,rp} = \Lambda_{rp}\boldsymbol{r}_{rp}(-\sigma_{rp}), \end{cases} \tag{4.9}$$

This behavior is a special case of the previous behavior where the desired task value σ_{rb} is here set to 0 in order to minimize the distance between the agent and the point $\boldsymbol{p}_p$, as with the previous behaviors, the symbol $\boldsymbol{p}$ in Eq. 4.5 is replaced, and $\boldsymbol{p}_p$ is used instead, which is the location of the point to reach given in the local coordinate system of the agent.

Move perpendicular

The move perpendicular behavior allows the agent to move perpendicular to a given point $\boldsymbol{p}_{mp}$. In this scenario, the point is sampled from the border, and the tangent based on the direction from the agent to the point is calculated. The direction of the tangent is dependent on the patrolling direction, clockwise or counterclockwise. For a tangent with a clockwise direction the behavior equation becomes:

$$\text{Move perpendicular} \begin{cases} \boldsymbol{v}_{d,mp,cw} = \Lambda_{mp} \boldsymbol{r}_{mp,cw} \\ \boldsymbol{N}_{mp,cw} = \boldsymbol{I}_2 - \boldsymbol{r}_{mp,cw} \boldsymbol{r}_{mp,cw}^T, \end{cases} \tag{4.10}$$

where $\boldsymbol{r}_{mp,cw} = [-k_y, k_x]^T \in \Re^{3\times 1}$ and $[k_x, k_y] = \frac{\boldsymbol{p}_{mp}}{\|\boldsymbol{p}_{mp}\|}$. For a counterclockwise direction for the tangent, $\boldsymbol{r}_{mp,cw}$ is replaced by $\boldsymbol{r}_{mp,ccw} = -\boldsymbol{r}_{mp,cw}$. This behavior is illustrated in figure 4.7.

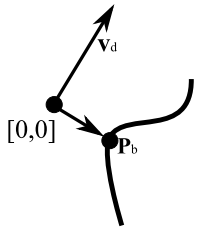


Figure 4.7: Generic avoidance behavior

The desired speed $\boldsymbol{v}_d$ from the patrol border behavior is the tangent from the direction from the agent down to the border.

Wander

In order for an agent to search for the border or other agents in the environment it has to have a behavior that creates coherent but random movements. Instead of creating a pure, random direction for each step that would produce a twitchy and unnatural movements, a technique outlined by [30] is used to create smooth and sustained behavior. The idea is to limit the steering force to a cylinder with radius r_c placed directly ahead of the agent by some offset o_c. Each step uses the displacement value of the previous step with the addition of an random displacement with a magnitude determined by the value m_d. This new value is

again constrained to the surface of the cylinder. The equation for the random walk is then:

$$\text{Wander behavior} \begin{cases} \boldsymbol{c}_c = \hat{\boldsymbol{v}} o_c, \boldsymbol{a}_{c,i-1} = [a_{c,x}, a_{c,y}]^T \\ \boldsymbol{d} = [\cos a_{c,x}, \sin a_{c,y}]^T r_c \\ \boldsymbol{a}_{c,i} = \boldsymbol{a}_{c,i-1} + \boldsymbol{z} m_d - \boldsymbol{z} 0.5, \boldsymbol{z} = [n \in \boldsymbol{z} : 0 < n < 1]^T \\ \boldsymbol{v}_{d,w} = \boldsymbol{c}_c + \boldsymbol{d}, \end{cases} \tag{4.11}$$

Where $\boldsymbol{c}_c$ is the center of the cylinder that constrains the speed force, $\hat{\boldsymbol{v}}$ is the normalized speed of the agent, o_c is the offset of the circle from the agent, d is the displacement force, $\boldsymbol{a}_{c,i-1}$ is the value of the angle that the displacement force is based on,$\boldsymbol{a}_{c,i}$ is the next random step angle value. m_d is the maximum change of the angle step value, $\boldsymbol{z}$ is a random vector with values ranging from 0 to 1. Finally, $\boldsymbol{v}_w$ is the final wander speed. An illustration of the behavior can be seen in figure 4.8.

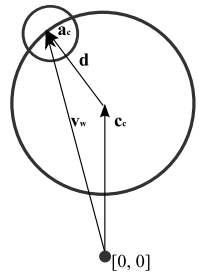


Figure 4.8: Wander behavior

The wander behavior used here are never used in an *action* where it has a priority over other behaviors. This means that the calculation of the null-space projection matrix for the wandering task is not needed and is omitted.

4.5 Actions Definition

intro goes hiero.

The first N task are taken from [1], the next two from [28].

4.5.1 Patrol Border Actions

These following behaviors are defined in the patrolling mission described in [1], and are concerned with how agents should behave when they encounter a border to patrol, while avoiding other agents and obstacles. These action is denoted with the prefix *Apb* to distinguish similar actions defined in [28].

Reach Position Action

This is a simple action that allows an agent to move towards a given position, and it directly maps the output from a single behavior, namely the *reach position* behavior, to the output of the action itself. This action is used when the only task of the agent is to move towards a point. The output is:

$$\boldsymbol{v}_{Apb,rp} = \boldsymbol{v}_{d,rp}, \tag{4.12}$$

Avoid Teammate Action

The avoid teammate action is tasked with avoiding a teammate that is in the safety zone of the agent. A teammate is an another agent in the simulation. At the same time, the secondary behavior is to try and reach the border or stay on it. This allows the agent to quickly return to patrolling the border when the teammate is out of the safety zone. This action consist of the *maintain distance behavior*, more precisely defined as the *avoid object* behavior as the top-prioritized behavior and the *reach position* behavior as the second. The output for the action then becomes using Eq. 3.10 from the NSB-framework:

$$\boldsymbol{v}_{Apb,at} = \boldsymbol{v}_{d,ao} + \boldsymbol{N}_{ao}\boldsymbol{v}_{d,rp}, \tag{4.13}$$

Avoid Friend Action

This action is tasked with avoiding friend agents while moving closer to the border. This action is basically the same as the *avoid teammate action*, the only difference is the distance the agent desires to keep to friend agents compared to the distance it desires to keep to teammate agents and obstacles. It is defined here to clarify the differences between the two actions. The output for this action is thus:

$$\boldsymbol{v}_{Apb,af} = \boldsymbol{v}_{d,af} + \boldsymbol{N}_{af}\boldsymbol{v}_{d,rp}, \tag{4.14}$$

Patrol Position Action

The patrol position action allows the robot to patrol along a position, in this case a border, while maintaining as close as possible to the it. This action is composed of the *reach position* behavior which allows it to be close to the border, and the *move perpendicular* behavior which moves the agent along the border. The output of this action does not rely on which priority the two task has since the output of the tasks are always perpendicular to each other, and consequently the second task is not cut off by the null-space projection of the first task. One of two possible definitions for the action output is then:

$$\boldsymbol{v}_{Apb,pb} = \boldsymbol{v}_{d,rp} + \boldsymbol{N}_{rp}\boldsymbol{v}_{d,mp}, \tag{4.15}$$

where $\boldsymbol{v}_{d,mp}$ is either $\boldsymbol{v}_{d,mp,cw}$ or $\boldsymbol{v}_{d,mp,ccw}$ depending on the direction the agent patrols the given point.

4.5.2 Lattice Formation Actions

The following behaviors are defined in [28] and their usage discussed in section 4.2.1. These actions are concerned with grouping the agents together in a lattice-formation while performing other tasks such as object avoidance and moving towards a rendezvous point. The actions is denoted using the Alf prefix.

Lattice Formation Actions

This action is tasked with maintaining a distance to a single object. In this scenario it is used when only one agent is in the view-distance of the current and the current agent has no other tasks to complete. The definition is then the output of the *lattice formation* behavior:

$$\boldsymbol{v}_{Alf,slf} = \boldsymbol{v}_{d,lf}, \tag{4.16}$$

If two agents or more agents are in the formation range of an agent, it takes the two closest agents and uses a combination of two *lattice formation* behaviors where the closest object is the input for the top-prioritized behavior, denoted

$\boldsymbol{v}_{d1,lf}$, while the second object is the input for the other one, denoted $\boldsymbol{v}_{d1,lf}$. The output from this action is as follows:

$$\boldsymbol{v}_{Alf,dlf} = \boldsymbol{v}_{d1,lf} + \boldsymbol{N}_{d1,lf}\boldsymbol{v}_{d2,lf}, \tag{4.17}$$

Avoid Obstacle Action

The avoid obstacle action is tasked with moving away from any objects that are in the safety zone of an agent, while trying to maintain the lattice-formation task against the closest teammate if it exists. It uses the *avoid object* behavior as the top-prioritized behavior and the *lattice formation* behavior as the second behavior. The action definition then becomes:

$$\boldsymbol{v}_{Alf,ao-lf} = \boldsymbol{v}_{d,ao} + \boldsymbol{N}_{d,ao}\boldsymbol{v}_{d,lf}, \tag{4.18}$$

If no teammates are in formation range of the agent, the action just outputs the avoid object behavior output:

$$\boldsymbol{v}_{Alf,ao} = \boldsymbol{v}_{d,ao}, \tag{4.19}$$

Move to Rendezvous Point Action

If a rendezvous point is defined for an agent and it has teammates in the formation range, it takes the closest agent and tries to remain in formation with it while moving towards the rendezvous point. The behavior used are thus the *lattice formation* and the *reach position* behavior prioritized from top to bottom respectively. The definition is then:

$$\boldsymbol{v}_{Alf,mr-o} = \boldsymbol{v}_{d,lf} + \boldsymbol{N}_{d,lf}\boldsymbol{v}_{d,rp}, \tag{4.20}$$

If no teammates are in the formation range of the agent, the action output is just the reach position behavior output:

$$\boldsymbol{v}_{Alf,mr} = \boldsymbol{v}_{d,rp}, \tag{4.21}$$

4.5.3 Border Search Actions

These actions are based off the approach described in section 4.2.2. The actions ensures that the agents collectively searches, retrieves and recover the border while maintaining a flocking behavior and avoiding objects. No new basic behaviors are used, the reuse of the ones defined in section 4.4 are sufficient.

Avoid - Maintain Formation Action

When an object is in the safety zone of an agent, and the agent is surrounded with one or more teammate agents it should primarily avoid the object and try to stay in the formation with the other teammates. Using the *avoid object* behavior and the *maintain distance* behavior will satisfy these conditions, and the output becomes:

$$\boldsymbol{v}_{A,amf} = \boldsymbol{v}_{d,ao} + \boldsymbol{N}_{d,ao}\boldsymbol{v}_{d,lf}, \tag{4.22}$$

if the object to avoid is an friend object, the definition becomes:

$$\boldsymbol{v}_{A,amf} = \boldsymbol{v}_{d,af} + \boldsymbol{N}_{d,af}\boldsymbol{v}_{d,lf}, \tag{4.23}$$

Avoid - Wander Action

The avoid - wander action is tasked with avoiding friend agents in the safety zone, and at the same time wander randomly in the environment. This action is used when the agent has no other task besides avoiding the intruding friend agent. The behaviors used in the action is *avoid friend* behavior and the *wander* behavior. The output definition for this action is:

$$\boldsymbol{v}_{A,aw} = \boldsymbol{v}_{d,af} + \boldsymbol{N}_{d,af}\boldsymbol{v}_{d,w}, \tag{4.24}$$

Wander Action

This action outputs directly the output from the *wander* behavior, and has no other sub-prioritized behaviors. This action is used when no other agents or objects are in the interaction range of the agent, and allows the agent to wander freely around. The output is then:

$$\boldsymbol{v}_{A,w} = \boldsymbol{v}_{d,w}, \tag{4.25}$$

Avoid - Follow Leader Action

If an agent is following a leader i.e. moving towards a rendezvous point and a teammate gets in the safety zone of the agent, it should move away from the agent while at the same time try to move towards the rendezvous point. This can be done by combining the two behaviors *avoid object* and *reach position*, where the avoidance behavior has top priority. The definition for this action is:

$$\boldsymbol{v}_{A,afl} = \boldsymbol{v}_{d,ao} + \boldsymbol{N}_{d,ao}\boldsymbol{v}_{d,rp}, \tag{4.26}$$

Follow Leader - Maintain Formation Action

This action is tasked with moving towards a rendezvous point while at the same time maintaining the desired lattice formation position against the closest teammate agent. It is combining the *reach position* behavior and the *lattice formation* behavior, and outputs the following:

$$\boldsymbol{v}_{A,flmf} = \boldsymbol{v}_{d,rp} + \boldsymbol{N}_{d,rp}\boldsymbol{v}_{d,lf}, \tag{4.27}$$

4.6 Supervisor Definitions

The task of selecting a suitable action from a pool of possible actions is done by using a finite state machine. Such a supervisor is easy to implement and uses no notable computational time in selecting the proper output. This means that a set of states an agent can find oneself in has to be defined and a set of conditions that makes an agent transition from one state to another. For a system in a highly dynamical and unpredictable environment a finite state machine would be very complex in order to reflect all the possible states and transitions between them. In order to keep complexity down the number of state and the transitions between these states should be minimized, since creating transitions from each state to every other would be a very tedious and convoluted process and is hard to keep track of.

4.6.1 Original Finite State Machine Supervisor

The supervisor used in this thesis is a modified version of the one proposed in [1]. In order to describe and justify these modifications a brief overview of the original supervisor is presented. This supervisor has one goal in mind, namely moving closer to the border and patrol it. This is done in separate stages and takes into account different events that may occur. These events are:

- Friend avoidance: A friend agent is detected and the agent maintains a minimum distance to it at all times.
- A teammate or object is in the safety zone and must be avoided similar to the friend agent event, but the minimum distance is smaller compared to the distance maintained to the friend agents.
- Border detected, start patrolling it. If other agents are detected when patrolling, start to patrol in the opposite direction.

Based on the events described above, an agent can at each time be in one of four *macro-states* as defined below, and can bee seen in figure 4.9:

- MS0: This state gets activated when a *friend* agent is in the safety area of an agent.
  - The only action selectable action in this macro-state is the *avoid friend action*.
  - MS1: This state is activated if the distance from the agent to the border is above a border visibility threshold. The possible actions to select from this state are:
    - If teammates are in the safety area of the agent, the *avoid teammate action* is selected.
    - If no teammates are in the safety area, the *reach frontier action* is selected.
  - MS2: This state is activated if the distance from the agent to the border is below a border visibility threshold and larger than a border patrol threshold. The possible actions to select from this state are:
    - If teammates are in the safety area of the agent, the *avoid teammate action* is selected.
    - Otherwise, the *patrol position action* is selected.

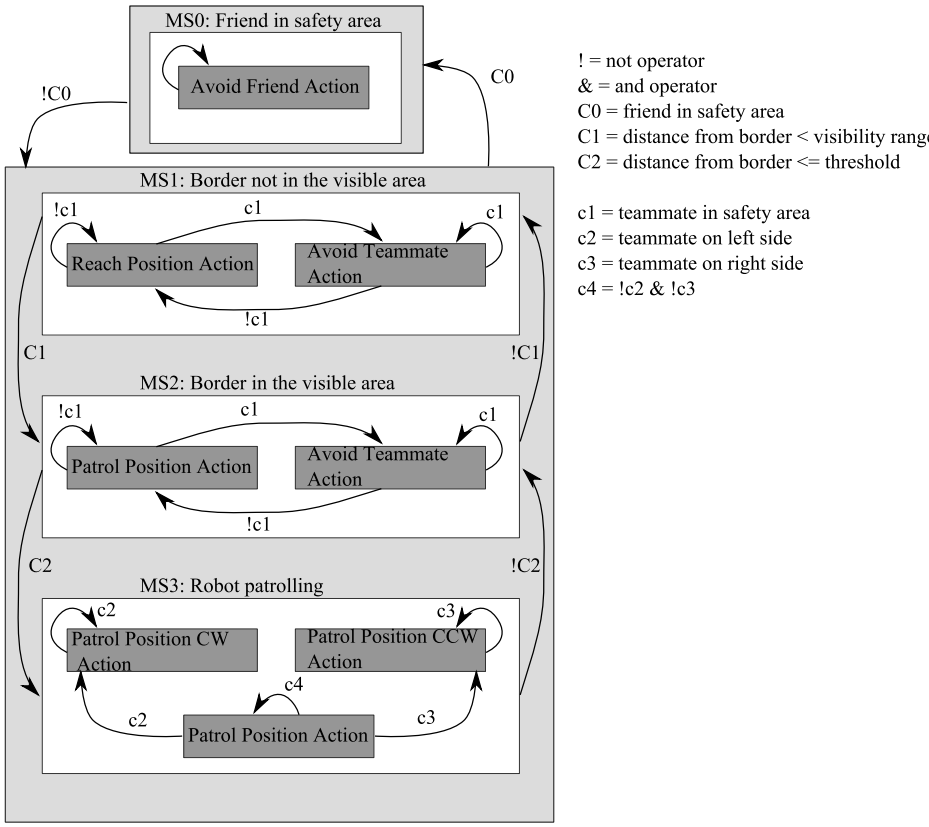


Figure 4.9: Patrolling mission finite state machine supervisor

– MS3: The last state is activated if the distance from the agent to the border is below a border patrol threshold. The possible actions to select from this state are:

  * If no teammates are detected the *patrol position action* is selected.
  * If a teammate is on the left side of the agent, the *patrol clockwise action* is selected.
  * Similar, if a teammate is detected on the right, the *patrol counter clockwise action* is selected.

The MS1, MS2 and MS3 states only becomes active if the MS0 state is not active since the avoidance of any friend agents is always a top priority. The MS2 and

MS3 states are the two macro-states that are directly concerned with selecting proper actions for patrolling the border and will be untouched in the modified version of the supervisor. In the MS1 state, the system always assumes the agent knows where the border is located even if the agent can not *see* it, and will move the agent closer to the border.

4.6.2 Modified Finite State Machine Supervisor

The supervisor used in this thesis redefines the actions which are selected when an agent has no notion of where the border is. It removes the assumption that an agent has knowledge about where the location of the border is, and instead uses a search module as outlined in section 4.2.2. This means that changes has to be made in two of the four *macro-states* defined in section 4.6.1. The states that are only concerned with the actual patrolling of the border will be untouched in terms of the action selection process. The main events to modify in the MS0 and MS1 states are the events that need to consider other teammates in terms for creating a flocking structure or events that implies that an agent should start wandering, searching for the border. Object avoidance should always be a top priority and agents in a flocking structure should always try to respond to the motion of the flock as defined in section 4.2.2. The different transition conditions for each of the four states are the same as before.

The redefined macro-states MS0 and MS1 as well as the previously defined MS2 and MS3 are shown in figure 4.10 and are defined below:

- MS0: This state is activated if a friend agent is detected in the safety zone of an agent. The following actions can be selected from this state:
  - If the border is not detected, two possible actions are available:
    - If teammates are detected, the *avoid - maintain formation action* is selected.
    - Otherwise, the *avoid - wander action* is selected.
  - If the border is detected, the *avoid friend action* is selected, where the *reach position* behavior in that action uses the border position as input.
  - MS1: This state gets activated if an agent cannot detect the border. This state is the implementation of the search module defined in section 4.2.2. This state has three sub-states: SS0, SS1 and SS2:

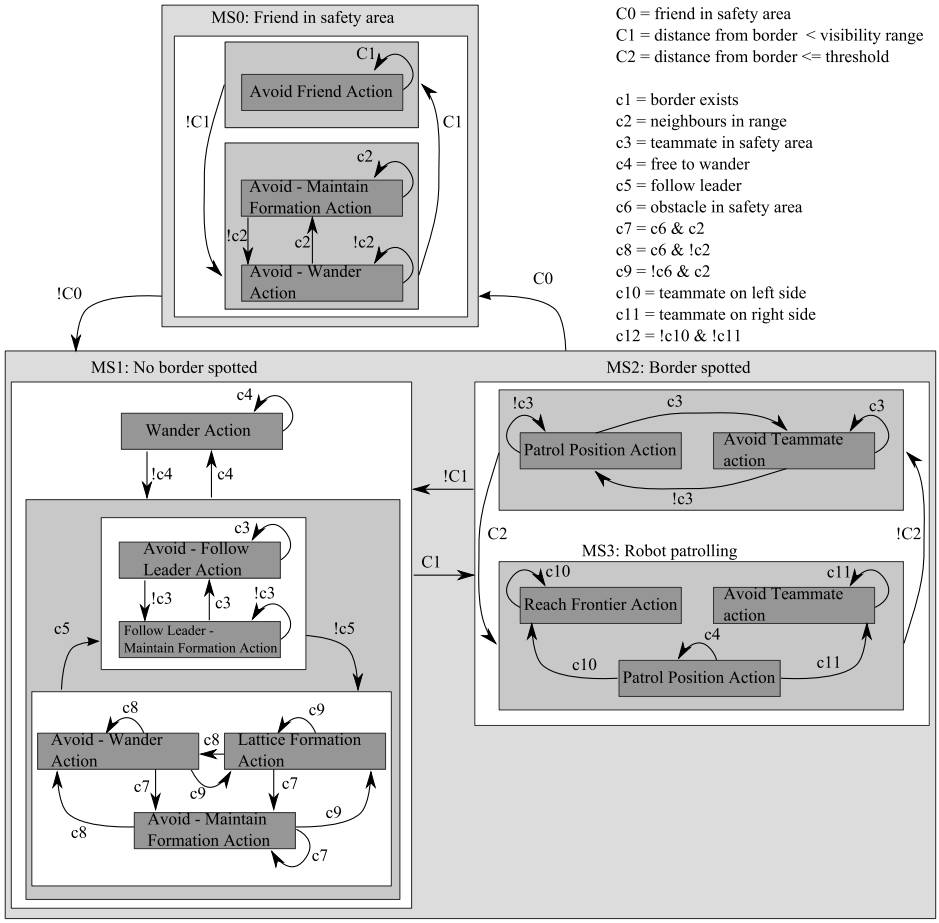


Figure 4.10: Modified finite state machine supervisor

* SS0: If an agent detects no obstacles or teammates in the safety zone, or no teammates in detection range or it is in *leader mode* the *wander action* is selected. SS1 and SS2 are only valid states if SS0 is not valid. They are defined as:

* SS1: If an agent is in a formation structure and detects that this structure is in motion, i.e a leader is present in the group, it can select between two actions:

  · If no obstacles or teammates are in the safety zone of an agent

the *follow leader - maintain formation action* is selected.

· Otherwise, if it detects obstacles or teammates in the safety zone the *avoid - follow leader action* is selected.

∗ SS2: If an agent detects no-one to follow, three actions are then available to be selected:

· If an agent detects a teammate or obstacle in the safety zone and no other teammates are detected outside of that zone, the *avoid - wander action* is selected.

· If an agent detects other teammates and nothing in the safety zone, the *lattice formation action* is selected.

· If an agent detects an obstacle or teammates in the safety zone and at the same time is a part of a formation structure, i.e. it has detected teammates outside the safety zone, the *avoid - maintain formation action* is selected.

– MS2 and MS3 becomes active when an agent detects the border, and are selected as previously described in section 4.6.1.

The MS0 state is extended to take into account the possible wander and formation tasks an agent now can have as well as the task of moving closer to the border - which now only is active when an agent actually detects the border. Avoiding the friend agent is still the top prioritized behavior in all of the actions defined in this macro-state. The MS1 state handles three different modes, SS0, SS1 and SS2, and their transitions states was defined above.

- Wandering/searching: An agent is either selected as a leader by a probability p_{leader}, and is wandering around in the environment. The conditions for staying in this mode if teammates are detected are previously defined in section 4.2.2. If no teammates are detected, the agent is free to wander as well, with the possible added task of object avoidance if objects are detected in the safety zone.

- Maintain formation: An agent will always try to maintain the lattice formation structure if it detects teammates. If no objects or teammates are in the safety zone, an agent can try to maintain the formation with up to two other agents simultaneously, otherwise the formation structure is a secondary task, while object avoidance is the primary task.

- Follow leader: An agent that is in a flocking formation, be it with one or more teammates would follow any agent that is in motion and is above a distance threshold for following teammates. Additional task would be

maintaining the formation structure as a secondary task or avoiding objects or teammates as primary and the follow task as secondary.

4.7 Simulations

In order to test out the the behavioral control for a swarm of agents in a dynamic environment, a custom simulator was built in C♯ using the XNA framework [31] on top of a ported version of Enki 2D robot simulator framework [32]. Using a proprietary simulator allows for easily integration of more advanced customization such as the dynamic border, which in reality is an 2d iso-surface mapped fluid simulation. The Enki framework handles the basic simulations such as rigid body dynamics, and the low level controls of the actors in the simulator including their sensors and actuators. The XNA framework handles the rendering and the overall update loop, while Windows presentation foundation is used for handling input and user interface. A detailed summary of this can be seen in figure 4.11, and in figure 4.12 a screen shot of the simulator in action is presented.

4.7.1 Dynamic Border

In order to investigate if the framework is adequate for patrolling *non-static* borders, a suitable model for a dynamic border is needed. The model should be interactive and manageable in terms of controlling and manipulating the overall shape and size of the border in the environment, but should also be able to contract and expand on its own based on some state in the model or by randomization. Real life scenarios requiring these features includes

In scenarios where the border to patrol represent fluid-like objects, for example oil spills, would thus require a fluid model.

Model

The dynamic border is modeled after a fast and stable 2-dimensional fluid solver [33] that creates the overall motion that fluids inhibit. The solver is based on the famous *Navier-Stokes* equations, but is modified to substitute physical correctness for stability, speed and interactiveness while maintaining an approximation of the familiar characteristics that a fluid has, such as viscosity and diffusion. The fluid, which consists of a density field, is manipulated in a contained environment by a velocity field, see figure 4.13. Changes in a part of the velocity field propagates through the field manipulating the corresponding densities in the density field.

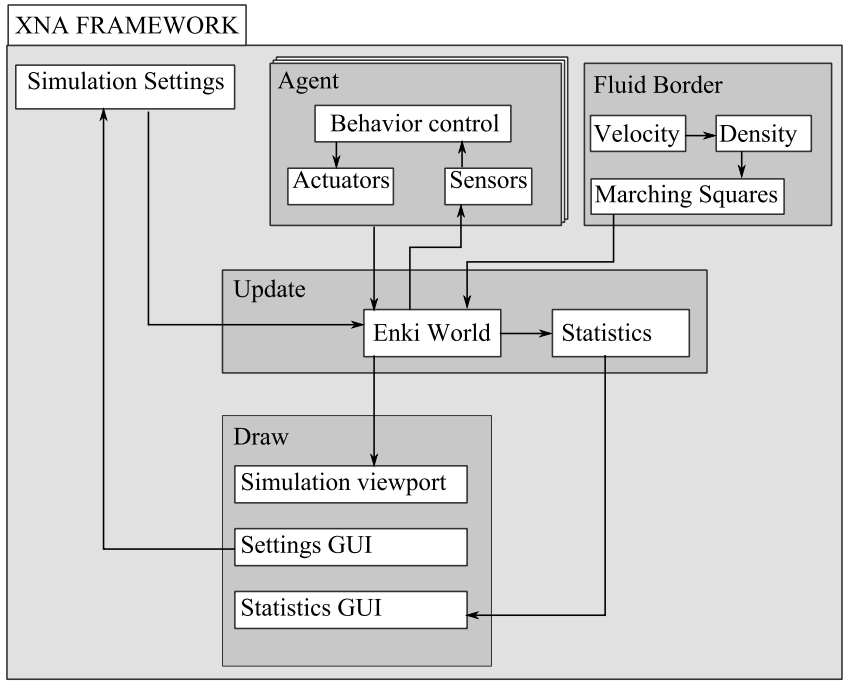


Figure 4.11: The overall simulation update loop

The density and velocity field can at any time be modified in the simulation, as well as the time step, which controls the speed of the simulation. The diffusion parameter represents the decay of density, and the viscosity parameter represents the resistance the fluid has to change. The resolution of the fluid simulation is also an important factor, the computational time for each time step grows with a higher resolution value.

Creating the border

To create a physical border out of the fluid model for the agents in the simulation to patrol, the density field is thresholded with a value into a binary scalar field, effectively creating a border between the areas in the field where the density values is below and above respectively. See figure 4.14 for an illustration. As a result, a dynamic border that follows the collection of densities that are above a certain threshold value is created. Depending on the settings new borders can

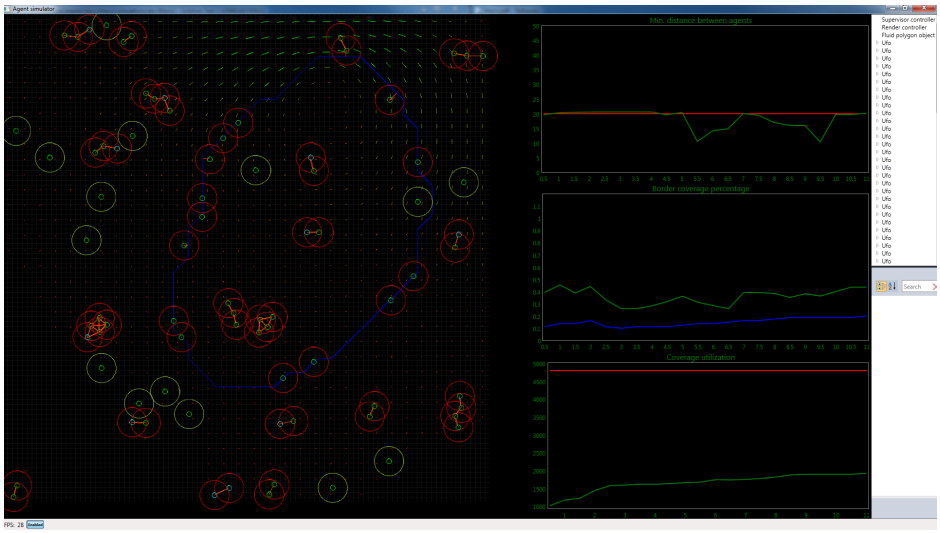
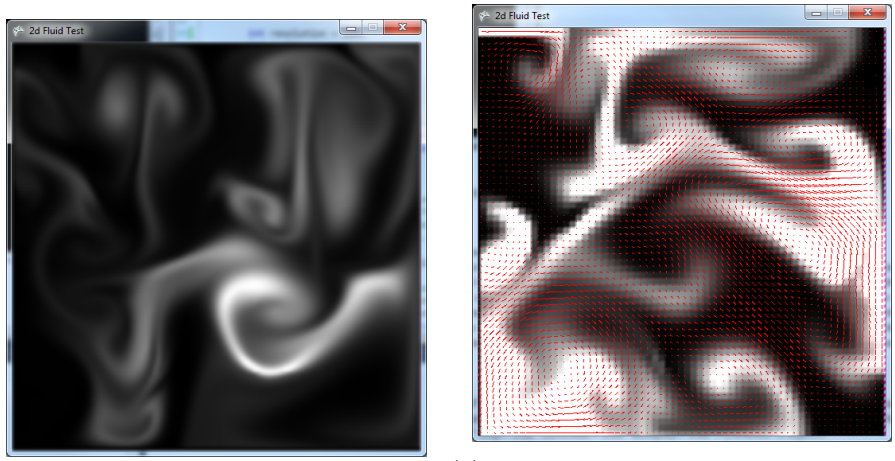


Figure 4.12: The simulatior

The green circles with yellow or red circles around them represent the agents and their field of view. A red circle is used when an agent detects something. The orange lines that some agents have is the ray that goes from the agent and to the object it has detected. Blue agents are agents in a flocking structure that has been selected as a leader. The blue shape in the middle is the dynamic border the agents try to patrol. The green and red grid represent the active velocity grid where green lines represent a high velocity and red represent low velocity. In the middle right, different real-time performance graphs can be created, and to the far right each component in the simulation can be selected and their settings can be changed.

appear and disappear based on the density and velocity distribution in the fluid model. For example, if the diffusion value is high, the calculated border will quickly disappear as the density decays. Low values will keep the density more in place, but if the system generates more density then the diffusion can remove, the border will eventually grow as big as the environment. A low viscosity value will also spread out the density when velocity is applied, while a high value will need a larger velocity force to manipulate the density. This illustrates the care that must be taken when selecting and adjusting the diffusion and viscosity parameters in the simulation.

In order for the agents to discover the border in the simulation, it has to be



(a) Showing the density field

(b) Showing the velocity over the denisty field

Figure 4.13: The 2d fluid solver

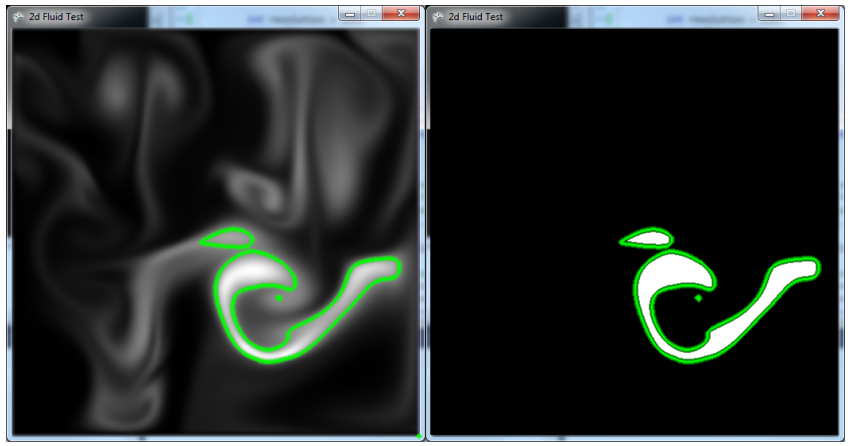


Figure 4.14: Before and after thresholding the density field

A side-by-side comparison before and after thresholding. The density field to the left and the scalar field to the right.

discretized into a polygon consisting of line segments, thus allowing the agents to accurately calculate the shape of the border that is visible to them. This is done by traversing the scalar field and upon discovering a new border, march

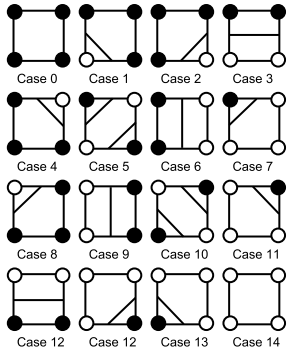


Figure 4.15: The marching squares line lookup-table

A figure showing the 15 different cases the 2×2 block of cells in each step in the traversal of the scalar field. A white dot means that the cell has the binary value of 1, 0 otherwise.

along it while creating line segments to make up the new border. Each new line added to the polygon is determined from a set of predefined lines according to the *marching squares* algorithm [34]. At each step a 2×2 block of cells are considered against 16 values of predefined lines, and is selected based on which of the cells in the 2×2 block is has the binary value 1, see figure 4.15 for the lines look-up table. When the new line has been found, the search continues at the endpoint of the new line.

Chapter 5

Experiments and Results

5.1 Experimentation

In order to test the robustness of the system several scenarios have been setup using varying conditions and settings. All of the new actions and architectural changes new to the system were tested separately while they were developed to ensure that they worked as intended under different configurations and environmental conditions. When the different, new designs were working as planned, the system as a whole was tested, and these tests will be presented in this chapter with the necessary information for recreating each experiment.

The main goal of the experiments is to investigate the ability of the system to find borders to patrol, as well as maintain and recover the patrolling formation when a patrolled border is changing its shape, in either growing, shrinking, splitting or deforming.

5.1.1 Experimental Plan

The experiments are divided into five different scenarios in order to benchmark the system under a varied set of configurations in both the setup of the agents and in the environment. These five scenarios are:

- Border search.
- Uniform border contraction.

- Uniform border expansion.
- Partially contracting border.
- Border split.

In most of the scenarios the agents are spawned evenly around the border in order to see how the system reacts to changes when it is initially successfully patrolling the border.

Border Search

This scenario investigates the capability of the system of locating any border in the environment. The agents are randomly spawned in the environment, and a static border is placed in the middle of it. A concern is that the proposed flocking behavior is a possible hindrance for the system in terms of finding a border to patrol, compared to a behavior where each agent walks around independently trying to locate the border. The way the system is designed to work implies that there could be some amount of waiting when a group of agents in a flocking structure is waiting for a leader to be created. Other questions that this experiment could answer is how the system reacts when an agent in a flocking structure that is not a leader discovers the border.

Uniform Border Contraction

In this scenario the agents spawned around a border shaped like a circle close to each other, and after a period of time the border contracts with a speed too high for the agents to follow, see figure 5.1. Similar to the border search scenario, the agents have no notion of where the border is, the difference is that they all start searching for it while being in formation mode. The idea is to see how well the system is able to relocate the border when all the agents are most likely to be in formation with one or more agents.

Partially Contracting Border

This scenario investigates how the system recovers from a border that is partially changing and decreasing in size at the same time. The border is initially circular and all the agents are uniformly situated around it. At a given time, the border is changed into an arbitrary shape and made smaller. The difference from this scenario compared to the uniform border contraction scenario, is that some agents is still able to detect the border right after it changed, meaning the border is

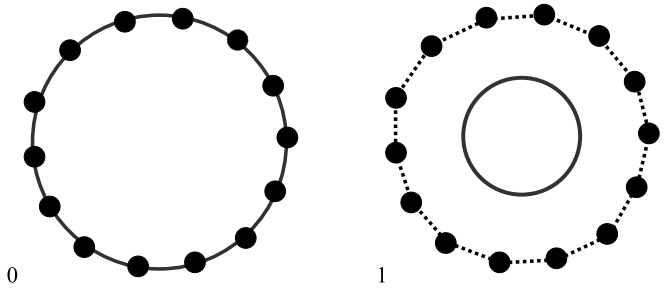


Figure 5.1: Uniform border contraction

At 0 the agents are patrolling the border and at 1 the border contracts with a speed too fast for the agents to follow and ends up in a circular formation. The rest of the scenario investigates how the agents perform when trying to relocate the border. The solid line represents the border, the black circles the agents and the dotted lines the lattice formation structure.

changing its form differently across its surface so that some agents lose track of it and some are able to patrol it. The aim of this scenario is to see if the flocking behavior of the system is able to recover the agent that lost track of the border.

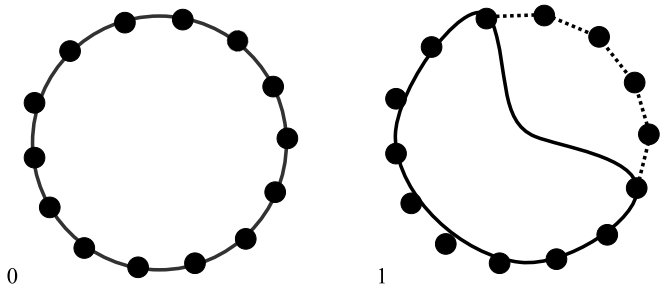


Figure 5.2: Partial border contraction

At 0 the agents are patrolling the border and at 1 the border partially deforms in a way that a group of agents lose track of the border. The solid line represents the border, the black circles the agents and the dotted lines the lattice formation structure.

Uniform Border Expansion

This scenario is similar to the uniform border contraction. Instead of contracting the border, it is expanded. Each agent is initially situated around the border, and they all collectively loose track of the border when it expands. After the expansion all agents are now *inside* the bounds of the border and should be able to locate border faster compared to when the border contracted uniformly, see figure 5.3 for an illustration.

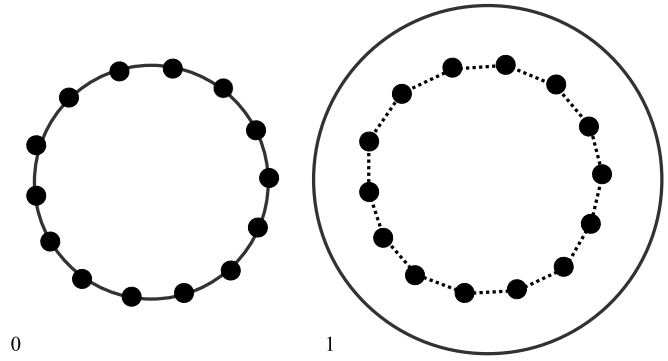


Figure 5.3: Uniform border expansion

At 0 the agents are patrolling the border and at 1 the border expands with a speed too fast for the agents to follow and ends up in a circular formation. The rest of the scenario investigates how the agents perform when trying to relocate the border. The solid line represents the border, the black circles the agents and the dotted lines the lattice formation structure.

Border Split

In this scenario the system is observed in how it performs when a border is split into two separate borders. The border is initially patrolled by a set of agents when at a given time it is divided into two new borders. Depending on the position of the agent and the shape of the new borders, an agent can find itself between two borders or situated in a formation structure that has one or more leader agents pulling in different directions. The goal here is to see if the flocking behavior allows for proper border aggregation when multiple borders are present.

Performance Measures

Each scenario uses the same performance measure, namely the percentage of robots that has detect the border. In addition, each scenario is tested with a modified version of the system where the flocking module is replaced with a random wander module, denoted as *random agents*. This means that instead of agents flocking together when they discover each other, they avoid each other. Otherwise, these agents will act as the other agents as specified by the supervisor. The system is compared against these agents since they have no behavioral traits that enables cooperation between agents that discover each other, and have no other tactics than to avoid anything that they detect that is not the border.

Each scenario is usually tested using 3 different values for p_{leader}, unless noted otherwise. These values are $0.001, 0.01$ and 0.025. This value is the probability of an agent becoming a leader when it detects that the formation it is a part of is stable, i.e. that there is not motion in the perceived formation. Using the 0.001 value it is a very low probability that an agent becomes a leader. Initial testing using such a low value shows that a group of flocking agents are very unlikely to be pulled in different directions by multiple leaders. When an agent becomes a leader it takes some time to propagate its motion to the following agents throughout the formation structure and in the meantime other agents can become a leader since they have not detected the initiated motion from the other leader(s). Another consequence using this low value is that formation structures with few agents can have no movement for relatively long periods of time since no agents is selected as leader. This is not an issue when selecting $p_{leader} = 0.025$. This is a relatively high probability for an agent to become a leader agent. A consequence with this is that larger formation structures can have multiple leaders splitting the structure into smaller ones.

5.1.2 Experimental Setup

The simulated environment is a quadratic box with a side length of $1000cm$. If an object is positioned outside of one of the bounds it is then moved to the opposite side of the environment. This way no objects disappear when they move out of bounds. Most of the experiments have agents initially successfully patrolling the border close enough to each other so every agent at least detects another agent when patrolling the border. This done in order for the system to quickly form flocking behaviors so it is easier to evaluate their performance instead of waiting for enough agents to meet up by wandering into each other.

Agent Setup

For all of the different tests conducted, the only change in the configurations of the agents is their initial positions and the probability p_{leader} of becoming a leader when an agent detects that the formation it is a part of is in a *stable state*. Each of the experiments are tested with different values for this parameter. The remaining settings for an agents is as follows:

- Sensor range: 30 cm. This is the range of the detection sensor of the agent.
- Safety radius: 20 cm. This is the closest any obstacle or agent can be to this agent.
- Lattice scale: 25 cm. This is the desired distance each agent should maintain to other agents when they are in a formation.

The gains for the different actions are taken from four values. Each behavior could use different gains, but for simplicity four values were found practical when tuning the system. The four values are:

- Avoidance gain: $\Lambda_{ao} = \Lambda_{at} = 10$. This gain is used for all behaviors where an agent is trying to move away from other agents or obstacles.
- Patrol gain: $\Lambda_{mp} = 10$. This gain is used for all behaviors which are concerned with patrolling the border.
- Reach gain: $\Lambda_{rp} = 15$. This gain is used by behaviors where the agent is tasked with moving towards a specific position, such as moving closer to the border or following an agent.
- Formation gain: $\Lambda_{lf} = 1$. This gain is used by the formation behavior.

The reach gain is using the highest gain value in order to assure that the agents following the border or other agents does not easily lose track of what they are pursuing. In contrast, the formation gain is set to a pretty low value. This is done in order to ensure a smooth creation of a formation and to avoid agents moving into the safety radius of each other when forming the formation structure. The avoidance gain is set to a high value in order to move agents quickly out of the safety radius of each other. The patrol gain is also set to a high value since the task output is always a normalized vector and thus needs to be scaled accordingly to the desired speed of the system.

Border Setup

Besides the agents, the environment has holds only one other entity, namely the border to be patrolled. As outlined in section 4.7.1, the border is based on a fluid model in order to *mimic* but not correctly replicate the behavior of viscous substances such as oil. This object is controlled directly by the fluid solver and its parameters are as follows:

- Resolution: 32. A fairly low resolution, but enables smooth, real-time simulations of the border. The lower the value the coarser the border and less computational time.
- Viscosity: 0.1. This is a relatively high value, making the border somewhat resistant to small changes made upon the border.
- Diffusion: 0. There is no diffusion in the density field. This means that the density in the system is constant.
- Border threshold: 2. A density value in the density field should have a value of 2 or more if a border is to be created at this position.

With these settings it is manageable to manipulate and deform the border with ease, without it deforming too fast or uncontrollable.

Border Search Setup

The following setup were used for the search scenario:

- 40 and 80 agents randomly spawns throughout the environment.
- A circular, *static* border covering 15% of the environment is placed in the center of the environment. The density field that make up the border has an initial value of 100.
- 3 different values for p_{leader} were tested: $0.001, 0.01$ and 0.025.
- 1 set of *random* agents were also tested.
- Each of the 8 different configurations were run 25 times each.
- Each run lasted 50 seconds.

Uniform Border Contraction Setup

The following setup were used for the uniform border contraction scenario:

- Initially, a circular border is placed in the center of the environment and covers 15% of it. After some time, when the agents have spawned and is successfully patrolling the border, it contracts about 30% so none of the agents can detect it.The density field that make up the border has an initial value of 100.
- 40 agents spawns evenly around the border.
- 3 different values for p_{leader} were tested: $0.001, 0.01$ and 0.025.
- 1 set of *random* agents were also tested.
- Each of the 4 different configurations were run 25 times each.
- Each run lasted 50 seconds.

Partially Border Contraction Setup

The following setup were used for the partially border contraction scenario:

- A circular border covering 15% of the environment is placed in the middle of the environment. The density field that make up the border has an initial value of 100. After a short period of time when all agents are successfully patrolling the border, two forces in random direction with a magnitude of 1000 is added from the center of the border, and the border threshold values is changed from 2 to 80. This two velocity impulses changes the shape of the border and the threshold value makes the overall shape of the border smaller where the two velocity impulses hit the border. The areas that are not affected from the velocity changes does not have their density values changed so some agents are most likely to detect the border after it has been changed.
- 40 agents spawns around the border and patrols it..
- 3 different values for p_{leader} were tested: $0.001, 0.01$ and 0.025.
- 1 set of *random* agents were also tested.
- Each of the 4 different configurations were run 25 times each.
- Each run lasted 50 seconds.

Uniform Border Expansion Setup

The following setup were used for the uniform border expansion scenario:

- A circular border is placed in the center of the environment, covering 15% of it. After a short period of time the border expands 30% quickly so no agents are able to detect it. The density field that make up the border has an initial value of 100.
- 40 agents spawns around the border and patrols it.
- 3 different values for p_{leader} were tested: $0.001, 0.01$ and 0.025.
- 1 set of *random* agents were also tested.
- Each of the 4 different configurations were run 25 times each.
- Each run lasted 50 seconds.

Border Split Setup

- The border is made up by a circle stretched in the horizontal axis, covering about 25% of the environment. When all agents are successfully patrolling the border, a column of density from the center of the border is removed in the vertical direction, effectively creating two borders. Two random velocity impulses are added to the velocity field towards the center of the border from each end where the column of density was removed. The magnitude of these impulses vary from 500 to a 1000 in order to create somewhat different borders each test run.
- 40 agents spawns around the border and patrols it.
- 3 different values for p_{leader} were tested: $0.001, 0.01$ and 0.025.
- 1 set of *random* agents were also tested.
- Each of the 4 different configurations were run 25 times each.
- Each run lasted 50 seconds.

5.2 Experimental Results

The results from the tests using the scenarios described in section 5.1.2 are presented in this section, as well as an analysis for each scenario result. The overall performance is of course tightly coupled to the size of the environment and the size of the border, but it is the difference in the performance of the various setups that is of interest.

5.2.1 Border Search Scenario

The averaged results for the border search scenario over 25 iterations for each configuration are presented in figure 5.4 for the first four configurations using 40 agents, and in figure 5.5 using 80 agents.

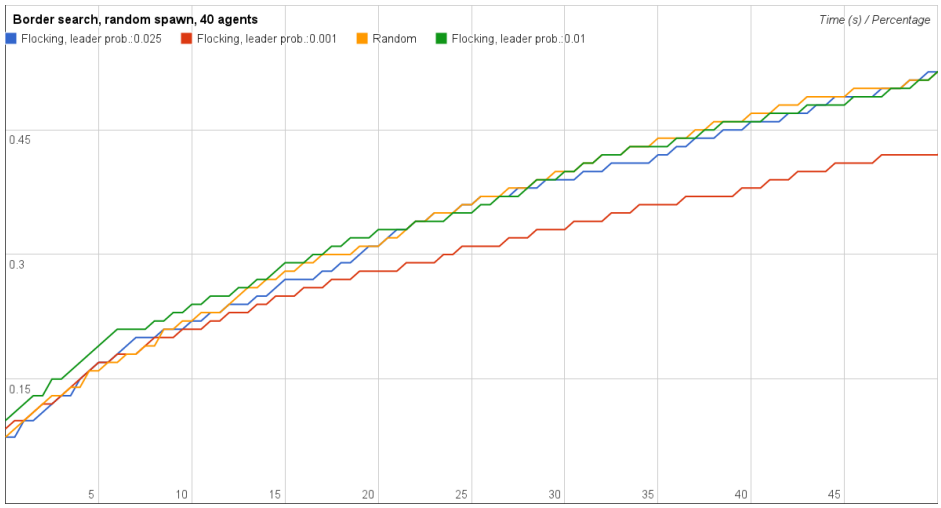


Figure 5.4: Averaged static border search performance using 40 agents

This graph shows the average percentage of agents that has detected a border over 25 iterations for each configuration.

The performances from the configurations with 40 agents shows that with a p_{leader} value of 0.025 or 0.01 is about as good compared to the configuration with *random agents*, where about 50% of the agents are able to find the border. The setup using $p_{leader} = 0.001$ is showing a stable result about 7% worse performance after 30 seconds from the other 3 setups.

The same scenario was tested using 80 agents in addition to the 40 agent tests. The results reports an overall lesser performance in all of the configurations tested. The *random agents* configuration is the best with about 3-4% better results than the configurations using a p_{leader} value of 0.01 and 0.025. The worst configurations is still the one using a leader probability value of 0.001.

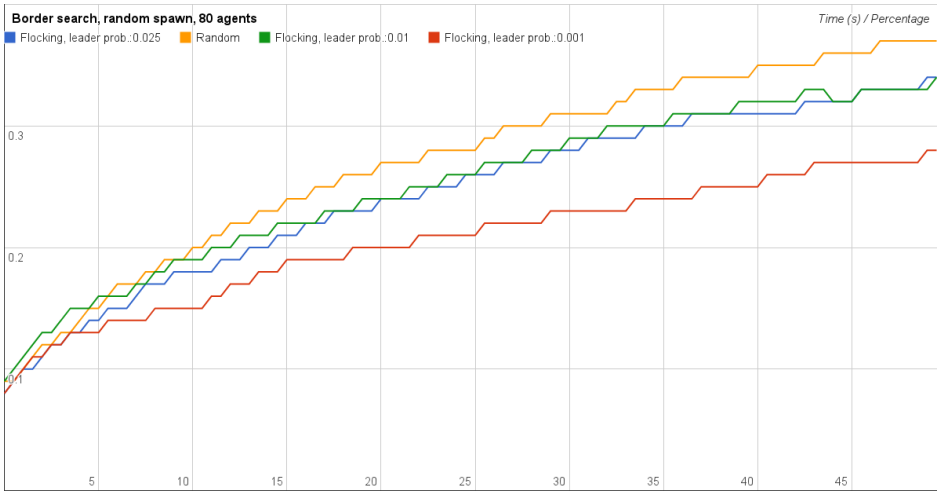


Figure 5.5: Averaged static border search performance using 80 agents

This graph shows the average percentage of agents that has detected a border over 25 iterations for each configuration.

Analysis

The experiments shows that with a $p_{leader} = 0.001$, the agents in the flocking structure is observed waiting for longer periods of time before one of them takes the leader role. This of course makes an impact on the ability to search the environment. There is some notable waiting time for the agents in the setups using 0.01 and 0.025 as value for p_{leader}, but this performance-hit in regards to searching time is evened out when a leader agent in a flock detects the border the rest of the flock aggregates to the border as well. The longer a flocking structure waits for a leader to be selected, the more agents arrive to the structure.

The reason for the overall performance decrease in the tests performed with 80 agents is observed for random agents when the environment area is so densely packed with agents so that they are confined in a small region of space where they are stuck detecting and avoiding other agents. Similar for the flocking agents, the environment is so full of agents that many flocking structures are formed, and when a flocking structure has a leader moving the group it is very likely to locate another group to form a larger structure with. This leads to more waiting time for a formation structure to have a leader selected. A conclusion to draw from this is that 80 agents is too many for the size of the environment used in these scenarios.

The results from the border search scenario also answers a few interesting questions. The first thing observed from this scenario is that a group of flocking agents are able to patrol the border when one of the agents in the structure detects it. An agent that is previously in either wandering, leader or flocking mode will automatically go into patrolling mode when it detects the border. An agent in a flocking structure that detects the border will consequently go into patrolling mode, while the neighboring agents in the structure will detect the motion and start following it and detect the border themselves. This behavior will propagate through the whole structure so that, normally, all of the agents are able to detect and patrol the border. So this means that the border aggregation behavior proposed in section 4.2.2 is working as designed.

Another behavior observed from this scenario is when a group of agents in a flocking structure that is lead by a leader agent which has not detected the border, is divided into two structures when an agent in the structure, that is not a leader, detects the border. At this point, there are two agents that moves independently and consequently the flocking structure is divided. Where this division occurs is dependent on the speed of the independent moving agents. If they are about equal, the number of agents between the independent moving agents are divided equally. If one is moving faster, it will acquire more agents.

5.2.2 Uniform Border Contraction Scenario

The averaged results from this scenarios running 25 iterations for each configuration, depicted in figure 5.6, shows how the system performs when all of the patrolling agents in the system collectively loses track of the border when it contracts. It clear that the *random agents* are able to find the border much faster the first seconds after the contraction which happens at time 0. After 20 seconds, all of the four configurations shows a steady increase of agents being able to detect the border, but with a clear advantage for the *random agents* with about 14% better performance than the best flocking configuration.

Analysis

This scenario clearly shows the disadvantage of the flocking approach compared to the *random agents* configuration. Immediately after the border contraction the random agents starts searching for the border, while the different flocking configurations starts flocking and has to wait for a leader to be selected based on the p_{leader} value. This initial flocking is show in a snapshot taken from the simulation in figure 5.7. Again, it is the configuration with the lowest probability

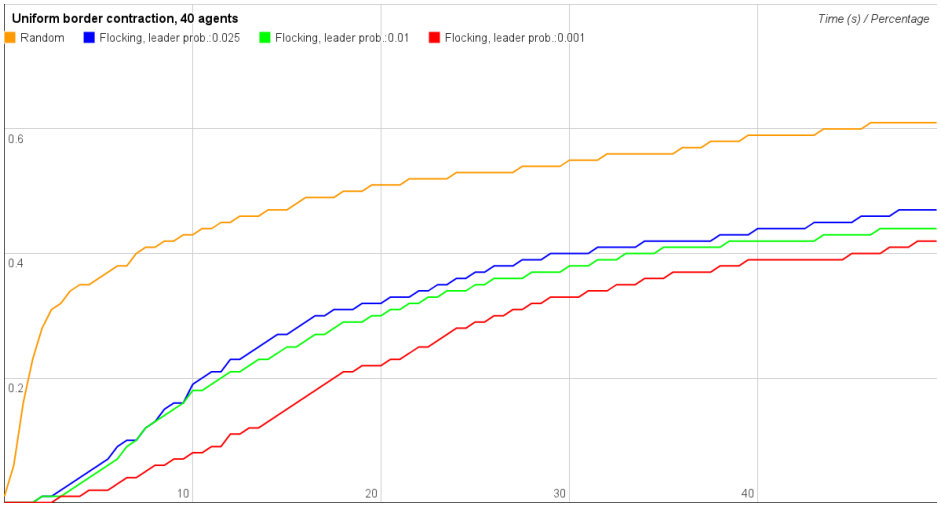


Figure 5.6: Averaged uniform border contraction performance using 40 agents

This graph shows the average percentage of agents that has detected a border over 25 iterations for each configuration, after the border uniformly contracted.

$p_{leader} = 0.001$ of an agent becoming a leader that has the worst performance, where it takes 4-6 seconds before at least on agent is able to detect the border. In addition, most of the time it is the responsibility of the leader agents to find the border since the constructed flocking structures are long lines of agents, where each agent is directly behind the agent they are following. An implication of this is that if a leader of such a structure is randomly wandering away from the border, all of the following agents will also follow in that direction and consequently be far away from the border. On the other hand, when a leader agent finds the border, normally, all the following agents will also aggregate towards the border.

Compared to the border search scenario results presented in section 5.2.1, whose scenario is similar in the way that no agents has initially detected the border, shows that when agents starts searching when they are in relatively large flocking structures they perform significantly lesser than the border search scenario. The only exception is the *random agents* in this scenario, which has a better performance that the *random agents* in the border search scenario. The reason for this is that the initial distance from the border to each agent is not far as seen in figure 5.7, so each agent has a good chance of locating it quickly, as the results show in figure 5.6 where the *random agents* quickly has over 30% of the agents patrolling the border after 2-4 seconds.

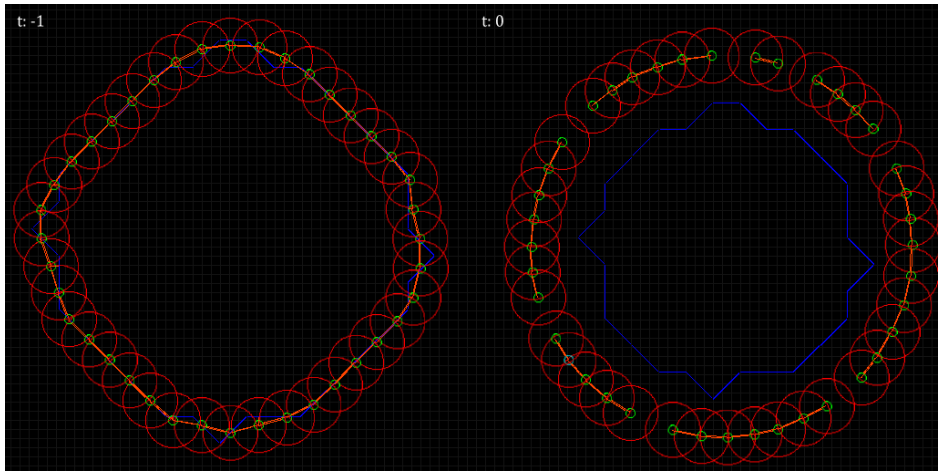


Figure 5.7: Uniform border contraction flocking structure

This picture is taken from the simulator and shows the initial setup of the agents patrolling the border at time -1, at time 0 the border contracts and the agents starts flocking with each other. The orange line shows that an agent has detect another agent.

5.2.3 Partially Contracting Border Scenario

The results for the partial border contraction scenario where around 25% of the agents loses track of the border, shows that the *random agents* perform around 10% worse than the regular flocking configurations using a p_{leader} value of $0.001, 0.01$ and 0.025. The best performer of these three configurations is the one using $p_{leader} = 0.001$ with a stable average of 95% after 22 seconds. The best configuration is a setup that uses $p_{leader} = 0$, meaning no agents in a flocking structure is selected as a leader. This setup has an average of 99% of the agents patrolling the border after 17 seconds. This setup was added after seeing the positive results from the three regular flocking configurations over the *random agents* configuration.

In addition to the standard performance test, a graph showing the percentage of agents that has detected the border or is in a flocking structure where one or more agents has detected the border is presented in figure 5.9. It shows that over 90% of the agents on average in all the three flocking configurations has either detected or is a part of a structure that has detected the border after 10 seconds. As a reference, the percentage of *random agents* patrolling the border is shown.

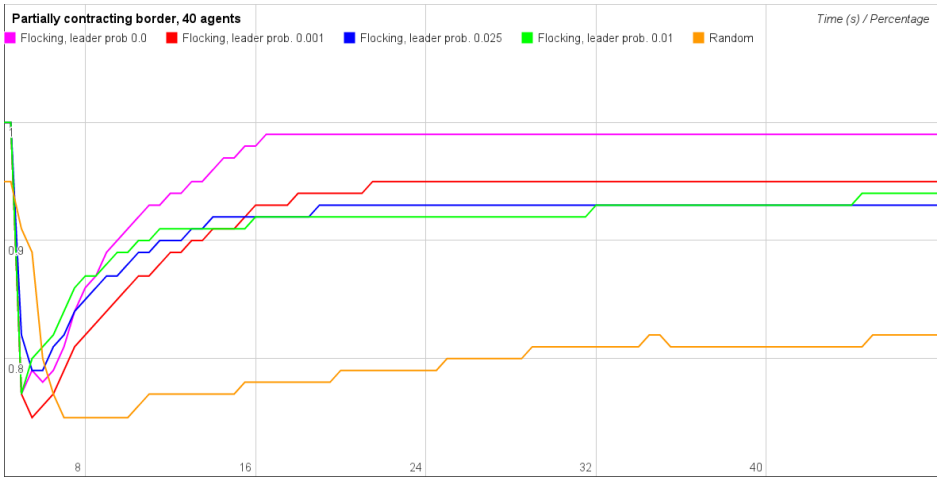


Figure 5.8: Averaged partially contracting border scenario performance using 40 agents

This graph shows the average percentage of agents that has detected a border over 25 iterations for each configuration, after the border partially contracted.

Analysis

The main reason for the good results for the flocking configurations compared to the *random agents* configuration is that most agents are initially a part of a flocking structure that has spotted the border. In the setup used in this scenarios, about 25% of the agents lose track of the border when in deforms and contracts as described in section 5.1.1. These structures tend to aggregate to the border since they have detected agents that are moving - in this case patrolling the border - and will consequently follow them. The *random agents* would wander off and it is by chance that they are able to detect the border. This is reflected in the results in figure 5.8. The difference between this scenario and the uniform border contraction scenario is that at least some agents are able to detect the border after it has contracted, making it possible for agents in a flockings structure connected to these agents to aggregate towards the border, in the latter scenario, the agents had to have a leader and that leader had to find the border for the whole flocking structure.

The configuration that has no leader agents has the best performance. This is because when the border contracts, the agents that rediscovers the border will congest the area around the border that is rediscovered since many agents from

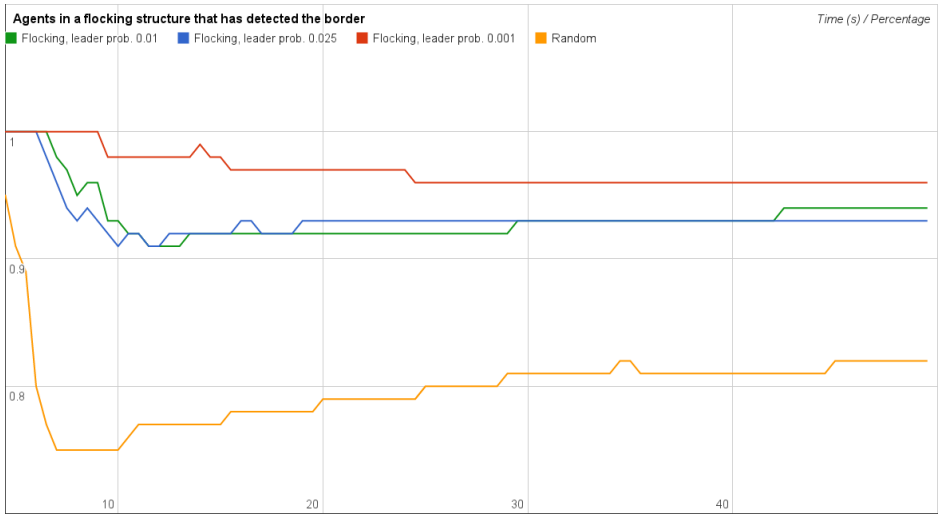


Figure 5.9: Average percentage of flocking structures detecting a border after partial contraction

The graph shows the percentage of agents that are part of a flocking structure which has one or more agents patrolling the border after the border partially contracted. The *random agents* are included as a reference.

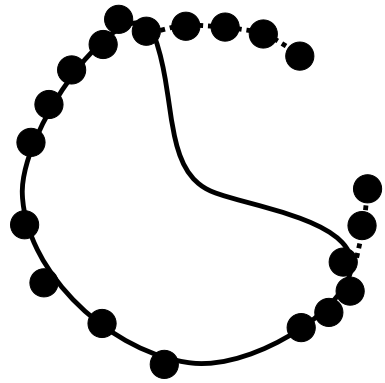


Figure 5.10: Border retrieval congestion

Congestion can occur if too many agents try to aggregate towards the border at the same time.

the flocking structure tries to move closer to it simultaneously. This congestion, depicted in figure 5.14, can lead to a full stop in the movement of a flocking structure that is aggregating towards the border, and consequently new leader agents can be selected from the stagnant flocking structure. The configuration that has a zero probability of selecting leader agents will of course avoid this problem, and the performance graph clearly shows this. The configuration with the lowest probability above zero, $p_{leader} = 0.001$, is also the best one. This is also evident in the graph shown in figure 5.8, where the same configuration has the least decline of agents that is a part of a structure that can detect the border. Agents using this configuration are less prone to wander off when stagnation occurs.

5.2.4 Uniform Border Expansion Scenario

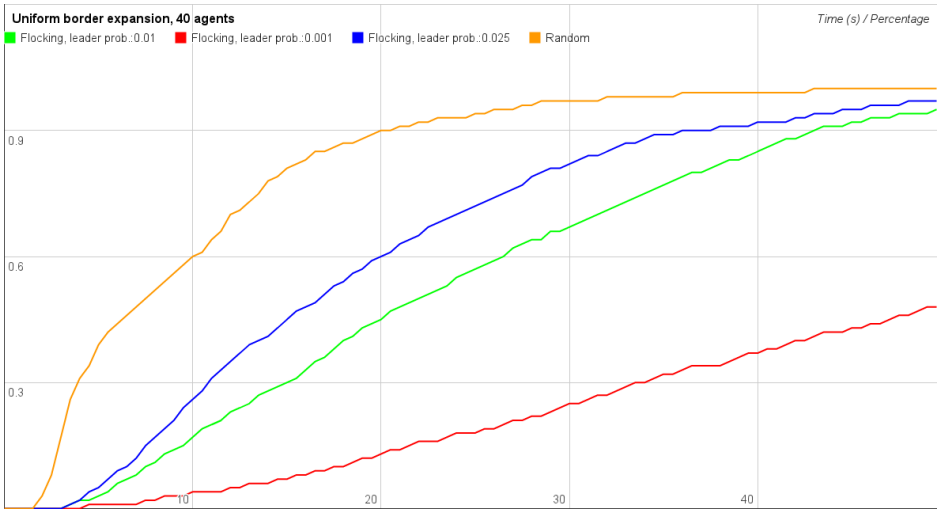


Figure 5.11: Averaged uniform border expansion performance using 40 agents

This graph shows the average percentage of agents that has detected a border over 25 iterations for each configuration, after the border uniformly expanded.

The performance results from the scenario where the whole border expands to an extent that all agents lose track of it, are presented in figure 5.11. It shows that the *random agents* aggregate faster to the border compared to the different flocking configurations. The percentage of agents patrolling the border after 50 seconds is also the highest with this configuration, with the flocking configurations using

p_{leader} values of 0.01 and 0.025 a few percentages below. These two configuration also converges to these values much slower than the *random agents* configuration. The flocking configuration using $p_{leader} = 0.001$ ends up with about 50% of the agents detecting the border after 50 seconds similar to a linear growth function.

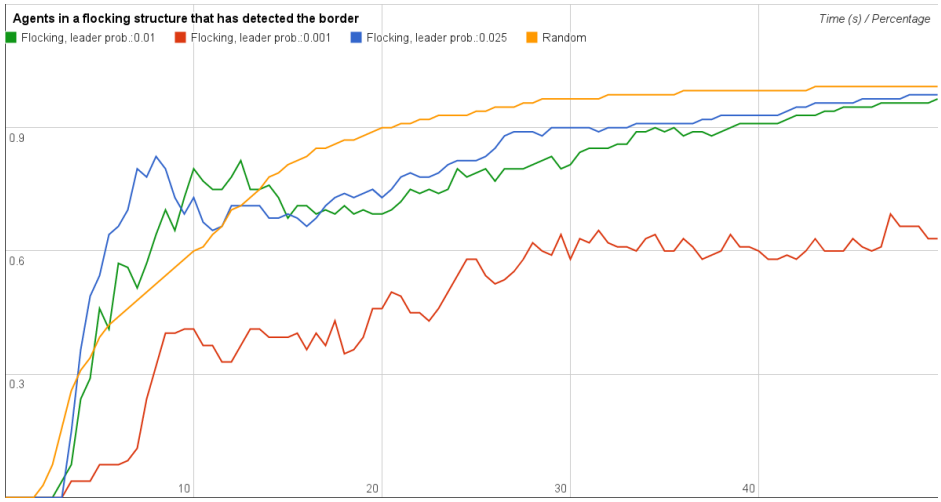


Figure 5.12: Average percentage of flocking structures detecting a border after uniform expansion

The graph shows the percentage of agents that are part of a flocking structure which has one or more agents patrolling the border after the border uniformly expanded. The *random agents* are included as a reference.

The graph showing the percentage of agents using the flocking configurations that has detected the border or is in a flocking structure that has detected the border is shown in figure 5.12. It shows that the two configuration with the highest probability for agents becoming leaders, has over 80& of the agents in a flocking structure that has detected the border before the *random agents* configuration catch up after 15 seconds. The flocking configuration using $p_{leader} = 0.001$ shows a lower amount of agents in such structures and peaks around 75%.

Analysis

Although being a novel scenario where all the agents are *inside* the border after it expands, and are surely to find the border after some time wandering in any direction, it offers some insight in how seemingly the *random agents* configuration

is able to find the border first, as seen in figure 5.11. Looking at the data in figure 5.12 shows how two of the flocking agents configuration are able to find the border with more agents faster than the *random agents* configuration, all though they use more time to aggregate all of the agents in a flock to the border. The flocking configuration with the lowest probability of selecting leader agents shows that it cannot keep up with the other configurations when it comes to locating the border, too much time is spent for the flocking structures to have a leader selected.

5.2.5 Border Split Scenario

The results from the tests performed according to the border split scenario are presented in figure 5.13. They show that all of the flocking configuration tested are able to normally, after around 20 seconds, fully recover from a border that splits into two. The flocking configuration with the highest probability of selecting a leader when a flocking structure experiences stagnation shows a 1-2% lower performance than the other two configurations. The *random agents* configuration shows around 93% recovery after 20 seconds, and immediately after the border split around 73% of the agents lose track of the border compared to 79% of the flocking configurations.

Analysis

This scenario presents similar results as seen in the *partially border contraction* scenario presented in section 5.2.3 where the *random agents* configuration persistently performs lesser than the flocking configurations in terms of agents successfully returning to the border in order to patrol it. This is because agents using the flocking configurations that lose track of the border almost always are a part of a flocking structure that has agents detecting the border. With the *random agents* configurations, agents losing track of the border after it has changed will independently wander randomly, and only rediscover the border again by chance.

Compared to the results from the partially border contraction scenario, the results presented in this scenario shows less difference between the flocking configurations in terms of agents detecting the border, as seen in figure 5.13. The reason for this difference is that right after the border splits in two large areas of the border emerges that has no agents patrolling it, which again leads to less congestion in the areas where flocking agents aggregates towards the border. An outcome of this action is that there are less stagnation periods in the flocking structures, so

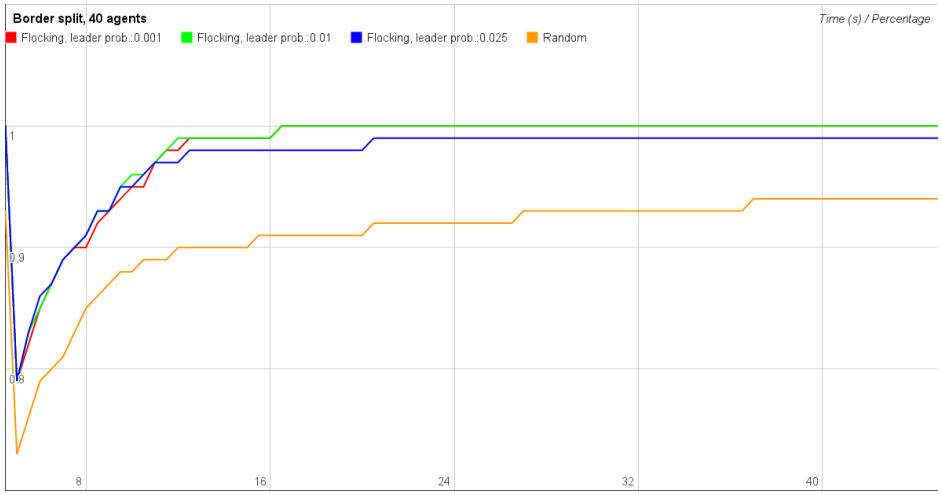


Figure 5.13: Averaged border split scenario performance using 40 agents

This graph shows the average percentage of agents that has detected a border over 25 iterations for each configuration, after a patrolled border has been split into two new borders.

that fewer leader agents are selected and leads the flock away from the border in with their wandering behavior.

This scenario also shows how a flocking structure is divided between two forces pulling in different direction when agents in the same flocking structure detects the two new borders after the split. The agents with the highest velocity will results in more agents in the structure aggregating towards them, the same way they were observed doing in the *border search scenario* in section 5.2.1. Usually, the agents patrols the border with the same velocity. Factors that affect this velocity, seen from a aggregation flocking structure, is the direction of the border from the flocking structure, and the presence of other agents patrolling the border forcing the agent to reverse its patrolling direction when they detect each other. This means that agents in a flocking structure that is pulled towards two different border, where one of the borders is congested with agents, and the other border has almost no patrolling agents, the latter border will end up attracting more agents from the flocking structure since the leader agents most likely have a higher velocity.

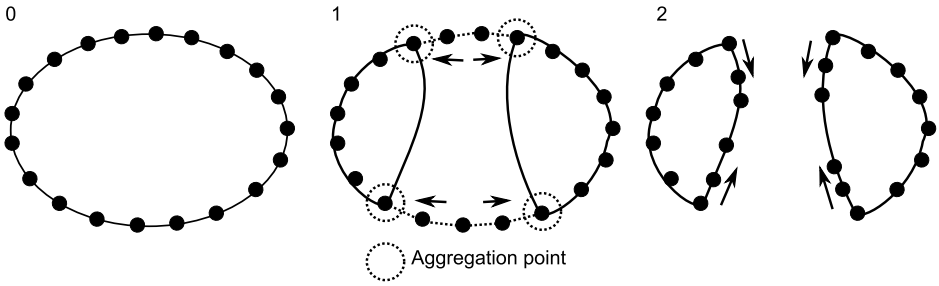


Figure 5.14: Border split

The agents in the border split scenario are less prone to congestion around the aggregation points since new border areas are created after the border split in two.

5.3 Evaluation

The experiments described and conducted in sections 5.1.1 and 5.2 gives reason to discuss the system as a whole with concerns to the challenges it faces with locating and patrolling a dynamically changing border.

5.3.1 Test Setup

After observing the tests and analyzing the data output from them, it became clear that some of the environmental choices made were not optimal. When an agent reached the outer limits of the environment, it were transported to the opposite side of the environment. The idea behind this approach was to keep a fixed number of agents in the simulation instead of discarding them when they went out of bounds. This decision was observed giving the *random agents* configuration an advantage over the flocking configurations. In scenarios where the *random agents* lost track of the border they would wander off, and when they wandered far enough they would encounter the limits of the environment and be transported to the opposite side, and consequently just wander straight into the simulation again. This also holds for the agents in the flocking configurations, but the effect of it was a bit different. In scenarios where some agents lost track of the border, they flocked with neighboring agents and often this flock had one or more agents detecting the border, eventually leading to the entire flock patrolling the border. By the time the whole flock had aggregated towards the border, random agents had time to wander the environment multiple times and just be repositioned if the went out of bounds instead of being completely

disregarded. Off course, flocking structures could also wander towards the bounds and be repositioned, but only the agents that actually crossed the bounds, which generally were leading agents since they are the spearhead of the flock, leaving the rest of the flock stagnant at the border in need of selecting a new leader to guide them. Overall, this choice is assumed to saturate the results in favor of the *random agents* configuration.

5.3.2 Flocking, Searching and Aggregation

The mechanism for creating flocking structures between agents based on simple interactions works remarkably well for creating robust structures, which is well documented in the works of Antonelli, Arrichiello, and Chiaverini [28]. Extending this behavior in pursuance of supporting movements among flocking structures in order to search or aggregate towards a border using the notion of motion as a positive stimulant, demonstrated both positive and negative results, but the concept proved to work as designed in the experiments.

The negative sides of this approach was prominent when larger flocking structures had no notion of where the border was. The process of creating motions in the system was highly dependent on the value of p_{leader} in each configuration. Low values would often results in dormant structures growing bigger in size as more agents detected the structure and when a leader was elected there was a significant chance that the leader would just wander into its own flocking structure, effectively stopping the movement of the structure again. Searching with larger flocking structures is also not optimal because the whole flock is usually dependent on the leader agent to find the border since they directly follow the movements of their neighbors and effectively creating a somewhat straight line behind the leader agent. This is fairly prominent when a group of patrolling agents flock together after having lost track of a changing border. The shape of their initial formation structure is practically a snapshot of how the border was shaped before it changed. Agents in a flock could sometimes experience congestion when trying to aggregate towards the border, leading to stagnation in the flock and possibly new leaders in the flock would be selected, guiding parts of the flock away from the direction of border. This is of course a consequence of the design of the system trying to avoid stagnation and this resulting behavior is very much dependent of the probability of a leader being selected as the results from the partially border contraction scenario shows in section 5.2.3 where the configuration with zero leader probability has the best performance.

The positives sides of this approach is that it is shows very good results for deforming borders where a larger group of agents can simultaneously lose track of

the border and still be able to retrieve as long as they are in a flocking structure that has one or more agents capable of detecting it. The flocking structures created are rigid enough to adapt to changes such as object avoidance in order to maintain the structure, but at the same time be flexible enough in order for large groups of agents to follow a single agent. Both the searching and aggregation behaviors emerge from the simple interactions in the flocking structure, leading to a self organizing structure with no need of micro-managing the behaviors of each agent. Using the flocking structure as a basis for for searching and aggregation also simplifies the implementation, creating a system with only a single parameter to configure, namely the p_{leader} value. From the experiments it is evident that using a low probability value, i.e $p_{leader} = 0.001$, is too low, often causing dormant flocking structures, on the other hand using high values, i.e 0.01 and 0.025, can often result in multiple leaders in a flock being selected, which is not necessarily always a negative thing, but is often a negative effect from congestion among border aggregating flocking structures. Each scenario in the experiments represents a single way the border could behave since it is much easier to evaluate and measure single events instead of a combination, but the results from these scenarios leads to the thought that the system woudl might outperform the *random agents* configuration when using one of the flocking configuration, preferably one with a high p_{leader} value. This assumption is based on the good results seen from the partial deformation experiments and the ability of flocking configuration to have some agents patrolling the border leading flocking agents towards it at the same time.

Overall, the system is capable of both locating and finding a border that is dynamic in shape, size and position, implemented in an independent module integrated nicely with the existing system proposed by Marino, Parker, Antonelli, *et al.* [1] without interfering the original patrolling task. The use of indirect communication with respect to the flocking and motion responsiveness in flocking structures is a great addition to the system in order to keep it robust and this communication approach is scalable using many agents. This also concludes that it is possible to extend the original solution using no direct communication, as research question 2 investigates.

In order to answer research question nr. 1 the results from the experiments points in both direction. As observed in the experiments, some flocking structures often became dormant for longer periods of time, contributing little to the overall performance, and on the other hand it shows great results when it comes to retrieving the border.

Chapter 6

Conclusion and Further Work

In this thesis a multi-agent system has been developed in order to tackle the patrolling task - the task of traversing an area in order to supervise or protect it. Research on different behavioral approaches to this problem has been undertaken in order to design a system that is both robust and scalable and capable of searching, retrieving and patrolling a dynamically changing border.

Inspired by several existing multi-agent system approaches to the patrolling task, the system proposed in this thesis in an extension of the architecture proposed by Marino, Parker, Antonelli, *et al.* [1] in order to support the capability of searching and retrieving a dynamic border, initially not supported by the original architecture.

The system is developed in the null-space-based behavioral control framework, which can be seen as a compromise of the two paradigms in behavioral control, namely the cooperative approach and the competitive approach. There the former combines different behavior to generate an output, and the latter selects a single behavior as output, the NSB approach allows a combination of multiple behaviors in a non-conflicting way arranged in a hierarchical structure. Which behaviors to combine is determined by a Finite State Automata supervisor.

The system is designed to encourage cooperation between the agent without the use of direct communication. Instead the agents are naturally attracted to each other and will follow any changes made by the other agents they perceive with their sensors. From these mechanics, a flocking behavior emerges that is both

suitable for searching and retrieving borders to patrol.

A custom simulator was also developed to support the representation of the dynamic border which is modeled using a fluid solver as the underlying controller. This was done in order to create a model capable of imitating the behaviors of borders found in real life scenarios such as oil spills.

Further Work

The experiments conducted in this thesis showed room for improvements in the developed system. The most prominent flaw of the system is how the searching behavior among a group of flocking agents is conducted where the whole structure is dependent of the leader agent in order to locate the border. This issue can be improved in a way that allows the following agents to move more adjacent to the agents they are following, much like the V-formation flocking birds exhibit when flying, allowing the system to create broader searching structures.

Larger flocking structures often created congestions of agents trying to move closer to the border, effectively creating a standstill in the flocking structure. Possible solutions to this is for agents to start patrolling the border even there are agents between them and the actual border in order to keep motion in the flocking structure. Initial tests of this approach has been conducted and showed promising results, although they require some changes in the supervisors structure, and was not included in the design presented in this thesis.

The usage of *friend* vehicles in the experiments was not tested, all though the support for they are implemented in the system. The focus of this thesis was on the searching and retrieving and was therefore prioritized when it came to testing the system.

Bibliography

[1] A. Marino, L. Parker, G. Antonelli, and F. Caccavale, "A decentralized architecture for multi-robot systems based on the null-space-behavioral control with application to multi-robot border patrolling", English, *Journal of Intelligent and Robotic Systems*, pp. 1–22, 2012, ISSN: 0921-0296. DOI: 10.1007/s10846-012-9783-5. [Online]. Available: http://dx.doi.org/10.1007/s10846-012-9783-5.

[2] M. Wooldridge, *An introduction to multiagent systems*. Wiley, 2008.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Swarm intelligence", *From Natural to Artificial Systems,(UK: Oxford University Press, 1999)*, 1998.

[4] C. R. Kube and H. Zhang, "Collective robotics: from social insects to robots", *Adaptive Behavior*, vol. 2, no. 2, pp. 189–218, 1993.

[5] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul, "Multi-agent patrolling: an empirical analysis of alternative architectures", in *Multi-Agent-Based Simulation II*, Springer, 2003, pp. 155–170.

[6] D. W. Casbeer, D. B. Kingston, R. W. Beard, and T. W. McLain, "Cooperative forest fire surveillance using a team of small unmanned air vehicles", *International Journal of Systems Science*, vol. 37, no. 6, pp. 351–360, 2006.

[7] J. Clark and R. Fierro, "Mobile robotic sensors for perimeter detection and tracking", *ISA transactions*, vol. 46, no. 1, pp. 3–13, 2007.

[8] J. Clark and R. Fierro, "Cooperative hybrid control of robotic sensors for perimeter detection and tracking", in *American Control Conference, 2005. Proceedings of the 2005*, 2005, 3500 –3505 vol. 5. DOI: 10.1109/ACC.2005.1470515.

[9] J. Pita, M. Jain, J. Marecki, F. Ordóñez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus, "Deployed armor protection: the application of a game theoretic model for security at the los angeles international airport", in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 125–132.

[10] H. Everett, "Robotic security systems", *Instrumentation & Measurement Magazine, IEEE*, vol. 6, no. 4, pp. 30–34, 2003.

[11] R.-j. Yan, S. Pang, H.-b. Sun, and Y.-j. Pang, "Development and missions of unmanned surface vehicle", English, *Journal of Marine Science and Application*, vol. 9, no. 4, pp. 451–457, 2010, ISSN: 1671-9433. DOI: 10.1007/s11804-010-1033-2. [Online]. Available: http://dx.doi.org/10.1007/s11804-010-1033-2.

[12] H. R. Everett and D. W. Gage, "Third-generation security robot", in *Photonics East'96*, International Society for Optics and Photonics, 1997, pp. 118–126.

[13] A. R. Girard, A. S. Howell, and J. K. Hedrick, "Border patrol and surveillance missions using multiple unmanned air vehicles", in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, IEEE, vol. 1, 2004, pp. 620–625.

[14] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre, "Recent advances on multi-agent patrolling", in *Advances in Artificial Intelligence–SBIA 2004*, Springer, 2004, pp. 474–483.

[15] A. Kolling and S. Carpin, "Multi-robot surveillance: an improved algorithm for the graph-clear problem", in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008, pp. 2360–2365.

[16] N. Agmon, S. Kraus, and G. A. Kaminka, "Multi-robot perimeter patrol in adversarial settings", in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008, pp. 2339–2345.

[17] Y. Chevaleyre, "Theoretical analysis of the multi-agent patrolling problem", in *Intelligent Agent Technology, 2004. (IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*, 2004, pp. 302–308. DOI: 10.1109/IAT.2004.1342959.

[18] R. Brooks, "A robust layered control system for a mobile robot", *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[19] R. Arkin, "Motor schema based navigation for a mobile robot: an approach to programming by behavior", in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4, 1987, pp. 264–271. DOI: 10.1109/ROBOT.1987.1088037.

[20] D. Payton, "An architecture for reflexive autonomous vehicle control", in *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, vol. 3, 1986, pp. 1838–1845. DOI: 10.1109/ROBOT.1986.1087458.

[21] P. Pirjanian, "Behavior coordination mechanisms-state-of-the-art", Citeseer, Tech. Rep., 1999.

[22] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[23] B. Krogh, "A generalized potential field approach to obstacle avoidance control", in *Proc. SME Conf. Robotics Research: The Next Five Years and Beyond*, 1984.

[24] D. Whitney, "Resolved motion rate control of manipulators and human prostheses", *Man-Machine Systems, IEEE Transactions on*, vol. 10, no. 2, pp. 47–53, 1969, ISSN: 0536-1540. DOI: 10.1109/TMMS.1969.299896.

[25] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for realtime kinematic control of robot manipulators", *IEEE Transactions on Robotics and Automation*, pp. 398–410, 1997.

[26] G. Antonelli, F. Arrichiello, and S. Chiaverini, "Stability analysis for the null-space-based behavioral control for multi-robot systems", in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 2008, pp. 2463–2468. DOI: 10.1109/CDC.2008.4738697.

[27] G. Antonelli, F. Arrichiello, and S. Chiaverini, "The null-space-based behavioral control for autonomous robotic systems", English, *Intelligent Service Robotics*, vol. 1, pp. 27–39, 1 2008, ISSN: 1861-2776. DOI: 10.1007/s11370-007-0002-3. [Online]. Available: http://dx.doi.org/10.1007/s11370-007-0002-3.

[28] ——, "Flocking for multi-robot systems via the null-space-based behavioral control", English, *Swarm Intelligence*, vol. 4, pp. 37–56, 1 2010, ISSN: 1935-3812. DOI: 10.1007/s11721-009-0036-6. [Online]. Available: http://dx.doi.org/10.1007/s11721-009-0036-6.

[29] A. Marino, F. Caccavale, L. Parker, and G. Antonelli, "Fuzzy behavioral control for multi-robot border patrol", in *Control and Automation, 2009. MED '09. 17th Mediterranean Conference on*, 2009, pp. 246 –251. DOI: 10.1109/MED.2009.5164547.

[30] C. W. Reynolds, "Steering behaviors for autonomous characters", in *Game Developers Conference*, vol. 1999, 1999, pp. 763–782.

[31] (May 2013), [Online]. Available: `http://msdn.microsoft.com/en-us/aa937791.aspx`.

[32] (May 2013). Enki the fast 2d robot simulator, [Online]. Available: `http://home.gna.org/enki/`.

[33] J. Stam, "Stable fluids", in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.

[34] (May 2013), [Online]. Available: `http://www.mathworks.com/matlabcentral/fileexchange/30525`.