Multi-Robot Exploration Using Potential Games

by

George Philip, B.A.Sc.

A Thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfilment of
the requirements for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
September 2013

The undersigned hereby recommends to the

Faculty of Graduate and Postdoctoral Affairs

acceptance of the Thesis

Multi-Robot Exploration Using Potential Games

Submitted by **George Philip, B.A.Sc.**

in partial fulfilment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Professor Howard Schwartz, Thesis Supervisor

Professor Sidney Givigi, Thesis Co-supervisor

Professor Roshdy Hafez, Chair
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

September, 2013

Abstract

In this thesis, we consider exploring a 2-D environment with multiple robots by modelling the problem as a Potential Game rather than using conventional frontier-based dynamic programming algorithms. A potential game is a type of game that results in coordinated behaviours amongst players. This is done by enforcing strict rules for each player in selecting an action from its action set. As part of this game, we define a potential function for the game that is meaningful in terms of achieving the greater objective of exploring a space. Furthermore, an objective function is assigned for each player from this potential function. We then create algorithms for the exploration of obstacle-filled bounded spaces, and demonstrate through simulation how it outperforms uncoordinated algorithms by reducing the time needed to uncover the space.

Acknowledgments

I would like to thank my supervisors Professor Howard Schwartz and Professor Sidney Givigi for their guidance and mentorship through the course of my studies and research. I would not have been able to complete my work successfully without their invaluable support and expertise in the related field of study.

I would like to express my appreciation for the staff and professors I have had the pleasure of meeting and interacting with over the course of my study at Carleton University.

Finally, I would like to thank my mother for her unwavering support and encouragement during the difficult long hours and joyous breakthroughs.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronyms	Definition
CML	Concurrent Mapping and Localization
DMDP	Deterministic Markov Decision Process
LRF	Laser Range Finder
MDP	Markov Decision Process
NE	Nash Equilibrium
SAP	Spatial Adaptive Play
SLAM	Simultaneous Localization and Mapping
WLU	Wonderful Life Utility

List of Symbols

Symbols	Definition
A_i	action set consisting of all possible actions a player i can play at a given time
a_i	the action a player i decides to play at time t from its action set A_i (i.e. $a_i \in A_i$)
a_i^b	baseline action
a	the joint actions of all players played at time t (i.e. $a = (a_1, a_2, ..., a_n)$)
A	the set of all joint actions that can be played
a_{-i}	joint actions of all players except player i
n	number of players in a Potential game
N	the set of all players in a Potential game (i.e. $N = \{1, 2, 3, ..., n\}$)
N_i	the set of all players, except the ith player (i.e. $N_i = N \backslash \{i\}$)

U_i	local objective function of player i in a game
U	the set of all local objective functions of all players in a game
G	representation of a game (i.e. $G = (N, A, U)$)
$\Delta(A_i)$	probability distribution over a player i's action space A_i
$\Delta(A)$	probability distribution over joint action space A
$p_i(t)$	strategy of a player i in Spatial Adaptive Play (SAP) in selecting an action from its action set A_i (i.e. $p_i(t) \in \Delta(A_i)$)
ϵ	exploration rate determining how often a player is allowed to update its action
β	exploration parameter in SAP that determines how likely a player will select a suboptimal action
α_i	action sequence set consisting of all possible two-step action sequences a player i can play over two time steps
γ	the joint two-step action sequence of all players played over two time steps
α	the set of all joint two-step actions sequences that can be played over two time steps in a Potential game
(a_i^x, a_i^1)	the two-step action sequence a player i decides to play at time t from the set α_i (i.e. $(a_i^x, a_i^1) \in \alpha_i$)
(a'_{-i}, a''_{-i})	joint two-step action sequences of all players except i

ϕ	potential function or global objective function in a Potential game
$gridpts$	set of 2-D Cartesian coordinates of all the grid points
$pose_i(t)$	pose or orientation of player i at time t
$loc_i(t)$	2-D Cartesian coordinates of player i
$pos_i(t)$	vector consisting of $loc_i(t)$ and $pose_i(t)$; $pos_i(t) = (loc_i(t), pose_i(t))$
$pos_{-i}(t)$	position and heading vector of every player except player i
$sensRange$	range of the sensor in terms of the number of grid points that can be detected vertically or horizontally from where the sensor is located
$discPts_i(t)$	set of Cartesian coordinates of the grid points that have been discovered by player i up until time t
$discPts_{-i}(t)$	set of Cartesian coordinates of the grid points that have been discovered by all players except player i up until time t
asf_i	action selection function, which compares the utility for each two-step action sequence and selects the action sequence that would provide i the most utility
Z	number of horizontal and vertical divisions in a grid game

Chapter 1

Introduction

The field of robotics has seen much development and research in recent years. The problem of exploring an unknown environment and generating a map for it remains an active area for research and is at the heart of mobile robotics. There are many applications for this such as in planetary exploration, reconnaissance, rescue, etc., in which complete coverage of a terrain is important [2]. Recently, these applications have been extended to include underwater systems in accomplishing various tasks using Autonomous Underwater Vehicles (AUVs). This includes mapping of mines underwater and the mapping of the topography under polar ice caps [3]. Furthermore, applications involving the use of multiple robots in achieving cooperative tasks have received a considerable amount of attention. A multi-agent system consists of a number of intelligent agents that interact with other agents in a multi-agent environment. An agent is an autonomous entity that observes the environment and takes an action to satisfy its objective based on its knowledge [4]. The major challenge in multi-agent systems arise from the fact that agents have limited knowledge about the status of other agents, except perhaps for a small subset of neighbouring agents. Agents are endowed with a utility function or reward that depends on their own strategies and the strategies of other agents. As such, in situations where agents know nothing about the structure of their utility functions or how their own utility

depends on the actions of other agents, the only course of action for them is to observe rewards based on experience and "optimize" on a trial and error basis [5]. Also, as all agents are trying simultaneously to optimize their own strategies, even in the absence of noise, an agent trying the same strategy twice may see different results because of the non-stationary nature of the strategies of other agents.

There are often three categories of problems that are referred to within this umbrella of mobile robotics. They are Guidance, Navigation, and Control. Guidance involves intelligent decision making of where the vehicle should go and which path or trajectory should be taken to reach the goal [6]. Navigation is the ability of the robot to perceive or localize itself in the environment in order to ensure it is on the proper course. Finally, control is the method by which the robot determines the signals to send to the vehicle actuators [6].

The exploration and the mapping of an environment is a challenge in that the environment is completely unknown and there is no preexising map for a robot to localize itself within. Having multiple robots explore and map out the space adds to the complexity because as with any multi-agent system the environment becomes dynamic and complex [4]. Not only do robots have to simultaneously explore the environment while avoiding obstacles and barriers and locate these features on a map, but they also have to coordinate themselves so that their numbers can be used to efficiently navigate the space. In addition, since each robot does not follow the same path or trajectory to explore the space, they have a different view of the environment. These different views of the environment have to be merged together to create a unified view or map of the environment. Robots that simultaneously explore an environment to map it while localizing itself within that environment is solving what is known as the Simultaneous Localization and Mapping Problem (SLAM) or the Concurrent Mapping and Localization problem (CML) [7, 8].

1.1 Motivation

In most applications today that involve the use of multiple robots to explore a space, a variation of the frontier-based dynamic programming algorithm introduced in [2] is utilized. This approach involves choosing appropriate target points for the individual robots so that they simultaneously explore different regions of the environment. Coordination is achieved by simultaneously taking into account the cost of reaching a target point and its utility [2]. Whenever a target point is assigned to a specific robot, the utility of the unexplored area visible from this target position is reduced for the other robots. In this way, different target locations are assigned to indvidual robots [2]. Using this approach every robot is required to keep track of the frontier cells, which is the boundary between unexplored and explored cells. To determine the cost of reaching the current fontier cells, the optimal path from the current position of the robot to all frontier cells is computed based on a deterministic variant of value iteration, a popular dynamic programming (DP) algorithm [9, 10]. Thus, the cost of reaching each cell in the explored space must be calculated. As it can take several iterations to converge to a final cost value for each cell, the computational complexity grows as the robots explore more space. In fact, the computational complexity is of quadratic order of the size of the explored area. The second issue with this approach is that much information has to be shared among robots to achieve coordination. Among other variables that must be shared, each robot has to share with other robots its cost of reaching the frontier cell that is closest to it. Furthermore, as part of the coordination scheme each player has to consider the decision the other players would take at a given time and the respective payoffs they would receive before the player can evaluate its own payoff for a particular action choice. Thus, considering the aforementioned facts about frontier-based dynamic programming algorithms, it would be of interest to investigate a method that may reduce computational complexity for

large search spaces; have a robot determine the action it should take without having to calculate the decision of other robots; and that reduces the information that needs to be shared among robots. It is generally known that game theory offers advantages in that it leads to decentralized systems and reduces computational complexity [1]. In this regard, we come up with a method of exploring a space using multiple robots by modelling the problem as a Potential Game.

1.2 Problem Statement

In this thesis we look at using multiple robots for the purpose of navigating and exploring a bounded 2-D space that consists of obstacles. We are more interested in the navigation algorithms employed by robots to fully explore a space than the mapping aspect.

In [1], a mechanism known as a Simple Forward Turn Controller was devised to solve the Consensus problem as a Potential game. The Consensus problem entails getting multiple robots to meet at a common location in a distributed fashion without any one robot having pre-existing knowledge of where that location is. As part of the Potential game, a global objective function or a Potential function was defined that captures the overall objective of consensus among robots. Moreover, a local objective function was assigned for each robot to work towards so that consensus could be realized in maximizing the function. The Simple Forward Turn Controller served as the framework for simulating nonholonomic behaviour of real-world robots by coercing robots to play certain actions at certain times. In this thesis, we aim to create a similar Potential game under the same framework of the Simple Forward Turn Controller, but instead of Consensus being the objective, we seek to solve the exploration problem. The exploration problem can be described as follows: while there is an unknown territory, allocate each robot a target to explore and coordinate

team members in order to minimize overlaps [11]. Given this, we define a meaningful Potential function and assign each player an objective function that is coherent with the overall goal.

1.3 Contributions and Publications

The main contributions of this thesis are:

1. The collaborative mapping of an unknown environment with a team of robots using Potential games. As part of this contribution, we extend the definition of a Potential game so that it can be modelled under the framework of the Simple Forward Turn Controller.

2. We define the potential function for our Potential game, which will be a summation of the objective function of each player in the game. The objective function itself will be based on the number of grid points that a player would discover in playing an action sequence from its action set. Moreover, update rules for crucial variables in the Potential game will be presented. The combination of the potential function and the objective function for each player will be shown to satisfy the WLU family of utility structures. We then prove that our Potential function in conjunction with the local objective function of a player constitutes a Potential game.

3. The modification of the Simple Forward Turn Controller so that a player chooses its action from its best response set. The significance of this modification to our game will be discussed.

4. Simulations in NetLogo signifying how our Potential game algorithm outperforms an uncoordinated algorithm in terms of the time required to explore a

finite bounded space. Numerous simulation results using different parameter values will be presented.

5. Runtime analysis of our Potential game algorithm, which will be found to have a lower runtime order than frontier detection algorithms.

6. Improvement of our initial Potential game algorithm. The improvement stems from having a robot predict the future location of every other robot when it decides to turn. New update rules will also be presented for key variables as part of this new algorithm.

The related publications are:

1. G. Philip, S. N. Givigi, and H. M. Schwartz, "Multi-Robot Exploration Using Potential Games," in *IEEE International Conference on Mechatronics and Automation (ICMA), 2013*, (Takamatsu, Japan).

2. G. Philip, S. N. Givigi, and H. M. Schwartz, "Cooperative Exploration Using Potential Games," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2013*, (Manchester, UK). to be published..

The publication to the IEEE ICMA 2013 conference has been nominated for best conference paper.

1.4 Organization of Thesis

The outline of this thesis is as follows:

Chapter 2 - Background and Literature Review. We review some of the earlier work on using multiple robots for the exploration of spaces as well as the use of Potential games in solving other problems such as the Consensus problem.

Furthermore, fundamental concepts and knowledge relating to this thesis will be introduced including game theory, cooperative and noncooperative games, cooperative control problems, Potential games, Nash equilibrium, holonomic and nonholonomic constraints, and algorithm runtime analysis.

Chapter 3 - Weakly Acyclic and Potential Games. The Potential game and its superclass, the Weakly Acyclic game, will be formally defined. As part of our Potential game, we define a meaningful potential function and objective function for each player. We briefly discuss how the Consensus problem was modelled in previous work as a Weakly Acyclic game under the framework of the Simple Forward Turn Controller, and adapt the Simple Forward Turn Controller for our goal of exploring a space. Finally, the resulting Potential game algorithm will be simulated for an unbounded or infinite space.

Chapter 4 - Modified Potential Game Algorithms. The Potential game algorithm introduced in Chapter 3 will be modified so that bounded finite spaces can be explored by a team of robots. Simulation results with varying parameter values will be presented, which show that the modified Potential game algorithm reduces exploration time compared to an uncoordinated exploration algorithm. Furthermore, the runtime of the modified Potential game algorithm will be analyzed and shown to be less than the runtime of frontier-detection algorithms. Finally, an improvement to the modified Potential game algorithm will be suggested that is based on having a robot predict the future location of every other robot when it is deciding on a turn. Simulations results indicate this algorithm to perform slightly better than the modified Potential game algorithm.

Chapter 5 - Conclusions. We conclude the thesis by reviewing the main findings and contributions in prior chapters along with future research directions in multi-robot exploration using Potential games.

Chapter 2

Background and Literature Review

The goal of exploration is to gain as much new information as possible of the environment within bounded time [11]. As mentioned in the previous section, frontier-based dynamic programming algorithms have been widely used for coordinating multiple robots to explore a space. However, there are other methods too that have been identified or have been implemented in systems to solve mobile robotics problems. For example, before game theory had been considered for solving cooperative robotics problems, an established way of solving problems in multi-robot applications was using the leader and follower approach where one robot assigns individual behaviours and tasks to other robots [12]. With this approach, however, the leader is a high value target, which in military applications can have disastrous consequences. Another method that has recently been used in the field is based on decentralized partially observable Markov decision processes (MDP) that provide a mathematical framework for cooperative planning in an uncertain environment [1,13]. However, as the group of robots get large, the computational complexity of this method increases substantially.

More recently, the emergence of cooperative behaviours in the realm of simultaneous multi-agent learning have been studied. In [14], co-evolutionary processes involving genetic programming (GP) methods were used to achieve cooperative behaviours among multiple robots playing a soccer game.

With regard to the use of game theory in cooperative robotics, Marden et al. showed in [15] how a Weakly Acyclic game could be implemented in a decentralized manner in a holonomic environment. The Dynamic Sensor Coverage problem, which is the problem of arranging a group of sensors throughout a space so as to provide the maximum probability of detection of an intruder, was formulated as a Weakly Acyclic game [15]. Furthermore, the Consensus problem, which is the problem of getting a group of autonomous robots to meet at a point without having a centralized algorithm telling the robots where that point is, was formulated and solved in [5] as a Weakly Acyclic game. In [1], a mechanism known as a Simple Forward Turn Controller was devised as part of a Weakly Acyclic game to solve the Consensus problem in a nonholonomic environment. In this thesis, we integrate the Simple Forward Turn Controller in our multi-robot Potential game for the exploration of a space.

The following subsections briefly discuss some of the major concepts and knowledge that form the foundation of this thesis.

2.1 Game Theory

Game theory was first introduced by Von Neumann and Morgenstern in their publication "*The Theory of Games and Economic Behaviour*" in 1944 [16]. Since then it has found major applications in the field of economics, behavioural and social science, evolutionary biology, political science, and military strategy.

With regard to our focus in this thesis, game theory has been applied to many problems in the field of robotics and autonomous systems in recent years. In this setting, each player $i \in N$ is assigned an action set A_i and a local objective function $U_i : A \to \mathbb{R}$, where $A = \prod_{i \in N} A_i$ is the set of joint actions [15].

A game, G, can be expressed as a triplet [1].

$$G = (N, A, U) \tag{2.1}$$

Section 2.4 will provide two examples of games formulated using the above expression.

2.2 Cooperative and Noncooperative Games

Most games can be either categorized as a cooperative game or a noncooperative game. A cooperative game is a game in which the players form coalitions, and the coalition reinforces the cooperative behaviour [17]. In a noncooperative game or competitive game on the other hand, players interact with one another to maximize their own rewards or achieve their own goals, and each player cannot enforce contracts on what other players may do on future moves [1]. Therefore players compete and do not work together. An important assumption that each player makes about other players in noncooperative games is that they are rational. Checkers and chess are examples of noncooperative games.

Cooperative and noncooperative games can be distinguished from one another based on the reward structure or function of each of the players. If all players have the same reward function, the game is a fully cooperative game or a team game [4]. If one player's reward function is always the opposite sign of the other player's, the game is called a fully competitive or zero-sum game. A zero sum game receives its name from the fact that the gains of one player is offset by the losses of another player. Cooperative and noncooperative games can be thought of as two ends of a spectrum. Games that have all types of reward functions fall between these extremes and are known as general-sum stochastic games [4]. The stag hunt game and the prisoner's dilemma game are examples of a cooperative and noncooperative game respectively.

Both these games will be presented in Section 2.4.

2.3 Cooperative Control Problems and Potential Games

Cooperative control problems entail several autonomous players seeking to collectively accomplish a global objective [15]. The challenge in cooperative control problems is obtaining local objective functions for each of the players so that collectively they accomplish the desired global objective. In [15], a game theoretic approach to cooperative control problems was introduced that involved assigning each player a local objective function, albeit it was for solving the consensus problem rather than for exploring a space using multiple robots as is our interest in this thesis. The formulation of a game as seen in Section 2.1 was used to represent the game. The idea was that if each player's assigned utility function, U_i, were to fall under a suitable category of games, one could appeal to algorithms with guaranteed properties for all games within that category [15]. More precisely, a category of games that guaranteed a Nash equilibrium (which will formally be defined in the following subsection) was sought to solve the Consensus problem. This category of games was identified to be the Potential game. Not only does the Potential game guarantee a Nash equilibrium, but it also asserts the notion of utility alignment (as presented in [18]) so that each player's objective function is appropriately "aligned" with the objective of the global planner [15]. This addresses how a player's local objective contributes to the greater cooperative objective or goal in a cooperative control problem. As stated in the contributions of this thesis, we design a utility U_i for each player as part of a Potential game for the purpose of exploring a space. Furthermore, we utilize the same learning dynamics for each player in the game as [15] where the decision of any player i at

time t is made using only observations from the game played at time $t-1$. This is referred to as single stage memory dynamics [15].

2.3.1 Potential Games

In this subsection, we present the definition of a Potential game as it appeared in [15]. Suppose that the objective of a the global planner is captured by a potential function $\phi : A \to \mathbb{R}$, and let

$$a_{-i} = (a_1, ..., a_{i-1}, a_{i+1}, ..., a_n)$$

denote the collection of actions of players other than player i. The joint action a will be expressed as (a_i, a_{-i}) henceforth.

Definition 2.1 (Potential Games) *Player action sets* $\{A_i\}_{i=1}^n$*, together with player objective functions* $\{U_i : A \to \mathbb{R}\}_{i=1}^n$*, constitute a* **potential game** *if, for some potential function* $\phi : A \to \mathbb{R}$*,*

$$U_i(a_i'', a_{-i}) - U_i(a_i', a_{-i}) = \phi(a_i'', a_{-i}) - \phi(a_i', a_{-i})$$

for every player $i \in N$*, for every* $a_i', a_i'' \in A_i$*, and for every* $a_{-i} \in \times_{j \neq i} A_j$*.*

The implication of the definition above for a Potential game is the that there be perfect alignment between the global objective function and each player's local objective function. In other words, if a player unilaterally changed its action (i.e. from a_i' to a_i'' in Definition 2.1), the change in its objective function has to be equal to the change in the potential function [15]. This is what is expressed by the equality in Definition 2.1, and is what captures the notion of utility alignment that Potential games are known for. Therefore, any gain in utility that a player receives from

changing its action must be fully accounted for in the potential function. A very good example of a class of problems being modelled as a Potential game can be found in [15], which details how the Consensus problem can be formulated as a Potential game. There is a weaker notion of a Potential game, known as a Weakly Acyclic game, which will be discussed in Section 3.1. As stated earlier, learning algorithms for Potential games guarantee convergence to a (pure) Nash equilibrium, which is of importance to us as it will be seen in Section 3.4. Furthermore, we extend the definition of a Potential game in Section 3.2 as one of our contributions.

A Nash equilibrium is a set of strategies of players such that "the strategy of each player is a best response to the strategies chosen by all other players" [19]. Another way of perceiving a Nash equilibrium is that it is a set of strategies where no player can do better by unilaterally changing its strategy. The formal definition of a Nash Equilibrium is as follows [15], and is premised on the assumption that all players are rational.

Definition 2.2 (Nash Equilibrium) *An action profile $a^* \in A$ is called a pure Nash equilibrium if for all players $i \in N$,*

$$U_i(a_i^*, a_{-i}^*) = \max_{a_i \in A_i} U_i(a_i, a_{-i}^*).$$

From the definition of a Potential game and the Nash equilibrium above, it can be seen that in Potential games, any action profile maximizing the potential function is a pure Nash equilibrium. Thus, every potential game has at least one such equilibrium, but there may also be suboptimal pure Nash equilibria that do not maximize the potential function [15]. The definition of a Nash equilibrium will be illustrated through the example games presented in the following section.

2.4 Example Games

In this section, we provide two examples of games to illustrate the concepts discussed in Section 2.1, Section 2.2, and Section 2.3; namely the definition of a game, the difference between a cooperative and a noncooperative game, and Nash equilibrium.

The first example we present is the prisoner's dilemma, which is a very well known noncooperative game. As in (2.1), we denote this game G as $G = (N, A, U)$. In this game, two prisoner's are held in separate cells and interrogated simultaneously. They are offered deals in the form of lighter jail sentences for betraying their fellow inmate. They can "cooperate" with the other prisoner by not snitching, or "defect" by betraying the other prisoner. Thus, each prisoner has two actions available for them to choose from. The set of all joint actions A is represented in Table 2.1, which is a typical payoff matrix for this game. Each entry in the matrix represents the payoff for both prisoners for a joint action. Note that lower jail sentences correspond to higher payoffs in the matrix. There are thus 4 possible joint actions in A, and the payoffs for both prisoner's over each of the joint actions define U. According to this reward structure, the top-left corner of the matrix cannot be a Nash equilibrium because a rational prisoner would snitch on the other prisoner to increase his/her reward from 3 to 5. We again emphasize a rational player here as the definition of a Nash equilibrium deems it to be necessary (see Section 2.3.1). Likewise, the top-right and bottom-left corner are not stable because the other prisoner could defect to increase their reward from 0 to 1. In this situation, the Nash equilibrium is for both prisoners to not cooperate and defect each other out. Neither prisoner can unilaterally change their action from defecting on the other player to increase their reward.

The second game we look at is the stag hunt game, which is a cooperative game that was first introduced by Jean Jacques Rousseau in "*A Discourse on Inequality*" [20]. In this game two players may choose to hunt a stag or a rabbit. Hunting a stag

Table 2.1: Payoff matrix for prisoner's dilemma

Prisoner1 \ Prisoner2	Cooperate	Defect
Cooperate	3,3	0,5
Defect	5,0	1,1

Table 2.2: Payoff matrix for stag hunt

Player1 \ Player2	Stag	Rabbit
Stag	10,10	0,7
Rabbit	7,0	7,7

is difficult and failure would be the result if one was to hunt a stag alone. Also, a stag provides more meat than a rabbit. Table 2.2 is a typical payoff matrix for this game. Note that there are two different Nash equilibriums. They are $(10, 10)$ and $(7, 7)$, but $(10, 10)$ is clearly the optimal Nash equilibrium. This game is different from the prisoner's dilemma game because the maximum reward for each player is obtained when the players' decisions are the same. Therefore, a coalition is enforced in a game by ensuring that by working together, players generate more utility than working by themselves.

2.5 Holonomic and Nonholonomic Constraints

In robotics, a holonomic system is a system in which the robot faces only holonomic constraints. Holonomic constraints are restrictions on the movement of the robot such that path integrals in the system depend solely on the initial and final states of the system, completely independent of the trajectory of transition between those states [21]. It only describes the movement of the robot's chassis and does not account for external forces such as gravity or intertia [1]. A simple example of a holonomic system would be a ball attached to a chord of fixed length and rotating around a

point. If the angular velocity of the ball rotating around the point is given, all that is needed to determine the angular position of the ball or the state of the system at any time is the initial angular position of the ball. In most real world examples of wheeled robots, however, nonholonomic systems are encountered. A nonholonomic system is usually a mechanical system where the constraints imposed on the motion of the system are not integrable; i.e. the constraints cannot be written as time derivatives of some function of the generalized coordinates [21]. In a nonholonomic system, the state of the system depends on prior states or on the trajectory of states in its state space. Thus, the transition of states matter in computing a final state, and so the system cannot be represented by a conservative potential function. An example of a nonholonomic system is a four-wheeled car as shown in Fig. 2.1 where only the front wheels can be turned. When turning a car towards a specific direction, the front wheels do not maintain a constant angle with the heading of the car for the whole duration of the turn. We denote the angle between the wheels and the heading of the car as u as shown in Fig. 2.1. To determine the exact heading and position of the car at the end of the turn, it is necessary to know the intermediate values of u and the intermediate values of the heading and the position of the car throughout the turn. In fact, the parameter u depends on the intermediate heading and positions values of the car. Thus, the history of its states are required to determine its final state making it a nonholonomic system. Figure 2.2 and Figure 2.3 are examples from [1] that show a holonomic and nonholonomic path respectively of a bicycle-like robot.

As stated earlier, [5] and [15] solved the Consensus problem and the Dynamic Sensor Coverage problem as Weakly Acyclic games. However, they were solved for holonomic environments and thus, the algorithms could not be implemented as controllers on actual robots that may, for example, use a differential drive system. The main contribution of [1] was devising a mechanism known as the Simple Forward Turn Controller to solve the Consensus problem in a nonholonomic environment.

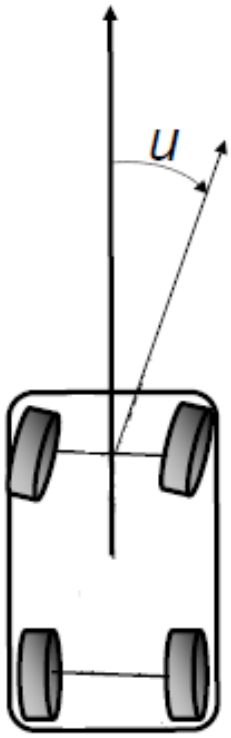

Figure 2.1: Angle between the heading and the wheels of a car [1]

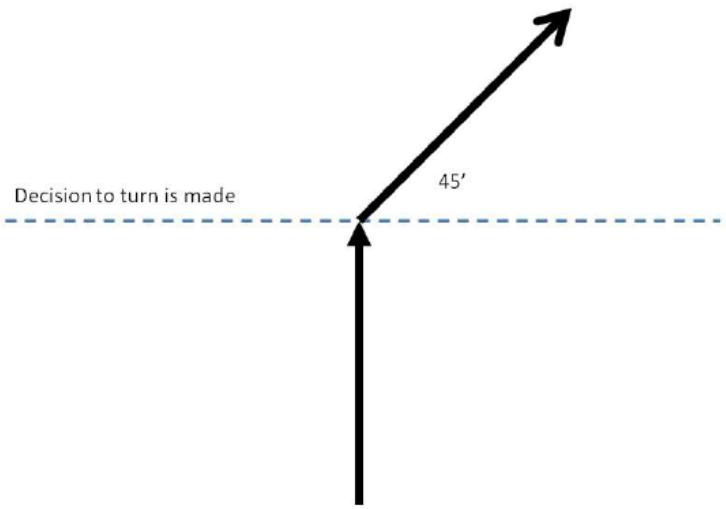

Figure 2.2: Holonomic 45 degree turn

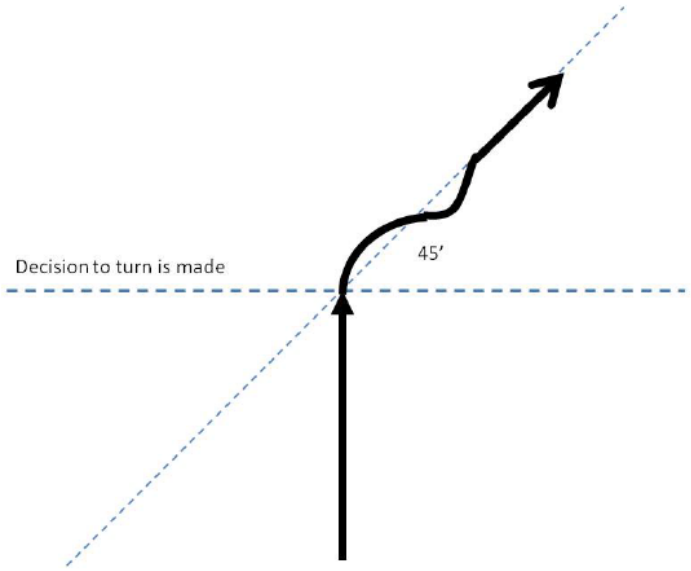

Figure 2.3: Nonholonomic 45 degree turn [1]

This mechanism will be discussed in detail in Section 3.1 when the Weakly Acyclic Game is introduced, and more thoroughly in Section 3.3.1. We build on the Simple Forward Turn Controller in our solution for exploring a space because the algorithms we develop in this thesis become more intuitive and foreseeable to implement on real systems.

2.6 Algorithm Runtime

The performance of algorithms are usually compared on the basis of their running time or the time it takes to execute. It is a measure of the efficiency or the computational complexity of the algorithm. The runtime of an algorithm is often described by a function of some parameter that determines the size of the input to the algorithm. Different functions with the same growth rate or asymptotic properties are classified together in what is known as Landau notation [22]. Landau notation captures or describes the bounds on the asymptotic growth rate of a function. The most common of these Landau notation is the big O notation, which provides an upper bound on the

growth rate of a function [22]. This is also referred to as the "order" of the function. There are several related notations associated with the big O notation, including o, Ω, ω, and Θ, which describe other kinds of bounds on asymptotic growth rates. We only concern ourselves with the big O and Θ notation in this thesis.

A function $f(x)$ is said to be on the order of another function $g(x)$ if and only if there exists a positive constant M such that for sufficiently large values of x, $f(x)$ is at most M multiplied by $g(x)$ in absolute value. This is mathematically written as

$$f(x) \in O(g(x)) \text{ as } x \to \infty, \tag{2.2}$$

and carries the following mathematical meaning.

$$|f(x)| \leq M|g(x)|, \ \forall x > x_0, \text{ where } x_0 \in \mathbb{R}$$

This can also be stated as [22]

$$\exists M > 0, \ \exists x_0 \in \mathbb{R} \ : \ |f(x)| \leq M * |g(x)|, \ \forall x > x_0.$$

The Θ notation is similar to the big O notation, except that it has a lower bound on the growth rate of a function in addition to an upper bound so that

$$f(x) \in \Theta(g(x)) \text{ as } x \to \infty, \tag{2.3}$$

if and only if

$$|k_1 * g(x)| \leq |f(x)| \leq |k_2 * g(x)|, \ \forall x > x_0, \text{ where } x_0, k_1, k_2 \in \mathbb{R}.$$

Thus, f is bounded both above and below g asymptotically. The Θ notation is more

formally stated as [22]

$$\exists k_1 > 0,\ \exists k_2 > 0,\ \exists x_0 \in \mathbb{R}\ :\ k_1 * |g(x)| \leq |f(x)| \leq k_2 * |g(x)|,\ \forall x > x_0.$$

2.7 Summary

In this chapter, we briefly reviewed some of the various methods that exist in the literature for enforcing cooperative behaviour in mobile robotics. We also presented some introductory material on major areas of knowledge and concepts, which will form the backbone of this thesis. The formal definition of game theory was presented, and it was mentioned how game theory has been used in the past in the form of Weakly Acyclic games to solve the Dynamic Sensor Coverage and Consensus problems. The Potential game and Nash equilibrium were formally defined as well, and it was discussed how every Potential game has at least one Nash equilibrium that maximizes the potential function. Furthermore, as Potential games uphold the notion of utility alignment, it would seem that they would serve as a good model to solve cooperative control problems. This connection will be confirmed through the results in this thesis. The stag hunt game and the prisoner's dilemma game were presented as examples of cooperative and noncooperative games, and their Nash equilibriums were identified. Holonomic and nonholonomic systems were then introduced, and the mechanism known as a Simple Forward Turn Controller for solving the Consensus problem in a nonholonomic environment was mentioned. In the following chapter, the Simple Forward Turn Controller will be modified and integrated into our solution for exploring a space with multiple robots. Finally, the use of Landau notation to express the running time of an algorithm was discussed.

Chapter 3

Weakly Acyclic and Potential Games

3.1 Weakly Acyclic Game

Weakly Acyclic games are a class of games that unlike what is often encountered in cooperative robotics, provides robust group behaviours for robots while only placing gentle restrictions on the robots' selection of actions [1]. "Informally, a Weakly Acyclic game is one where natural distributed dynamics, such as better-response dynamics, cannot enter inescapable oscillations" [23]. This definition implies that players can start with any action and so long as there exists a pure nash equilibrium, the players will reach it by changing their actions throughout the course of the game, which will result in a corresponding increase in their utility. The following definitions have to be established to formalize a Weakly Acyclic game.

Definition 3.1 (Better-response actions) *An action $a_i' \in A_i$ is a better-response of player i to an action profile (a_i, a_{-i}) if $U_i(a_i', a_{-i}) > U_i(a_i, a_{-i})$* [23],

where as noted before, a_{-i} refers to the joint actions of all the players except i and U_i refers to the utility or objective function of player i.

Definition 3.2 (Better response path) *A better response path in a game G is a sequence of action profiles $a^1, ..., a^k$ in that for every $j \in [1, ..., k-1]$ two conditions are met*:

1. a^j and $a^j + 1$ only differ in the action of a single player i.

2. player i at time step j+1 is a better response action, i.e. $U_i(a_i^{j+1}, a_{-i}^j) > U_i(a_i^j, a_{-i}^j)$. [1, 23]

The second part of Definition 3.2 implies that the utility received by the player changing its action at a given time must be greater than the utility it would receive if it did not change its action.

Definition 3.3 (Weakly Acyclic Games) *A game G is Weakly Acyclic if for any action profile $a \in A$, there exists a better response path starting at action a, and ending at some pure Nash equilibrium of G* [1, 23],

where as noted in Section 2.1, A represents the set of all joint action vectors for all the players in the game.

A limiting factor in a Weakly Acyclic game lies in the first part of Definition 3.2, which requires that only one player changes its action at every time step. Thus if a strict Weakly Acyclic game is used as a solution to solve a cooperative robotics problem it would require that a centralized entity determine which player will change its action at every time step. This is, however, very undesirable because it would make it a centralized system. Another option is to let the players change their actions at a random specified rate, ϵ, which is known as the exploration rate [5]. It was found in [5] that using the exploration rate option, it is never guaranteed that a Nash equilibrium will be found, but if ϵ is small and if the time step t is significantly large, the Nash

equilibrium will be found with a high probability. This theory was incorporated in [1] knowing that there is a slight probability that for a small number of tests, a Nash equilibrium consensus point would not be reached.

As mentioned in Section 2.5, the Simple Forward Turn Controller was devised as an improvement over [5] and [15] in solving the Consensus problem in a nonholonomic environment. The lateral motion experienced by a robot when it turns is accounted for in the Simple Forward Turn Controller by having the robot change its pose or orientation in one time step, and then having it move forward with the new pose in the following time step for one time step. Thus, when a robot turns it does so over two time steps over a sequence of two different actions (a "turn action" and a "move forward" action). We call this a two-step action sequence. This restriction in having to perform a turn over two times steps is how the nonholonomic behaviour of a robot is modeled. The idea of having a two-step action sequence is that it would impact the utility the robot would receive over the course of the turn sequence in comparison to the utility it would get if a whole turn (which includes a robot's lateral movement and its movement forward) was executed in one time step. This will be seen in Section 3.2 and Section 3.3.1. Section 3.3.1 discusses the use of restricted action sets to model a nonholonomic system, and it will be seen how the Simple Forward Turn Controller allows these restrictions to be realized.

The following subsections discuss the workings of the Simple Forward Turn controller as seen in [1], which includes initialization and the action-selection policy based on the expected utilities. However, we first need to establish an action set for each of the robots in a similar manner as [1]. We will arbitrarily assign each robot in our game an action set, A_i, that has without loss of generality four actions. This is sufficient for any robot to get to any point in a 2-D environment.

$$A_i = \{a_i^1, a_i^2, a_i^3, a_i^4\}, \tag{3.1}$$

where a_i^1 is the action "move forward", a_i^2 "turn 90°", a_i^3 "turn 180°", a_i^4 "turn −90°".

3.1.1 Initialization

At the first time step, $t = 0$, each player will randomly select a pose. In the next time step each of the robots will execute a_i^1 ("move forward"). The combination of the "move forward" command and the pose of a robot constitutes what is known as its baseline action a_i^b [1].

3.1.2 Action Selection

At each time step, each robot is given a choice to play its baseline action by moving forward with a probability of $(1-\epsilon)$ or to explore by performing a turn sequence with a probability ϵ.

3.1.3 Baseline Action and Turn Sequence

When player i plays the baseline action and does not explore, it moves in the direction that the baseline action specifies. We denote a two-step action sequence of a player i as $(a_i^x, a_i^1) \in \alpha_i$, where $a_i^x \in A_i$ in (3.1), and the sequence of actions is a_i^x followed by a_i^1, the "move forward" command. There are four two-step actions sequences that are possible for the action set specified in (3.1). They are

$$\alpha_i = \{(a_i^1, a_i^1), (a_i^2, a_i^1), (a_i^3, a_i^1), (a_i^4, a_i^1)\} \tag{3.2}$$

Notice here that $a_i^x = a_i^1$ represents the baseline action. Fig. 3.1 shows a robot i's predicted positions in a grid game for playing each two-step action sequence in α_i. The arrow in the figure indicates the direction the robot is facing. Points 2 and 1 are the positions i expects to be at at the end of time t and at the end of time $t+1$

respectively if it is to play its baseline action sequence (i.e. (a_i^1, a_i^1)). Points 3, 4, and 5 are the positions i expects to be at at the end of time $t+1$ if it were to perform turn sequences (a_i^2, a_i^1), (a_i^3, a_i^1), and (a_i^4, a_i^1) respectively. As discussed earlier, the robot's position does not change at the end of time t when it plays any of the turn sequences; only its heading changes. This is the defining characteristic of the Simple Forward Turn Controller to account for nonholonomic behaviour.

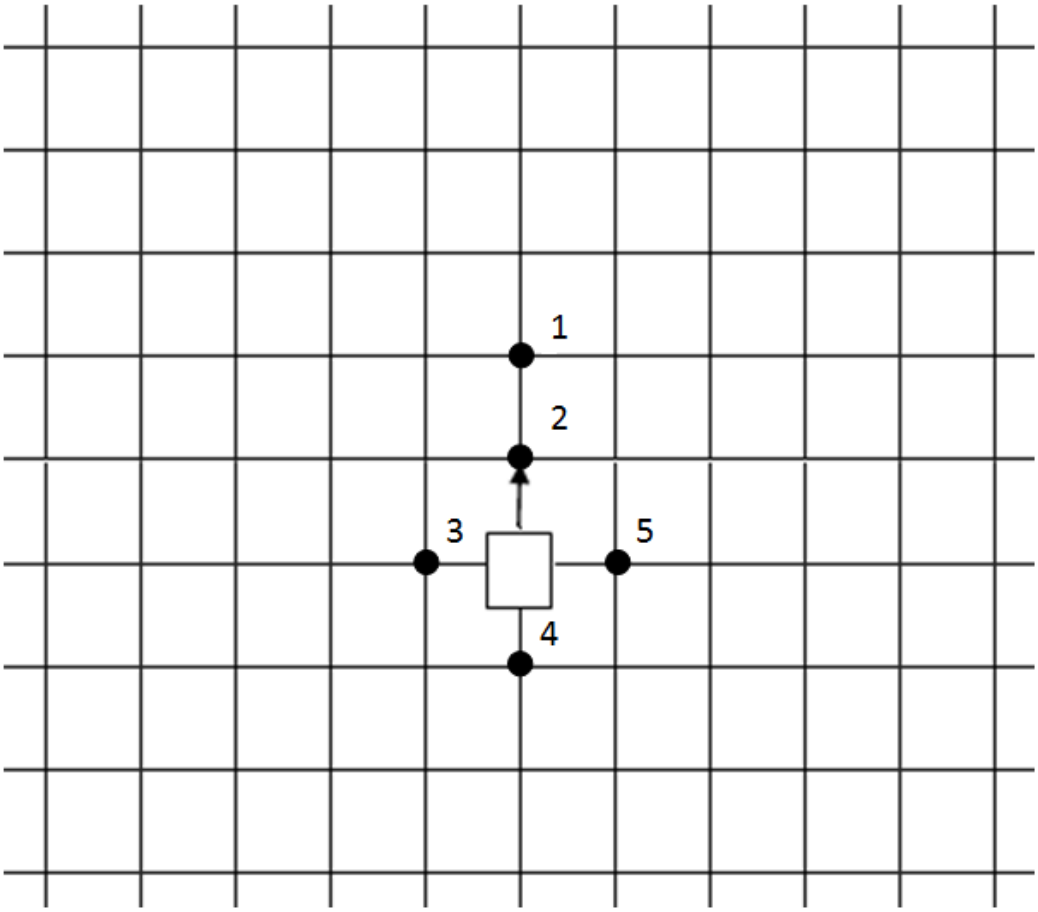

Figure 3.1: A robot's predicted future positions for each two-step action sequence

When player i explores, it randomly selects a two-step action sequence from the four possible action sequences and predicts its utility if it were to execute the action sequence. If the predicted utility of turning in a direction over a two-step action sequence is greater than the utility of playing the baseline action, the player will turn

in that direction. This is in accordance with the second part of Definition 3.2 in a Weakly Acyclic Game. It will then set the baseline action with the heading the player possesses after it has completed the turn sequence. This process will repeat itself until consensus is reached, which is synonymous with reaching the Nash equilibrium in the Consensus problem.

In [1], the utility that the controller receives is based on the Euclidean distance between the robots. In this thesis, however, as our focus is on having a group of robots explore a space, the utility will be based on the new grid points that have been discovered by the robots. This will be detailed in Section 3.2. Furthermore, we modify one aspect of the original Simple Forward Turn Controller above as it was presented in [1]. Instead of having a player i randomly select and predict the utility for one two-step action sequence from the four possible action sequences, we modify it so that the player predicts the utility for every two-step action sequence. The significance of making this modification with respect to the problem we are solving in this thesis will be apparent in Section 3.3.1.

3.1.4 Convergence of Weakly Acyclic Games

It was shown in [1] that even if a group of players used the Simple Forward Turn controller in a Weakly Acyclic game, there would be a high probability of reaching the Nash equilibrium. For the Consensus problem in particular, it was found through repeated simulations that using the Simple Forward Turn Controller with an exploration rate of 1/64 allowed the Nash equilibrium to be reached the most amount of times. However, it could take a long time to reach the Nash equilibrium as it was demonstrated in Section 3.3.4 of [1]. This is primarily because in a Weakly Acyclic game there is no systematic way for designing a player's objective function based on a global utility function [15]. In other words a player's objective function is not made explicit with respect to the greater cooperative goal. That is where a subclass of the

Weakly Acyclic game known as the Potential Game is particularly useful. A Potential Game speeds up the time it takes to reach an equilibrium at the expense of enforcing strict rules as to which action can be selected. In a Potential Game, every player's local objective function is "aligned" with the global objective function [1] so that the players' utility can be easily derived from the global utility function. In retrospect, due to a lack of rigid rules a player's utility function is "somewhat" aligned to the game's global utility function in a Weakly Acyclic game [1].

The following section will again introduce the Potential Game, but we extend its definition and introduce some new notation for the purpose of representing our game as outlined in the Contributions section of this dissertation. Furthermore, the objective function of each player and the global utility function will be defined for our game, which serves as yet another contribution of this thesis.

3.2 Potential Game

A potential game is a game "in which all payoff differences to all players can be described by first differences of a single real-valued function" [24], or in other words, a global utility function. This additional rule of having each player's utility function aligned with the global utility function is what sets a Potential game apart from a Weakly Acyclic game. An individual player's contribution to a global utility can be determined using the Wonderful Life Utility (WLU) [1]. The WLU is a family of utility structures wherein the utility of a player forms a marginal contribution made by the player to the global utility [25]. Mathematically, this is represented for every player i as

$$WLU_i = \phi(z) - \phi(z_{-i}), \tag{3.3}$$

where z represents the collection of all players, z_{-i} represents the collection of all players except player i, and $\phi()$ is the function that represents the global utility of

the players in its argument [1]. As it will be shown shortly, once a global utility function has been defined, it is easy to assign local objective functions for each of the players using the WLU so that the resulting game is a Potential game. In fact, it is known that the WLU leads to a Potential game with the global utility being the potential function [25]. It also makes a player's utility more learnable by removing unnecessary dependencies on other players' assignment decisions, while still keeping the player utilities aligned with the global utility [25]. This can be seen in [25], which uses the WLU to solve an autonomous vehicle-target assignment problem where a group of vehicles are expected to optimally assign themselves to a set of targets. In [25], the vehicles are assumed to be rational self-interested decision makers that want to optimize their utility. The utility function of each vehicle was set using the WLU so that the objectives of the vehicles are localized to each vehicle yet aligned with a global utility function. This allowed each vehicle to make their own individual decisions without any regard for the decisions of other vehicles. This aspect of the WLU that allows an agent to make decisions without considering other's decisions is highly beneficial over other methods such as reinforcement learning techniques and frontier based dynamic programming methods, which require each agent to know the actions taken by other agents.

Before we set up our potential game, we need to create a grid where the game will be played. The grid represents the space which the robots will explore. If we divide the space equally so that there is Z horizontal divisions and Z vertical divisions, we will have a $Z \times Z$ grid. A grid point is the intersection of a horizontal line and a vertical line. They serve as reference points in calculating utilities as it will be seen shortly. Furthermore, as before the group of players or robots in the potential game is represented by $N = \{1, 2, 3, ..., n\}$ where n is the number of players. In this setting, each player $i \in N$ is assigned a two-step action sequence set α_i and a local objective function $U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big) : \alpha \to \mathbb{Z}$ where $\alpha = \prod_{i \in N} \alpha_i$ is the set of joint two-step

action sequences. Before we define the function $U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big)$, however, we first define an intermediary objective function for a player for a single time step as opposed to two time steps in a two-step action sequence to make the definition easier to follow. In a similar manner as [1] we assign a player i's objective function for a single time step at a time t for a given action $a_i^x \in A_i$. Note that t is an instance of time in the discrete time domain.

$$U_i^{a_i^x}(pos_i(t)) = \sum_{pt \in gridpts} f[pos_i(t+1), pt], \tag{3.4}$$

where

$$f[pos_i(t+1), pt] = \begin{cases} 1, & \text{if } pt \in discPts_i(t) \\ 1, & \text{if } cond.1 \text{ or } cond.2 \text{ is } true \\ 0, & \text{otherwise} \end{cases}$$

$cond.1$ evaluates to $true$ if

$$||pt - loc_i(t)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t),$$

$cond.2$ evaluates to $true$ if

$$||pt - loc_i(t+1)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t),$$

and $gridpts$ is the set of 2-D cartesian coordinates of all the grid points. For example, if we have a square grid of 3×3 points we would have $3^2 = 9$ grid points, and if the grid's bottom-left corner is situated at the origin $(0, 0)$, we have $gridpts = \{(0,0), (1,0), (2,0), (0,1), (1,1), (2,1), ...\}$. The term $loc_i(t)$ is the 2-D cartesian coordinates of player i and $loc_i(t) \in gridpts$. In Fig. 3.2 for example, $loc_i(t) = (4, 1)$ for Player 1. The term $pose_i(t)$ represents the pose of player i at time

t and is composed of the unit vectors $\hat{x} = \{1, 0\}$ and $\hat{y} = \{0, 1\}$ of a 2-D cartesian space. Since a robot can be facing one of four directions in our game according to the action set defined in (3.1), we effectively have $pose_i(t) \in \{\hat{x}, \hat{y}, -\hat{x}, -\hat{y}\}$. In Fig. 3.2 for example, $pose_i(t) = (1, 0)$ for Player 1. The term $pos_i(t)$ is a vector that consists of $loc_i(t)$ and $pose_i(t)$ so that $pos_i(t) = (loc_i(t), pose_i(t))$. For example, in Fig. 3.2, $pos_i(t) = (4, 1, 1, 0)$ for Player 1. The term $pos_i(t + 1)$ is the predicted location and pose of player i at the next time step $(t+1)$ after taking an action a_i^x at time t so that $pos_i(t + 1) = (loc_i(t + 1), pose_i(t + 1))$. The term $discPts_i(t)$ is the set of Cartesian coordinates of the grid points that have been discovered by i up until time t so that $discPts_i(t) \subset gridpts$. The term $discPts_{-i}(t)$ is the set of Cartesian coordinates of the grid points that have been discovered by all players except i up until time t so that $discPts_{-i}(t) \subset gridpts$. Finally, the term $sensRange$ represents the range of the sensor, which we assume has 360° of coverage in our game, and has units that represent the number of grid points that can be detected vertically or horizontally from the location of the sensor assuming the grid is a square. Note that $pos_i(t+1)$ is present in the argument of function $f()$ because $loc_i(t + 1)$, which is a component of $pos_i(t + 1)$, is used in evaluating $cond.2$. The conditions $cond.1$ and $cond.2$ exist to ensure that points in $gridpts$ that have the prospective of increasing player i's utility have not already been discovered by other players and that it falls within the sensor scan of player i at time step t or $t + 1$ respectively.

At the beginning of a time step $pos_i(t)$ is updated. Then $discPts_i(t)$ and $discPts_{-i}(t)$ are updated to include new grid points that have been discovered so that

$$discPts_i(t) = \big\{pt \in gridpts \mid f[pos_i(t), pt] = 1\big\}, \tag{3.5}$$

$$discPts_{-i}(t) = \big\{pt \in gridpts \mid \big(\sum_{j \in N_i} f[pos_j(t), pt]\big) \geqslant 1\big\}, \tag{3.6}$$

where $N_i = \{x \mid x \text{ is a player in the game and } x \neq i\}$ or $N_i = N \setminus \{i\}$.

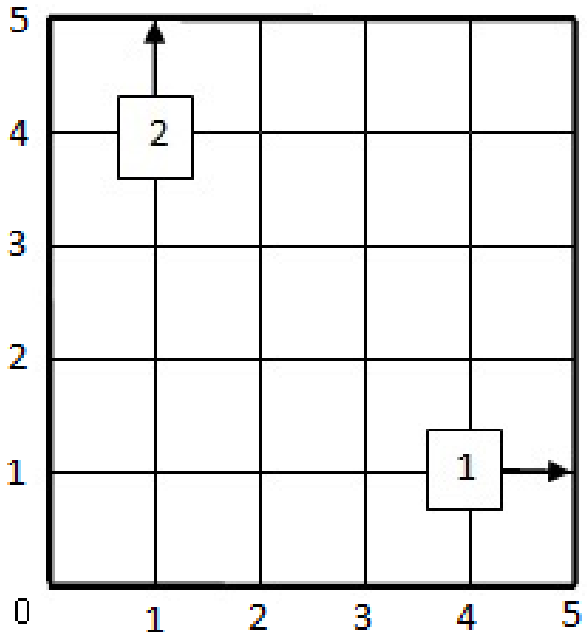

Figure 3.2: 6×6 grid game with two robots

Note that $f()$ in (3.5) and (3.6) is a function of time t rather than time $t+1$ as it was defined in (3.4). This signifies that new grid points are added to the set $discPts_i(t)$ and $discPts_{-i}(t)$ only if they have actually been discovered rather than just predicted in the previous time step when (3.4) was evaluated. There is a greater than or equal to symbol present in (3.6) because there can be times when two or more robots have overlapping sensor coverage, which can lead to a grid point being discovered by more than one robot. More will be said about this in Section 3.3.2. Finally, the predicted utility for each two-step action sequence (a_i^x, a_i^1) is calculated. This is done as opposed to just predicting the utility of one action because if we assume a player has a 360° view or sensor coverage, the turn action alone at time t will not change its utility at time $t+1$. This is because the robot will remain in the exact same position at the end of the time step t as seen in Fig. 3.1. It has to move forward at time $t+1$ if its predicted utility is to increase. This can be seen in the definition of the objective function of a player in (3.4), where only the discovery of

new grid points at a time $t+1$ causes the objective function $U_i^{a_i^x}(pos_i(t))$ to change from $U_i^{a_i^x}(pos_i(t-1))$. Hence, we now define the objective function for a player i for a two-step action sequence based on its objective function for a single time step.

$$\begin{aligned} U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big) &= U_i^{a_i^1}(pos_i(t+1)) \\ &= \sum_{pt \in gridpts} f[pos_i(t+2), pt], \end{aligned} \tag{3.7}$$

where as previously mentioned $U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big) : \alpha \to \mathbb{Z}$ and $\alpha = \prod_{i \in N} \alpha_i$. Based on the definition of $f[pos_i(t+1), pt]$ in (3.4), we evaluate $f[pos_i(t+2), pt]$ to be the following.

$$f[pos_i(t+2), pt] = \begin{cases} 1, & \text{if } pt \in discPts_i(t+1) \\ 1, & \text{if } cond.1 \text{ or } cond.2 \text{ is } true \\ 0, & \text{otherwise} \end{cases}$$

$cond.1$ evaluates to $true$ if

$$||pt - loc_i(t+1)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t+1),$$

$cond.2$ evaluates to $true$ if

$$||pt - loc_i(t+2)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t+1),$$

The left-hand side of (3.7) represents the predicted utility of a two-step action sequence $(a_i^x, a_i^1) \in \alpha_i$ for a player i at time t amidst the sequence of actions taken by the other players during the two-step action sequence, denoted here by (a'_{-i}, a''_{-i}). In the right-hand side of the equation, $pos_i(t+1)$ represents the position and pose of player i after it has executed the first action a_i^x in the two-step action sequence (a_i^x, a_i^1).

Therefore, the right-hand side of the equation represents player i's predicted utility at the end of the second time step $t+1$ after it has executed the second action of the two-step action sequence, which by the definition of a player's objective function in (3.4) is inclusive of the utility it would have had in the previous time step t. Thus, effectively, $U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big)$ is the utility player i predicts to have by the end of the time step $t+1$ at time t by following the two-step action sequence (a_i^x, a_i^1). We note two important points here. The first is that player i's predicted utility of playing the action sequence (a_i^x, a_i^1) is independent of the sequence of actions (a'_{-i}, a''_{-i}) played by other players because player i's objective function as it is defined in (3.4) is independent of the actions taken by the other players at time t. The position and the pose of the other players are not predicted nor utilized in any way. Secondly, instead of using $discPts_{-i}(t+1)$ for evaluating the right-hand side of (3.7) as it would be expected based on (3.4), $discPts_{-i}(t)$ is used. This is because the prediction is done over two time steps and at time t player i cannot know $discPts_{-i}(t+1)$. This can only be determined in the next time step after every player has taken an action and has communicated the set of grid points it has discovered to the rest of the players. Thus, it uses the latest knowledge it has, which is $discPts_{-i}(t)$. Another way of stating this is that we assume $discPts_{-i}(t+1) = discPts_{-i}(t)$. Once the utilities of every two-step action sequence have been predicted, an action is taken based on an action-policy that will be presented in Section 3.3.1.

We define the potential function of the game as

$$\phi(t) := \sum_{i \in N} U_i^{a_i^x}(pos_i(t)), \tag{3.8}$$

Given the potential function in (3.8), we can see as in [15] that with the assignment of the objective function in (3.4) each player does not have to observe the decision of all players to evaluate its payoff for a particular action choice. This is because if

we observe the definition of the potential function and its relation to the objective function of each player in (3.4), we see that it satisfies the WLU; and as stated in the beginning of this section, the WLU removes unnecessary dependencies of a player's decisions on other players' assignment decisions. We define a corresponding potential function $\phi(\gamma) : \alpha \to \mathbb{Z}$ for (3.8) that is a function of the two-step action sequences of all n players rather than time t.

$$\phi(\gamma) := \sum_{i \in N} U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big), \tag{3.9}$$

where $\gamma \in \alpha$. By using the WLU formulation in (3.3), (3.9) can be written as

$$\phi(\gamma) = U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big) + \sum_{j \in N_i} U_j\big((a_j^x, a_j^1), (a'_{-j}, a''_{-j})\big). \tag{3.10}$$

We now formally define a Potential Game as it is defined in [9], but we extend the definition so that it is for a two-step action sequence rather than a single action.

Definition 3.4 (Potential Games) *Player action sets $\{\alpha_i\}_{i=1}^n$, together with player objective functions $\{U_i : \alpha \to \mathbb{Z}\}_{i=1}^n$, constitute a Potential Game if, for some potential function $\phi : \alpha \to \mathbb{Z}$,*

$$\begin{aligned} &U_i\big((a'_i, a_i^1), (a_{-i}^1, a_{-i}^1)\big) - U_i\big((a''_i, a_i^1), (a_{-i}^1, a_{-i}^1)\big) \\ &= \phi\big((a'_i, a_i^1), (a_{-i}^1, a_{-i}^1)\big) - \phi\big((a''_i, a_i^1), (a_{-i}^1, a_{-i}^1)\big) \end{aligned} \tag{3.11}$$

for every player $i \in N$, for every $(a'_i, a_i^1) \in \alpha_i$, and for every $(a''_i, a_i^1) \in \alpha_i$

Notice in (3.11) that the second argument of the objective function of player i and the potential function is (a_{-i}^1, a_{-i}^1) implying that all players must "move forward" for two time steps or equivalently play their baseline actions while i is playing its

two-step action sequence. This is as per the test for a potential game as seen in [9], which requires that all players other than player i continue to play their previous action. Since player i is the one changing its action from its baseline action, none of the other players are allowed to change their actions from their respective baseline actions, and thus, must continue to play it. This is also consistent with the first part of Definition 3.2 of a Weakly Acyclic game (Section 3.1), which a potential game is a subclass of.

Claim 3.1 *Player objective functions* (3.7) *constitute a potential game with potential function* (3.9).

Proof. A similar approach as [15] will be used to prove the claim. We assume a player i is contemplating at time t whether to turn in one direction by performing the action sequence (a_i', a_i^1) or to turn in another direction by performing (a_i'', a_i^1) so that $a_i' \neq a_i''$. The change in the objective function of player i by switching from the action sequence (a_i'', a_i^1) to the action sequence (a_i', a_i^1), provided that all other players collectively play (a_{-i}^1, a_{-i}^1), is

$$\triangle U_i = U_i\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) - U_i\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big)$$

The first difference equation for the potential function of the game for the two different action sequences of player i is

$$\triangle \phi = \phi\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) - \phi\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big)$$

Substituting (3.10) into the above difference equation, we get

$$\begin{aligned}\triangle \phi = U_i\big((a'_i, a^1_i), (a^1_{-i}, a^1_{-i})\big) + \sum_{j \in N_i} U_j\big((a'_i, a^1_i), (a^1_{-i}, a^1_{-i})\big) \\ - \Bigg[U_i\big((a''_i, a^1_i), (a^1_{-i}, a^1_{-i})\big) + \sum_{j \in N_i} U_j\big((a''_i, a^1_i), (a^1_{-i}, a^1_{-i})\big)\Bigg] \quad (3.12)\end{aligned}$$

As previously mentioned, a player's predicted utility for a two-step action is independent of the sequence of actions played by the others. Due to this and the fact that every player j has to play its baseline action for two time steps while player i completes its turn sequence, we have

$$U_j\big((a'_i, a^1_i), (a^1_{-i}, a^1_{-i})\big) = U_j\big((a^1_j, a^1_j), (a^x_{-j}, a^y_{-j})\big) = U_j\big((a''_i, a^1_i), (a^1_{-i}, a^1_{-i})\big) \quad (3.13)$$

Now, we substitute (3.13) into (3.12) to get

$$\triangle U_i = \triangle \phi$$

□

Considering that the greater objective of this thesis is to get a group of robots to explore a space as quickly as possible, a solution that organizes these robots to achieve this can be thought of as projecting cooperative behaviour. After all, as mentioned earlier, the goal of exploration is to gain as much new information as possible of the environment within bounded time. Therefore, if a robot follows the tracks of another robot as part of a solution, which is to say that it moves through already explored space, the solution would not be portraying cooperative behaviour. This is because in the time that the robot spend moving through explored spaces, it could have been moving in a different path and exploring previously uncovered spaces, and possibly, reduce the overall time needed for exploration. Thus, a solution that engages robots

to take different paths or that minimizes overlaps can be thought of as instilling cooperative behaviour. Now, in this regard if we consider the objective function of a player, it is evident that no robot has anything to gain from following the path of another robot since it does not increase its utility in any way. Robots seek to follow different paths from one another, and thus, our objective function encourages cooperative behaviour in terms of achieving the greater objective.

3.3 Potential Game Setup

Based on (3.7), the goal of each robot is to maximize its utility by discovering new grid points. To do this each robot i has four action sequences at its disposal from the set α_i defined in Section 3.1. However, each robot has restrictions on the action it can use from one time step to the next if the team of robots is to reach a Nash equilibrium or if it is to project cooperative behaviour in the sense discussed in the previous section. Also, as mentioned in Section 3.1, modelling a nonholonomic system imposes further restrictions on the actions of a robot. Specifically, we analyze Potential games with state-dependent action sets where the set of actions available for a given player depends on the player's previous action. State-dependent action sets of this form are referred to as (range) restricted action sets [15]. As it will be seen in the following subsection, all these restrictions on the actions are abided by the potential game if it adheres to a modified version of the learning algorithm known as Spatial Adaptive Play (SAP) [15, 26–28] under the framework of the Simple Forward Turn Controller. Furthermore, in Section 4.1 we see that having our Potential game adhere to the SAP allows for strong coordination or cooperation among players to have the space explored quickly.

3.3.1 Spatial Adaptive Play (SAP) and Restricted Actions

In SAP, at each time $t > 0$, one player $i \in N$ is randomly chosen and allowed to update its action. All other players must repeat their actions so that $a_{-i}(t) = a_{-i}(t-1)$ [15]. At time t, the updating player i selects an action from A_i based on its strategy $p_i(t) \in \triangle(A_i)$, which is a probability distribution over its action space, so that the a_ith component $p_i^{a_i}(t)$ of its strategy is given as

$$p_i^{a_i}(t) = \frac{exp\{\beta U_i(a_i, a_{-i}(t-1))\}}{\sum\limits_{\bar{a}_i \in A_i} exp\{\beta U_i(\bar{a}_i, a_{-i}(t-1))\}} \tag{3.14}$$

for some exploration parameter $\beta \geq 0$. Note that β is different from the exploration rate ϵ discussed earlier. Constant β determines the likelihood that player i will select a suboptimal action. If $\beta = 0$, player i will select any action $a_i \in A_i$ with equal probability [15]. As $\beta \to \infty$, player i will select an action from its best response set

$$\left\{ a_i \in A_i : U_i(a_i, a_{-i}(t-1)) = \max_{a_i' \in A_i} U_i(a_i', a_{-i}(t-1)) \right\} \tag{3.15}$$

with arbitrarily high probability.

In [27], it was shown that in a repeated Potential game that adheres to the SAP, the stationary distribution $\mu \in \triangle(A)$ of the joint action profiles is given as

$$\mu(a) = \frac{exp\{\beta\phi(a)\}}{\sum\limits_{\bar{a} \in A} exp\{\beta\phi(\bar{a})\}} \tag{3.16}$$

The stationary distribution μ can be interpreted as follows: For sufficiently large times $t > 0$, $\mu(a)$ is equal to the probability that $a(t) = a$ [15]. As β approaches ∞, all the weight of the stationary distribution is on the joint actions that maximize the potential function [15]. In a Potential game, joint actions that maximize the Potential function induce cooperation among robots. Thus, a Nash equilibrium can be

asymptotically reached with a sufficiently large β when the Potential game adheres to SAP. For this reason we have our Potential game with potential function (3.8) adhere to the SAP algorithm, and assume a value of ∞ for β so that actions are always chosen from the best response set. This forms the action-policy or the strategy $p_i(t) \in \triangle(A_i)$ of a player i in our Potential game. In other words, each of the players' action-policy in our Potential game is defined by setting β in the corresponding SAP algorithm to infinity. This will be seen in Section 3.3.2.

We can now provide an explanation to the modification we made to the original Simple Forward Turn Controller in Section 3.1.3. In our solution, for a player to choose an action sequence from the best response set, it has to know the utilities for every two-step action sequence. If it were to randomly select an action sequence from the action sequence set, there is no guarantee that it would be from the best response set. This justifies the modification made to the Simple Forward Turn Controller in this thesis.

We note that in the SAP algorithm, the restriction of allowing only one player to update its action while having the rest of the players repeat their actions has a striking resemblance to the first part of Definition 3.2 of a Weakly Acyclic game (Section 3.1). This makes sense because a Potential game is a subclass of the Weakly Acyclic game with additional restrictions. Thus, one would expect the same restrictions of a Weakly Acyclic game to apply to a Potential game. However, as noted in Section 3.1, it is not practical to have a centralized entity to determine which robot will change its action at every time step. Thus, the part of the SAP algorithm that requires a player to be randomly selected to allow it to update its action cannot be held here. Instead as discussed in Section 3.1, we allow each robot to change their actions at a small specified rate, ϵ, knowing that there is a small probability that the Nash equilibrium will not be reached, which is equivalent to knowing that cooperation between robots will marginally decrease. This forms the basis of the modification to the original SAP

algorithm. In the Simple Forward Turn Controller, allowing a robot to change its action with a rate of ϵ corresponds to the robot deviating from playing its baseline action to perform a turn sequence over a two-step action sequence (see Section 3.1.2). Any robot that does not perform a turn sequence plays its baseline action with a probability of $1 - \epsilon$.

Analyzing the Simple Forward Turn Controller further, we note that there are more restrictions on actions that can be played, which stem from accounting for inflexibilities in the range of motion experienced in a nonholonomic environment. Consider for example that a robot wants to change its baseline action and then play its new baseline action all within one time step. Recall from Section 3.1.1 that a baseline action is the combination of the "move forward" command (a_i^1) and the pose of the robot at a particular time. Under the framework of the Simple Forward Turn Controller, the robot cannot do so because it needs a full time step to change its heading. It cannot simultaneously change its heading and move forward in a single time step. It needs to do so over the course of a two-step action sequence. Thus, the ability of a robot to execute a "move forward" command in a particular direction at a particular time t is dependent on its baseline action in the previous time step $t-1$. If the direction the robot wants to move in coincides with its baseline action from the previous time step, the robot can "move forward" in that direction at time t. Otherwise, it has to perform the appropriate turn action from the action set $(A_i \setminus a_i^1)$ at time t, and then "move forward" in the specific direction it wants to move in, in the subsequent time step. Once the robot has performed the turn sequence, it can set the direction it is facing towards as the new baseline action. Thus, if a robot is changing its baseline action, it has a limited set of actions to choose from at time t. This subset of actions of the action space A_i are referred to as a restricted action set (i.e. $(A_i \setminus a_i^1) \subset A_i$) [15]. As these restrictions are a fairly accurate representation of the dynamics of a nonholonomic system, we use the Simple

Forward Turn Controller in our algorithms to make them realizable on real robotic platforms. Furthermore, since the movements of a robot in a Simple Forward Turn controller resemble the movements of a robot in our Potential game as it was defined in Section 3.2; namely in that they are both defined by a two-step action sequence, the framework of the Simple Forward Turn Controller is ideal for our Potential game. The following subsection will introduce an algorithm for exploration that is premised on the modified SAP algorithm discussed in this subsection and the Simple Forward Turn Controller.

3.3.2 An Algorithm for Exploration

Using the Simple Forward Turn Controller, if a robot is allowed to update its action at time t based on the exploration rate ϵ, it will predict the utility it would receive by performing each of the four two-step actions as discussed in Section 3.1.3. This is done for the baseline action as well where $a_i^x = a_i^1$ in (a_i^x, a_i^1). The action selection function, asf_i, then compares the utility for each two-step action sequence and selects the action sequence that would give it the most amount of utility.

$$asf_i = \underset{(a_i^x, a_i^1) \in \alpha_i}{\operatorname{argmax}} U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big) \tag{3.17}$$

Equation (3.17) essentially represents the best response set in the SAP algorithm, and thus, choosing an action from this set is in accordance with what was discussed in Section 3.3.1 about setting the value of β to infinity. Any ties for the predicted utility are broken arbitrarily unless the two-step action sequence involving the baseline action (i.e. (a_i^1, a_i^1)) happens to have the same utility as the maximum utility in which case the baseline action is performed. This is in accordance with the second part of Definition 3.2 in Section 3.1. This means that there is a very low probability that at a time t a robot i will change its action because firstly ϵ is small, and secondly, even

if the robot has the option to change its action with a probability of $1-\epsilon$, the action that it is changing to must provide it with higher utility than the baseline action. If at time t a robot is not allowed to change its action based on ϵ, it has to play its baseline action. Algorithm 1 summarizes the algorithm for exploration. Note that U_i is short for $U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big)$ in Algorithm 1, and the action *Play* (a_i^x, a_i^1) *based on* asf_i takes into account the tie-breaking rule just discussed.

Algorithm 1 Potential Game Exploration Algorithm

```
Initialize # of time steps, sensRange, and ε

for t ← 1, # of time steps do
    for all player i ∈ N do

        Update pos_i(t), discPts_i(t), and discPts_{-i}(t)    //eq. (3.5) and eq. (3.6)

        if player should explore based on ε then
            Compute U_i , ∀(a_i^x, a_i^1) ∈ α_i                //eq. (3.7)
            Play (a_i^x, a_i^1) based on asf_i                 //eq. (3.17)
        else
            Play (a_i^1, a_i^1)
        end if

    end for
end for
```

At a time t a robot i only knows where it is and all the grid points it has discovered. It then queries all the other robots for their position $(loc_j(t), j \in N_i)$ and all the grid points they have discovered to calculate $discPts_{-i}(t)$. Recall that $discPts_{-i}(t)$ is the set of Cartesian coordinates of the grid points that have been discovered by all players except i up until time t. As mentioned in Section 3.2 it is not necessary that at time t all the grid points discovered by a robot i were exclusively discovered by it. This is because there can be times when two or more robots have overlapping sensor coverage, which can lead to a grid point being discovered by more than one robot. However, due to the fact that a robot mostly moves straight (since $\epsilon \ll 1$), in a large environment

they quickly spread apart if they all begin in relatively the same location with different orientations so that overlapping sensor coverages quickly diminish. Furthermore, recall from Section 3.2 that our Potential game leads to cooperative behaviour in the sense that robots seek to follow different paths from one another when exploring. This is because based on the objective function defined in (3.7), no robot has anything to gain from following another robot's path or running into another robot's path. In comparison, if robots were completely uncoordinated and could perform any action whenever they wanted (i.e. $\epsilon = 1$), they would have much more frequent run-ins or overlaps with other robots over uncovering the same grid points so that there would be a higher probability that robots would explore the same areas. This would make the exploration process inefficient. This is how coordination is achieved in Algorithm 1 over an uncoordinated algorithm. Section 4.2 will present results that show how a variant of Algorithm 1 outperforms an uncoordinated algorithm in the exploration of a finite space.

3.4 Unbounded Game Simulation

In this section, we look at simulating Algorithm 1. Before we do so, however, we note a very important limitation of the algorithm. This will be exemplified through the scenario shown in Fig. 3.2, which shows the poses and positions of two robots in a 6×6 grid at time t. The boundaries of the grid indicated by bold lines represent walls or obstacles. If at time t both robots were to play an action according to the exploration policy ϵ, it is highly likely that both would play their baseline actions (i.e. "move forward" with their current poses). In fact, there is a probability of $(1 - \epsilon)$ that either robot will play its baseline action. This would cause them to run into the walls or obstacles. If there were no obstacles present and if the space to be explored was unbounded this would not be an issue, and so we could just use the

algorithm discussed in the previous section in its entirety. However, since our interest lies in exploring a finite space that has well-defined boundaries and obstacles within it, in practice we cannot have the robots running into the obstacles. Thus we need to modify the algorithm. This modification and its effects on the game are discussed in Section 4.1.

To simulate the game for the unbounded scenario, we create a very large square grid in MATLAB and place the robots at locations so that the distance between a robot and the other robots are negligible in comparison to the dimensions of the grid. The exact positions and poses of the robots can be arbitrarily selected so long as the collection of robots are situated somewhere near the centre of the grid to prevent them from hitting the outter walls. Figure 3.3 illustrates an arbitrary set of positions and poses for 5 robots that we use in our simulation. Each of the robots have a 360° sensor coverage (i.e. $sensRange$) with a radius of 5 grid points and ϵ is 0.3. The simulation is averaged over 500 games, with each game consisting of 35 times steps. The game was averaged over 500 games because it resulted in a smooth profile for the graph. Averaging over more than 500 games would perhaps result in a smoother profile, but it would take longer to obtain the results. Each game was chosen to have 35 time steps because it is past the time that Nash equilibrium is reached as it will be seen shortly, and 35 time steps is sufficient for the values of the graph to settle. Figure 3.4 is an example from the 500 games of the final orientation of each of the robots. Notice that robots 1 and 3 have changed their orientations in Fig. 3.4 in comparison to Fig. 3.3. This is because at some time between the events in Fig. 3.3 and Fig. 3.4, robots 1 and 3 would have been allowed to update their actions to receive more utility. They would have made a turn as a result of running into other robots' explored spaces. The robots keep executing the "move forward" command (a_i^1) with their respective orientations (i.e. their baseline actions) for every time step that follows the time the snapshot in Fig. 3.4 was taken. This is because

the environment is unbounded, which means that as time goes to infinity, the infinite space cannot be fully explored. Thus, after a certain time once the robots have had a chance to spread out, robots will only play (a_i^1, a_i^1) with their final orientations (i.e. baseline action sequence) because they would receive the most amount of utility in doing so for every time step till infinity compared to if they had to make additional turns. This is a direct result of the definition of the objective function for a robot in (3.7), where in the presence of unexplored space ahead of the robot, the highest rewarding action sequence is the baseline action sequence. The final orientations of the robots is a Nash equilibrium because no robot can unilaterally change its action sequence from its baseline action sequence to gain more utility than their baseline action sequence would provide. In other words, a Nash equilibrium occurs when the robots stop changing actions. To determine the Nash equilibrium in our simulation we keep a running total of the number of turn-actions performed by each robot for every time step and then add them up. As the Nash equilibrium is approached, there is very minimal change in this running total. For the particular example in Fig. 3.4, 12 times steps were taken to reach the Nash equilibrium. Fig. 3.5 shows the result of the simulation averaged over the 500 games. The weight of the stationary distribution μ in this Nash equilibrium is fully on the joint action vector $(a_1^1, a_2^1, a_3^1, a_4^1, a_5^1)$. That is each player plays their respective baseline action for each time step once the Nash equilibrium has been reached. This is as expected because as discussed in Section 3.3.1, setting β to infinity puts all the weight of μ on the joint actions that maximize the potential function. In our game, the joint actions that maximize the potential function once the Nash equilibrium has been reached is each player's baseline action.

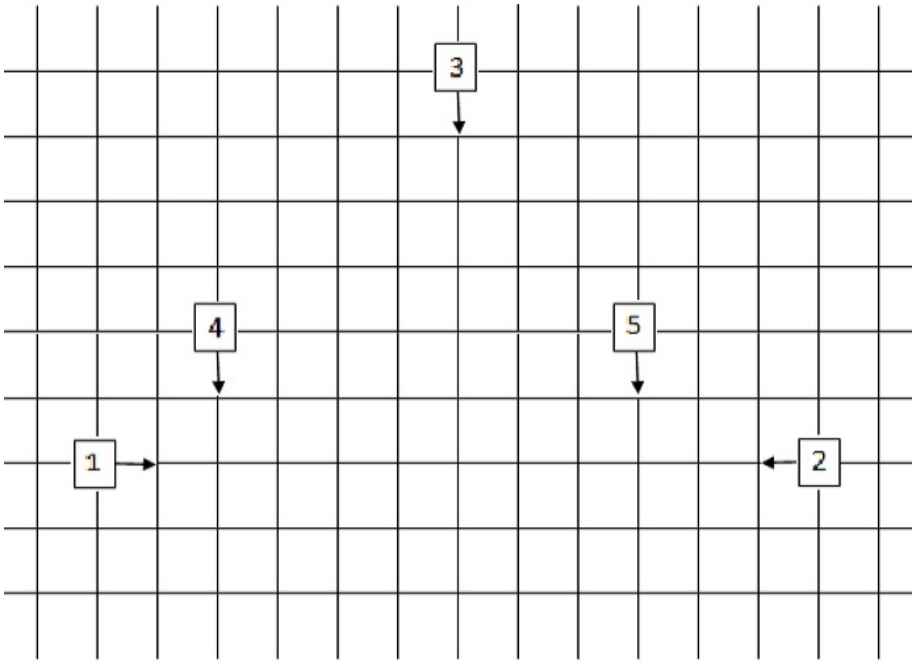

Figure 3.3: Unbounded simulation with 5 robots

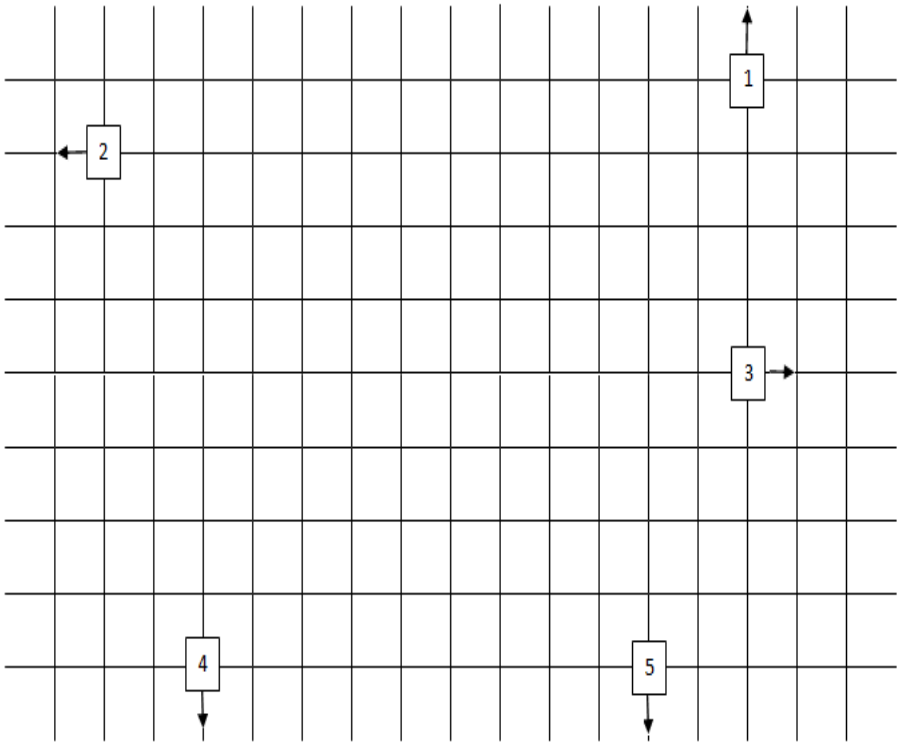

Figure 3.4: Final orientation of robots after reaching Nash equilibrium

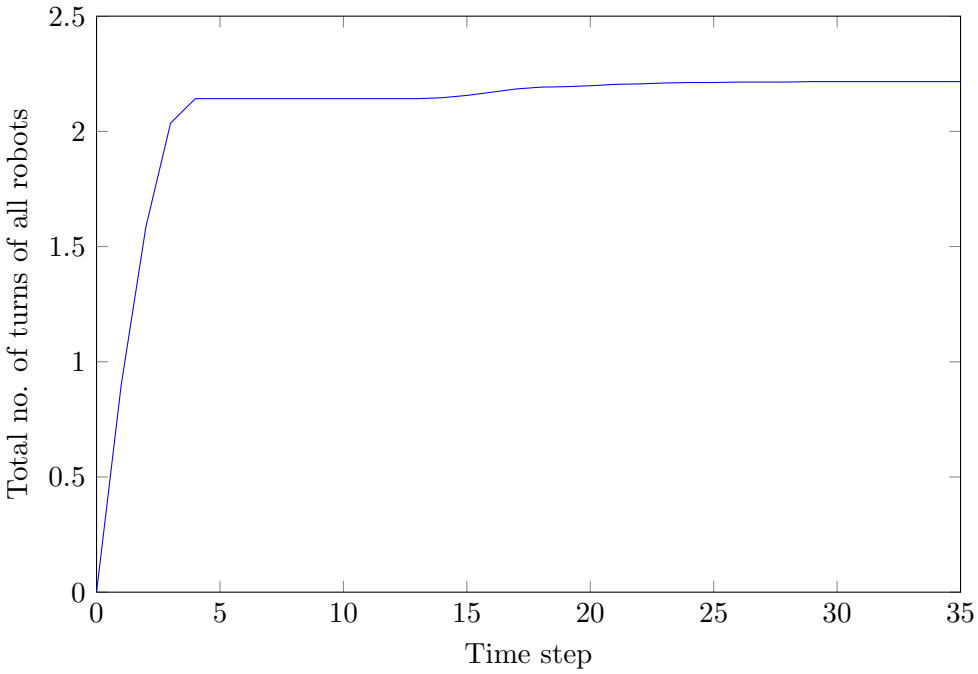

Figure 3.5: Nash equilibrium in unbounded grid game

3.5 Summary

In this section, we presented the Weakly Acyclic game and definitions associated to it such as better-response actions and better-response path. It was discussed how the Simple Forward Turn Controller was devised to solve the Consensus problem as a Weakly Acyclic game in [1]. However, the Weakly Acyclic game was deemed to be inadequate to solve the Consensus problem because it did not assert the notion of utility alignment between a player's objective function and the global objective function. Instead the Potential game, a subclass of the Weakly Acyclic game, was found to be better suited in solving the Consensus problem under the framework of the Simple Forward Turn Controller. In our solution for exploring a space, we proposed a Potential game similar in structure to that in [1], but defined a new potential function and objective function to reflect our goal. Furthermore, the definition of a Potential game itself was extended in this chapter for a two-step action sequence, which serves

as a contribution of this thesis. This Potential game was hypothesized to promote cooperative behaviour in the sense that robots would try to move in different paths when exploring a space, which in turn would reduce exploration time compared to an uncoordinated algorithm. This hypothesis will be confirmed in Section 4.2. Also, it was seen how unlike frontier-detection algorithms, each player does not have to observe the decision of all players in evaluating its payoff for a particular action choice. Another modification we proposed in this chapter was to the Simple Forward Controller; namely, when a player is allowed to update its action, it chooses an action from the best response set (3.15) rather than randomly selecting an action and playing it if it provides more utility than the current action (baseline action). It was seen that having a player select actions from its best response set could be realized by setting β to infinity in the corresponding SAP game. It was then discussed how the provision of restricted action sets in the Simple Forward Turn Controller is useful for modelling a nonholonomic environment. Finally, an algorithm was proposed for the exploration of an unbounded or infinite space, which when applied to a group of five robots resulted in a Nash equilibrium.

Chapter 4

Modified Potential Game Algorithms

4.1 Modified Algorithm for Bounded Spaces

To address the limitation identified in Section 3.4 of robots running into walls or obstacles, we seek to modify Algorithm 1 so that robots can traverse environments with obstacles. Since we do not want robots running into obstacles and walls, a simple solution to this problem would be for a robot to change its heading when it encounters an obstacle in front of it even if at that particular moment it is not allowed to perform a turn sequence as dictated by its exploration policy ϵ. The direction the robot would turn would be the direction that results in the highest utility. This is shown in Algorithm 2. We can immediately perceive the repercussions of this modification as the obstacles would cause robots to change actions more often than ϵ. In this respect, the presence of obstacles can be considered to have the equivalent effect of increasing ϵ from the value it was initialized to, which as discussed in Section 3.1 and Section 3.3 would decrease the probability that the Nash equilibrium will be reached. However, reaching a Nash equilibrium as it is described in Section 3.4 is not our goal here. Our goal is to fully explore a finite space in as little time as possible. We demonstrate that the Modified Potential Game Algorithm (Algorithm 2) reduces exploration time compared to a completely uncoordinated exploration

Algorithm 2 Modified Potential Game Algorithm

Initialize $\#\ of\ time\ steps$, $sensRange$, and ϵ

for $t \leftarrow 1, \#\ of\ time\ steps$ **do**
  for all player $i \in N$ **do**

    Update $pos_i(t)$, $discPts_i(t)$, *and* $discPts_{-i}(t)$ //eq. (3.5) and eq. (3.6)

    if *player should turn because of obstacle* **then**
      Compute U_i , $\forall(a_i^x, a_i^1) \neq (a_i^1, a_i^1)$ //eq. (3.7)
      Play (a_i^x, a_i^1) *that maximizes* U_i
    else if *player should explore based on* ϵ **then**
      Compute U_i , $\forall(a_i^x, a_i^1) \in \alpha_i$
      Play (a_i^x, a_i^1) *based on* asf_i //eq. (3.17)
    else
      Play (a_i^1, a_i^1)
    end if

  end for
end for

algorithm. Before we can perform a comparison though, we need to define one such uncoordinated exploration algorithm. A simple example would be an algorithm in which each robot always moves in the direction that provides it the most amount of utility with any ties in the utility being arbitrarily broken. This is shown in Algorithm 3. In contrast, in Algorithm 2, robots can only move in the direction that provides its the most amount of utility when ϵ allows them to do so. It can be shown through simulations that using Algorithm 3, all the areas that need to be explored in a finite space will be explored in finite time.

4.2 Simulation of Exploration Algorithm

To compare Algorithm 2 and Algorithm 3 we perform simulations in a programmable multi-agent modeling environment known as NetLogo. NetLogo uses an agent-based

Algorithm 3 Uncoordinated Exploration Algorithm

Initialize $\#\ of\ time\ steps,\ sensRange,$ and ϵ

for $t \leftarrow 1, \#\ of\ time\ steps$ **do**
  for all player $i \in N$ **do**

    Update $pos_i(t),\ discPts_i(t),\ and\ discPts_{-i}(t)$ //eq. (3.5) and eq. (3.6)

    if *player should turn because of obstacle* **then**
      Compute $U_i\ ,\ \forall(a_i^x, a_i^1) \neq (a_i^1, a_i^1)$ //eq. (3.7)
      Play (a_i^x, a_i^1) *that maximizes* U_i
    else
      Compute $U_i\ ,\ \forall(a_i^x, a_i^1) \in \alpha_i$
      Play (a_i^x, a_i^1) *that maximizes* U_i
    end if

  end for
end for

programming language. To compare the two algorithms we use the setup in Fig. 4.1 consisting of three robots and three obstacles indicated by the rectangular boxes. In both algorithms *sensRange* is set to 2 grid points, and ϵ is set to 0.3 for Algorithm 2. Note that ϵ does not apply to Algorithm 3 as in Algorithm 3, each robot always moves in the direction that provides it the most amount of utility. The simulation is run for 2000 time steps or iterations and averaged over 20 games. Fig. 4.2 shows the result of the simulation where the number of grid points that remain to be explored at every time step is plotted for both algorithms. It can be seen that the Modified Potential Game Algorithm requires less time to explore the entire space. In fact it can be seen that the Uncoordinated Exploration Algorithm has not even uncovered the entire space in 2000 time steps.

Further simulations were performed to see the affect of changing the parameters ϵ and *sensRange* on exploration time. Fig. 4.3 compares Algorithm 2 and Algorithm 3 for $\epsilon = 0.3$ and $sensRange = 3$ grid points. Fig. 4.6 and Fig. 4.7 compares the

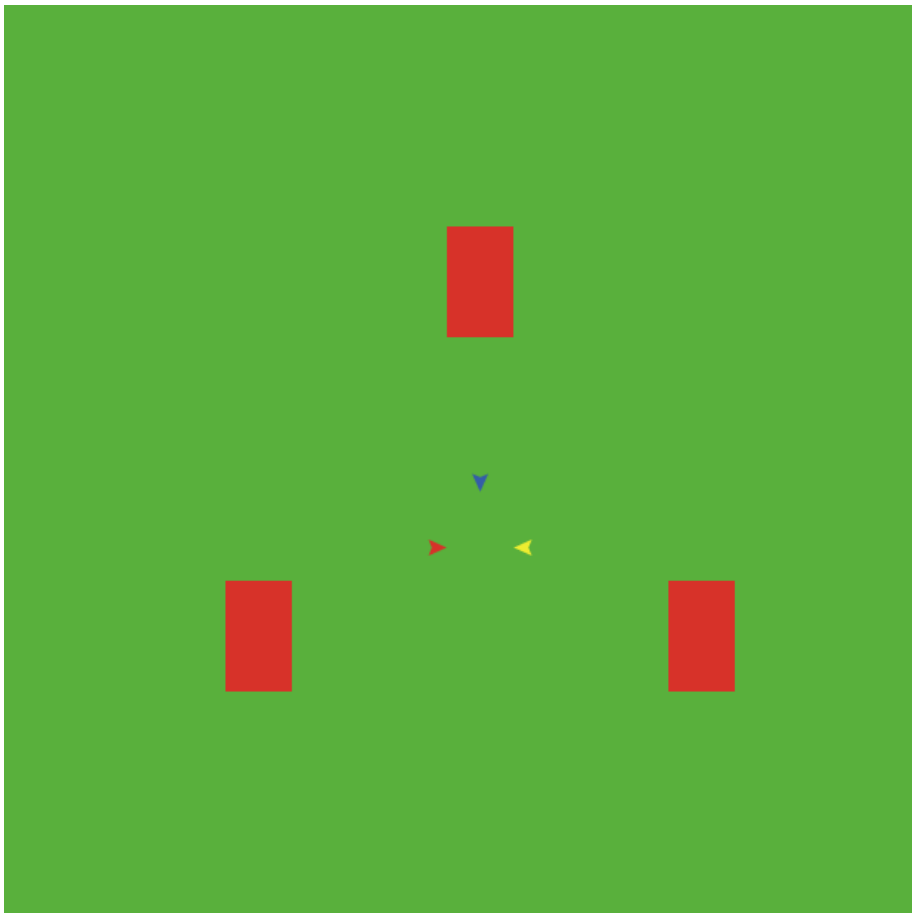

Figure 4.1: Simulation setup for Algorithm 2 and Algorithm 3

algorithms for $\epsilon = 0.3$ and $\epsilon = 0.1$ respectively, while keeping *sensRange* constant at 4 grid points. We observe that in comparing Fig. 4.2 and Fig. 4.3, Algorithm 2 does not provide as significant an improvement in terms of exploration time over Algorithm 3 as *sensRange* is increased from 2 to 3 grid points. In fact, in Fig. 4.3 the rate of exploration is greater with Algorithm 3 than Algorithm 2 in the first 200 time steps. This can also be seen in Fig. 4.6 where *sensRange* is 4 grid points. Fig. 4.4 shows how significantly exploration time improves for Algorithm 3 as *sensRange* increases. Exploration time is also reduced for Algorithm 2 as *sensRange* increases, but the improvement is not as profound as Algorithm 3. This can be observed in Fig. 4.5 for increasing values *sensRange* and a constant value of 0.3 for ϵ. Despite

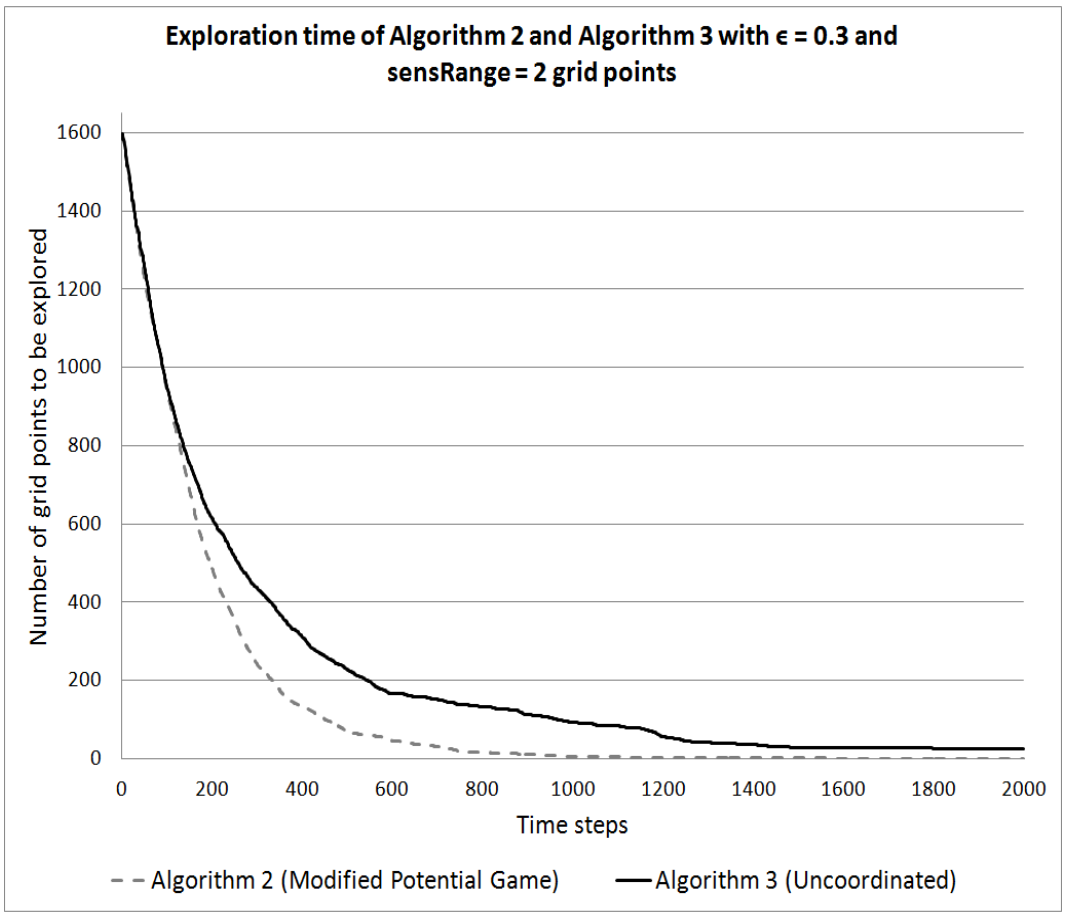

Figure 4.2: Comparison of exploration time of Algorithm 2 and Algorithm 3 with $\epsilon = 0.3$ and $sensRange = 2$ grid points

these performance improvements of Algorithm 3 with increasing *sensRange* values, it is evident through Fig. 4.2, Fig. 4.3, and Fig. 4.6 that the overall time needed for exploration is still lower with Algorithm 2 than Algorithm 3. Moreover, in real-world multi-robot platforms, it can be costly to equip individual robots with sensors such as laser range finders (LRF) that have a longer range. This is especially true if there is a large fleet of robots. Thus, the potential benefits of an uncoordinated algorithm such as Algorithm 3 in increasing the rate of exploration cannot be realized in these situations.

Comparing Fig. 4.6 and Fig. 4.7, we note that there is not much difference in the exploration time of Algorithm 2. In fact, it was found that as long as ϵ remained

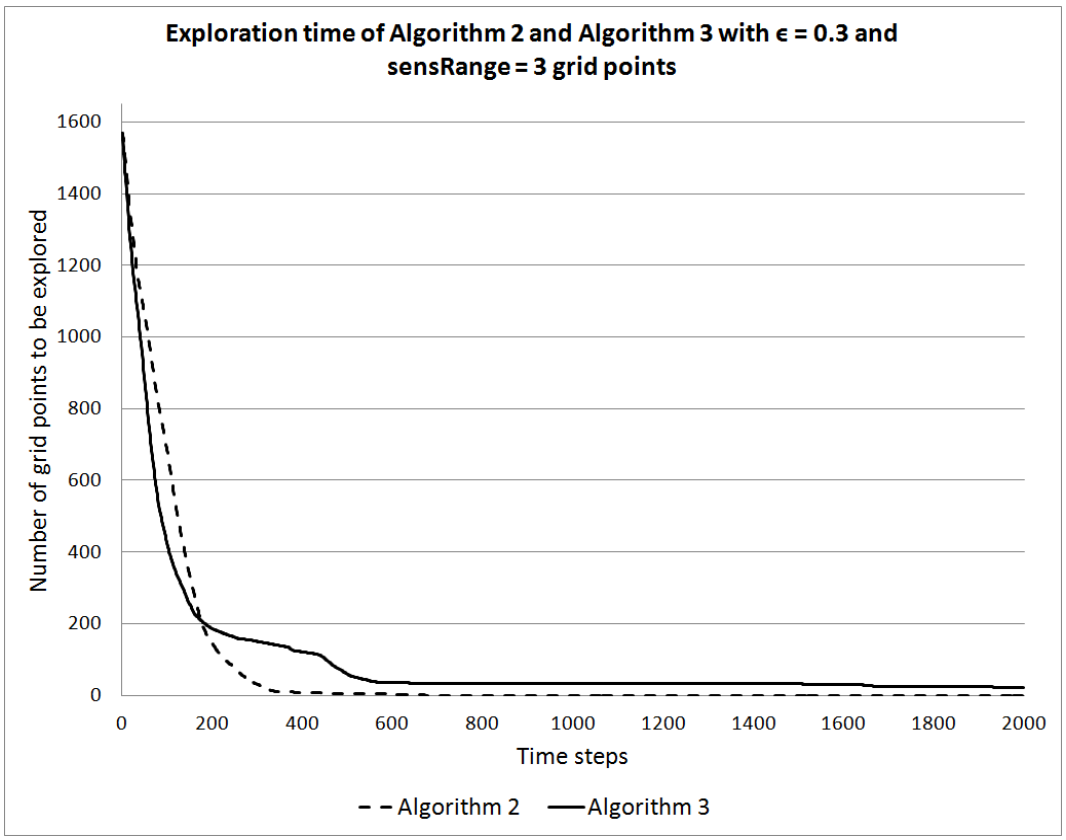

Figure 4.3: Comparison of exploration time of Algorithm 2 and Algorithm 3 with $\epsilon = 0.3$ and $sensRange = 3$ grid points

under a value of 0.3, the exploration time of Algorithm 2 did not change appreciably.

The Modified Potential Game Algorithm has an advantage over uncoordinated algorithms in that it introduces a certain degree of coordination among the robots even if they do not reach an equilibrium so that they do not all follow the same path when they are exploring. Sensor overlaps are reduced in this way. Thus, the time it would theoretically take to explore a space would be less than an algorithm that does not utilize any form of coordination, which explains the differences in performance of the algorithms in Fig. 4.2. However, the more obstacles there are in a given space, the more the robots have to employ obstacle avoidance manoeuvres in the Modified Potential Game Algorithm making the game deviate more from the original potential game. Thus coordination would be lost in terms of having robots move in different

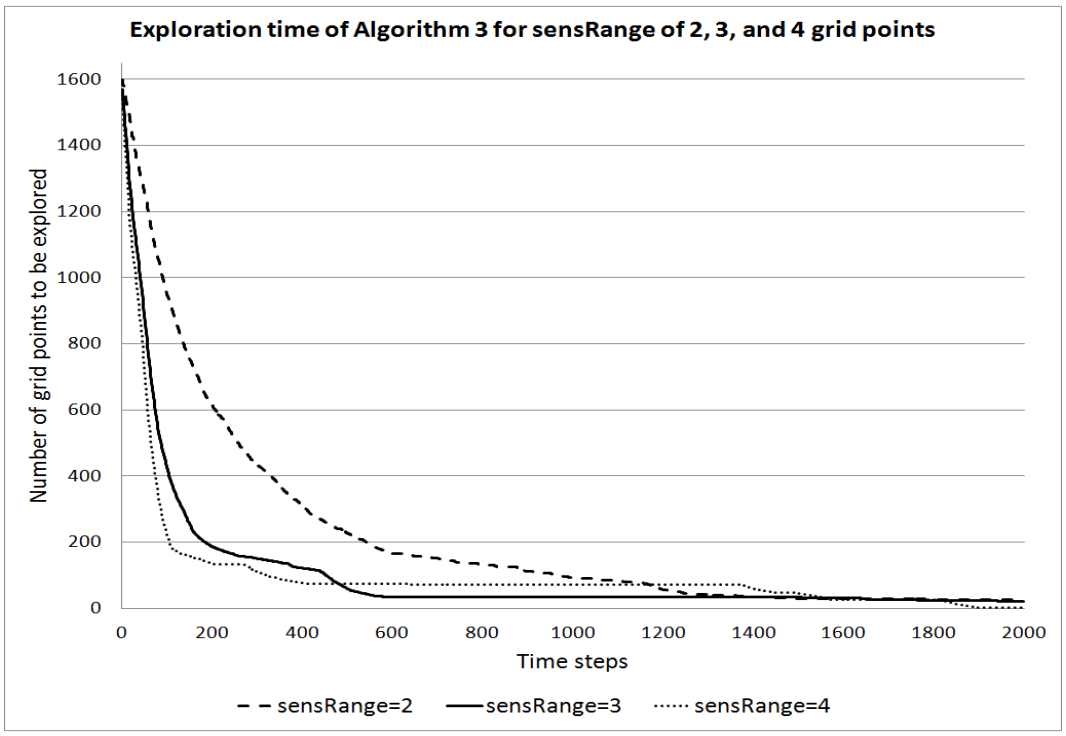

Figure 4.4: Comparison of exploration time of Algorithm 3 for a *sensRange* of 2,3, and 4 grid points

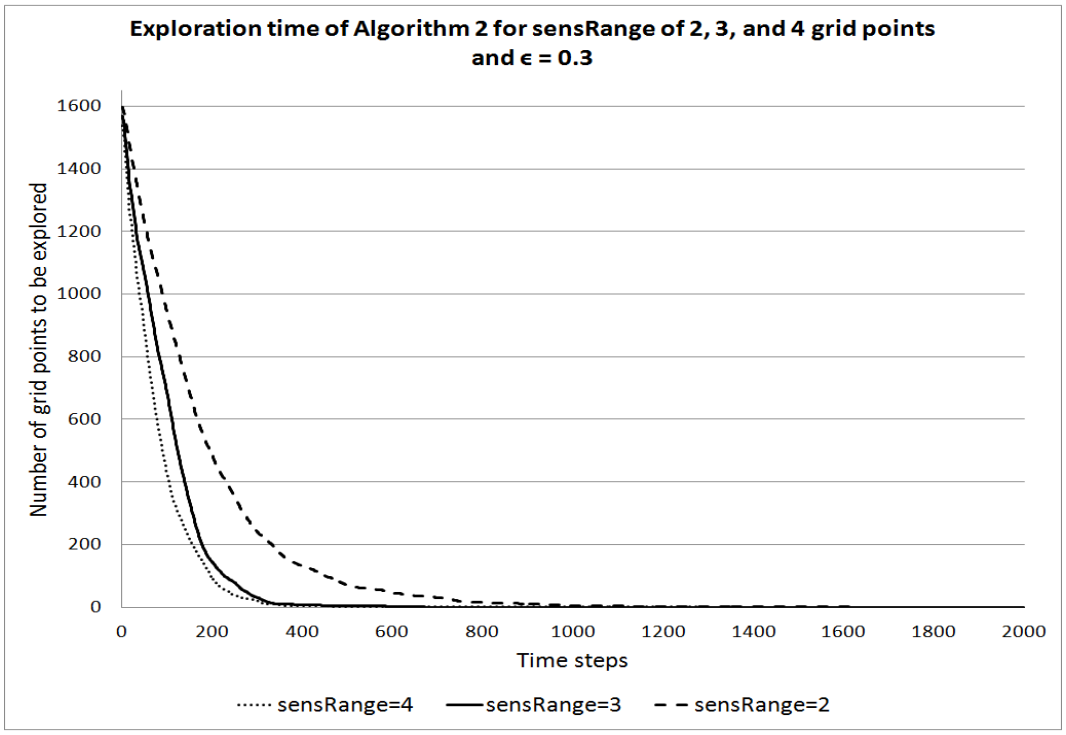

Figure 4.5: Comparison of exploration time of Algorithm 2 for a *sensRange* of 2,3, and 4 grid points, and $\epsilon = 0.3$

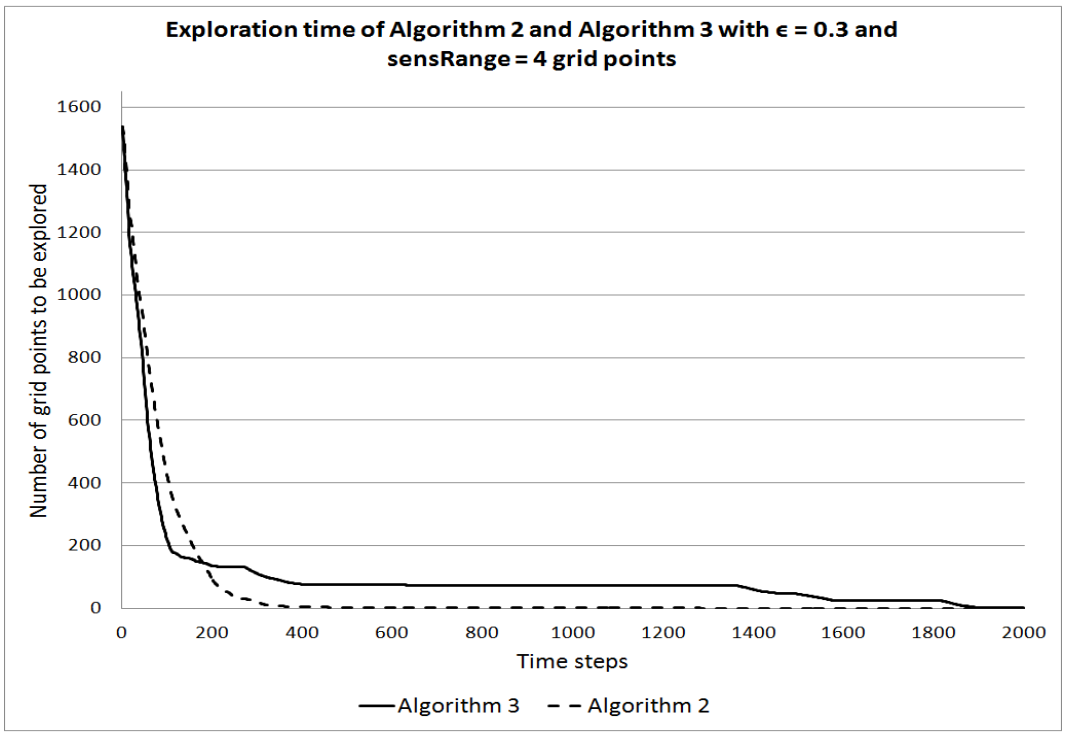

Figure 4.6: Comparison of exploration time of Algorithm 2 and Algorithm 3 with $\epsilon = 0.3$ and $sensRange = 4$ grid points

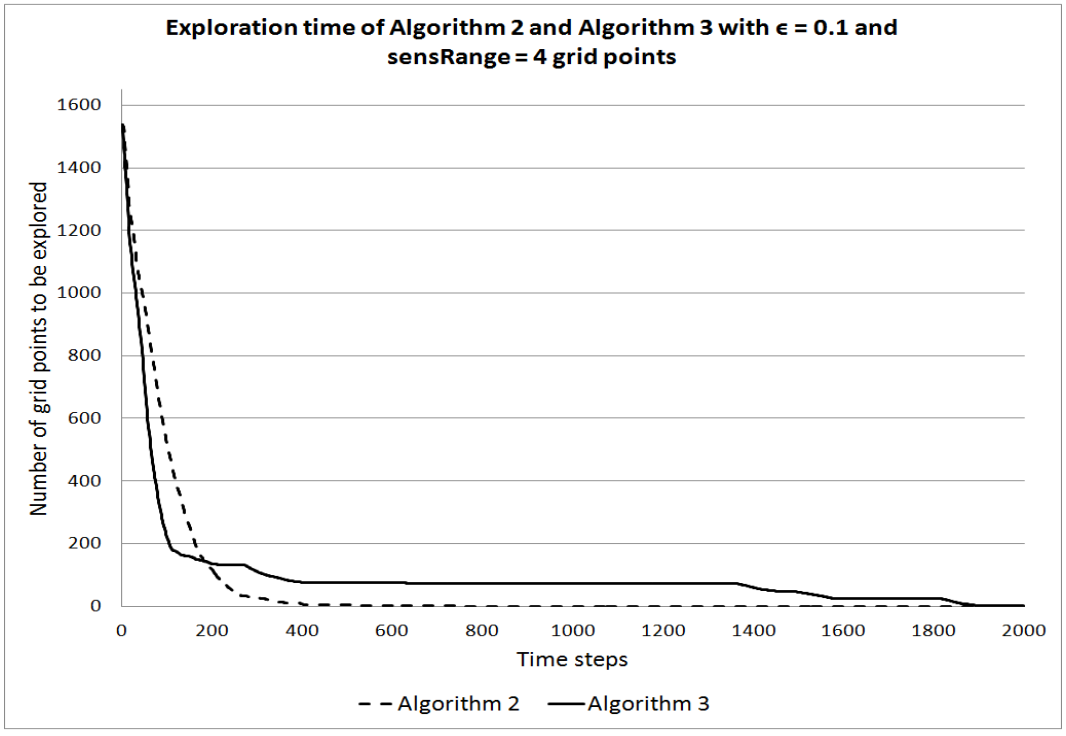

Figure 4.7: Comparison of exploration time of Algorithm 2 and Algorithm 3 with $\epsilon = 0.1$ and $sensRange = 4$ grid points

paths and improvements in exploration time over an uncoordinated algorithm would not be as profound. This will be demonstrated through a simulation in Section 4.4.1. Another important point to notice is that a coordinated behaviour would provide significantly greater improvements the more robots we have in a given space. Therefore, using many robots for a given space with a minimal number of obstacles, we can expect to receive significant improvements in exploration time using the Modified Potential Game Algorithm compared to uncoordinated algorithms.

4.3 Computational Complexity of Algorithm

In this section, we analyze Algorithm 2 to determine its computational complexity. The material presented in Section 2.6 serve as a brief, but concise review of the big O notation, which we use in this section to represent the runtime order of our algorithm. Before we analyze Algorithm 2, however, we investigate the computational complexity of frontier-based exploration algorithms. This gives us a base upon which we can compare and comment on the performance of our algorithm.

As mentioned in Section 1.1, most approaches today use frontier-based exploration for having a space explored using multiple robots. In frontier-based exploration, robots explore by repeatedly computing and moving towards frontiers, which is the boundary that separates known regions from unknown regions [11]. Computing the cost of reaching frontiers or *frontier detection* as it is referred to, involves the use of a deterministic variant of *value iteration*, a popular dynamic programming algorithm [9, 10]. In [29], it was shown that for deterministic Markov Decision problems (DMDP), basic value iteration takes $\Theta(Z^2)$ iterations, where Z denotes the number of states. Thus, it cannot do better than an $O(Z^2)$ algorithm in terms of execution time if we just consider the upper bound of its growth rate (See (2.2) and (2.3) in Section 2.6). In frontier detection algorithms, the states correspond to the cells in the explored

area. Considering that frontier detection algorithms processes all the states every time it performs frontier detection, it can be a time consuming process which slows down exploration [11]. In fact, even on powerful computers, state-of-the-art frontier detection algorithms can take a number of seconds to run for every execution of the algorithm, and if a large region is explored, the robot actually has to wait in its spot until the frontier detection algorithm terminates [11]. To make matters worse, there are frontier-based algorithms such as the algorithm presented in [30] that suggest calling frontier detection every time-step of the coordination algorithm.

There are two important points we note about Algorithm 2 before we analyze its runtime. The first point is that Algorithm 2 is a distributed algorithm. Hence, when we make a statement about its computational complexity, we are referring to an instance of its execution on one of the robots. Secondly, as a robot does not have to make a decision when it is forced to play its baseline action, it does not have to compute anything. In fact, it only seldom needs to calculate values. One occasion that it needs to compute values is when it is allowed to update its action as dictated by ϵ. The other occasion when it needs to compute values is when it needs to avoid an obstacle, and it needs to decide which direction to turn towards. As $\epsilon \ll 1$ and if we assume an environment with a large open space with few obstacles and few outer walls in comparison to the area of the overall space, robots would be moving straight most of the time with respect to the total time needed for exploration. They would not need to make many decisions resulting in a drastic reduction in the number of computations needed. More will be said about this in Section 4.4.1. Considering the aforementioned points, it is only of interest to us to analyze the computational complexity of Algorithm 2 for a robot i that is allowed to update its action at time t, and we proceed bearing this in mind.

As discussed earlier, the $argmax$ operator in the function asf_i (Equation (3.17)) selects the two-step action sequence from the two-step action sequence set α_i that

would give i the most amount of utility or that maximizes objective function (3.7). Based on the actions we defined in the action set A_i and the resulting two-step action sequence set α_i that was derived from it, player i has to compute the utility it expects to receive from performing each of the four two-step action sequences to make a decision. It namely has to compute the utility it would receive by playing (a_i^1, a_i^1), (a_i^2, a_i^1), (a_i^3, a_i^1), and (a_i^4, a_i^1). Figure 4.8 shows player i contemplating each of the two-step action sequences at time t in a $Z \times Z$ grid. The robot is the black box and the arrow on top of it represents the direction it is facing. The two solid dots ahead of the robot represents its predicted positions if it were to play its baseline action for the following two time steps (i.e. (a_i^1, a_i^1)). The solid dots to the left, bottom, and right of the robot represents its predicted positions at the end of the second time step after playing (a_i^2, a_i^1), (a_i^3, a_i^1), and (a_i^4, a_i^1) respectively. Recall from Section 3.2 and Section 3.3.1 that under the framework of the Simple Forward Turn Controller, when a robot performs a two-step action sequence that involves a turn, the robot remains in the same position for the first time step. Thus, over the course of a two-step action sequence the position would only change once, and this is why there is only one dot present to the left, bottom, and right of the robot in Fig. 4.8. The circles represent the 360° coverage of the sensor from the future positions, and the range of the sensor, $sensRange$, has been set to 2 grid points. In the objective function (3.7) for a player, the function $f[pos_i(t+2), pt]$ is evaluated for every point $pt \in gridpts$. This leads to Z^2 iterations of $f[pos_i(t+2), pt]$ as the grid is $Z \times Z$ in dimension. Evaluating $f[pos_i(t+2), pt]$ for a particular point pt is not intensive computationally because the majority of the function involves verifying whether or not pt belongs to the set $discPts_i(t)$ or $discPts_{-i}(t)$. This is as simple as maintaining a lookup table in memory in the form of an array and having a simple array indexing operation. Since retrieving a value from memory is very fast, cross-checking pt with already discovered points is an inexpensive operation. In the clauses $cond.1$ and $cond.2$ in (3.7), the operations

to determine whether $pt \notin discPts_{-i}(t)$ must be executed first because if it does not hold true, the magnitude function to determine if $||pt - loc_i(t+1)|| \leq sensRange$ is true or if $||pt - loc_i(t+2)|| \leq sensRange$ is true, does not need to be evaluated. Even if the magnitude function is required to be evaluated, the computation needed for it does not have any affect on the runtime order for iterating through all the points. Since there are four two-step actions sequences to be considered, $4Z^2$ iterations are needed, which is of order Z^2. Thus, the runtime order would be $O(Z^2)$.

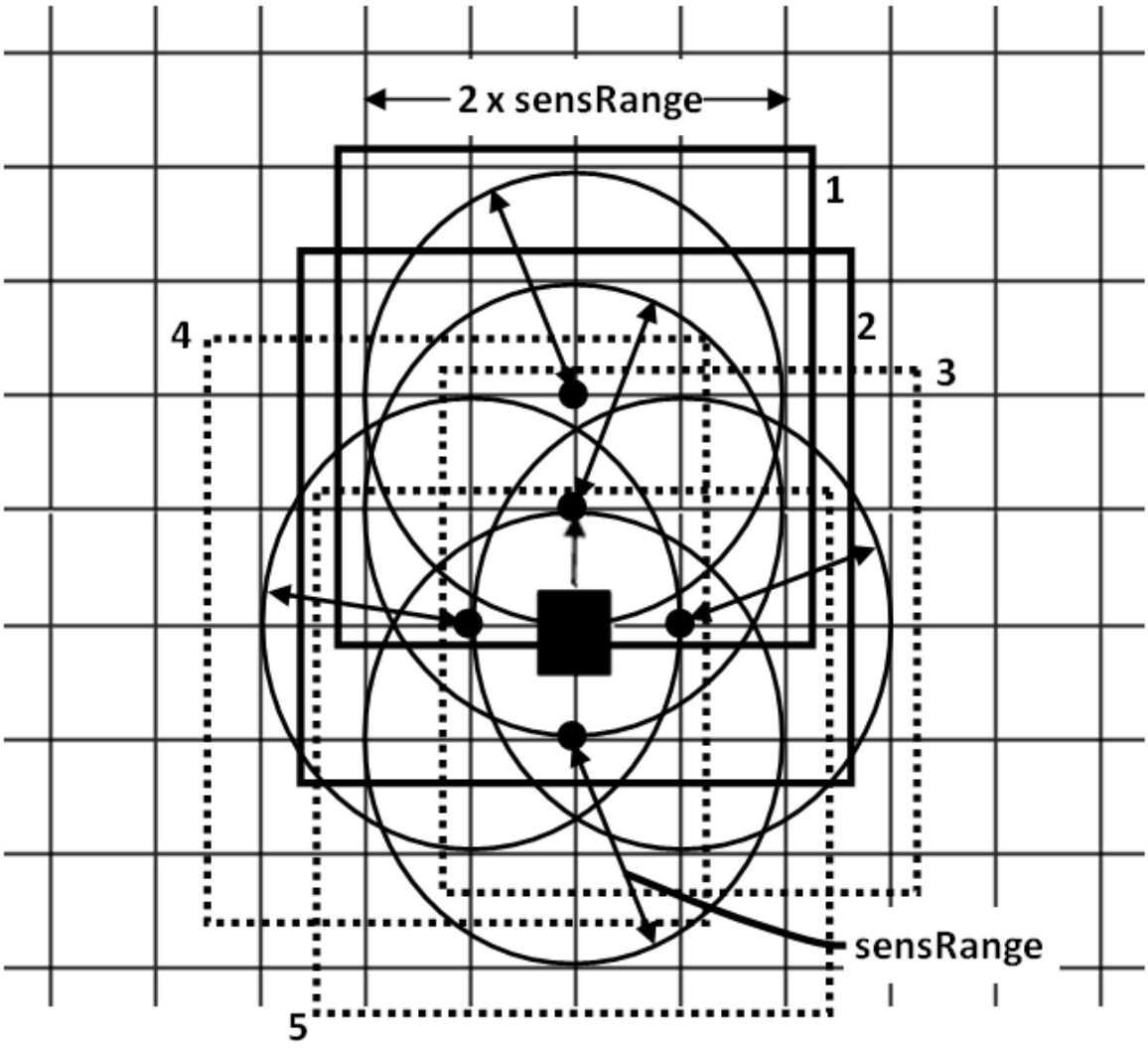

Figure 4.8: Player i considering each two-step action sequence

If the function $f[pos_i(t+2), pt]$ did not have to be evaluated for every point in

the grid, the complexity of computing the objective function (3.7) for a player i for an action sequence (a_i^x, a_i^1) could be reduced. Since the points that have already been discovered by player i, namely $discPts_i(t)$, are present in the lookup table, the function $f[pos_i(t+2), pt]$ does not have to be evaluated for them to determine their contribution to the overall utility of player i. Instead, a very simple operation can be used to query the number of elements in $discPts_i(t)$, which would indicate the utility of player i prior to time t. The problem then becomes to iterate through only a subset of points in the grid that have the potential of increasing player i's utility in the following two-step action sequence. It would be necessary to at least scan through the points that would be in range of the sensor in the future positions. Since the sensor has a circular coverage, a solution would be to enclose the points that would be covered by the sensor's range using a square, and scan through all the points that would fall under the square. This is illustrated in Fig. 4.8. Squares 3, 4, and 5 enclose the points that need to be scanned to determine the utility of turning right, left, and back respectively. Squares 1 and 2 enclose the points that need to be scanned to determine the utility of playing the current baseline action. There are two squares because there are two future positions associated with playing the baseline action. The squares associated with the baseline action have a solid boundary, whereas the squares associated with any of the turn actions have dotted boundary lines. Figure 4.9 shows a detailed view of how a square encloses a sensor's radial coverage so that in scanning all the grid points inside the square (denoted by the bold line), all the points that would be in range of the sensor are also scanned. The solid dot indicates a future position of the robot and the hollow dot indicates a point that needs to be scanned. We say that a point pt is "bounded at a location $loc_i(t)$" if at time t, pt is inside the square that encloses the circle created by the sensor centered at $loc_i(t)$. Since the radius of the sensor's coverage is $sensRange$, a side of the square is $2 * sensRange$ in dimension. We note that one point that certainly does not need to be scanned or

that cannot contribute to increasing player i's utility in the two-step actions sequence is the point where the robot would be situated (i.e. the solid dot in Fig. 4.9). This point would already have been accounted for in the previous time step, and is based on the assumption that $sensRange \geq 1$ grid point; that is, the range of the sensor is large enough to at least detect adjacent points in the grid. Thus, the number of points that would need to be iterated through or scanned in a square is

$$\begin{aligned} \textit{points to be scanned in square} &= (2|sensRange| + 1)^2 - 1 \\ &= 4|sensRange|^2 + 4|sensRange| \end{aligned} \tag{4.1}$$

This equates to 24 in our particular example with $sensRange = 2$ grid points. We note that *points to be scanned in square* is a $O(sensRange^2)$ function of $sensRange$. It cannot be of order $sensRange$ because for sufficiently large values of $sensRange$, *points to be scanned in square* would not be bounded by $M * |sensRange|$, where M is a constant factor (see Section 2.6).

We need to restate the objective function in (3.7) in a different way now because presently the objective function iterates through all the points in the grid. Before we do so, however, we need to define the following set.

$$squareSet = \{pt \in gridpts \mid cond.3 \text{ is } true\}, \tag{4.2}$$

where,

$$cond.3 = \begin{cases} true, & \text{if } a_i^x = a_i^1 \text{ in } (a_i^x, a_i^1) \text{ and } pt \text{ is bounded at } loc_i(t+1) \text{ or } loc_i(t+2) \\ true, & \text{if } a_i^x \neq a_i^1 \text{ in } (a_i^x, a_i^1) \text{ and } pt \text{ is bounded at } loc_i(t+2) \\ false, & \text{otherwise} \end{cases}$$

If the first statement in $cond.3$ evaluates to true, then player i must be playing its

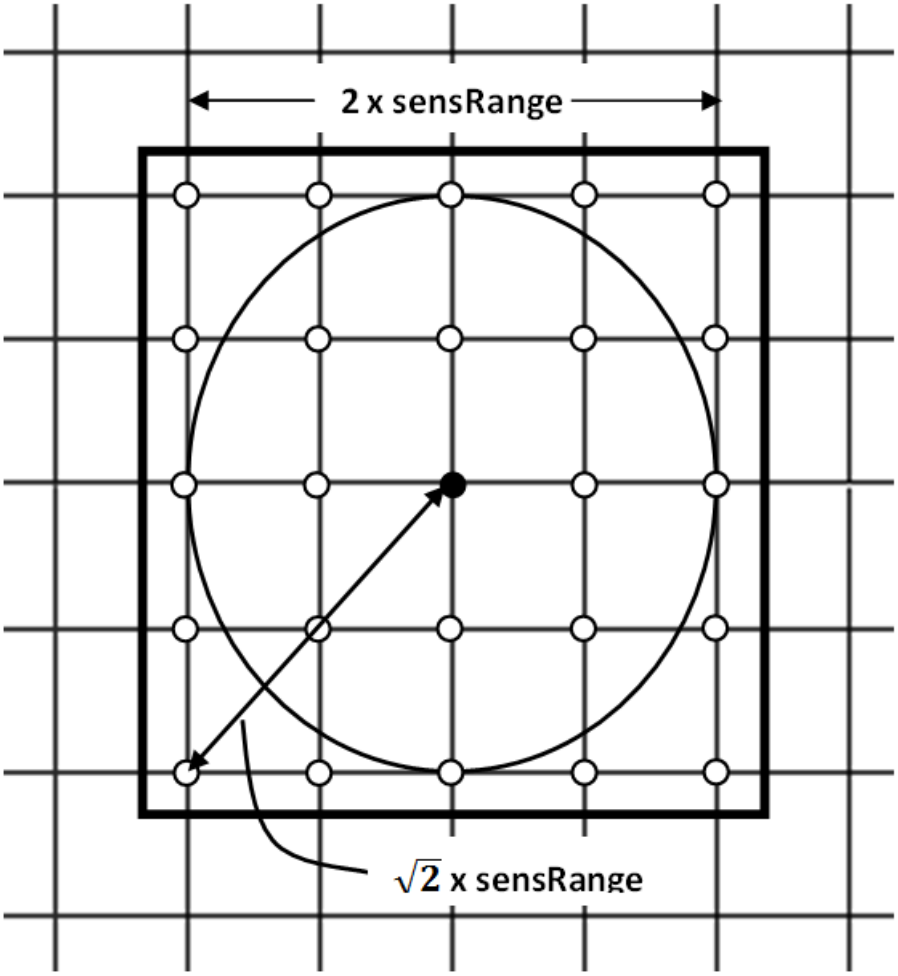

Figure 4.9: Example of a square enclosing sensor's radial coverage

current baseline action for the two-step action sequence. If the second statement in *cond*.3 evaluates to true, then player i must be making a turn. Given this, we now restate $U_i^{a_i^1}(pos_i(t+1))$ in terms of a function $h[pos_i(t+2), pt]$.

$$U_i^{a_i^1}(pos_i(t+1)) = |discPts_i(t)| + \sum_{pt \in squareSet} h[pos_i(t+2), pt], \tag{4.3}$$

where,

$$h[pos_i(t+2), pt] = \begin{cases} 1, & \text{if } cond.4 \text{ or } cond.5 \text{ is } true \\ 0, & \text{otherwise} \end{cases}$$

cond.4 evaluates to *true* if

$$||pt - loc_i(t+1)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t+1) \text{ and } pt \notin discPts_i(t),$$

cond.5 evaluates to *true* if

$$||pt - loc_i(t+2)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t+1) \text{ and } pt \notin discPts_i(t),$$

and $|discPts_i(t)|$ represents the cardinality of $discPts_i(t)$. We note that the only difference between *cond*.4 and *cond*.1, and *cond*.5 and *cond*.2, is the inclusion of the clause $pt \notin discPts_i(t)$. It is present to ensure that $h[pos_i(t+2), pt]$ only equates to 1 for a point pt if pt is expected to be discovered by i in the following two-step action sequence. This prevents double-counting previously discovered points. As before, it is assumed that $discPts_{-i}(t+1) = discPts_{-i}(t)$, and,

$$U_i\big((a_i^x, a_i^1), (a'_{-i}, a''_{-i})\big) = U_i^{a_i^1}(pos_i(t+1)).$$

Considering that (4.3) has to be computed for four two-step actions sequences, a total of $20(|sensRange|^2 + |sensRange|)$ iterations are needed. This is calculated using (4.1) as follows.

$$\begin{aligned} total\ iterations &= 5 * points\ to\ be\ scanned\ in\ square \\ &= 5[(2|sensRange| + 1)^2 - 1] \\ &= 5(4|sensRange|^2 + 4|sensRange|) \\ &= 20(|sensRange|^2 + |sensRange|) \end{aligned}$$

The coefficient 5 above is present rather than 4 because as mentioned earlier there are two future positions associated with playing the baseline action, and so, two squares

are required. From the definition of the big O notation (see Section 2.6), we have

$$\begin{aligned} &total\ iterations \in O(20(|sensRange|^2 + |sensRange|)) \\ &\Rightarrow total\ iterations \in O(sensRange^2) \end{aligned}$$

The calculation above for the runtime order is for a robot i that is allowed to update its action in the absence of any obstacles near it. It thus has the full set α_i available to it to select a two-step action sequence. If on the other hand the robot faces an obstacle in front of it and has to manoeuvre around it, it cannot play its baseline action. Hence, the action sequence (a_i^1, a_i^1) would not be available for it to select from α_i. This gives us $12(|sensRange|^2 + |sensRange|)$ iterations as the calculation below shows, which is still $O(sensRange^2)$.

$$\begin{aligned} total\ iterations &= 3 * points\ to\ be\ scanned\ in\ square \\ &= 3[(2|sensRange| + 1)^2 - 1] \\ &= 3(4|sensRange|^2 + 4|sensRange|) \\ &= 12(|sensRange|^2 + |sensRange|) \end{aligned}$$

$$\begin{aligned} &\Rightarrow total\ iterations \in O(12(|sensRange|^2 + |sensRange|)) \\ &\Rightarrow total\ iterations \in O(sensRange^2) \end{aligned}$$

Since $O(sensRange^2) \leq O(Z^2)$ and assuming $sensRange << Z$, we can conclude that Algorithm 2 is computationally more efficient than frontier-based exploration algorithms.

4.4 Improved Exploration Algorithm

This section discusses an improvement to Algorithm 2 in terms of the time taken to explore a space. The improvement stems from each robot predicting the location of every other robot when deciding on a direction to turn. This allows a robot to change its heading to avoid exploring the same areas as other robots, and as a result achieve a greater degree of coordination. The basis of the prediction is that when a robot i is allowed to perform a turn sequence based on its exploration rate ϵ, it can be reasonably sure that every other robot will play their baseline action or move forward for the two time steps required to complete i's turn. In fact, the prediction becomes more accurate the smaller ϵ is set to because robots will turn less often, and thus when a robot is allowed to turn it can be reasonably sure that other robots will not turn. A robot that is deciding to turn needs to know the heading and the location of every other robot (i.e. $pos_{-i}(t)$) in the time step it is deciding on turning on so that it can predict the locations of all the robots in the two time steps it will take to perform its turn (i.e. $pos_{-i}(t+1)$ and $pos_{-i}(t+2)$). Taking into account the aforementioned, we now redefine the objective function of a player i that is able to update its action at time t as

$$\begin{aligned} U_i\big((a_i^x, a_i^1), (a_{-i}^1, a_{-i}^1)\big) &= U_i^{a_i^1}(pos_i(t+1), pos_{-i}(t+1)) \\ &= \sum_{pt \in gridpts} f[pos_i(t+2), pos_{-i}(t+2), pt] \end{aligned} \tag{4.4}$$

where

$$f[pos_i(t+2), pos_{-i}(t+2), pt] = \begin{cases} 1, & \text{if } pt \in discPts_i(t) \\ 1, & \text{if } cond.6 \text{ is } true \\ 1, & \text{if } cond.7 \text{ is } true \\ 0, & \text{otherwise} \end{cases}$$

cond.6 evaluates to *true* if

$$||pt - loc_i(t+1)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t)$$
$$\text{and } ||pt - loc_{-i}(t+1)|| \nleq sensRange$$
$$\text{and } ||pt - loc_{-i}(t+2)|| \nleq sensRange$$

cond.7 evaluates to *true* if

$$||pt - loc_i(t+2)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t)$$
$$\text{and } ||pt - loc_{-i}(t+1)|| \nleq sensRange$$
$$\text{and } ||pt - loc_{-i}(t+2)|| \nleq sensRange$$

Note in comparison to (3.7), equation (4.4) uses (a^1_{-i}, a^1_{-i}) rather than (a'_{-i}, a''_{-i}) signifying that the objective function is calculated under the assumption that other robots move forward in the following two time steps. The clause $||pt - loc_{-i}(t+1)|| \nleq sensRange$ or $||pt - loc_{-i}(t+2)|| \nleq sensRange$ evaluate to *true* if pt is not in range of any of the robots aside from robot i in the respective time step. Equation (4.4) is the objective function for any player i that is able to change its action at time t. For every other player that is not allowed to change its action at time t from its baseline action (i.e. moving forward) as dictated by ϵ, its predicted utility for the two-step action sequence that follows is calculated using (4.3).

It can be seen that in maximizing (4.4), a player i avoids heading in a direction that it predicts other robots are going to move towards. Fig. 4.10 will be used to illustrate this. In Fig. 4.10, two robots are simultaneously exploring a 20×20 grid free of obstacles except for the outer walls. The heading and position of each robot shown in the diagram is for a time t. Note that the grid points are represented differently here. They are inside squares, which we refer to as a cell in the diagram. In Fig. 4.10, Robot 1 has reached the edge of the grid and cannot play its baseline

action. It thus has to perform a turn sequence and move left, right, or down to the respective cells marked in the diagram by the chequered squares. The green cells represent grid points that have already been discovered by Robot 1 and Robot 2. It is assumed that Robot 2 is not allowed to update its action at time t, and hence, must play its baseline action for the following two time steps. The positions that Robot 2 would assume in the following two time steps are marked by the chequered squares in Fig. 4.10. The red squares indicate the new cells or the grid points that would be discovered by Robot 2 in the following two time steps. As Robot 1 has to perform a turn sequence, its objective function would be dictated by (4.4). Therefore, Robot 1 would not expect to receive gains in utility from cells 4 and 6 if it turned left because it would predict that Robot 2's sensor scan would pick up those points at time $t+2$. The clause $||pt - loc_{-i}(t+2)|| \nleq sensRange$ in (4.4) would not be satisfied for cells 4 and 6, and thus, $f[pos_i(t+2), pos_{-i}(t+2), pt] = 0$ for cells 4 and 6. Robot 1, however, would expect to receive gains in utility from cells 1, 2, 3, 5, and 7 amounting to a total utility gain of 5 in turning left. On the other hand, Robot 1 would expect to receive a gain of 7 in utility from turning right by discovering cells 8 to 14. Thus, turning right is clearly a superior decision to turning left for Robot 1 as it provides more utility. We disregard Robot 1 moving down in this example because in reality Robot 1 would have moved upwards to get to the position it is at in Fig. 4.10. Therefore, it would have already discovered points below it. Had Robot 1 used objective function (3.7) to determine which direction to turn towards, it would have chosen indiscriminately between turning left and turning right. Turning left in this situation would mean that less of the space would be explored by the end of time step $t+2$ compared to turning right. As objective function (4.4) ensures that Robot 1 only turns right in this situation, one would expect that in making similar decisions in all situations such as the one illustrated in Fig. 4.10, the exploration time needed with (4.4) would be less than the exploration time needed with (3.7). Simulation results

will be presented shortly that confirm this hypothesis.

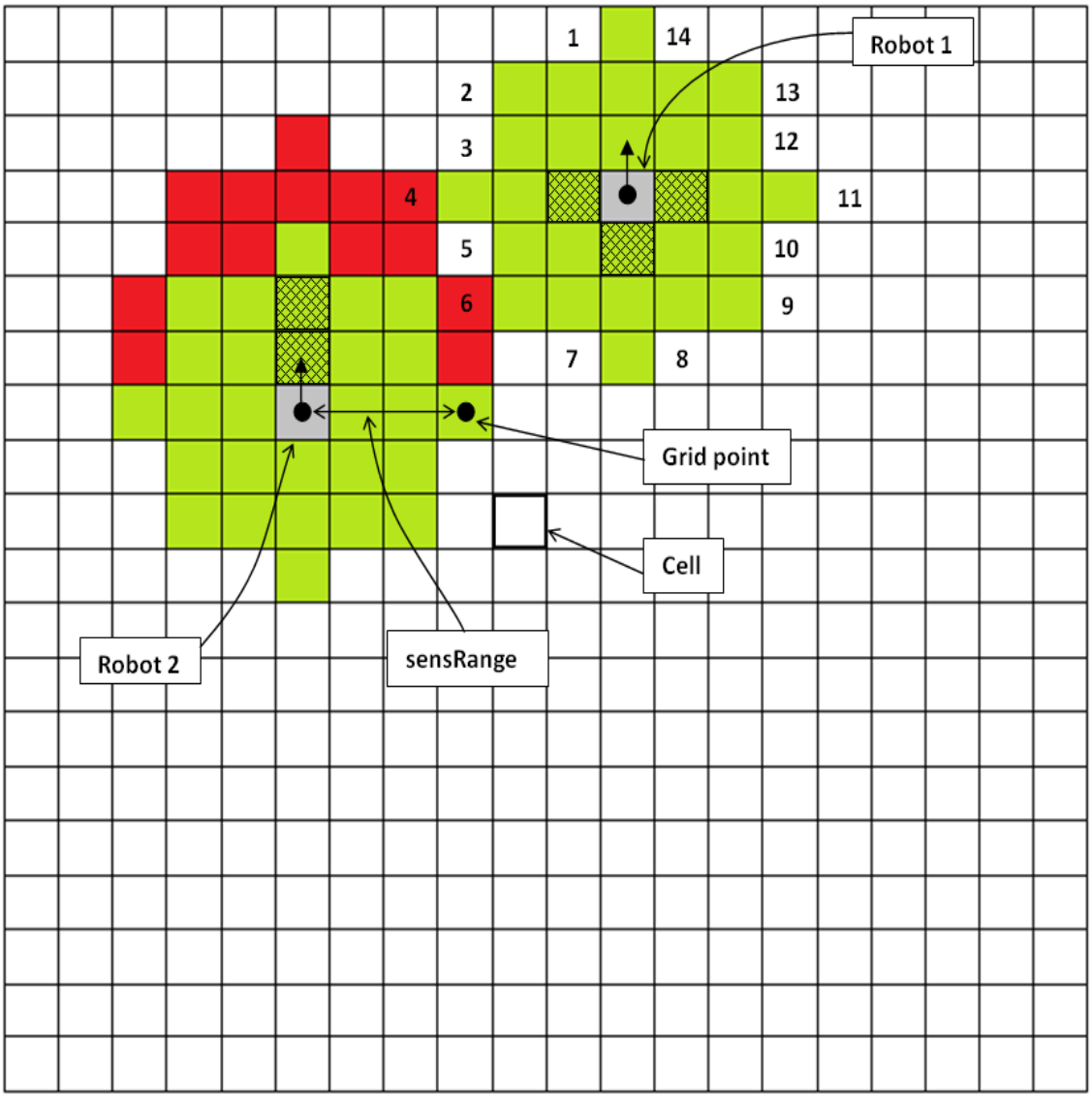

Figure 4.10: Example to illustrate decision process of a robot using objective function (4.4)

We need to introduce new rules for updating $discPts_i(t)$ and $discPts_{-i}(t)$ that are consistent with the objective function (4.4). As with the previous objective function (3.7), at the beginning of a time step $pos_i(t)$ is updated, except if a robot i is in the middle of a turn sequence in which case it needs to update $pos_{-i}(t)$ as well. Then $discPts_i(t)$ and $discPts_{-i}(t)$ are updated. However, depending on whether a

robot has the option of performing a turn sequence or not at a time t as dictated by ϵ, we differentiate how $discPts_i(t)$ is updated. Based on *cond*.3 and *cond*.4 in (4.4), a robot i that has the option of performing a turn sequence at a time t does not expect to increase its utility by discovering new grid points in the following two time steps if it predicts those grid points would be discovered by other robots in those two times steps. Thus, to be consistent with the prediction, i must ensure those grid points do not get included in $discPts_i(t)$ when it updates the set after taking an action. This is reflected in (4.5).

If a robot i has the option of performing a turn sequence at time t, $discPts_i(t)$ is updated as follows in the following two time steps.

$$discPts_i(t) = \{pt \in gridpts \mid cond.8\} \cup discPts_i(t-1) \tag{4.5}$$

cond.8 evaluates to *true* if

$$||pt - loc_i(t)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t-1)$$
$$\text{and } ||pt - loc_{-i}(t)|| \not\leq sensRange$$

If on the other hand a robot i can only play its baseline action, $discPts_i(t)$ is updated as follows in the following two time steps.

$$discPts_i(t) = \{pt \in gridpts \mid cond.9\} \cup discPts_i(t-1)$$

cond.9 evaluates to *true* if

$$||pt - loc_i(t)|| \leq sensRange \text{ and } pt \notin discPts_{-i}(t-1)$$

Finally, $discPts_{-i}(t)$ is updated for all robots as follows.

$$discPts_{-i}(t) = \bigcup_{j \in N_i} \{pt \in gridpts \mid pt \in discPts_j(t)\}$$

Claim 4.1 *Player objective function* (4.4) *constitute a potential game with potential function* (3.9).

Proof. The proof is very similar to the proof of Claim 3.1 presented in Section 3.2. However, we present it here for clarity and completeness. Again, we assume a player i is contemplating at time t whether to turn in one direction by performing the action sequence (a_i', a_i^1) or to turn in another direction by performing (a_i'', a_i^1) so that $a_i' \neq a_i''$. The change in the objective function of player i by switching from the action sequence (a_i'', a_i^1) to the action sequence (a_i', a_i^1), provided that all other players collectively play (a_{-i}^1, a_{-i}^1), is

$$\triangle U_i = U_i\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) - U_i\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big)$$

The first difference equation for the potential function of the game for the two different action sequences of player i is

$$\triangle \phi = \phi\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) - \phi\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big)$$

Substituting (3.10) into the above difference equation, we get

$$\triangle \phi = U_i\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) + \sum_{j \in N_i} U_j\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) - \Bigg[U_i\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big) + \sum_{j \in N_i} U_j\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big)\Bigg] \quad (4.6)$$

A robot $j \in N_i$ computes its objective function using (3.7). Since this value is not influenced by robot i's actions, it is irrespective of a_i' and a_i''. Therefore, we have

$$U_j\big((a_i', a_i^1), (a_{-i}^1, a_{-i}^1)\big) = U_j\big((a_j^1, a_j^1), (a_{-j}^x, a_{-j}^y)\big) = U_j\big((a_i'', a_i^1), (a_{-i}^1, a_{-i}^1)\big) \quad (4.7)$$

Now, we substitute (4.7) into (4.6) to get

$$\triangle U_i = \triangle \phi$$

□

As previously (Section 3.3.1), we have players update their actions with a probability of ϵ to allow for decentralized decision-making. However, this comes at the expense of losing some coordination or cooperation in the sense that when a player i decides to make a turn in a direction believing that no other player is going to come in its way, it may be wrong and it may actually run into another player's path. Turning and running into another player's path could be considered a bad decision on player i's part because it could perhaps have turned in another direction in that time and not have any overlaps with other players. Thus, better spatial distribution of robots could have been had, which could reduce the time needed for exploration. However, in comparison with a completely uncoordinated algorithm such as Algorithm 3, overlaps would be significantly lower resulting in faster exploration times.

To asses the effect of using (4.4) on exploration time, we replace the objective

function used in Algorithm 2 (i.e. (3.7)) with (4.4), and use the new rules for updating $discPts_i(t)$ and $discPts_{-i}(t)$ as discussed in this section. This is shown in Algorithm 4. The term $pos_{-i}(t)$ refers to the combined location and heading vector for each player $j \in N_i$, and $U_{i\_improved}$ is short for objective function (4.4).

Algorithm 4 Improved Exploration Algorithm

Initialize $\#\ of\ time\ steps,\ sensRange,$ and ϵ

for $t \leftarrow 1, \#\ of\ time\ steps$ **do**
  for all player $i \in N$ **do**

    if *player in middle of turn sequence* **then**
      Update $pos_i(t),\ pos_{-i}(t),\ discPts_i(t),$ *and* $discPts_{-i}(t)$
    else
      Update $pos_i(t),\ discPts_i(t),$ *and* $discPts_{-i}(t)$
    end if

    if *player should turn because of obstacle* **then**
      Compute $U_{i\_improved}$, $\forall (a_i^x, a_i^1) \neq (a_i^1, a_i^1)$
      Play (a_i^x, a_i^1) *that maximizes* U_i
    else if *player should explore based on* ϵ **then**
      Compute $U_{i\_improved}$, $\forall (a_i^x, a_i^1) \in \alpha_i$
      Play (a_i^x, a_i^1) *based on* asf_i
    else
      Play (a_i^1, a_i^1)
    end if

  end for
end for

We simulate Algorithm 4 for the same test environment in Fig. 4.1 with *sensRange* set to 2 grid points, but use a value of 0.1 for ϵ rather than 0.3. Again, the simulation is run for 2000 time steps and averaged over 20 games. Fig. 4.11 compares the performance of Algorithm 2 with Algorithm 4. It can be seen that Improved Algorithm discovers more grid points from the $200th$ time step to the $700th$ time step, but in terms of the total time it takes to explore the whole space there is no difference between the two algorithms. However, it can be argued that if there is only

a limited time given to explore the space (400 time steps for example), Algorithm 4 would explore more of the space than Algorithm 2. On the contrary it is important to note that with Algorithm 4, a robot that is performing a turn sequence requires more information from the other robots compared to Algorithm 2. Specifically, it needs $pos_{-i}(t)$, which includes both the heading and position of the other robots (i.e. $pose_{-i}(t)$ and $loc_{-i}(t)$ respectively). Recall that in Algorithm 2, a robot only needs to know the position of all the other robots (i.e. $loc_{-i}(t)$). Thus, the improvement in exploration time of the Improved Algorithm over Algorithm 2 comes at the expense of more information having to be shared among the robots. No analysis was done in this thesis to examine bandwidth requirements for communications, and the simulations performed does not account for any communication overhead or latency between robots as this falls outside the scope of this thesis.

Fig. 4.12 captures the results of a simulation of Algorithm 2 and Algorithm 4 for $\epsilon = 0.3$ and $sensRange = 2$ grid points. Algorithm 4 performs worse than Algorithm 2 between the 200th and 800th time step. Increasing ϵ from 0.1 to 0.3 resulted in a significant amount of coordination to be lost among robots with Algorithm 4. Thus, many robots would have run into each others' paths while updating their actions through the course of the game resulting in significant sensor overlaps. Since Algorithm 4 performed better than Algorithm 2 when $\epsilon = 0.1$ and worse than Algorithm 2 when $\epsilon = 0.3$, a natural comparison would be to see if Algorithm 4 with $\epsilon = 0.1$ would perform better than Algorithm 2 with $\epsilon = 0.3$. The result of this comparison is illustrated in Fig. 4.13. It is clear that Algorithm 4 performs better than Algorithm 2 even though both algorithms take approximately the same time to explore the whole space.

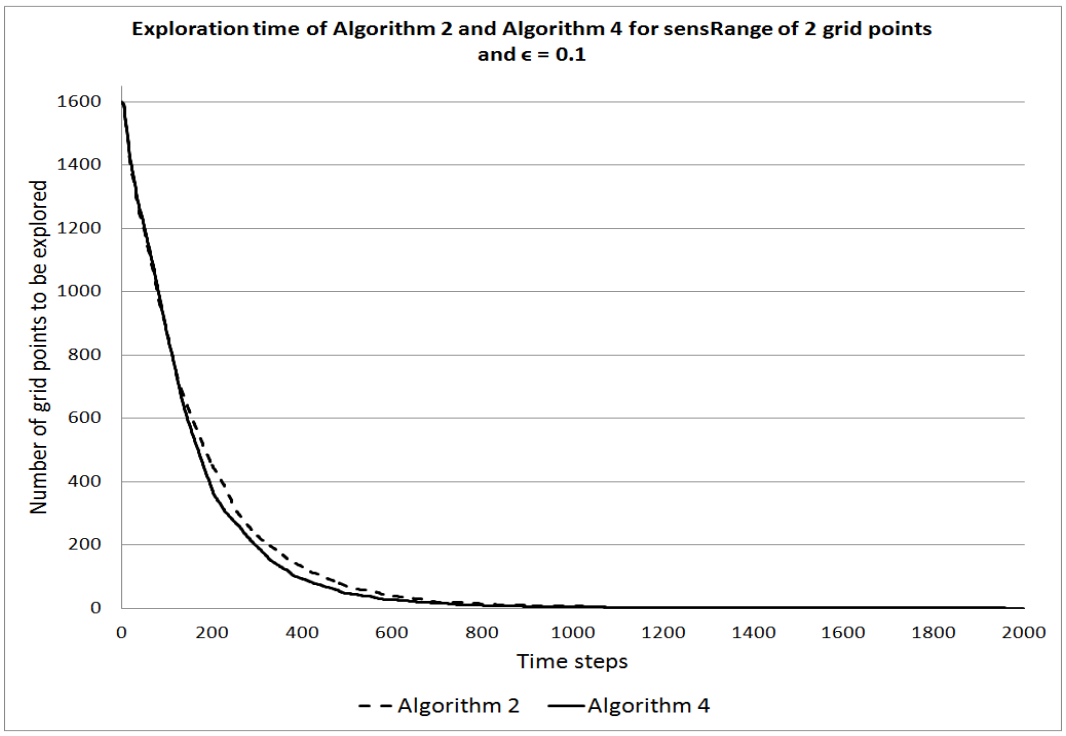

Figure 4.11: Comparison of exploration time of Algorithm 2 and Algorithm 4 with $\epsilon = 0.1$ and $sensRange = 2$ grid points

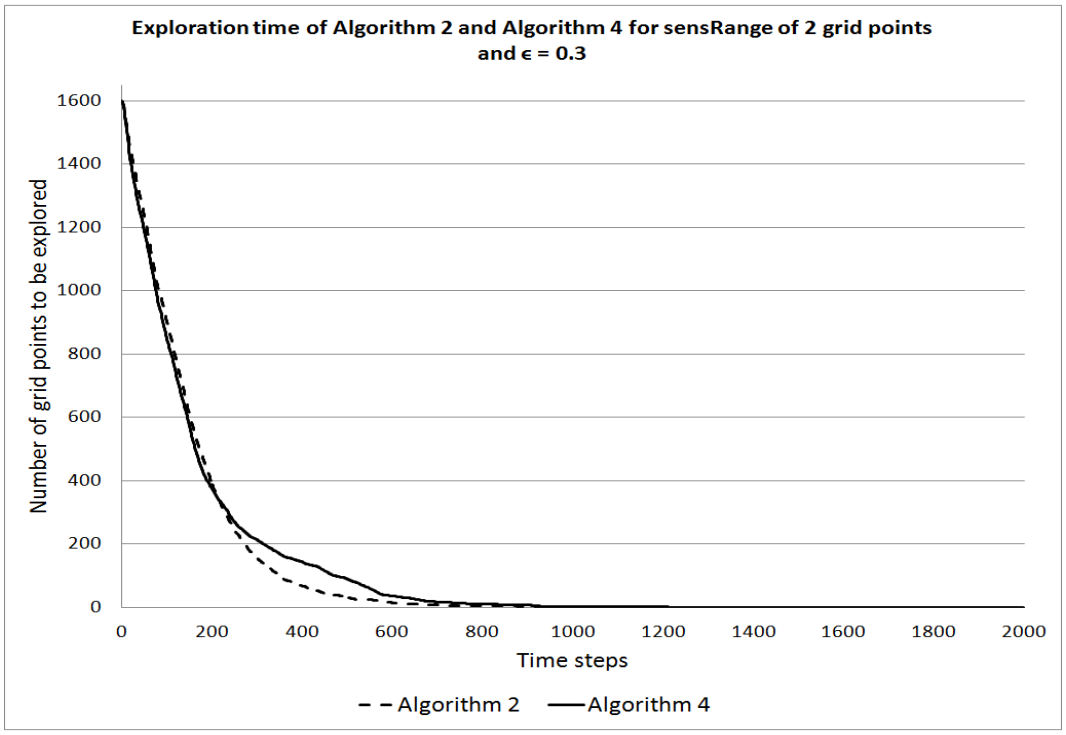

Figure 4.12: Comparison of exploration time of Algorithm 2 and Algorithm 4 with $\epsilon = 0.3$ and $sensRange = 2$ grid points

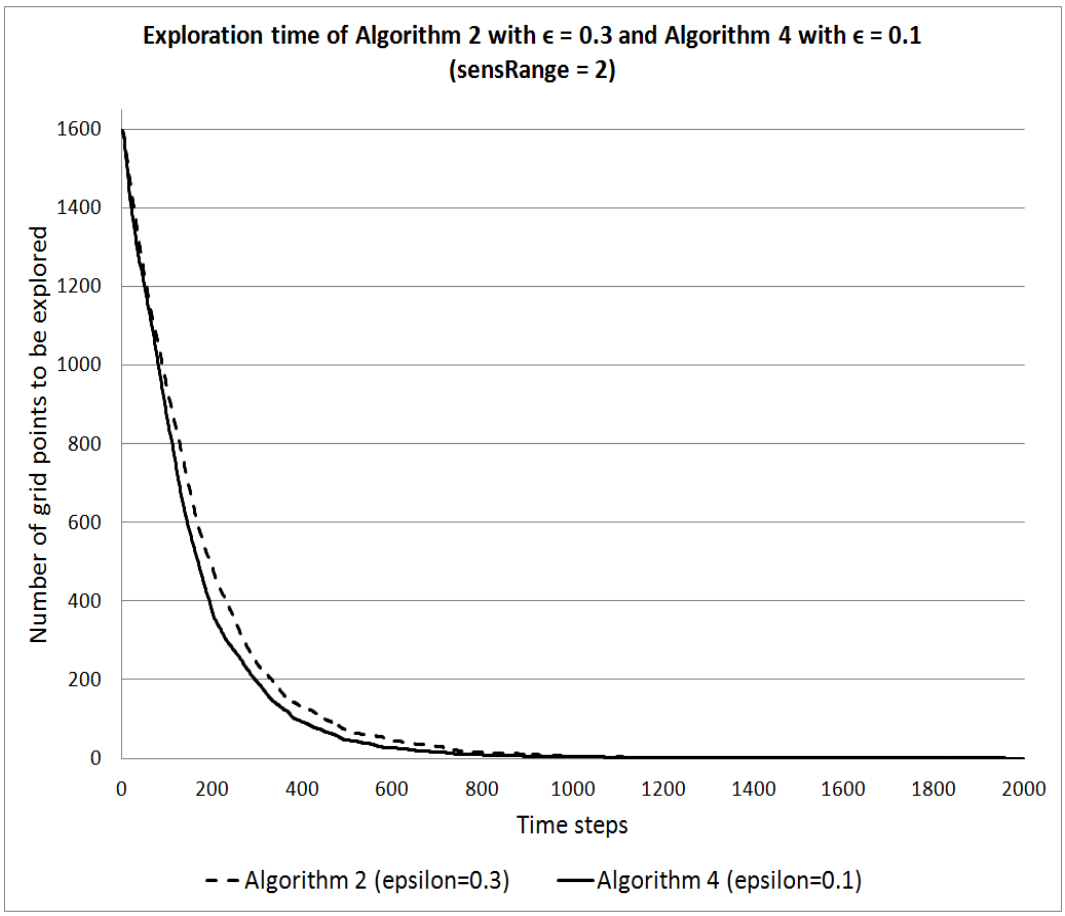

Figure 4.13: Comparison of exploration time of Algorithm 2 with $\epsilon = 0.3$ and Algorithm 4 with $\epsilon = 0.1$, and $sensRange = 2$ grid points for both algorithms

4.4.1 Effect of Obstacles on Exploration Time

In this section, we investigate the effect that obstacles have on the exploration time required to uncover a space. An obstacle is an object in the environment that causes a robot to turn even if at that particular moment it is not allowed to do so based on ϵ. Therefore, objects inside a space that is to be explored as well as walls that enclose the space are considered obstacles. As discussed in Section 4.1, the presence of obstacles can be effectively thought of as increasing ϵ from the value it was initialized to, which as seen in Section 4.4 reduced coordination among robots. Fig. 4.14 represents a test setup that is similar to Fig. 4.1, except it omits two of the obstacles in Fig. 4.1.

Thus, the density of the obstacles in terms of the area they occupy compared to the area of the whole space has effectively been reduced in Fig. 4.14 in comparison to Fig. 4.1. We then simulate Algorithm 4 in the setup in Fig. 4.14 with $\epsilon = 0.1$ and $sensRange = 2$ grid points, and compare it to how it performed in the setup in Fig. 4.1 with the same values for ϵ and $sensRange$. Fig. 4.15 shows the results of the simulation. Despite ϵ being the same in both tests, Algorithm 4 performed slightly better in the setup with one obstacle than the original setup with three obstacles. This confirms the thinking that increasing the number of obstacles in a space effectively increases the value of ϵ. In Fig. 4.15, the exploration time of Algorithm 3 is displayed for reference. There is no change in the curve of Algorithm 3 from the setup consisting of three obstacles to the setup consisting of only one obstacle.

We can consider the effect that outer walls, which bound an entire space, would have on exploration time if we assume a square environment without any obstacles embedded inside it; that is the only obstacles are the outer walls. If l represents the length of a side of this environment, its area $Area$ is proportional to l^2 (i.e. $Area \propto l^2$), whereas its perimeter $Perim$ is proportional to l (i.e. $Perim \propto l$). Therefore, if l doubles for example, $Area$ would quadruple and $Perim$ would only double. Given these relations of $Area$ and $Perim$ to l, it is foreseeable that as l gets large, the total exploration time will be increasingly comprised of the time that robots spend moving forward rather than performing obstacle avoidance manoeuvres. This is because the boundaries that impose restrictions on the actions that a robot can play would be insignificant in comparison to the overall area. Moving forward reduces the time needed for exploration in comparison to making frequent turns because more grid points are discovered if a robot plays its baseline action. Thus, scenarios in which the exploration process is mostly comprised of robots moving forward would more so justify using Algorithm 4. Otherwise, using Algorithm 4 may not be as effective, and other exploration algorithms such as frontier-detection algorithms that have more

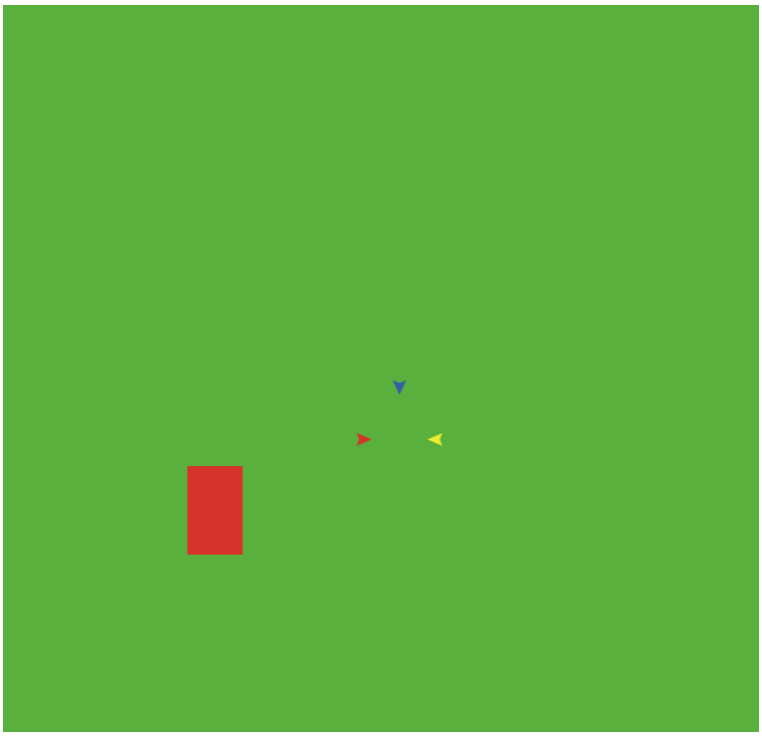

Figure 4.14: Simulation setup for analyzing the effect of obstacles on exploration time

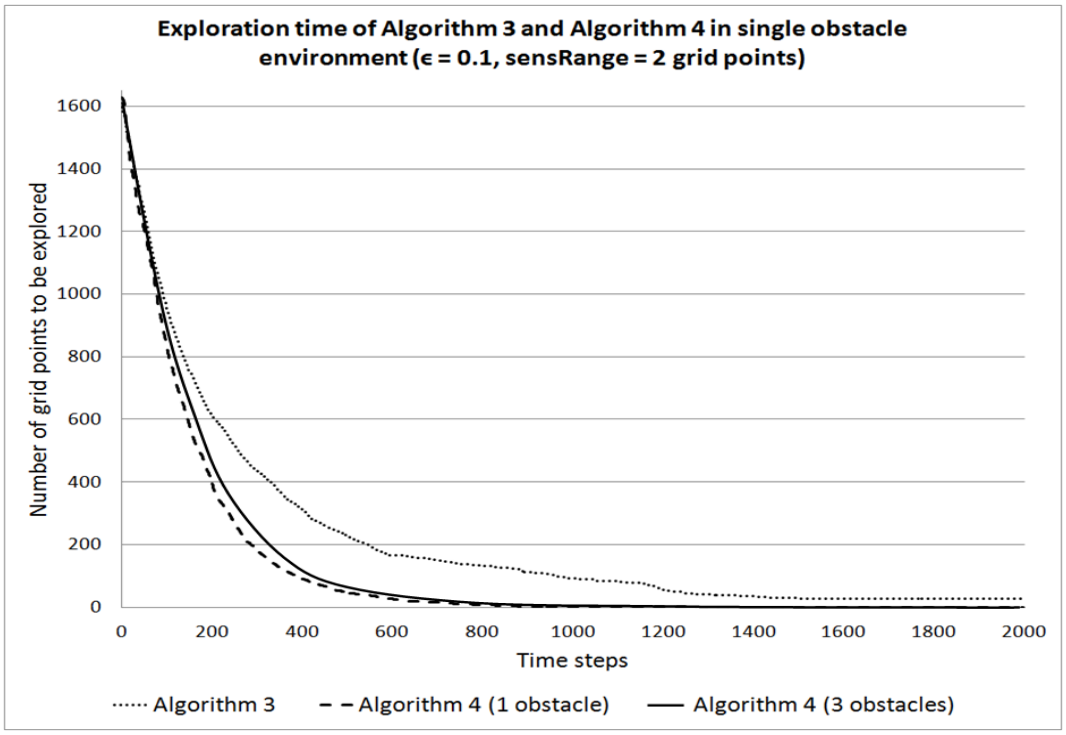

Figure 4.15: Comparison of exploration time of Algorithm 4 in a setup with one obstacle and in a setup with two obstacles

explicit coordination schemes may prove to be more beneficial. As discussed earlier, frontier-detection algorithms employ frontiers to maintain coordination amongst robots, which can greatly reduce sensor overlaps and result in efficient exploration of the environment. Even if it has a greater computational complexity (i.e. $O(Z^2)$) than the Potential game algorithms discussed in this paper, for small environments where Z is small, the running time becomes negligible. Instead its greater efficiency over the Potential game algorithms in having robots move in non-overlapping paths may reduce overall exploration time.

From the analysis in this section, we can conclude that Algorithm 4 performs best in large open areas with few obstructions.

4.5 Summary

In this section, we modified the exploration algorithm introduced in Section 3.3.2 so that bounded spaces embedded with obstacles can be explored. To assess the performance of the modified exploration algorithm (Algorithm 2), an example of an uncoordinated exploration algorithm (Algorithm 3) was presented. It was then shown through simulation that Algorithm 2 reduced the time needed for exploration compared to Algorithm 3. Moreover, it was found that Algorithm 2 fared even better than Algorithm 3 when the range of the sensors that robots were equipped with was decreased. Algorithm 2 has an advantage over Algorithm 3 in that it introduces a certain degree of coordination among robots so that they do not all follow the same path when they are exploring. Furthermore, Algorithm 2 was analyzed to have a running time of $O(sensRange^2)$ compared to a running time of $O(Z^2)$ for frontier-detection algorithms. This makes our Potential game computationally more efficient than frontier detection algorithms. Finally, an improvement to Algorithm 2 was suggested (Algorithm 4), which is based on having a robot predict the future location

of every other robot when it decides to turn. Although Algorithm 4 required the same amount of time to explore a space as Algorithm 2, it was found to have a slightly greater exploration rate than Algorithm 2 in the first few hundreds of time steps. It was further shown through simulation that as the density of obstacles in a space increased, coordination among the robots decreased resulting in a decrease in exploration rate at the beginning of the exploration process. This was attributed to the fact that increasing the obstacles in a space has the equivalent affect of increasing ϵ, which in turn reduces the accuracy of a robot's prediction of where other robots would be situated in a two-step action sequence. Also, having more obstacles in a space means that robots would make more turns and would need to make more decisions, which increases computational complexity (see Section 4.3). Based on these results, it was concluded that Algorithm 4 performs best in large open areas with only a few obstructions.

Chapter 5

Conclusions

In today's age, there are increasing number of applications that demand the use of multi-robot systems in mapping unknown spaces. Although frontier-based dynamic programming methods are currently the most common in achieving these objectives, new methods are being sought to address its shortcomings. This thesis presents a method for exploration with multiple robots using Potential games. It is based on previous work in solving the Consensus problem using Potential games. Instead of consensus being the collaborative goal, however, our main goal in this thesis is the exploration of closed spaces. The exploration process in a multi-robot platform involves allocating each robot a target to explore such that overlaps are minimized. As part of this work, a meaningful Potential function was defined and a local objective function for each player was derived that is coherent with the overall goal. Moreover, the Potential game defined in this thesis is built atop the Simple Forward Turn Controller, which was introduced in prior work to simulate the nonholonomic behaviour of real-world robots.

5.1 Summary of Contributions

This thesis presented a method of exploring a space with multiple robots using a Potential game. The method was developed and validated through numerous simulations with varying parameters. In summary, the thesis has provided advancements to this area of research through the following contributions:

1. A method of collaboratively mapping an unknown environment with a team of robots using Potential games was proposed. The definition of a Potential game was extended so that it could be modelled under the framework of the Simple Forward Turn Controller.

2. In Chapter 3.2, we defined the potential function for our Potential game, which is a summation of the objective function that was assigned to each player in the game. The objective function itself was based on the number of grid points that would be discovered by a robot if it played an action sequence from its action set. Moreover, update rules were created for the variables $discPts_i(t)$ and $discPts_{-i}(t)$ for every time step. The combination of the potential function and the objective function for each player was seen to satisfy the WLU family of utility structures, and a proof was presented signifying that they made up a Potential game.

3. The Simple Forward Turn Controller was modified so that when a player is allowed to update its action, it chooses an action from the best response set rather than randomly selecting an action as it was suggested originally.

4. In Chapter 4.2, it was demonstrated through NetLogo simulations how the algorithm that is based on our Potential game (Algorithm 2) required less time to explore a bounded space compared to an uncoordinated exploration algorithm (Algorithm 3). Numerous simulation results were presented comparing

both algorithms for different values of ϵ and $sensRange$. Also, Algorithm 2 was analyzed and found to have a lower runtime order than frontier detection algorithms.

5. In Chapter 4.4, an improvement was suggested to our first algorithm (Algorithm 2), which is based on having a robot predict the future location of every other robot when it decides to turn. New update rules were also presented for variables $discPts_i(t)$ and $discPts_{-i}(t)$ as part of this new algorithm (Algorithm 4). Algorithm 4 was found to have a slightly greater exploration rate than Algorithm 2 at the beginning of the exploration process.

5.2 Future Work

The work presented in this thesis opens new research directions and future work to be conducted in multi-robot exploration using Potential games. They are as follows.

1. Implement the algorithms presented in this thesis on real robotic platforms such as a team of Husky robots. The algorithm needs to be tested on indoor and outdoor environments with different types of obstacles.

2. Integrate frontier-detection algorithms with the Potential game algorithm introduced in this thesis in such a way that the benefits of both algorithms can be exploited.

3. Conduct thorough analysis of the bandwidth requirements in communications between robots for the algorithms introduced in this thesis so that their applicability in severely bandwidth constrained mediums such as water can be assessed. Further along the lines of bandwidth limited mediums, future work will involve the development of a game where robots do not communicate with

their neighbours every time step or only communicates with a subset of their neighbours. This would reflect limitations in communication range of robots in large multi-robot platforms where robots could be situated far from each other.

4. Improve or altogether come up with a new algorithm whose reward calculation for each action in a robot's action set can be run in parallel. This could offer improvements in execution time and system responsiveness, especially when robots have large action sets.

The applications involving multi-robot systems to solve real-world problems are numerous. There is an increasing need today for decentralized robotic systems in planetary exploration, reconnaissance, etc. that require simple yet cooperative agents to accomplish an overall goal. Potential games offer a promising research direction in this regard.

List of References

[1] J. Lindsay. *Nonholonomic Consensus in Cooperative Robotics: A Game Theoretical Approach.* Master's thesis, Royal Military College of Canada, Kingston, Ontario (2011).

[2] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. "Collaborative multi-robot exploration." In "Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)," pages 476–481. San Francisco, California, USA (2000).

[3] P. Wadhams. "The use of autonomous underwater vehicles to map the variability of under-ice topography." *Ocean Dynamics* **62**(3), 439–447 (2012).

[4] X. Lu. *Multi-Agent Reinforcement Learning in Games.* Ph.D. thesis, Carleton University, Ottawa, Ontario (2012).

[5] J. R. Marden, H. P. Young, G. Arslan, and J. S. Shamma. "Payoff-based dynamics for multiplayer weakly acyclic games." *SIAM J. Control Optim.* **48**(1), 373–396. ISSN 0363-0129 (2009).

[6] P. Dinnissen. *Using Reinforcement Learning in Multi-Robot SLAM.* Master's thesis, Carleton University, Ottawa, Ontario (2011).

[7] J. Castellanos *et al.* "The spmap: A probabilistic framework for simultaneous localization and map building." *IEEE Trans. Robot. Autom.* **15**(2), 125–137 (2001).

[8] G. Dissanayake *et al.* "A computationally efficient solution to the simultaneous localization and map building (slam) problem." In "Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)," pages 1009–1014 (2000).

[9] R. E. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ (1957).

[10] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press and Wiley (1960).

[11] M. Keidar and G. A. Kaminka. "Robot exploration with fast frontier detection: theory and experiments." In "Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1," pages 113–120. International Foundation for Autonomous Agents and Multiagent Systems (2012).

[12] A. Farinelli, L. Locchi, D. Nardi, and V. A. Ziparo. "Assignment of dynamically perceived tasks by token passing in multi-robot systems." In "Proceedings of the IEEE Special Issue on Multi-Robot Systems," pages 1271–1288 (2006).

[13] S. Seuken and S. Zilberstein. "Formal models and algorithms for decentralized decision making under uncertainty." *Autonomous Agents and Multi-Agent Systems* **17**(2), 190–250 (2008).

[14] E. Uchibe, M. Nakamura, and M. Asada. "Co-evolution for cooperative behavior acquisition in a multiple mobile robot environment." In "Proc. of the IEEE International Conference on Intelligent Robots and Systems," pages 4254–430. Springer-Verlag (1999).

[15] J. Marden, G. Arslan, and J. Shamma. "Cooperative control and potential games." *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* **39**(6), 1393–1407. ISSN 1083-4419 (2009).

[16] J. V. Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press. ISBN 0691119937 (1944).

[17] P. Straffin. *Game theory and strategy*. Number v. 36 in ANNELI LAX NEW MATHEMATICAL LIBRARY. Mathematical Assoc. of America. ISBN 9780883856376 (1993).

[18] D. H. Wolpert and K. Tumer. "An introduction to collective intelligence." Technical report, Handbook of Agent technology. AAAI (1999).

[19] Gintis. *Game Theory Evolving: A Problem-centered Introduction to Modeling Strategic Interaction*. Economics / Princeton University Press. Princeton University. ISBN 9780691009438 (2000).

[20] J. Rousseau and M. Cranston. *A Discourse on Inequality*. Penguin Books Limited. ISBN 9780141920009 (2003).

[21] I. Kolmanovsky and N. McClamroch. "Developments in nonholonomic control problems." *Control Systems, IEEE* **15**(6), 20–36. ISSN 1066-033X (1995).

[22] S. Chang. *Data Structures and Algorithms*, pages 21–24. Series on software engineering and knowledge engineering. World Scientific Publishing Company Incorporated. ISBN 9789812383488 (2003).

[23] A. Fabrikant, A. D. Jaggard, and M. Schapira. "On the structure of weakly acyclic games." In "Proceedings of the Third international conference on Algorithmic game theory," SAGT'10, pages 126–137. Springer-Verlag, Berlin, Heidelberg. ISBN 3-642-16169-3, 978-3-642-16169-8 (2010).

[24] L. Blume. "Population games." Game Theory and Information 9607001, EconWPA (1996).

[25] J. Marden and J. S. Shamma. "Autonomous vehicletarget assignment: a game theoretical formulation." *ASME Journal of Dynamic Systems, Measurement, and Control* pages 584–596 (2007).

[26] H. Young. *Individual strategy and social structure: an evolutionary theory of institutions*. Princeton paperbacks. PRINCETON University Press. ISBN 9780691086873 (2001).

[27] L. E. Blume *et al.* "The statistical mechanics of strategic interaction." *Games and economic behavior* **5**(3), 387–424 (1993).

[28] W. Arthur, S. Durlauf, D. Lane, and S. E. Program. *The Economy As an Evolving Complex System II: Proceedings*. A Proceedings volume in the Santa Fe Institute studies in the sciences of complexity. Addison-Wesley, Acfanced Book Program. ISBN 9780201328233 (1997).

[29] O. Madani. "Polynomial value iteration algorithms for deterministic mdps." In "Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence," pages 311–318. Morgan Kaufmann Publishers Inc. (2002).

[30] K. Wurm, C. Stachniss, and W. Burgard. "Coordinated multi-robot exploration using a segmentation of the environment." In "Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on," pages 1160–1165 (2008).

Appendix A

Code of Exploration Algorithms in NetLogo®

A.1 Uncoordinated Exploration Algorithm

```
1  globals [obstacle obstacle1 obstacle2 obstacle3 explored_pts
       no_games sensRange]
2  turtles-own [explore]
3  patches-own [explored visible_no_pts]
4
5  to setup
6    clear-all
7    set no_games 0
8    set sensRange 3      ;set sensRange
9    setup-turtles
10   setup-patches
11
12   reset-ticks
13 end
14
15 to setup_two
16   clear-turtles
17   clear-patches
18   plot-pen-up
19   ;clear-plot
20   setup-turtles
21   setup-patches
22
23   reset-ticks
24 end
```

```
25
26 ;;;; CREATE OBSTACLES  ;;;;
27 to setup-patches
28   set obstacle1 (patch-set patch 10 10 patch 11 10 patch 12
       10 patch 10 11 patch 11 11 patch 12 11 patch 10 12 patch
        11 12 patch 12 12 patch 10 13 patch 11 13 patch 12 13
       patch 10 14 patch 11 14 patch 12 14)
29   set obstacle2 (patch-set patch 30 10 patch 31 10 patch 32
       10 patch 30 11 patch 31 11 patch 32 11 patch 30 12 patch
        31 12 patch 32 12 patch 30 13 patch 31 13 patch 32 13
       patch 30 14 patch 31 14 patch 32 14)
30   set obstacle3 (patch-set patch 20 26 patch 21 26 patch 22
       26 patch 20 27 patch 21 27 patch 22 27 patch 20 28 patch
        21 28 patch 22 28 patch 20 29 patch 21 29 patch 22 29
       patch 20 30 patch 21 30 patch 22 30)
31   set obstacle (patch-set obstacle1 obstacle2 obstacle3)
32
33   update-explored
34 end
35
36 to setup-turtles
37   create-turtles 3     ;create 3 robots
38   ask turtle 0 [
39     setxy 23 16
40     face patch-at -1 0
41     set color yellow
42   ]
```

```
43    ask turtle 1 [
44      setxy 21 19
45      face patch-at 0 -1
46      set color blue
47    ]
48    ask turtle 2 [
49      setxy 19 16
50      face patch-at 1 0
51      set color Red
52    ]
53    ask patches [
54      set explored false
55    ]
56    set explored_pts nobody
57  end
58
59  to go
60    if ticks >= 2000 [        ;2000 time steps
61      ifelse no_games >= 20 [      ;20 games
62        stop
63      ] [
64        set no_games no_games + 1
65        setup_two
66        plot-pen-down
67      ]
68    ]
69    move-turtles
```

```
70    update-explored
71    tick                        ;increment the tick counter and
        update the plot
72  end
73
74  ;;;; UPDATE EXPLORED AREAS  ;;;;
75  to update-explored
76    ask turtles[
77      ask patches in-radius sensRange[
78        set explored true
79        set explored_pts (patch-set self explored_pts)
80      ]
81    ]
82    ask patches [
83      set pcolor ifelse-value (explored = true) [black] [green]
84    ]
85    ask (patch-set obstacle1 obstacle2 obstacle3)[ set pcolor
        red ]
86  end
87
88  to move-turtles
89    let selected_patch nobody
90    let maxUtility 0
91
92    ask turtles [
93      let following_patch_1 patch-ahead 1
94      let following_patch_2 patch-ahead 2
```

```
95
96      ;;;;    explore every time step    ;;;;
97      ifelse (following_patch_1 = nobody or member?
           following_patch_1 obstacle = true) [
98        set explore true
99      ] [
100       set explore true
101     ]
102     ;;;;;;;;;;;;;;
103
104     ifelse (explore) [
105       let old_dx dx
106       let old_dy dy
107
108       ask neighbors4 [
109         ifelse (self = nobody or member? self obstacle = true
              ) [
110           set visible_no_pts 0
111         ] [
112           ifelse self = following_patch_1 [
113             set visible_no_pts baseline-utility
                  following_patch_2
114           ] [
115             set visible_no_pts turn-utility
116           ]
117         ]
118       ]
```

```
119       set selected_patch max-one-of neighbors4 [
              visible_no_pts]
120       ask selected_patch [
121         set maxUtility visible_no_pts
122       ]
123
124       if following_patch_1 != nobody [
125         ifelse member? following_patch_1 obstacle = false [
126           if selected_patch != following_patch_1 [
127             ask following_patch_1 [
128               if (maxUtility = visible_no_pts and maxUtility
                      != 0) [
129                 set selected_patch self
130               ]
131             ]
132           ]
133         ] [
134           if selected_patch = following_patch_1 [
135             let modified_neighbors4 nobody
136
137             ask neighbors4 [
138               if self != following_patch_1 [
139                 set modified_neighbors4 (patch-set self
                        modified_neighbors4)
140               ]
141             ]
142
```

```
143             set selected_patch max-one-of modified_neighbors4
                    [visible_no_pts]
144           ]
145         ]
146       ]
147
148       ifelse selected_patch = following_patch_1 [
149         fd 1        ;move forward
150       ] [
151         face selected_patch
152       ]
153     ] [
154           fd 1        ;move forward
155     ]
156   ]
157 end
158
159 ;;;;  UPDATE ACTION BASED ON EPSILON    ;;;;
160 to-report explore-epsilon
161   report ifelse-value (random 10 < 3) [true][false]
162 end
163
164 ;;;;PREDICTED GAIN IN UTILITY FROM PLAYING BASELINE ACTION
    ;;;;
165 to-report baseline-utility [following_patch_2]
166   let tempUtility 0
167   let two_patches_blobs nobody
```

```
168
169   set two_patches_blobs patches in-radius sensRange
170
171   if following_patch_2 != nobody[
172     ask following_patch_2 [
173       set two_patches_blobs (patch-set patches in-radius
              sensRange two_patches_blobs)
174     ]
175   ]
176
177   ask two_patches_blobs [
178     update-plots
179     if explored != true [
180       set tempUtility tempUtility + 1
181     ]
182   ]
183
184   report tempUtility
185 end
186
187 ;;;;PREDICTED GAIN IN UTILITY FROM TURNING IN A SPECIFIC
      DIRECTION;;;;
188 to-report turn-utility
189   let tempUtility 0
190
191   ask patches in-radius sensRange[
192     if explored != true [
```

```
193         set tempUtility tempUtility + 1
194       ]
195     ]
196
197     report tempUtility
198 end
```

A.2 Algorithm 2

This is the first algorithm that was introduced for exploring bounded spaces.

```
1  globals [obstacle obstacle1 obstacle2 obstacle3 explored_pts
       no_games sensRange]
2  turtles-own [explore]
3  patches-own [explored visible_no_pts]
4
5  to setup
6    clear-all
7    set no_games 0
8    set sensRange 3           ;set sensRange
9    setup-turtles
10   setup-patches
11
12   reset-ticks
13 end
14
15 to setup_two
16   clear-turtles
17   clear-patches
18   plot-pen-up
19   ;clear-plot
20   setup-turtles
21   setup-patches
22
```

```
23    reset-ticks
24  end
25
26  ;;;; CREATE OBSTACLES  ;;;;
27  to setup-patches
28    set obstacle1 (patch-set patch 10 10 patch 11 10 patch 12
        10 patch 10 11 patch 11 11 patch 12 11 patch 10 12 patch
         11 12 patch 12 12 patch 10 13 patch 11 13 patch 12 13
        patch 10 14 patch 11 14 patch 12 14)
29    set obstacle2 (patch-set patch 30 10 patch 31 10 patch 32
        10 patch 30 11 patch 31 11 patch 32 11 patch 30 12 patch
         31 12 patch 32 12 patch 30 13 patch 31 13 patch 32 13
        patch 30 14 patch 31 14 patch 32 14)
30    set obstacle3 (patch-set patch 20 26 patch 21 26 patch 22
        26 patch 20 27 patch 21 27 patch 22 27 patch 20 28 patch
         21 28 patch 22 28 patch 20 29 patch 21 29 patch 22 29
        patch 20 30 patch 21 30 patch 22 30)
31    set obstacle (patch-set obstacle1 obstacle2 obstacle3)
32
33    update-explored
34  end
35
36  to setup-turtles
37    create-turtles 3     ;create 3 robots
38    ask turtle 0 [
39      setxy 23 16
40      face patch-at -1 0
```

```
41      set color yellow
42    ]
43    ask turtle 1 [
44      setxy 21 19
45      face patch-at 0 -1
46      set color blue
47    ]
48    ask turtle 2 [
49      setxy 19 16
50      face patch-at 1 0
51      set color Red
52    ]
53    ask patches [
54      set explored false
55    ]
56    set explored_pts nobody
57  end
58
59  to go
60    if ticks >= 2000 [              ;2000 time steps
61      ifelse no_games >= 20 [       ;20 games
62        stop
63      ] [
64        set no_games no_games + 1
65        setup_two
66        plot-pen-down
67      ]
```

```
68    ]
69    move-turtles
70    update-explored
71    tick                        ;; increment the tick counter and
          update the plot
72  end
73
74  ;;;; UPDATE EXPLORED AREAS  ;;;;
75  to update-explored
76    ask turtles[
77      ask patches in-radius sensRange[
78        set explored true
79        set explored_pts (patch-set self explored_pts)
80      ]
81    ]
82    ask patches [
83      set pcolor ifelse-value (explored = true) [black] [green]
84    ]
85    ask (patch-set obstacle1 obstacle2 obstacle3)[ set pcolor
          red ]
86  end
87
88  to move-turtles
89    let selected_patch nobody
90    let maxUtility 0
91
92    ask turtles [
```

```
93      let following_patch_1 patch-ahead 1
94      let following_patch_2 patch-ahead 2
95
96      ifelse (following_patch_1 = nobody or member?
            following_patch_1 obstacle = true) [
97        set explore true
98      ] [
99        set explore explore-epsilon    ;explore according to
              epsilon
100     ]
101
102     ifelse (explore) [
103       let old_dx dx
104       let old_dy dy
105
106       ask neighbors4 [
107         ifelse (self = nobody or member? self obstacle = true
                ) [
108           set visible_no_pts 0
109         ] [
110           ifelse self = following_patch_1 [
111             set visible_no_pts baseline-utility
                    following_patch_2
112
113           ] [
114             set visible_no_pts turn-utility
115           ]
```

```
116         ]
117       ]
118       set selected_patch max-one-of neighbors4 [
             visible_no_pts]
119       ask selected_patch [
120         set maxUtility visible_no_pts
121       ]
122
123       if following_patch_1 != nobody [
124         ifelse member? following_patch_1 obstacle = false [
125           if selected_patch != following_patch_1 [
126             ask following_patch_1 [
127               if (maxUtility = visible_no_pts and maxUtility
                     != 0) [
128                 set selected_patch self
129               ]
130             ]
131           ]
132         ] [
133           if selected_patch = following_patch_1 [
134             let modified_neighbors4 nobody
135
136             ask neighbors4 [
137               if self != following_patch_1 [
138                 set modified_neighbors4 (patch-set self
                       modified_neighbors4)
139               ]
```

```
140                ]
141
142                set selected_patch max-one-of modified_neighbors4
                       [visible_no_pts]
143              ]
144            ]
145          ]
146
147          ifelse selected_patch = following_patch_1 [
148            fd 1          ;move forward
149          ] [
150            face selected_patch
151          ]
152        ] [
153              fd 1        ;move forward
154        ]
155    ]
156 end
157
158 ;;;;  UPDATE ACTION BASED ON EPSILON    ;;;;
159 to-report explore-epsilon
160   report ifelse-value (random 10 < 3) [true][false]
161 end
162
163 ;;;;PREDICTED GAIN IN UTILITY FROM PLAYING BASELINE ACTION
       ;;;;
164 to-report baseline-utility [following_patch_2]
```

```
165
166   let two_patches_blobs nobody
167   let tempUtility 0
168
169   set two_patches_blobs patches in-radius sensRange
170
171   if following_patch_2 != nobody[
172     ask following_patch_2 [
173       set two_patches_blobs (patch-set patches in-radius
              sensRange two_patches_blobs)
174     ]
175   ]
176
177   ask two_patches_blobs [
178     update-plots
179     if explored != true [
180       set tempUtility tempUtility + 1
181     ]
182   ]
183
184   report tempUtility
185 end
186
187 ;;;;PREDICTED GAIN IN UTILITY FROM TURNING IN A SPECIFIC
      DIRECTION;;;;
188 to-report turn-utility
189   let tempUtility 0
```

```
190
191   ask patches in-radius sensRange[
192     if explored != true [
193       set tempUtility tempUtility + 1
194     ]
195   ]
196
197   report tempUtility
198 end
```

A.3 Algorithm 4

This is the improved algorithm for exploring bounded spaces.

```
1  globals [obstacle obstacle1 explored_pts no_games sensRange]
2  turtles-own [explore]
3  patches-own [explored visible_no_pts]
4
5  to setup
6    clear-all
7    set no_games 0
8    set sensRange 2           ;set sensRange
9    setup-turtles
10   setup-patches
11
12   reset-ticks
13 end
14
15 to setup_two
16   clear-turtles
17   clear-patches
18   plot-pen-up
19   ;clear-plot
20   setup-turtles
21   setup-patches
22
23   reset-ticks
```

```
24 end
25
26 ;;;; CREATE OBSTACLES  ;;;;
27 to setup-patches
28   set obstacle1 (patch-set patch 10 10 patch 11 10 patch 12
       10 patch 10 11 patch 11 11 patch 12 11 patch 10 12 patch
        11 12 patch 12 12 patch 10 13 patch 11 13 patch 12 13
       patch 10 14 patch 11 14 patch 12 14)
29   set obstacle (patch-set obstacle1)
30
31   update-explored
32 end
33
34 to setup-turtles
35   create-turtles 3      ;create 3 robots
36   ask turtle 0 [
37     setxy 23 16
38     face patch-at -1 0
39     set color yellow
40   ]
41   ask turtle 1 [
42     setxy 21 19
43     face patch-at 0 -1
44     set color blue
45   ]
46   ask turtle 2 [
47     setxy 19 16
```

```
48      face patch-at 1 0
49      set color Red
50    ]
51    ask patches [
52      set explored false
53    ]
54    set explored_pts nobody
55  end
56
57  to go
58    if ticks >= 2000 [                ;2000 time steps
59      ifelse no_games >= 20 [         ;20 games
60        stop
61      ] [
62        set no_games no_games + 1
63        setup_two
64        plot-pen-down
65      ]
66    ]
67    move-turtles
68    update-explored
69    tick                        ;; increment the tick counter and
        update the plot
70  end
71
72  ;;;; UPDATE EXPLORED AREAS  ;;;;
73  to update-explored
```

```
74    ask turtles[
75      ask patches in-radius sensRange[
76        set explored true
77        set explored_pts (patch-set self explored_pts)
78      ]
79    ]
80    ask patches [
81      set pcolor ifelse-value (explored = true) [black] [green]
82    ]
83    ask (patch-set obstacle1)[ set pcolor red ]
84  end
85
86  to move-turtles
87    let selected_patch nobody
88    let maxUtility 0
89
90    ask turtles [
91      let following_patch_1 patch-ahead 1
92      let following_patch_2 patch-ahead 2
93
94      ifelse (following_patch_1 = nobody or member?
           following_patch_1 obstacle = true) [
95        set explore true
96      ] [
97        set explore explore-epsilon        ;explore according
             to epsilon
98      ]
```

```
99
100     ifelse (explore) [
101       let old_dx dx
102       let old_dy dy
103       let others other turtles
104
105       ask neighbors4 [
106         ifelse (self = nobody or member? self obstacle = true
                ) [
107           set visible_no_pts 0
108         ] [
109           ifelse self = following_patch_1 [
110             set visible_no_pts baseline-utility
                    following_patch_2 others
111
112           ] [
113             set visible_no_pts turn-utility others
114           ]
115         ]
116       ]
117       set selected_patch max-one-of neighbors4 [
              visible_no_pts]
118       ask selected_patch [
119         set maxUtility visible_no_pts
120       ]
121
122       if following_patch_1 != nobody [
```

```
123         ifelse member? following_patch_1 obstacle = false [
124           if selected_patch != following_patch_1 [
125             ask following_patch_1 [
126               if (maxUtility = visible_no_pts and maxUtility
                      != 0) [
127                 set selected_patch self
128               ]
129             ]
130           ]
131         ] [
132           if selected_patch = following_patch_1 [
133             let modified_neighbors4 nobody
134
135             ask neighbors4 [
136               if self != following_patch_1 [
137                 set modified_neighbors4 (patch-set self
                        modified_neighbors4)
138               ]
139             ]
140
141             set selected_patch max-one-of modified_neighbors4
                    [visible_no_pts]
142           ]
143         ]
144       ]
145
146       ifelse selected_patch = following_patch_1 [
```

```
147           fd 1           ;move forward
148         ] [
149           face selected_patch
150         ]
151       ] [
152             fd 1          ;move forward
153       ]
154     ]
155 end
156
157 ;;;;  UPDATE ACTION BASED ON EPSILON    ;;;;
158 to-report explore-epsilon
159   report ifelse-value (random 10 < 1) [true][false]
160 end
161
162 ;;;;PREDICTED GAIN IN UTILITY FROM PLAYING BASELINE ACTION
      ;;;;
163 to-report baseline-utility [following_patch_2 others]
164
165   let two_patches_blobs nobody
166   let tempUtility 0
167
168   set two_patches_blobs patches in-radius sensRange
169
170   if following_patch_2 != nobody[
171     ask following_patch_2 [
```

```
172       set two_patches_blobs (patch-set patches in-radius
              sensRange two_patches_blobs)
173     ]
174   ]
175   ask two_patches_blobs [
176     let candidate self
177     update-plots
178     if explored != true [
179       let no_gaze true
180       ask others[
181         if no_gaze[
182           let following patch-ahead 1
183           let following_2 patch-ahead 2
184           if following != nobody[
185             ask following[
186               ifelse distance candidate <= 2 [
187                 set no_gaze false
188               ] [
189                 if following_2 != nobody[
190                   ask following_2[
191                     if distance candidate <= 2[
192                       set no_gaze false
193                     ]
194                   ]
195                 ]
196               ]
197             ]
```

```
198             ]
199           ]
200         ]
201         if no_gaze[
202           set tempUtility tempUtility + 1
203         ]
204       ]
205     ]
206
207     report tempUtility
208 end
209
210 ;;;;PREDICTED GAIN IN UTILITY FROM TURNING IN A SPECIFIC
      DIRECTION;;;;
211 to-report turn-utility [others]
212     let tempUtility 0
213
214     ask patches in-radius sensRange[
215       let candidate self
216       update-plots
217       if explored != true [
218         let no_gaze true
219         ask others[
220           if no_gaze[
221             let following patch-ahead 1
222             let following_2 patch-ahead 2
223             if following != nobody[
```

```
224             ask following[
225               ifelse distance candidate <= 2 [
226                 set no_gaze false
227               ] [
228                 if following_2 != nobody[
229                   ask following_2[
230                     if distance candidate <= 2[
231                       set no_gaze false
232                     ]
233                   ]
234                 ]
235               ]
236             ]
237           ]
238         ]
239       ]
240       if no_gaze[
241         set tempUtility tempUtility + 1
242       ]
243     ]
244   ]
245
246   report tempUtility
247 end
```