Bar-Ilan University
Department of Computer Science

Robust Multi-Robot Formations

by

Ruti Glick

Advisor: Dr. Gal Kaminka

Submitted in partial fulfillment of the requirements for the Master's degree
in the Department of Computer Science

Ramat-Gan, Israel
October 2005

This work was carried out under the supervision of

Dr. Gal A. Kaminka

Department of Computer Science, Bar-Ilan University.

Abstract

Robots in formations move while maintaining their relative positions in a predefined geometric shape. Previous work has examined formation-maintenance algorithms that would ensure the stability of the formation. However, for each geometric formation, an exponential number of stable controllers exists. Thus a key question is how to select (construct) a formation controller that optimizes other desired properties, such as sensor usage for robustness. This paper presents a monitoring multi-graph framework for formation controller selection, based on sensor-morphology considerations. We instantiate the monitoring multi-graph framework, and present several contributions. First, we show that graph-theoretic techniques can be used to compute optimal sensing policies that maintain a given formation. In particular, sensor-optimal control laws for separation-bearing (distance-angle) formation control can be automatically constructed. Second, we present a protocol allowing controllers to be switched on-line, to allow robots to adjust to permanent and intermittent sensory failures. We report results from comprehensive experiments using this technique with physical robots. The results reveal the efficacy of the technique in practice. In particular, we show that the use of the dynamic protocol allows formations of physical robots to move significantly faster and with greater precision, while reducing the number of formation failures due to sensor limitations. Finally, we demonstrate how the representation facilitates the selection of a formation leader according to different social criteria, and the control of sensor-heterogeneous robots.

Acknowledgments

First, I would like to express special gratitude and appreciation to my adviser Dr. Gal A. Kaminka. His extensive knowledge, ideas and experience, along with his availability for consulting, constant willingness to guide, and general support were of tremendous help to me. Working with Gal was a very pleasant experience.

In addition, I would like to thank Noa Agmon-Segal for sharing her knowledge with me and helping me in the development of this thesis.

I would like to thank the members of the MAVERICK. The lab was, and I'm sure still is, a very pleasant place to work. They are all good friends and great people to work with.

Finally I would like to thank my family and especially my parents Zeev and Hana Glick, simply for being there. The homey environment, encouragement and support they provided me were simply priceless.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Making a group of robots move autonomously in formation is a challenge which has been of significant interest in recent years. Addressing this challenge is important for many applications in the real world, from military missions, such as the use of autonomous vehicles for supply-chains, or deception, to robotic entertainment. Various formation maintenance algorithms have been suggested (e.g., [3, 4, 17, 11, 9, 20, 19, 21, 5, 16]).

In the popular *leader-referenced* approach to formation-maintenance (formation control) tasks, robots maintain their relative position with respect to their peers, according to a desired geometric shape. The algorithms assign each robot with a single or multiple neighboring robots (*targets*) that it must monitor, to maintain the given geometric shape while moving. The movement of the group is derived from the movement of a special target, called the *leader*, which does not follow any other target. Instead, the leader moves based on the location of the goal (or is operator-controlled). Those monitoring it automatically move to maintain their relative positions. Those monitoring them then react as well, etc. The set of assigned targets, the leader, and their associated controller type are called the *control graph* in [11, 9].

Previous work has examined constraints on a given control-graph, that would ensure the stability of the formation. In particular, two popular methods for such control are *Separation-Bearing Control* (SBC), and *Separation-Separation Control* (SSC) [15, 9]. In both, a single robot is chosen as the leader of the formation. In SBC, each robot (except the leader) must maintain a given distance (separation) and angle (bearing) with respect to an assigned target. In SSC, each robot (excluding the leader) maintains its distance with respect to two different targets. It has been shown that control-graphs, which induce SSC or SBC controllers for each robot, and satisfy other constraints (e.g., connectivity, a single leader, etc.), are sufficient to maintain stable formations. Both SSC and SBC have been used in formation-control of simulated and real robots (e.g., [3, 15, 16]).

However, for each geometric formation, an exponential number of stable possible control graphs exists [9]. Thus a key question is how to select (construct) a control graph that optimizes desired properties other than stability. Many of these desired properties have to do with each robot's sensor morphology—the type, placement, and configuration of sensors on robot bodies. For instance, a control-graph in which one robot must pan its camera backwards (relative to the direction of movement) is most likely less preferable than one in which the same robot can monitor a target ahead of it. Unfortunately, previous work has often ignored the role of sensor morphology in selecting between control graphs (see Section 2 for a detailed discussion, and notable exceptions).

This paper presents a graph-theoretic framework for control-graph selection based on sensor-morphology. The framework represents alternative sensing schemes in a *monitoring multi-graph*, in which directed weighted edges, denote monitoring capabilities (sensors or communications) and their associated costs. By applying graph-theoretic techniques, optimal control graphs can be efficiently constructed.

We instantiate the monitoring multi-graph framework, and present two contributions. First, we provide an efficient method for automatically constructing sensor-optimal control-graphs for SBC control. This construction also includes selecting the best leader for the group. Second, we present a protocol allowing control-graphs to be switched on-line, to enable the robots to adjust to handle permanent and intermittent sensory failures. Finally, we demonstrate the usefulness of the framework in automated construction of control graphs for sensor-heterogeneous robots, and in selecting leaders.

To evaluate these contributions, the monitoring multi-graphs framework has been fully implemented in simulation and with Sony AIBO robots. The Construction of a variety of control graphs, according to a defined team of robots, is presented. We show the results of extensive experiments, which demonstrates the robustness of the formations as a result of using the monitoring multi-graphs. We empirically show that use of the framework leads to significantly increased precision, better performance, and robustness to changing environmental conditions.

This thesis is organized as follows. Section 2 presents related work and background for the formation-maintenance problem in robotics. Section 3 introduces the notions of a monitoring multi-graph and control graph and describes the process of generating the latter from the former. This section also describes the importance of leader selection. Section 4 presents a dynamic control-graph switching algorithm for recovery from formation failures. Results of comprehensive experiments with physical robots and through simulation are presented in Section 5. Section 6 provides a summary and discusses possible directions of future work.

Chapter 2

Related Works

The literature on formation control is vast, thus we will focus the on most relevant studies. To the best of our knowledge, all previous works on formation-maintenance in robotics have made the assumption that sensor configuration matches the control algorithms. Moreover, existing works often assume all robots are homogeneous, and thus do not generate monitoring rules that are individually-tailored to the sensing capabilities of different agents within the formation. Our work addresses these open issues. Additional differences with existing work are discussed below.

Maintaining formation while moving requires the robots to locate themselves according to reference points. [2] examine three techniques for the robot to identify its position. The first one is *Unit-Center-Referenced* where the robots place themselves according to X, Y coordinates defined by the formation. The second is *Leader-Referenced* where all the robots situate themselves relative to one robot who has chosen to lead the group. The last method is *Neighbor-Reference* where each robot locates itself correspondingly to one predestined robot. Their study compares the methods by using teams of up to four physically homogeneous robots. Our approach is suitable for leader-referenced and neighbor-referenced approaches.

Desai et al.[9, 12, 8] expand the neighbor-reference method, and show that formation can be maintained if each robot monitors an angle and distance to another robot, or distances to two other robots. An un-weighted control graph describes the monitoring from a global perspective. Desai et al. do not discuss selection of an optimal control graph; indeed they assume omni-directional sensing. However, they discuss switching the geometric shape defining the formations (and their associated control graphs) to tackle terrain changes. Thus our framework complements theirs.

In order to construct control graph we elaborate the notion of the monitoring graph introduced by Kaminka and Bowling [18].They defined monitoring graphs, as directed graphs where vertices denote robots, and edges denote monitoring capabilities. An edge (a, b) exists in the graph if robot a is able to monitor (observe, communicate with) robot b. They were able to show that certain conditions of the structure of the monitoring graph, corresponding to monitoring conditions of the team-members, must exist in order for the team-members to detect disagreements despite uncertainty in monitoring. Our work extends the monitoring graph in two manners. First, we represent multiple ways in which monitoring can be done (e.g., multiple sensors) as multiple edges between a pair of vertices, i.e., we use multigraphs. Second, we use weights on edges to denote the robot's costs for using the monitoring device associated with the edge. We show how to assign costs to edges in the context of formation-maintenance tasks.

Fredslund and Mataric [16] describe a general algorithm that generates an angle-distance monitor rule for each robot in the formation. The location of robots in formation is determined only by the arbitrary robots' ID and the formation they aim to build. The position of the robot determines which robot it has to follow and which robot will lead the team. This work under assumed sensing capabilities. The monitoring rules were supplemented by communications for robustness. Our algorithms allow for automated selection of the leader, and consider the unique sensor configuration of each robot.

Fierro et al. [15] analyzed the stability of SBC and SSC controllers, and proposed using manually-constructed control laws to allow up to three robots to switch between alternative SSC and SBC schemes, in essence, switching between alternative control graphs on-line without relying on communications. Our work complements their results by providing (i) a method for optimal selection of alternative control graphs, for an unbounded number of robots; (ii) a protocol using communications for making synchronized control-graph decisions, in a distributed fashion.

Lemay et al. [19] and Michaud et al. [20] present a method for quantifying the cost of using the sensors to determine distance and angle to a neighbor. They provide an algorithm where each robot senses its distance and angle from the others. It then broadcasts this information. Using this knowledge, each robot finds the robot to follow according to the one with the smallest deviation of angles and distance combination to it. However, this is used only in selecting the formation positions of all robots. In contrast, our method uses cost information, after the positions have been assigned, to select the optimal target for each robot. Also, we use additional sensory cost factors for the initialized control graph **in order to allow** (Ruti: it was: as for allowing) dynamic switching of control graphs on-line.

Naffin and Sukhatme [21] suggest an on-line method allowing a group of robots to organize into formation. They designed an approach to grow a formation, one robot at a time. Vision limitations, such as lack of cameras pointing backwards, are used in determining the robots' arrangement in the formation. The order of gathering along with these limitations determine the selected robot to lead the formation, and the placement of the other members. While we have not addressed the issue of how robots choose their positions within a formation, we have developed a general method that allows heterogeneous robots to select the best sensing scheme that maintains the formation. We have also provided an algorithm for selecting the formation leader optimally.

Balch and Hybinette [4] apply social potential fields which use attraction and repulsion to position robots within their relative positions in a defined formation. This technique is robust to obstacles in the path of the robots. This is an important challenge our approach does not yet take into account. However, their technique cannot guarantee robots will form into the desired shape.

Dudenhoeffer and Jones [13] also based their work on the principle of social potential fields. They deal with a large scale group of robots. Each robot combines two main elements: sensors and behaviors. The sensors are designed to be able to scan an area of 220^0 (with expansion possibility). This enables the robot to detect some of its team members. The behavior is composed of a basic wandering behavior, group formation behavior above and collision avoidance at the top. This architecture is built for a diverse set of team goals, including the examination of surroundings. Our algorithm is designed for a specific purpose of movement in formation. The robots **must (Ruti: it was: have) to follow only one robot each, rather than compute a social potential field.

Carpin and Parker [5] present a platform for the formation maintenance problem. Their platform is based on a situated automaton in a team's operation level which is affected by states of the robot at the individual level. The group level contains three states: following, waiting and recovering. The waiting and recovering states are designed to get over dynamic obstacle by waiting for them to move on. Our algorithm is managed as a situated automaton composed from these main states. Recovery operation in our algorithm, performed by updating the control graph, demands coordination between the team's members. To ensure this coordination, the recovery state itself being composed from "wait" and "recover" states.

An alternative to our work, that increases robustness, is to utilize global knowledge and continuous communications. Parker [22] examines the need to combine local information with global knowledge in order to carry out cooperative tasks. She investigated this issue in formation-maintenance problems. As shown in her work, adding global knowledge to the local knowledge may improve task performance. However, since this information is very expensive and may even be unachievable, we try to reduce the need for general knowledge.

Another issue is tracking the trail of the target robot. Chiem and Cervera [6] propose using the distance and the angle from the tracked robot, detected by sensors, to build the Bézier curve. Following these curves enable more accurate tracking. Our work handles the issue of which robot to track and not how to track it, thus complementing their work.

Investigations have also been conducted on formation stability [23], splitting and joining formations [1, 21], obstacle avoidance with formations [4, 10] and switching formations [11, 9]. Our study does not deal with these issues and we leave these questions for future research.

Chapter 3

Cost-Optimal Formation Control Graphs

In this chapter we begin by describing the use of monitoring multi-graphs to analytically represent various ways in which a robot may monitor its peers by observation or communication (Section 3.1).

We then describe how a multi-graph can be used in formation-maintenance tasks to assist in the automatic generation of monitoring rules for robots, such that coordinated movement is maintained. We elucidate how our method is useful for a group of heterogeneous robots (Sections 3.2).

Finally we explain the importance of the formation leader, the robot that will lead the team in the formation-maintenance task. (Section 3.3).

3.1 Monitoring Multi-Graphs in Formation Control

We introduce the use of multi-graphs to represent the monitoring capabilities of robots in a multi-robot system. A *monitoring multi-graph* is a tuple $\langle V, E\rangle$ where V is a set of vertices denoting robots, and E is a *bag* (sometimes called multi-set) of weighted edges $\{\langle u, v, w\rangle\}$, each linking two vertices $u, v \in V$, and having a non-negative weight $w \in \mathbb{N}$. Since E is a bag, an edge linking two vertices may appear more than once (even with the same weight).

Edges denote monitoring capabilities. An edge $< u, v, w >$ exists if robot u is able to monitor (sense, communicate to, observe, or otherwise gain knowledge of the state of) robot v in some distinct fashion, e.g. through a specific sensor. The weight w indicates the monitoring robot's cost for using the sensor. As multiple sensors (or methods) may exist for one robot to monitor another, multiple edges may exist with various costs. When a robot can monitor another, the reverse is not always true. Thus edges are directed, i.e.,$< u, v, w > \in E \nRightarrow < v, u, w > \in E$.

In practice, most tasks require monitoring to be selective. A monitoring multi-graph can be useful in such reasoning, and can allow the robot to represent monitoring options which it has available, and the costs involved. The robots can reason about their monitoring decisions in the context of global monitoring constraints. The following sections will present techniques for such reasoning.

We use monitoring multi-graphs to represent the sensory capabilities of robots in the formation. Here, vertices (denoting robots) have an associated position which denotes the associated robot's position in the formation, relative to its peers. The multi-graph is constructed such that it takes into account the positions of the robots, and their (possibly heterogeneous) sensor configurations, in terms of range, field of view, and panning angles.

The input into the construction phase is a given formation that leads to the assignment of robots to places in the formation, essentially a multi-graph with no edges. The initial pose of all robots is towards the direction of the movement. Previous work typically assumes that teams are homogeneous, and thus do not address preferences of allocation of robots to places. Though we do not make such an assumption (see below), we leave allocation for future work.

For each robot (vertex), we add edges by considering its sensors, which can be used for monitoring other robots. We focus on sensors that can provide identification, distance, and bearing to other robots, for instance, a combination of cameras and distance sensors. We exclude sensors that cannot be used for monitoring others, such as location (e.g., GPS), distances traveled (e.g., odometry), etc.

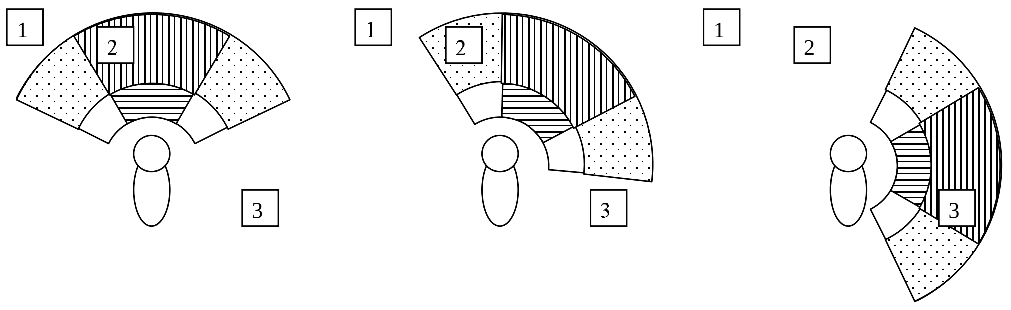
At the beginning of task each robot has information about all other members of its team. The information includes each robot's sensor morphology, and its place with respect to the leader of the group. Simple calculation reveals distances and angles between each pair of robots. Taking into account these parameters relative to a sensor's characteristics leads to the cost involved. The cost will be the weights of the edges.

The weight of an edge is an indication of its *expected* cost-of-usage: Smaller weights indicate better lower costs, and thus greater preference for usage. This cost can be computed based on any number of factors, however we empirically found the following three factors to be useful in practice: sensing distance limits, field of view limits, and panning angle (rotation of the field of view with respect to the center of the robot). We therefore focus on these factors in this paper. Other factors (e.g., for modeling energy consumption, communication bandwidth and/or reliability, etc.) will be addressed in future work.

We assign a cost to each factor for each relevant range of values. For instance, Table 3.1 shows an example of a set of such assignments, for a hypothetical robot (used in the simulation experiments). The first column (attribute) marks the sensor attribute in question—distance, field of view, or panning angle. The second

Attribute	Range	Cost
Distance (mm)	$[0, 450]$	0.4
	$(450, 600]$	0.75
Field of View	$[-30°, 30°]$	0.2
	$(30°, 50°]$	0.4
	$[-50°, -30°)$	0.4
Pan	$[-90°, 90°]$	0.6

Table 3.1: Type-1 Robot Sensor Configuration.



(a) Pan $0°$. (b) Pan $30°$. (c) Pan $90°$.

Figure 3.1: Monitoring possibilities change based on sensor panning.

column (range) marks the values (ranges of values) for which we wish to specify costs. The costs are noted in the final column. Notice that several ranges may be possible for each attribute, which may differ in their costs or range of values. Again recall that lower costs signify higher preferences.

Figure 3.1 shows the Type 1 robot using its single sensor at different pan angles. Each curved subregion denotes monitoring areas with different costs. The two arcs differentiate distance limitations. The numbered squares denote other robots. Figure 3.1-a, for instance, shows the robot panning straight ahead (at $0°$). Square 1 shows a robot that is outside of the distance range of the monitoring, regardless of the panning angle or the field of view. The bottom right robot (square 3) cannot be monitored given the current pan and field of view. The remaining robot (square 2) is currently within the central field of view. Figures 3.1-b,c show all robots in the same positions, but with different panning angles for the sensor.

To compare alternative sensing possibilities, we use edges from the monitoring robot to the other (monitored) robots. An edge will be created for each sensor.

This is done as follows.

First, we compute the area covered by a sensor, given its field-of-view possibilities, distance ranges, and pan options. For a field-of-view range $[f_{min}, f_{max}]$, a pan range $[p_{min}, p_{max}]$, and a distance range $[d_{min}, d_{max}]$, the area covered is a curved region enclosed by the distance range, and defined by the arcs at an angle $[p_{min} + f_{min}, p_{max} + f_{max}]$. Multiple pan, field-of-view, and distance range options give rise to multiple curved regions, which may overlap. For instance, based on Table 3.1, the leftmost field-of-view range covers the arc $[-50 + -90, -30 + 90] = [-140, 60]$ degrees, the central field of view covers $[-30 + -90, 30 + 90] = [-120, +120]$ degrees, etc.

Then, we find robots that are within each region. For each of these robots, we create a temporary directed edge from the monitoring robot. Since the positions of vertices in the multi-graph correspond to geometric positions in the formation, the distance between two robots corresponds to the length of the line connecting them, and the angle between any two robots can be computed relative to the initial pose which faces the direction of movement.

For instance, Figure 3.1 presents a robot with only one sensor. The left top robot is outside of the distance range of the robot's monitoring according to this sensor. Thus there would be no edge from the monitoring robot to this left top robot. Figures 3.1-a, and b, show multiple ways in which the robot ahead of the monitoring robot (and slightly to the left) can be monitored—within the central field of view (when the pan angle is set to $0°$) and within the left field of view (pan angle set to $30°$). Thus two temporary edges to robot would be created.

In the next step we compute the weight of each temporary edge, as a function of the costs of the distance, field-of-view and pan ranges involved. We use a weighted sum function to combine cost factors into a single cost value for the weight of the edge.

$$weight(e) = \sum_{i} w_i \cdot cost(i)$$

where i is the sensor attribute in question (distance, field-of-view, pan angle), w_i is the weight of the sensor attribute, and $cost(i)$ is the cost of using the sensor attribute in the given configuration (i.e., the appropriate table entry).

Finally, all temporary edges constructed according to the same sensor are merged to one edge. This edge receives the weight of the edge with the lowest weight from the edges that have merged. This edge is added to the monitoring graph. As a result, a maximum of one edge per combination of robot to monitor and sensor exists in the multi graph. The process of building the monitoring multi-graph is depicted in Algorithm refalg:MultiGraphGenerate

In real-world settings, robots may occlude each other. Thus the last step after initializing the monitoring multi-graph includes removal of physically occluded

Algorithm 1 MultiGraphGenerate()

```
1: Multi-graph $\mathcal{MG} \leftarrow \emptyset$
2: for each robot $r_i$ participates in task do
3:     Define $i$ as vertex donates $r_i$ in multi-graph $\mathcal{MG}$
4:     Locate $i$ in its place
5: for each vertex $i$ do
6:     for each vertex $j$ do
7:         $dist_{i,j} \leftarrow$ Calculate distance from $i$ to $j$
8:         $angle_{i,j} \leftarrow$ Calculate angle from $i$ to $j$
9:         for each vision sensor $k$ of robot $r_i$ do
10:            $w \leftarrow$ CalculateWeight($dist_{i,j}, angle_{i,j}, k$)
11:            if $w$ is a finite number then
12:                Add edge $e_{i,j,w}$ from $i$ to $j$ to $\mathcal{MG}$
13: Return $\mathcal{MG}$, directed weighted multi-graph
```

edges. As embodied robots occlude each other, any robot x positioned on an edge between a pair of other robots a, b will block their view of each other. Thus any edges $\{< a, b >, < b, a >\}$ are removed from the monitoring multi-graph. When applying this technique with physical robots, we have found it useful to consider occlusion even if x is not positioned exactly on the edges between a and b, to account for the size of the physical body of an occluding robot. This step is described in Algorithm 2.

In addition the body of a monitoring robot may block itself from sensing other robots. Edges suffering from this situation also have to be removed during this step. This issue will be handled in future work.

Algorithm 2 RemoveOcclusion(multi-graph $\mathcal{MG}$)

```
1: for each bag of edges $E_{i,j}$ from vertex $i$ to vertex $j$ do
2:     for each vertex $h$ do
3:         $dist_{i,j} \leftarrow$ Calculate distance from $i$ to $j$
4:         $dist_{i,h} \leftarrow$ Calculate distance from $i$ to $h$
5:         $angle_{i,j} \leftarrow$ Calculate angle from $i$ to $j$
6:         $angle_{i,h} \leftarrow$ Calculate angle from $i$ to $h$
7:         if $angle_{i,j} \approx angle_{i,h}$ and $dist_{i,h} < dist_{i,j}$ then
8:             Remove all $e_{i,j,w} \in E_{i,j}$ from $\mathcal{MG}$
9: Return $\mathcal{MG}$, directed weighted multi-graph
```

The result of this process (after it is repeated for all robots, and all sensors) is a weighted, directed, monitoring multi-graph where vertices denote robots (in their relative positions), and (multi-)edges represent all possible ways in which the robots can monitor each other, given their sensors and their limitations.

Assumptions about the similarity of robots in the group are not made during this process of graph generation. Edges from each vertex are defined according to a robot represented by this vertex. Each edge represents the features of one sensor keeping track of a given distance and angle. Thus the algorithm is suitable for a heterogeneous group of robots. Each robot may have its own sensors' configuration and morphology. The only condition is that robot's sensors details will be predefined and known by all members of the team. (see Section 5).

From this step forward, the algorithm manipulates only the graph and not the robots, as shown in the following sections. The only way in which the heterogeneity of the group and the sensors' configuration find expression is in the monitoring multi graph generation. All the knowledge that is relevant to the algorithm is represented in this multi-graph.

3.2 Computing Optimal Control Laws

Now that the monitoring multi-graph is complete for a given formation, it can be used to induce individual controllers for each robot, such that if all robots maintain the distances and angles represented by the selected edges, the formation will be correctly maintained. In particular, we show how to use a version of Dijkstra's single-source shortest paths (S3P) algorithm to construct SBC controllers for each robot, that guarantee optimal-cost formations.

In SBC, a robot—called *leader*—is responsible for determining the overall global path (e.g., by deferring to a human operator [14] or by using a path planner). Each of the other robots (*followers*) is given an individually-tailored control rule, restricting it to maintain a given distance and angle (with respect to the direction of movement) to its *target*—either the leader, or another follower—that in turn monitors its own target. Separation-Bearing control (SBC) thus relies on a single monitoring link for each robot.

We can deduce the SBC monitoring rules from the monitoring multi-graph, by choosing edges that signify sensor choices. The edges length and angle with respect to the initial position signify the separation and bearing, respectively. For now, we assume that the leader position has been pre-determined (however below we show how the leader may be optimally selected). Given the leader, a formation graph can be maintained using SBC under the following conditions:

1. The out-degree of the leader robot is 0.

2. The out-degree of every tracking robot is exactly 1 and the outgoing edge is pointing to its target.

3. A path exists from every follower to the leader.

The first condition guarantees that the leader does not have to monitor anyone to fulfill its role. The second condition guarantees that every robot (other than the leader) has target which it can monitor for separation and bearing. The final condition guarantees that the formation is connected such that all sequences of robots monitoring others will eventually monitor the leader. In other words, it guarantees that the leader robot is indeed positioned such that it is capable of leading, given how it is monitored.

We define a formation graph as optimal, if in addition to the conditions above, it also guarantees that each individual robot monitors the leader, directly or indirectly (transitively) using the minimal sensor cost. In general, this cannot be achieved by simply selecting the least costly edge of each robot's position, since such local selection may cause robots to form a cycle. In other words the robots monitor each other instead of the leader. Moreover, such local selection does not address a key challenge: a robot's overall monitoring cost in the context of a formation also depends on its target's monitoring cost. This is because a robot's position depends on its target, and thus shorter paths to the leader reduce latency in position update. To be precise, it may be better to monitor a robot at a higher local cost to guarantee that overall the path from the target to the leader is shorter and less expensive.

Fortunately, graph-theoretic algorithms have already been devised to address such challenges. In particular, we use a version of Dijkstra's S3P algorithm (described in [7]). However, rather than compute the shortest path from a source vertex to all others, we compute the Single *Target* Shortest Path. This is easily done by traversing edges backwards.

Another deviation from Dijkstra's algorithm is that it must be modified to work in multi-graphs. In particular, its edge-selection policy now must consider multiple edges between any two vertices. It can be shown that this does not change the optimality of the algorithm. Our proof relies on the optimality of Dijkstra's algorithm which in turn depends on a greedy step. The algorithm begins with a source vertex that donates the leader in our scenario. Each iteration the algorithm chooses the next vertex, the one with the lowest cost of the total path from the vertex to the leader, and updates the other vertex with the lowest cost possible. If two edges with different weights exist between the same vertices the one with lowest cost will be chosen, since this leads to a lower total cost. In fact we can look at an edges' bag as one edge with a cost equal to the minimal weight. Modifying this step such that it considers multiple edges does not modify the result of this step. After a vertex has been selected, all of its outgoing edges can be ignored. According to the Dijkstra algorithm, none of them will be used in the rest of the algorithm.

This step in which edges are selected also touches on a final modification of Dijkstra's algorithm for our purposes. In theory, any ties in alternatives (i.e., edges lead to the same total weight) can be broken arbitrarily by the algorithm, since the selection will not affect optimality. In practice, however, we have found it useful to reduce *hops*, i.e. the number of edges that leads from a given robot to the leader. The reason for this is that the more edges there are in the path the more robots it may effect. Each deviation of a robot in the path may cause deviation for all the robots following it directly or transitively. Moreover, response time increases since it is composed of the response times of all robots in the path. A long response time makes formation maintenance difficult.

To overcome these influences we break ties in such a manner as to prefer edges that minimize hops. This is done by introducing secondary uniform weights on each edge that are used to count hops to the leader. These weights are updated in the algorithm's iterations as the total path's weight. If two vertices have the same total paths' weight, the one with the smallest number of hops will be selected. In cases where the number of hops is also equal, the robot with the lowest ID will be chosen.

Using the modified Dijkstra's algorithm (Algorithm 3), a single edge is selected optimally for each robot except the leader. These edges form an SBC control-graph to be executed by the robots, i.e., the algorithm induces a control graph $\mathcal{G}$ from the monitoring multi-graph $\mathcal{MG}$. Because each edge is specific to the robot in which it originates, the SBC control law of each robot is individually tailored to the monitoring capabilities of the robot. This enables sensor-heterogeneous robots as mentioned before.

3.3 Leader Selection

We have seen above, that given a leader, we can calculate the best formation graph for its followers. However, it is also possible to find the best leader for the team. Leader selection has great influence on fulfilling the formation maintenance task. The leader is the one that is responsible for movement and determining the path and speed. Keeping the leader alive and maintaining contact with it is a necessary condition for completion of the task. The important assignment of choosing a leader should take under consideration two criteria: (i) The ability of the chosen robot to lead the group; and (ii) the ability of the group to follow this robot. These issues are discussed below.

As mentioned, the leader can choose its path by using a path planner, or by

Algorithm 3 ModifiedDijkstraAlgorithm()

STATIC: dist - vector hold total weight from each vertex to the leader target - vector hold target of each vertex (robot) sensor - vector hold sensor used for each robot monitoring its target weight - vector hold cost of using best sensor for each robot monitoring its target hops - vector hold number of hops from robot i to the leader

```
 1: for each vertex $i \in \mathcal{MG}$ do
 2:     $target[i] \leftarrow null$
 3:     $sensor[i] \leftarrow null$
 4:     $weight[i] \leftarrow null$
 5:     $hops[i] \leftarrow null$
 6: $\mathcal{S} \leftarrow \emptyset$
 7: $\mathcal{Q} \leftarrow$ All vertices in $\mathcal{MG}$
 8: while $\mathcal{Q} \neq \emptyset$ do
 9:     $u \leftarrow$ Closest vertex from $\mathcal{Q}$
10:     $\mathcal{Q} = \mathcal{Q} - u$
11:     $\mathcal{S} = \mathcal{S} \cup u$
12:     for all vertices $v$ where bag of edges $E_{v,u}$ from $v$ to $u$ exists do
13:         $w \leftarrow$ Minimal weight of edge in $E_{v,u}$
14:         $k \leftarrow$ Sensor used for $e_{v,u,w} \in E_{v,u}$
15:         if $weight[u] + w < weight[v]$ OR ($weight[u] + w = weight[v]$ AND
            $hops[u] + 1 < hops[v]$ then
16:             $weight[v] = weight[u] + w$
17:             $target[v] = u$
18:             $hops[v] = hops[u] + 1$
19: $\mathcal{MG}_{new} \leftarrow$ ExtractDijkstraGraph($\mathcal{MG}, target, weight$)
20: Return $\mathcal{MG}_{new}$
```

Algorithm 4 ExtractDijkstraGraph(multi graph $\mathcal{MG}$, targets' vector $target$, weights' vector $weight$)

```
1: $\mathcal{MG}_{new} \leftarrow \emptyset$
2: Copy vertices from $\mathcal{MG}$ to $\mathcal{MG}_{new}$
3: for all vertex i in multi graph $\mathcal{MG}_{new}$ do
4:     Add edge $e_{i,target[i],weight[i]}$ to $\mathcal{MG}_{new}$
5: Return $\mathcal{MG}_{new}$
```

control graph	Social optimal cost	Individual optimal cost
Figure 3.2-b	6	7
Figure 3.2-c	10	8

Table 3.2: Social Vs Individual optimal control graphs' cost

relying on a human operator. In any case it should be equipped with the appropriate capabilities. For example, a wide view of the environment can be useful in finding a path where the group can pass with minimal problems such as obstacles and turns. Stable communication is needed in order to receive commands from a human operator, and to remain attentive to the team members' state. In cases of a group of heterogeneous robots, choosing the robot that physically matches this function is very significant. However, we leave the issue of choosing the best robot to lead the team with the above parameters for future work.

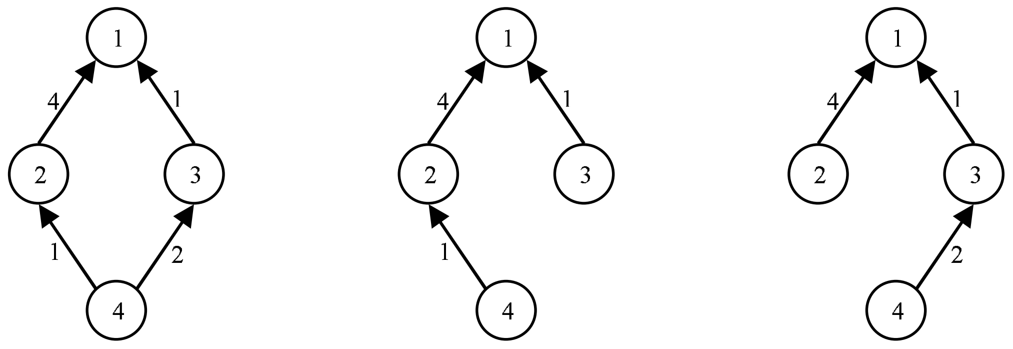
The criterion we used in our work for selecting the leader is the ability of the other robots on the team to monitor it. Each member of the team has to be able to directly or indirectly observe the leader. Placement of the robots relative to the leader and their sensors determine this ability. A control graph is defined as a graph that contains a path from each vertex to one specific vertex that denotes the leader. A graph for a group where vertex i represents the leader contains exactly one outgoing edge from each vertex except vertex i, which has no outgoing edge. Thus a different control graph is needed for each selected leader.

The selection of the best leader for the group depends on the task definition. Two optimality criteria for control graphs are possible: Minimal social (global) cost, or minimal cost for each robot. As in other areas of multi-robot systems, these two optimality criteria do not necessarily coincide, and indeed it is possible to construct graphs that are socially-optimal, yet they are not individually optimal.

Figure 3.2 shows the following for a given monitoring multi graph: (a) the socially-optimal control graph (b) the individually-optimal control graph (c). Different robots assigned as robot 4's target in the two control graphs. The total cost of formations (b) and (c) are shown in table 3.2.

When choosing to minimize the social (global) cost of monitoring in a team, a minimum spanning tree of the multi-graph could be used. However, we believe that this optimality criteria is inappropriate for SBC formation maintenance, because it allows pathological cases in which the global monitoring cost is minimized, while specific robots monitor the leader through very long or expansive paths. The movement of the leader triggers and controls the movement of its followers, and as a result, latency in their responses is a function of the number of monitoring edges that exist between each one and the leader and their quality.

In contrast, leader selection based on individual-optimality is much more ro-



(a) The monitoring multi-graph.

(b) A social-optimal control graph.

(c) An individual-optimal control graph.

Figure 3.2: Social- and individual-optimal control graphs.

bust. The total weight of the path from a robot to the leader represents the cost of monitoring the leader. The higher the total cost, the less reliable the monitoring using this monitoring rule, and thus less preferable. The sum of the evaluated total cost for each robot is used as an estimation for the quality of the total graph

We found that the best control graph for our task is the one with minimal cost for each robot. In order to find the best leader, based on the above criteria, we iterate through all team-members, setting each as the leader (temporarily) and computing the resulting formation graph. If a legal control graph is received (connected graph), we then compute the total weight of the formation graph. Once we have gone through all the leaders, we choose the leader for whom the total weight of all paths to the leader in the formation graph is the smallest. The Pseudo code of the algorithm presented in Algorithm 5.

Algorithm 5 LeaderSelection(multi-graph $\mathcal{MG}$)

```
1: for each vertex $i$ in $\mathcal{MG}$ do
2:     $\mathcal{D}_i \leftarrow$ ModifiedDijkstraAlgorithm($\mathcal{MG}$, $i$)
3:     if connected graph $\mathcal{D}_i$ exists then
4:         $c_i \leftarrow$ EvaluateGraphWeight($\mathcal{D}_i, i$)
5: Return $i$ where $c_i$ is minimal
```

Algorithm 6 EvaluateGraphWeight(control graph $\mathcal{D}_i$, vertex i)

```
1: $sum \leftarrow 0$
2: **for** each vertex $j$ in $\mathcal{D}_i$ **do**
3:     $sum_j \leftarrow 0$
4:     **while** $j \neq i$ **do**
5:         $e_{j,k,w} \leftarrow$ Find the outgoing edge from vertex j
6:         $sum_j = sum_j + w$
7:         $j \leftarrow k$
8:     $sum = sum + sum_j$
9: Return $sum$
```

Chapter 4

Dynamic Switching of Control Graphs

Generation of an SBC control law for each robot is done automatically, based on the *expected* cost of using the robot's sensors. However, during deployment, sensors may act differently from what is anticipated, due to catastrophic or intermittent failures. For instance, a camera may get stuck in a particular angle, or lighting conditions may inhibit the ability to track specific colors.

To address this issue, we propose a distributed protocol that allows robots to dynamically switch control-graphs while maintaining their formation (Section 4.1). This protocol assumes that new costs are assigned to edges that are affected by failures. Section 4.2 discusses the translation of perceived failures into cost changes.

4.1 A Protocol for Control-Graph Switching

A protocol for dynamically switching control-graphs of a given formation must explicitly or implicitly coordinate the robots in the process of switching. Uncoordinated switches may result in two or more robots following each other, cyclically, instead of the leader.

We present a distributed protocol for such coordinated switching. The protocol involves several steps (Algorithm 7).

1. If a robot fails to monitor its current target it first broadcasts a message to all team-members, to let them know that a re-computation of the graph is needed. During this phase, any number of robots may broadcast in parallel.

2. Each robot that receives the message halts the movement, and adds it to a local list of robots $\mathcal{R}$ that require re-assignment of targets and sensors. The

robot receiving the message no longer attempts to maintain the formation, and will not report on any readjustment it wishes to make while at this stage of the protocol.

3. All robots make sure that all messages have been received and processed. This can be done either by having receivers acknowledge received communications, or in a more simplified manner (but less reliably) by having a timeout mechanism that ensures no new messages are generated.

4. All robots call on Algorithm 8 to determine the set $\mathcal{R}_k$ of robots in the team that are potentially affected (i.e., transitively) by a change in the initial list of robots' target assignments.

5. All re-execute the ChangeTarget algorithm on the monitoring multi-graph $\mathcal{MG}$, to update the control graph. However, because only the subset of team-members $\mathcal{R}_k$ is affected, decisions for other robots do not have to be revisited.

Algorithm 7 DynamicSwitching()

```
 1: $\mathcal{R} \leftarrow \emptyset$
 2: $\mathcal{G} \leftarrow$ the current control graph
 3: **while** true **do**
 4:     **if** I lost my target **then**
 5:         Stop movement and wait
 6:         Add my name to $\mathcal{R}$
 7:         Broadcast "wait for me" message to all members
 8:         Start timeout counter
 9:     **if** got "wait for me" massage from robot $r_i$ **then**
10:         Add $r_i$ to $\mathcal{R}$
11:         Stop movement and wait
12:         **if** not running timeout counter allready **then**
13:             Start timeout counter
14:     **if** timeout over **then**
15:         $\mathcal{R}_k \leftarrow$ GetverteciesToUpdate($\mathcal{G}, \mathcal{R}$)
16:         $\mathcal{G}_{new} \leftarrow$ TargetsReassignment($\mathcal{MG}, \mathcal{R}_k$)
17:         **if** my name $\in \mathcal{R}_k$ **then**
18:             Arrange myself according $\mathcal{G}_{new}$
```

The GetverteciesToUpdate algorithm (Algorithm 8) essentially computes all robots that are *upstream* from an affected robot, where upstream is taken to mean traversing the control-graph edges backwards, from the leader to the outmost followers. The algorithm follows edges backward, from the initial set of robots, adding additional affected robots as it goes. It halts when no new affected robots can be discovered.

Robots not contained in the affected robots group $\mathcal{R}_k$ do not have to search for a new target. This is because of the optimality of Dijkstra's greedy selection of edges in constructing the control graph. When generating the control graph from the monitoring graph, each robot is assigned a target with the cheapest path to the leader. Any other choice of target would lead to a path with equal or higher costs. The only changes that take place in the monitoring graph, as a result of failures, involve increased costs of one or more edges. Paths composed of these edges might become more costly, so robots using these paths—and upstream of the modified edges—should look for cheaper alternatives. Robots downstream from the modified edges do not need to search for new targets: Since a cheaper path to the leader did not exist for them before, there is no possibility it will now.

Algorithm 8 GetverteciesToUpdate (control graph G, robots $\mathcal{R}$)

```
1: $\mathcal{R}_k \leftarrow \emptyset$
2: $V \leftarrow \mathcal{R}$
3: **while** $\exists v \in V$ **do**
4:     Remove $v$ from $V$, put into $\mathcal{R}_k$
5:     **for all** $e_{j,v}$, edges from robot $j$ to $v$ **do**
6:         Insert vertex $j$ to $V$
7: Return $\mathcal{R}_k$
```

The protocol above can be executed in parallel by all team-members, or using a centralized computation which will distribute the result. When executed in parallel, care must be taken to ensure that (i) the robots begin their decision-making in a synchronized manner (i.e., work on the same initial list of robots $\mathcal{R}$ and have the same knowledge about where the failure occurred); (ii) arrive at the same choices in the re-computation of the control graph. The first requirement can be enforced in several ways. We chose to enforce it by introducing a timeout mechanism: Once a robot announces that a re-computation is necessary, other robots have a certain time period in which they can add to the list. When a robot receives a message to wait it stops all its behaviors and waits. Thus it is impossible for it to lose its target at this period. The timeout ensures that all messages sent will be sent and received by all the robots before any of the members begin the new calculation. As for the second requirement, to prevent parallel execution of Dijkstra's algorithm from making different decisions, any ties are arbitrarily broken by preferring the robot with the lower ID.

Another possible method is to assign the task of centrally computing the switch

Algorithm 9 TargetsReassignment(control graph $\mathcal{G}$, robots $\mathcal{R}$)

```
 1: Copy vertecies and edge from $\mathcal{G}$ to multi graph $\mathcal{MG}$
 2: for all $i$, vertex representing robot $i$ in $\mathcal{R}$ do
 3:     for all vertex $j \in \mathcal{MG}$ do
 4:         $dist_{i,j}$ ← Calculate distance from $i$ to $j$
 5:         $angle_{i,j}$ ← Calculate angle from $i$ to $j$
 6:         for all vision sensor $k$ of robot $i$ do
 7:             $weight_{i,j,k} \leftarrow$ CalculateWeight($dist_{i,j}, angle_{i,j}, k$)
 8:             $w \leftarrow$ Weight $weight_{i,j,k}$ with the probability for monitoring's failure
                according to history
 9:             Add edge $e_{i,j,w}$ from $i$ to $j$ to $\mathcal{MG}$
10: $v_{leader}$ ← the vertex representing the leader in $\mathcal{G}$
11: $\mathcal{G}_{new}$ ← ModifiedDijkstraAlgorithm($\mathcal{MG}$, $v_{leader}$)
12: if no connected graph $\mathcal{G}_{new}$ exists then
13:     $v_{leader}$ ← LeaderSelection($\mathcal{MG}$)
14:     $\mathcal{G}_{new}$ ← ModifiedDijkstraAlgorithm($\mathcal{MG}$, $v_{leader}$)
15: Return $\mathcal{G}_{new}$
```

for the team to one robot (or an external computer). The centralized unit collects all sent massages announcing the need for re-computation, runs the recovery algorithm (Algorithm 8 and then Algorithm 9) in order to update the control graph, and broadcasts the results to each team member. Each robot receiving the new control graph starts following its target as indicated therein.

The weakness of the centralized method is its requirements of communication bandwidth and reliability. Additional communications are required since, in addition to the messages informing everyone of failures (and requesting them to stop), the centralized unit must also send the results of the computation to all robots. In contrast, in the distributed protocol, each robot made its own calculation.

Communication reliability is also significantly important in the centralized protocol. The centralized unit must have knowledge of all the failures that have taken place. In the distributed system, an unreceived message may affect only the robot that missed it, or may not affect it at all (if the messages concern a branch of the control graph that is separate from its own). In the centralized protocol, if the centralized unit missed a message, it will affect the entire group. Consequently the whole computation of a new control graph will be affected.

4.2 Failures as Cost Changes

The initial parameters of the sensors of each robot do not change by themselves as a result of the failure. If the switching protocol is based on the information in the initial monitoring graph, the same control graph will be generated. Thus some type of method is needed to make a decision as to whether and how to modify the costs in the monitoring graph, to cause the creation of a new control graph when a switch occurs. To do this, we must first decide on the nature of the failure.

Some failures are minor and intermittent, resulting from sudden and short changes to the environment in which the sensor operates, e.g., a temporary short-lived lighting change, or a slight tremor of the camera due to a slip. Such failures may not necessarily lead to a switch in the control graph.

In contrast, other failures require a more thorough treatment (i.e., switching a control-graph). Some of these may be temporary (but for a relatively long duration), such as when taking a sharp turn around an obstacle: The target may disappear from view (blocked by the obstacle) until the turn is completed. Other such failures are permanent, e.g., due to a catastrophic sensor malfunction.

The challenge is to tell the difference between these two types of failures at run-time. On the one hand, if we treat any failure as permanent (modifying the cost of the appropriate edge to infinite), then intermittent failures may cause the monitoring graph to quickly become unconnected, and break the formation. On the other hand, if we are too relaxed in treating failures, then no control-graph switching would occur, and a failure would not be treated by the system.

One possible solution is to record the occurrence of failures, and draw conclusions from the frequency and duration of repeating failures. For this purpose we keep a table, called the *impossible monitoring table*. This table records the relevant part of the target switching history. Intuitively, for each robot i, with target robot j, using sensor k, the value in the $\langle i, j, k \rangle$ entry indicates the likelihood of robot i to experience difficulties in monitoring robot j using its sensor k. Initially, all entries are set to zero. Each time a robot i loses its target j using sensor k and causes a dynamic switch, the entry for $\langle i, j, k \rangle$ grows. The higher the value, the less reliable the edge in question. The values can be normalized to produce a probability distribution.

When running the TargetsReassignment algorithm (Algorithm 9) the information in the *impossible monitoring table* is used to update the weight of the edges. The higher the value in an entry, the higher the weight of the matching edge will be set. When the value in the entry is 1, the corresponding edge receives an infinity value and it will be considered as having been removed from the monitoring graph. The function mapping the table entries into cost adjustments is likely domain-dependent, and is left for future work. When using the entry in a distributed switching protocol, it is important that all the robots will have the same values in the *impossible monitoring table* since this parameter takes part in updating the control graph. Difference in entry values may cause generation of different control graphs for the team members.

The above method considers edge cost changes in the monitoring multi-graph, and as a result, in the generated control graph. As a result, an arrangement where the previous selected leader continues to lead the group may become impossible or costly. Selecting a new robot as a leader is required in this case.

Switching a leader is a complicated and expensive task. The control graph experiences many changes as a result of assigning a new robot as a leader. Sometimes changes demand that robots turn in place. In the real world unnecessary turning results in significant loss of localization accuracy, and should be eliminated as much as possible. But when no other solution can be found leader switching is a possible solution for fulfilling the task. When no possible control graph is generated by the extended Dijkstra's algorithm (Algorithm 3) a new leader is chosen using the leader selection algorithm (Algorithm 5) and an appropriate control graph is generated. Each robot checks if it has been assigned a new target and rearranges itself accordingly. Now movement can resume.

Chapter 5

Experiments

Our evaluation of the use of monitoring multi-graphs in coordination movement involves several stages. We first exemplify a series of experiments on physical robots (Sony AIBOs) demonstrating how automatically-generated, static control graphs are used in the real-world (Section 5.1). The results show that fixed non-switching control graphs can result in diverse performance quality. Then, we prove** (Ruti: it was: show) that the use of the dynamic switching of control graphs solves this problem: In extensive experiments, dynamically-switching formations have proven to be more robust and to out perform a fixed control graph formation (Section 5.2). Finally, we show that the monitoring rules that are produced using our technique for heterogeneous robots, given the desired formation and the robots' sensor specification tables. This evaluation demonstrates that sensor configurations can significantly affect formation monitoring rules (Section 5.3).

5.1 Fixed Control Graphs

A question remains as to the efficacy of the approach in real-world settings, in which robots' monitoring constraints are real, and the resulting monitoring rules are actually used. To address this question, we conducted a series of experiments according to the technique described above with formation maintenance tasks using real world robots.

The first set of experiments uses fixed control-graphs, generated from the monitoring multi-graphs, to control formations of Sony AIBO ERS-7 robots. Colored stickers that were glued to these dog look-alike robots' rear side help to distinguish between them. Each of these robots has a single camera on its panning head which can be used to detect color blobs in its $\approx 120^\circ$ view-field ($[-59^\circ, 59^\circ]$), although in practical terms, the effective view-field is

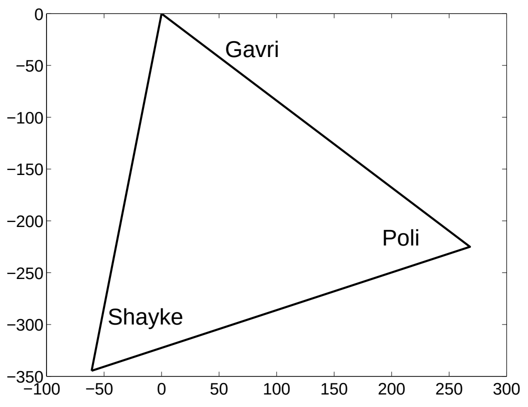
Attribute	Range	Cost
Distance	$[200, 1500]$	0.4
Field of View	$[-35, 35]$	0.5
Pan	$[-90, -40)$	0.7
	$[-40, 40]$	0.2
	$(40, 90]$	0.7

Table 5.1: Sensor Specification, AIBO

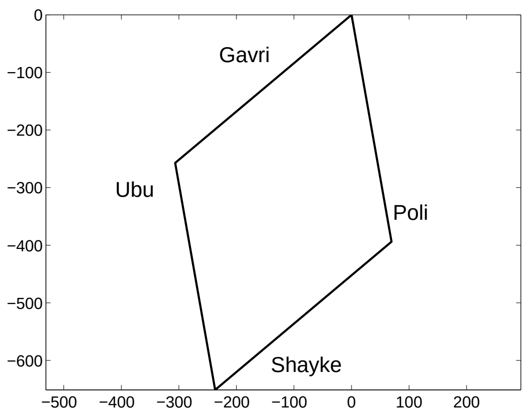
($[-35°, 35°]$). Using such color identification, the robot can identify others, when appropriately color marked. The head also contains an infra-red range sensor which can measure distances in the $[200_{mm}, 1500_{mm}]$ range, with some uncertainty. We treat the head (camera and distance sensor) as a single logical sensor, providing bearing to another robot (identified in the camera image, at a computable angle to the body of the observer), and distance. The head pans $90°$ left and right, and thus the maximal practical angle range for its vision, when combined with the practical field of view of $[-35°, 35°]$, is $[-125°, 125°]$. However, our experience has revealed that maintaining the pan angle in the $[-40°, 40°]$ range tends to produce better results. Based on the manufacturer's specifications and our experience with the robot, we generated sensor cost specifications for the robot (Table 5.1).

To test our technique, we used it to generate control graphs for the triangular formation specified in Figure 5.1-a, and the diamond formation in Figure 5.1-b. Both formations are tilted. For instance, in the triangle, Gavri (the leader) is 20 degrees to the right of Shayke, with respect to the direction of movement. We do not expect the formations to be maintained perfectly, and some leeway in angles and distances is built into the monitoring rules so as to account for sensor uncertainty.

Using Table 5.1, we used our technique to produce alternative control graphs for the formations. We discuss the triangle formation first. In the first (normal) case, the resulting SBC formation had Gavri as the leader (see Figure 5.1 for the robots' names), and the other two robots monitoring it directly (Figure 5.2-a). To experiment with different monitoring rules, we modified the sensor specification table such that the cost of panning in the range $[50°, 90°]$ was 0.7, and infinity anywhere else (forcing the AIBO to look sideways, with only $40°$of leeway). This case simulated a failure, for instance, where the camera pan motor got stuck. Providing this modified input to our algorithm produced a different monitoring graph, where Shayke monitored robot Poly, which in turn monitored robot Gavri (Figure 5.2-b).

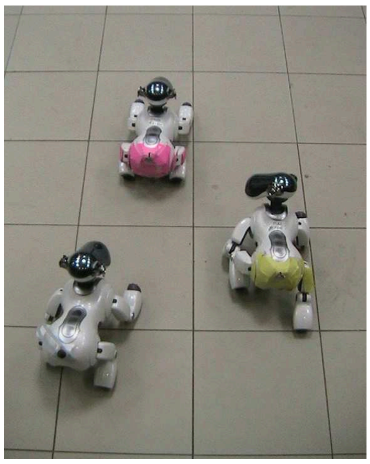


(a) Ideal Triangle.



(b) Ideal Diamond.

Figure 5.1: Ideal formations in fixed control-graph experiments. Robot names are shown.


(a) Shayke follows leader.

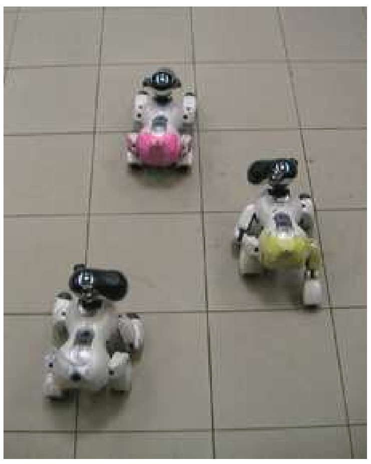

(b) Shayke follows Poly (right robot).

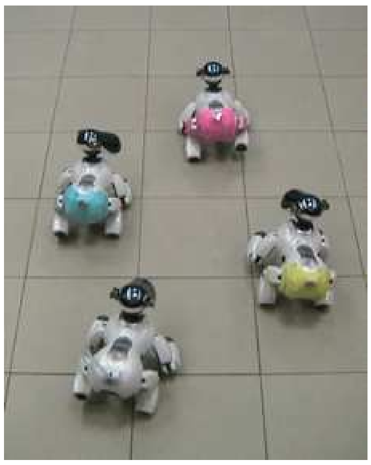
Figure 5.2: AIBO robots executing static triangle SBC control-graphs. Note the position of the bottom robot's (Shayke) head.

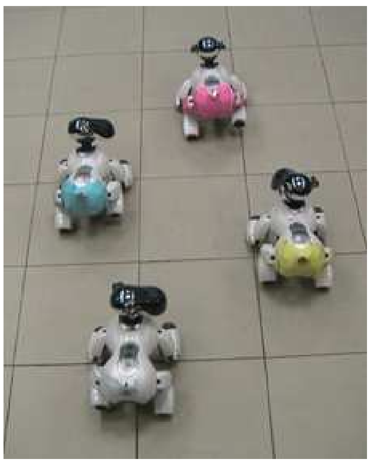
In the diamond formation, when Gavri is the leader of the formation, in principle, many control graphs are possible. They are all presented in Figure 5.4. In practice, when using AIBO ERS-7, only control graphs a-f are suitable to fulfill the task. Graphs j-l also describe possible control graphs according to the AIBO's specifications but they yielded very poor performance in practice since these specifications demand that at least one robot keep an angle of $110°$.

We restricted the algorithm to control graphs in which the last robot, Shayke, was the only one to select different targets. This was accomplished by tweaking its associated cost table, i.e. in effect rendering it heterogeneous from its peers. Thus we experimented with three control graphs: Shayke monitoring the leader (Figure 5.3-a), the right follower (Figure 5.3-b), and the left follower (Figure 5.3-c). All of the control graphs were generated automatically.

We ran 15 trials with each of these alternative SBC formations (a total of 75 trials). In these trials, the leader was controlled manually to determine an obstacle-free straight-line of about six meters in length. The objective was to contrast the stability and robustness of the different control-graphs under reasonable operating conditions.

Distance from the desired position is an important parameter for this task performance, as it is a measure of the error in formation maintenance. Figure 5.5 shows the average deviation from the desired place of all of the robots participat-


(a) Shayke follows Leader.


(b) Shayke follows Poly.

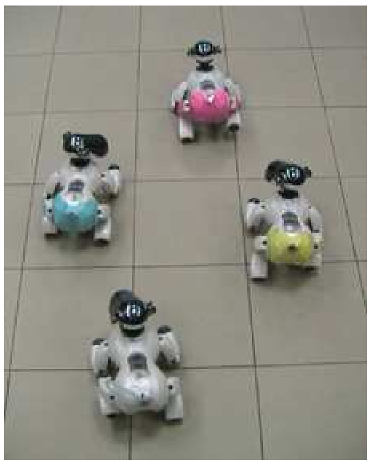

(c) Shayke follows Ubu.

Figure 5.3: AIBO robots executing static diamonds SBC control-graphs. Note the position of the bottom robot's (Shayke) head.

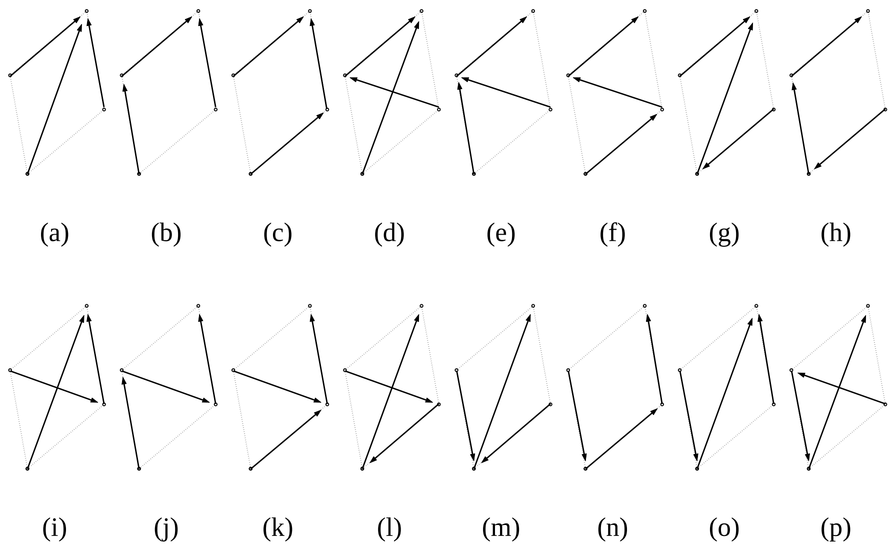


Figure 5.4: All possible control graphs for a group of 4 robots keeping a Diamond formation

ing in the formations, except the leader. In the triangle formation (Figure 5.5-a) we see that when Shayke followed the leader, the error was much smaller than in the other case. This is construed from the fact that the deviation in the position of the right robot (which followed the leader) was the same in both cases. However, when Shayke on the left followed the right robot (Poli) rather than the leader, the error in its position increased significantly. This can also be seen in Figure 5.6-a as explained below.

In contrast to the triangle formation, the results of the diamond formation (Figure 5.5-b) are more tricky. Although it seems that the best formation was maintained when Shayke followed the leader, the opposite is true. The reason for this is that deviation is measured as an absolute value. When Shayke monitored the right or left robot, deviation of the three robots (except the leader) was high but in the same direction. The received diamond was geometrically similar to the desired one. But when the leader monitored, while the left and the right robots were slightly behind the place they had to be, Shayke was too close to the leader. As a result the whole structure was not maintained. This can be seen clearly in 5.6-b.

Another interesting point raised by these graphs is the difference in position errors, of the robots that had the same target throughout all the experiments. The deviation was supposed to remain the same since their monitoring rule did not change. We believe the difference was the result of the effect of Shayke on the other team's member. Shayke's behavior was not the same in all the experiments since its control rule changed. When Shayke lost its target it signaled the whole group to wait while it re-acquired its target. The recovery from this waiting process affects the formation maintenance of these robots.

Figure 5.6 provides a visual presentation of the resulting formations, as represented by the average positions of robots with respect to each other. Figure 5.6-a shows the two triangle formations, while Figure 5.6-b displays the three diamond formations. Each figure also plots the ideal formation for comparison. The formations are shown in an X, Y coordinate system measuring millimeters. For the purpose of comparison, the leader is positioned at $(0, 0)$. The position of the other robots is determined using the measured distance and angle from their targets.

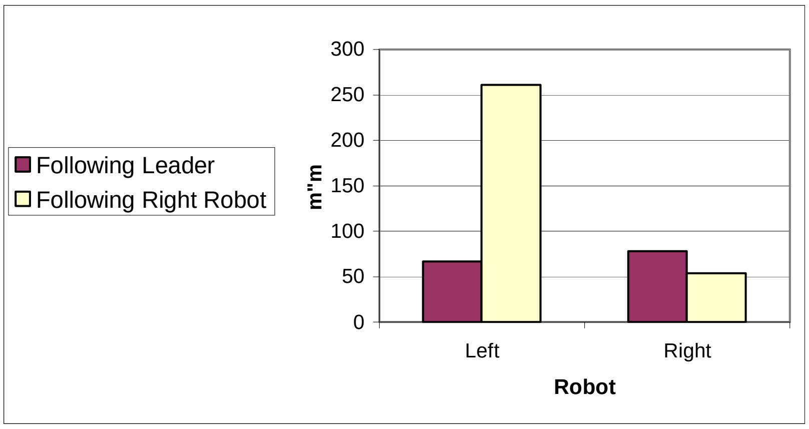
Qualitatively, it can be seen that large variances exist in the quality of the formations when maintained statically by different control graphs. In the triangle formation, the control graph in which Shayke monitors the leader directly yielded a much better formation maintenance than the control graph in which Shayke monitored the leader indirectly, through another robot. However, in the diamond case, the reverse is true. Here, two control graphs yielded good results; both of these control-graphs monitored the leader indirectly. In contrast, the control graph in which Shayke monitored the leader directly shows that the formation was not maintained.

Analysis of the results displayed in Figure 5.6 is not trivial, because the angles and distances are inter-related. For instance, a robot failing to maintain a distance from its target, will inevitably lead to apparent deviations in its angle with respect to the third robot. As a result several issues arise.

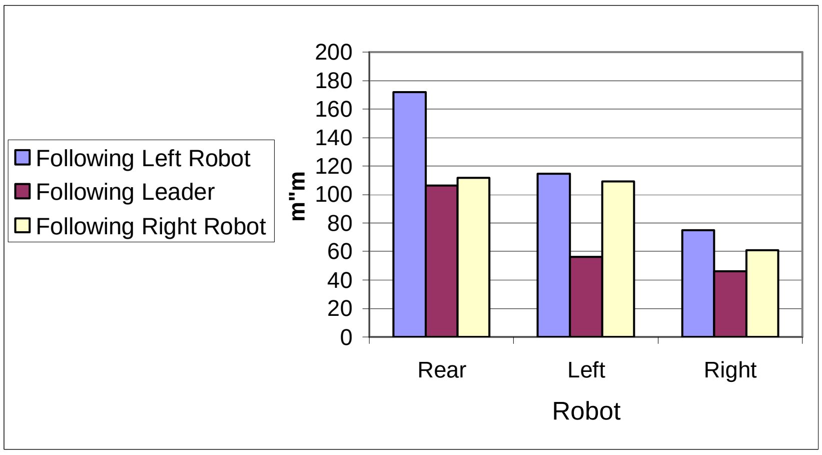
First we will explain the occurrence in the triangle formation. In terms of consistency, it is clear that Poli's (the right robot's) relative position was not very different in both formations, though it was slightly more behind in first case, where Shayke, the left robot, followed the leader. This was expected, as Poli's monitoring rule did not change between the two cases. The error in the position of Poli was 7.7cm in the first case, and 5.3cm in the second case (where Shayke followed Poli). Compared to the ideal length of this edge (35cm), the error in the first case was 22.2% and in the second case the error was approximately 15.3%.

On the other hand, Shayke's position errors were statistically significantly different between the two cases. A two-tails t-test (assuming different variances) results in a null hypothesis probability of ($p < 3.028 \times 10^{12}$). In the first case, the difference is 6.65cm (19%). In the second case, 26cm (74.6%).

We believe that Shayke's lagging behind was the result of its reliance in the second case on a target, which itself must rely on a target. Compared to the first case (where the left robot monitored the leader directly), there would likely be some latency in Shayke's responses to the movements of the leader, since they were moderated by Poli. Our choice of Dijkstra's algorithm as the basis for the formation generation is motivated by such latency. However, here, we forced

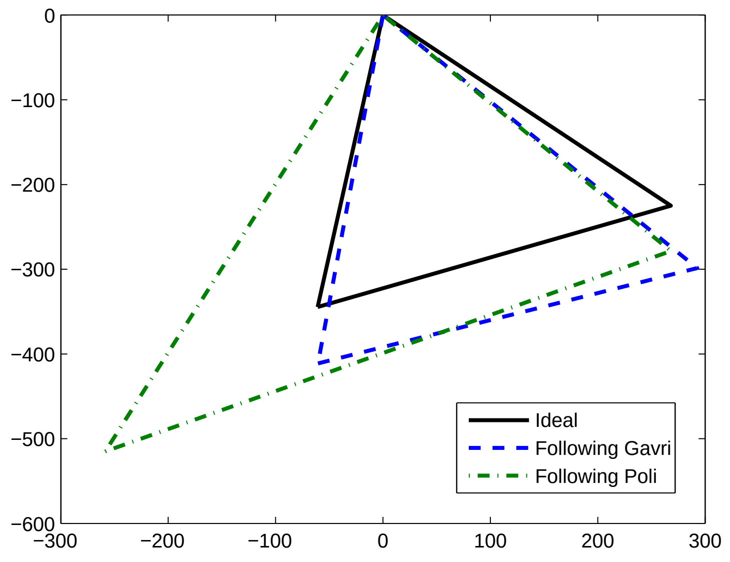


(a) Triangle.

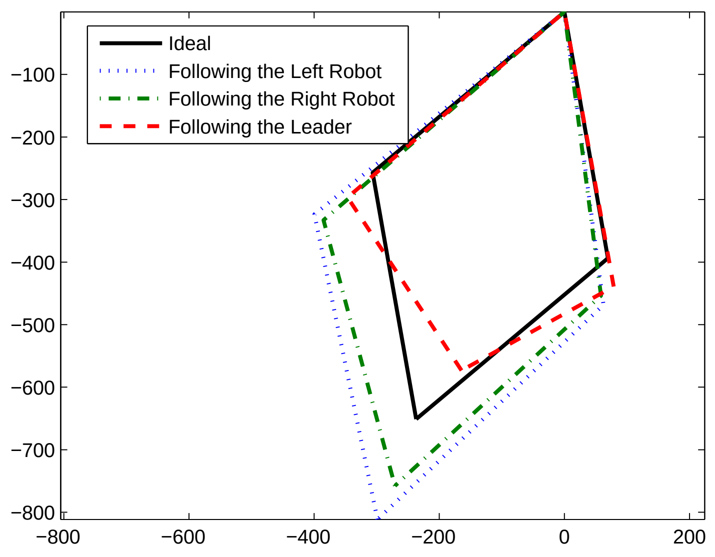


(b) Diamond.

Figure 5.5: Position Errors.



(a) Triangle.



(b) Diamond.

Figure 5.6: Ideal and actual robot positions.

the robot (in the second case) to look only sideways, and thus it had to find a longer route than necessary. One conclusion is that monitoring latency can affect a multi-robot system—in this case, a team moving in formation—even in small-scale, prototype settings such as those described in the experiment. For instance, the leader could have moved more slowly or communicated its movements, to reduce the effects of latency.

However, in the diamond formation, formations in which Shayke did *not* follow the leader directly, performed better. We believe that this is due to the effective sensor range of the robot, which causes unreliability in color detection. In the AIBO robot the vision's sensor we used is combined from both a color detection camera and a distance measurement device. Two parameters have to be taken under consideration when evaluating the costs over distances. The first is the distance measurement and the second is the color detection and identification. The specification we defined related only to the distance detection, which is equal for the entire $200_{mm} - 1500_{mm}$ range. Blob detection quality was not included in our specifications since we found that the detection of color blobs becomes unreliable with distance. This made Shayke lose the leader intermittently, causing it to move closer to the leader.

Another parameter associated with task performance in formation maintenance is the number of times robots lose their targets. Here again we compared this factor only relating to Shayke. We distinguished between two type of losses. The first, *little losses,* refers to a threshold for the number of consecutive video frames whose loss is considered normal. In our case, this threshold was set to 4, thus up to 4 consecutive frames in which the target is not tracked are considered normal, and are not declared a failure. The second type of loss (called *big losses*) refers to a larger threshold (11 in our case) over consecutive frames. When this threshold is passed, the robot activates the dynamic recovery algorithm.

Experiencing a large number of little losses or big losses indicates low performance. The recorded losses in these experiments, for the two formations, are presented in Figure 5.7. Results regarding the triangle formation (Figure 5.7-a) support our previous conclusions that following the leader yields better results than following Poli. In the diamond formation (Figure 5.7-b) the situation is different. According to these results Shayke should follow the robot located at the smallest angle (first Ubu with 10°, than the leader with 20° and finally Poli with a 50° angle).

Even if these results seem to contradict our conclusions from the average robots' positions (Figure 5.6), this is not the case. The formation was better kept when Shayke monitored the right robot than when it monitored the leader. But because of the wide angle between Shayke and Poli a little deviation from the desired place caused Shayke to lose Poli. When following the leader, since the angle needed was only 10°, even when the formation was broken, Shayke was able to continue sensing the leader. The conclusion is that in unpredictable environments, e.g. when unexpected sharp turns might happen, it is very important to select a target robot whose likelihood of loss is small, even at the expense of maintaining the correct position. Conversely, where recovery from losses is simple, monitoring robot leading to the formation being better maintained is being preferred.

5.2 Dynamically Switching Control-Graphs

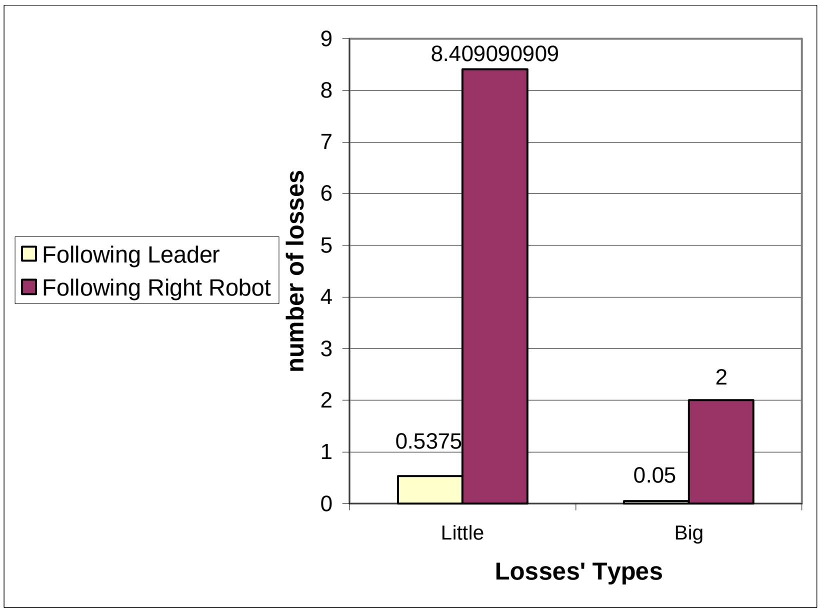
The principal lesson from the first set of experiments is that static formation control graphs can lead to markedly different results, depending on a number of factors. Thus, we wanted to evaluate the ability of dynamically-switching control graphs to compensate for such limitations, and yield better and more robust formations.

To experiment with this, we re-executed the diamond formation experiments, contrasting a static control graph with that of the dynamically-switching control graph, using the switching protocol described above. In both cases, the robots began with the same control graph. In the dynamic cases, they were allowed to switch to a different target while in the static cases the same control graph was generated at all times. To control and trigger such switches, we varied the *big loss* threshold determining whether a robot believed its target to be lost. A smaller number indicates that the robot is very quick to consider its target to be lost, and thus is a good simulation under noisy sensing conditions. A large number indicates that the robot is willing to wait a relatively long time before declaring it has lost its target [1]. We used values of 4, 20, and 40 to simulate noisy conditions. Each configuration was repeated 15 times, for a total of 90 trials. Unlike the previous set of experiments, the velocity of the leader was fixed, and thus less time for finishing the course indicates improved performance.

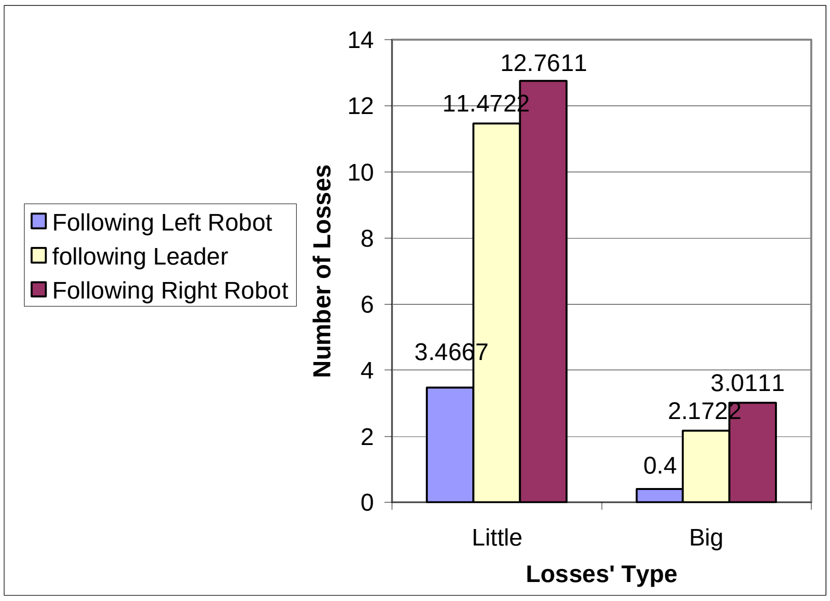
We focus on Shayke since it is the only robot whose target changed dynamically. At the beginning we compared the deviation of the actual distance and angles to the leader (Gavri) to the desired (distance of 69.28cm, angle of $20°$). The results are presented in Figures 5.8 and 5.9, respectively. In both figures the Y-axis represents the error.

Although our algorithm showed no improvement in the distance measurement, we believe that the formation was maintained better using the dynamic switching. The reason is that while deviation in distances shows no unequivocal improvement, using the dynamic switching shows great improvement in angle maintenance. Table 5.2 shows the results of two-tailed t-tests (assuming unequal variance) comparing the average deviations of distances and angles in the static and

[1]The numbers actually denote the number of consecutive frames in which the target was not identified



(a) Triangle.



(b) Diamond.

Figure 5.7: Losses of Target.

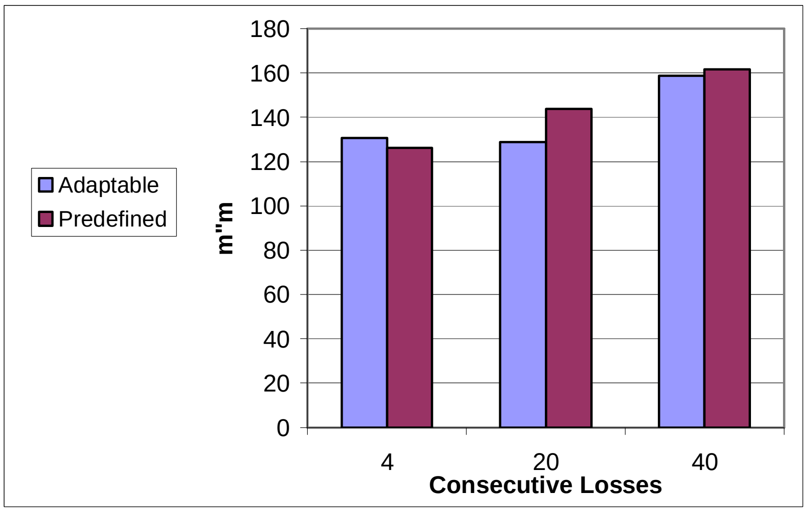


Figure 5.8: Distance Errors.

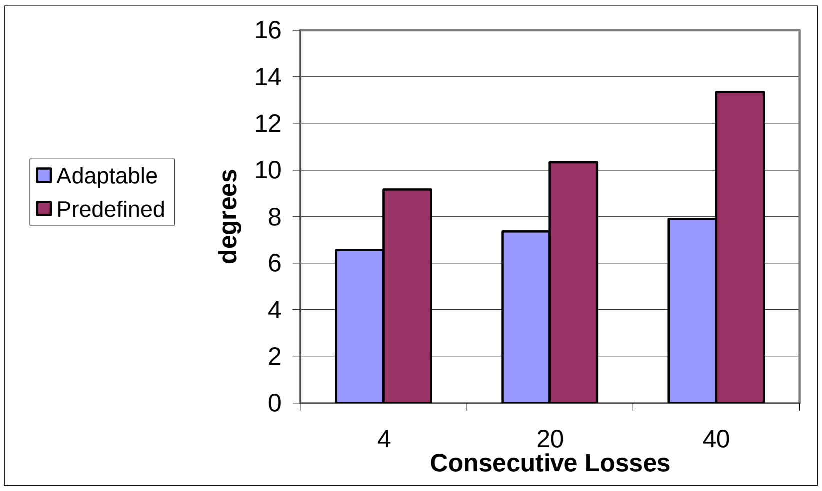


Figure 5.9: Angle Errors.

dynamic cases.

	Distances	Angles
Consecutive losses = 4	0.670389076	0.008502755
Consecutive losses = 20	0.038246941	0.000466375
Consecutive losses = 40	0.869657866	0.000013119
Total	0.736290326	0.000000015

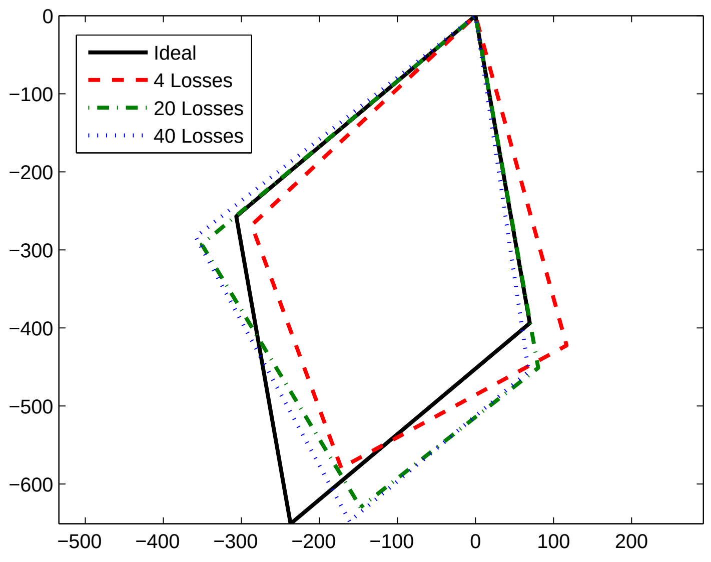
Table 5.2: Likelihood of null hypothesis, using a two-tailed t-test comparing the average deviation from the desired distance / angle of Shayke to the leader.

The explanation detailed above can be visualized in Figure 5.10. The figure draws the average positions of robots in the diamond formation, in the case of static and dynamic control graphs, for each value of the threshold. The positions are calculated by placing the leader at the (0, 0) point and using the detected angles and distances of the robots to their leader to find there relative place. As can be seen in the figures, the dynamic control graph yields results that are (i) more consistent across the experimental conditions; and (ii) closer to the ideal form of formation.

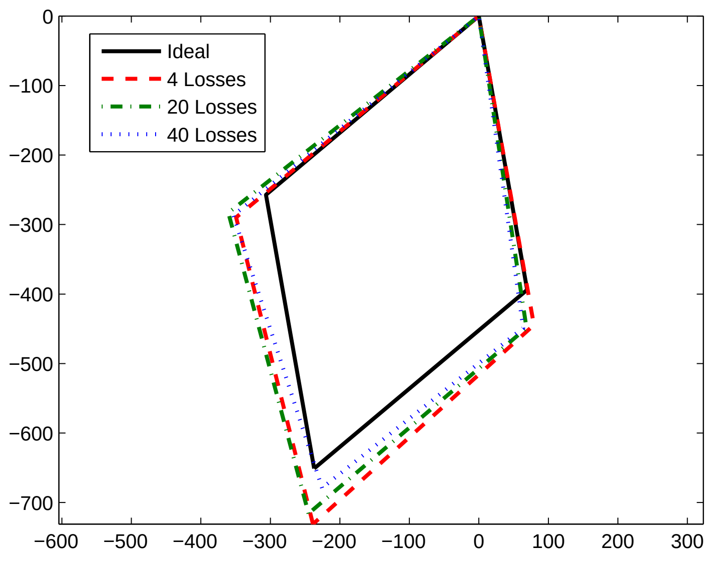
Figure 5.11 provides quantitative analysis of the same phenomenon. Here, the Y-axis represents the error in placement of Shayke in the fixed and dynamic control graph techniques. This time the values take both distances and angles under consideration and provide better evaluations of the formation maintenance. As can be seen in the figure, the dynamic switching technique leads to significantly smaller errors under all three conditions.

We examined additional performance measures. Figure 5.12 shows the time that it took the formation to finish the course, i.e., the smaller the value the better. The X-axis shows the different threshold settings, simulating different perception errors. The Y-axis represents the time. The figure shows that the dynamically-switching technique leads to significantly smaller durations needed to finish the task. Repeated two-tailed t-tests (assuming unequal variance) confirm a statistically significant difference between dynamic and static techniques, under all conditions, as depicted in Table 5.3.

In addition we examined the percentage of time needed to fulfill the task. First we checked the percentage of time the whole group remained connected, i.e., all the team members moved as a group, derived from the leader's movement. In Figure 5.13 the Y-axis displays the percentage of time the group remained connected, thus the larger the value the better. The figure shows two advantages of the dynamic-switching approach. First, the percentage of time the team remained connected was significantly higher than with the static control graph



(a) Fixed.



(b) Dynamic Switching.

Figure 5.10: Average Formation

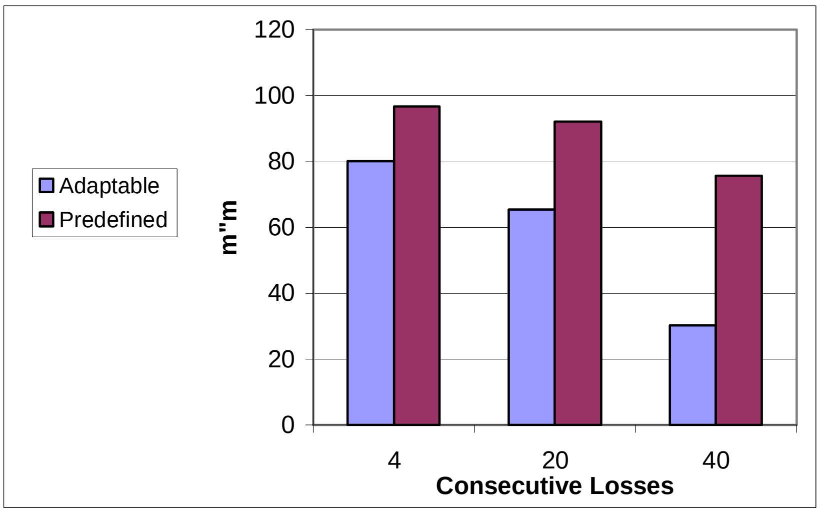


Figure 5.11: Position Errors.

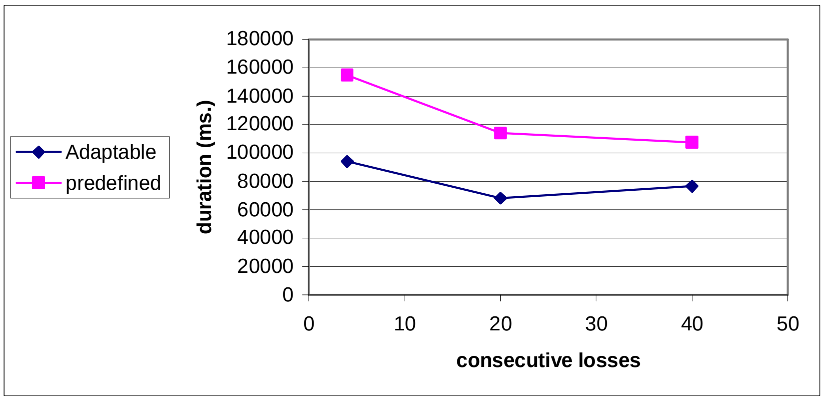


Figure 5.12: Time to complete course.

Case	*p-value*
Consecutive losses = 4	0.0000092176
Consecutive losses = 20	0.0000001005
Consecutive losses = 40	0.0000265963
Total	0.0000000000005

Table 5.3: Likelihood of null hypothesis, t-test for duration needed to fulfill task.

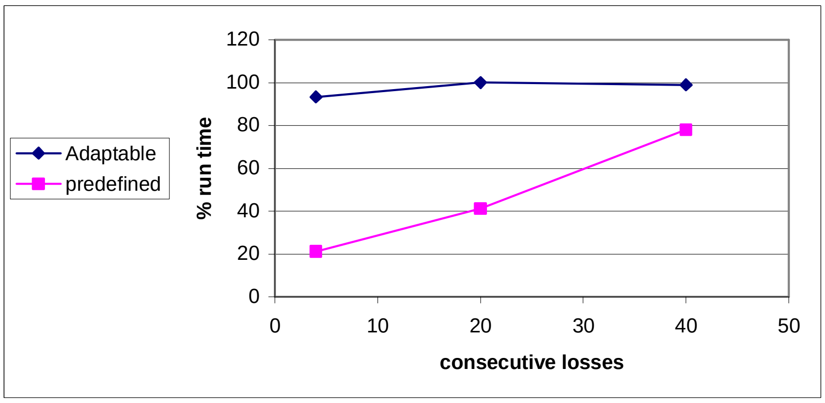


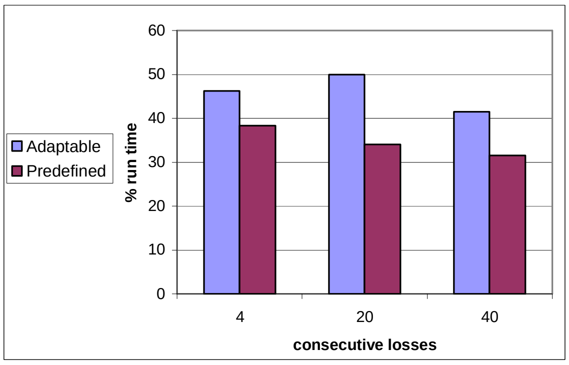
Figure 5.13: Percentage of time the group remained connected.

(Two tails t-test assuming an unknown variance shows statistical significance of $p < 2.40109 \times 10^{-15}$) . Second, and perhaps more importantly, the percentage essentially remains constant despite the significant change in the environmental conditions. This is in contrast to the static control graph approach, whose performance was lower, and also inconsistent across the controlled conditions.

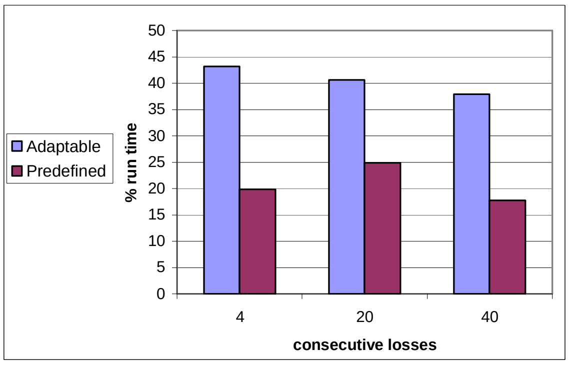
A second benefit was revealed when we examined the percentage of time the formation was maintained. i.e., angles and distances were within their tolerance levels. We examined the percentage of time the formation was maintained by defining "formation maintenance" in three ways: (i) distances were within their tolerance levels (ii) distances of the angles were within their tolerance levels (iii) both the distances of the angles and the distances were within their tolerance levels. The results appear in Figures 5.14-a, 5.14-b and 5.14-c, respectively. We defined the tolerance for angles as 10 degrees and for distances as 15cm.

According to each of these three parameters, dynamic switching is superior to the static control graph. These results are statistically significant for almost all factors (distance only, angle only and the intersection of distance and angle) and values of consecutive losses before activating the switching procedure. p-values for two-tailed t-test assuming unequal variance are presented in table 5.4

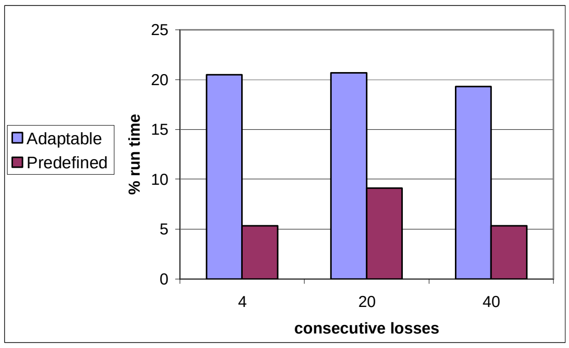
Finally we compared the number of calls to the dynamic switching procedure. As explained previously, in all sets of these experiments the dynamic switching procedure was activated when Shayke lost its target for a predefined number of sequential frames. This number was set at first to 4, than to 20 and finally to 40. In the static cases we used this procedure to generate the same control graph each time. The number of switches in the dynamic cases was statistically significantly lower than in the static cases. Table 5.5 provides the p-values resulting from two tailed t-tests (unequal variances) of the dynamic and static cases in each instance.



(a) Distances.



(b) Angles.



(c) Distances and Angles.

Figure 5.14: Percentage of time formation was maintained.

	Distances	Angles	Distances and Angles
Consecutive losses = 4	0.1293250	0.00000290	0.0000001
Consecutive losses = 20	0.0006321	0.0001473	0.0000085
Consecutive losses = 40	0.0440063	0.0000368	0.0000039
Total	0.0000613	0.0000000	0.0000000

Table 5.4: Likelihood of null hypothesis, t-test for percentage of time formation was maintained.

Case	*p-value*
Consecutive losses = 4	0.0000185102
Consecutive losses = 20	0.0000000990
Consecutive losses = 40	0.0000029815
Total	0.00000000000067

Table 5.5: Likelihood of null hypothesis, t-test for the number of switches in static Vs. dynamic cases.

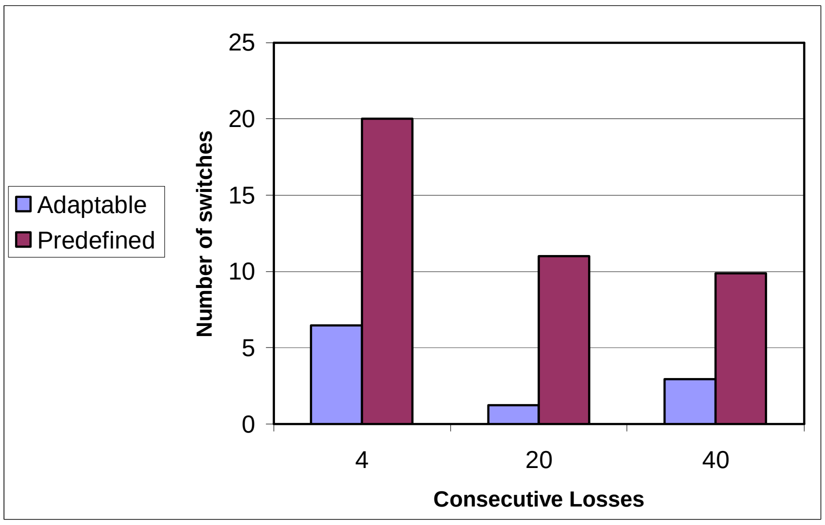


Figure 5.15: Target's switching.

Attribute	Range	Cost
Distance	$[0, 100]$	0
	$(100, 800]$	0.1
Field of View	$[-36°, 36°]$	0.1
Pan	$[-5°, 5°]$	0.1

Table 5.6: Type 2 Robot Sensor.

Attribute	Range	Cost
Distance	$[0, 310]$	0.4
	$(310, 675]$	0.7
Field of View	$[-30°, 30°]$	0.2
Pan	$[-90°, -5°)$	0.6
	$[-5°, 5°]$	0.4
	$(5°, 90°]$	0.6

Table 5.7: Type 3 Robot Sensor.

5.3 Multi-Graphs for Heterogeneous Teams

We defined four types of robots, with which we created a variety of formations. The sensor specification for each robot appears in Tables 5.6–5.8, which follow the same format as Table 3.1. Table 5.8 provides the data on multiple sensors. The details are shown for the first sensor, and the rest replicate its distance and field-of-view ranges, though at different panning values.

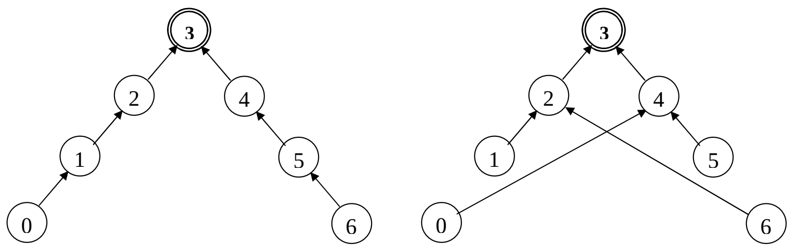
We experimented with different formations and different combinations of these robots, and produced monitoring multi-graphs and resulting formation graphs. All the graphs were produced automatically using the algorithms described above.

Figure 5.16 illustrates the formation graphs for triangular formations with seven robots, of type 3 (Fig. 5.16-a) and type 4 (5.16-b). In the latter, given the possibility of using a sensor at a pan angle unavailable to the first, it was cheaper for a trailing robot to directly monitor a robot farther from it, but only once removed from the leader), as opposed to monitoring a closer robot that is much more removed from the leader. This shows the importance of using individual monitoring costs as the optimal criterion, and thus the choice of the Dijkstra' algorithm.

Figure 5.17 displays the results for a square formation with eight robots. Here we show both homogeneous teams (Figures 5.17-a,b; all robots of the same type), and non-homogeneous teams (Fig. 5.17-c). In the latter, the robots in positions

Sensor	Attribute	Range	Cost
1	Distance	$[0, 400]$	0.1
		$(400, 900]$	0.2
	Field of View	$[-20°, 20°]$	0.3
	Pan	$[0°, 10°]$	0.1
2–8	Distance, F.o.V	Same as above	
	Pan°	$[-180, -170], [-82, -72]$ $[-36, -26], [36, 46], [72, 82]$ $[108, 118], [114, 124]$	0.1

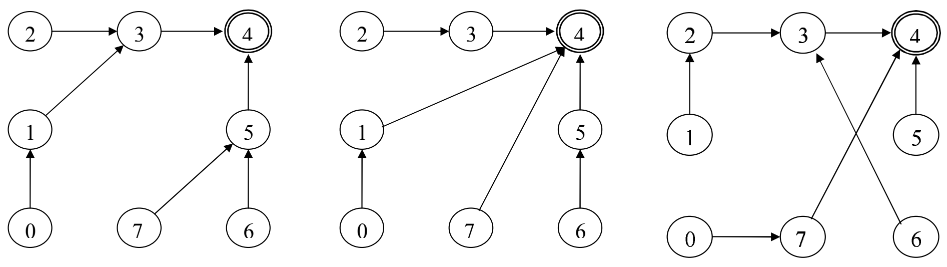
Table 5.8: Type 4 Robot Multiple Sensors.



(a) All type 3.

(b) All type 4.

Figure 5.16: Triangular, 7 Robots.



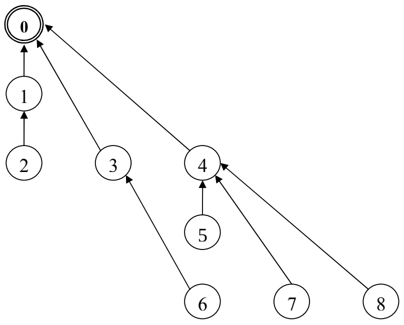
(a) All type 1. (b) All type 3. (c) Mixed types 1,2,3.

Figure 5.17: Square, 8 Robots.

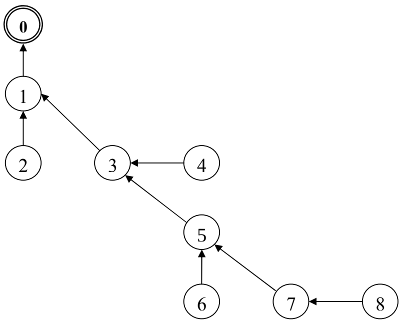
0,2,5 are of type 1, positions 1,4,6,7 contain type 2 robots, and position 3 holds a robot of type 3.

In a final set of experiments, we demonstrated the use of the complex stairs formation technique with 9 robots. Figures 5.18-a,b,c show homogeneous teams (type 4, 1, and 3, respectively). Figure **??**-d depicts the result of using a mixed team: Positions 1,4 have robots of type 1; positions 0,2,5,7 hold robots of type 2; positions 3,6 contain robots of type 3, and finally a robot of type 4 is in position 8.

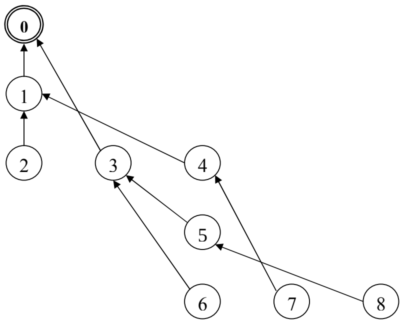
These results demonstrate that the technique optimizes for individual monitoring costs, and thus considers the sensor morphologies of the robots involved. Not all robots are created equal (at least in terms of sensors), and thus must be treated differently when coordinating between them. The results show that use of this automated technique facilitates the creation of monitoring rules for heterogeneous teams, where different members of the teams have different sensor morphologies.



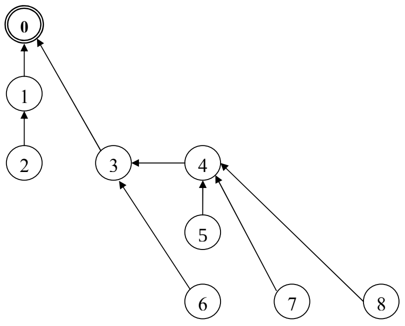
(a) All type 4.



(b) All type 1.



(c) All type 3.



(d) Mixed all types.

Figure 5.18: Stairs, 9 Robots.

Chapter 6

Conclusions and future work

We presented a novel representation for reasoning about formation control graphs, based on directed weighted monitoring multi-graphs. We have shown that the approach allows the use of graph-theoretic techniques, to address key open problems. In particular, we have provided novel techniques that (i) optimally account for sensor morphology and constraints in generating distributed formation-maintenance SBC controllers; (ii) allow sensor-heterogeneous teams; and (iii) allow robots to dynamically switch formation control graphs for added robustness. We have demonstrated the use of the technique in systematic experiments with physical robots, and have shown that the use of our techniques leads to significant improvement in both performance and robustness to environmental conditions.

Although a large body of literature exists on formation-maintenance, the area of constructing a formation controller that optimizes the control graph based on desired properties of robots has scarcely been discussed. In our work we initially describe a robust solution to this problem but much is left for future work.

First we should consider additional attributes relevant to determining the cost of an edge, e.g. the ability to detect specific color. Our experiments together with the manufacture's definitions show that in the AIBO's the quality of detecting blob depends on its color. In our experience, if the ability of robot to identify its target is based on color detection, then all incoming edges to a vertex with specific colors will involve higher costs, since the difficulty may lie in the target's color, not in their individual sensors.

Also the movement capabilities may affect target assignment. The control graph constructor has to suit the monitoring rule's to the group's movement direction and the possible walking directions of each robot. For example, if a robot cannot walk backwards it will not be able to follow a robot behind it even if it can monitor it. The first time this target will move with the predefined group movements direction, it will deviate from the desired position (which has changed), and may even totally lose its target.

The relative importance of the cost factors (attributes, e.g., distance, field of view and pan), has to be estimated and evaluated. In our current work we used $\forall_i w_i = 1$ in the cost function

$$weight(e) = \sum_i w_i \cdot cost(i)$$

. Reexamination may reveal that one parameter has less influence on the monitoring performance than the other. In this case, the appropriate weight w_i should receive a lower value than the others. In addition, more complicated cost functions can be considered. We use the above linear function were other function may prove better in reality.

Another key question is how to recognize which failure caused the robot to lose its target. Sometimes inability to monitor one robot points at difficulties monitoring other robots. Identification of the reason may help prevent more losses in the future by avoiding the assignment of inappropriate targets. For example, in the case where the robot sensors are stuck in a specific angle may completely obstruct a specific target from being monitored. Not only should this target not be chosen again but all robots placed along the current-impossible angles should not be taken into consideration. Another example is the case where the camera of a robot has lost the ability to recognize a specific color. In case all robots have signed in, that color should not be chosen as a target of that robot.

When the cause of the failure is known the weight given in the impossible monitoring table can be selected in a more accurate way and might be updated not only for the robot that experiences the failure and its current target. Weight should also be given to the history of the target reassignment process.

Evaluation of the other weights used in the algorithm is also an important point to be discussed in future work. All the algorithms are based on predefined cost definitions for each sensor and range. These costs are used to evaluate weights of edges in the monitoring multi graph and to select the best control graph. The more accurate the costs reflect reality, the better the selecting of a control graph and the less the unexpected failures (which are not the result of external factors).

Finally we have to discuses the allocation of the robots according to their properties. Our work defines a heterogeneous team of robots where each robot is accessorized with its own sensor types and configuration. We assume the place of each robot is predefined and the only question is which robot will be assigned to the best target to follow. In future work we will discuss how to best place each robot in the formation. The place of a robot in the formation defines its distance and angles for each of the other members. It facilitates determining the weight of the edge in the respective control graph. In the case of a heterogeneous team, each arrangement may lead to a different optimal control graph. The algorithm will search for the one that generates the best control graph.

A significant part of the allocation issue is the selection of the robot best fit to function as the leader. In addition to the monitoring the weights of the multi graph's edges, the ability of a robot to lead a group also has to be considered when selecting the best possible control graph.

Bibliography

[1] N. Agmon, G. A. Kaminka, and S. Kraus. Team member-reallocation via tree pruning. *In Proceedings of the National Conference on Artificial Intelligence (AAAI-2005)*, 2005.

[2] T. Balch and R. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, 1998.

[3] T. Balch and R. C. Arkin. Behavior-based formation control for multi-robot teams. *IEEE Transactions on Robotics and Automation*, 12 1998.

[4] T. Balch and M. Hybinette. Social potentials for scalable multirobot formations. In *Proceedings of the IEEE Conference on robotics and automation (ICRA-2000)*, San Francisco, 2000.

[5] S. Carpin and L. Parker. Cooperative leader following in a distributed multi-robot system. *IEEE International Conference on Robotics and Automation*, 2002.

[6] S. Y. Chiem and E. Cervera. Vision-based robot formations with bézier trajectories. *Intelligent Autonomous Systems 8*, IOS Press:191–198, 2004.

[7] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[8] J. P. Desai. Modeling multiple teams of mobile robots: A graph theoretic approach. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1:381–386, 2001.

[9] J. P. Desai. A graph theoretic approach for modeling mobile robot team formations. *Journal of Robotic Systems*, 19(11):511–525, 2002.

[10] J. P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. *IEEE International Conference on Robotics and Automation*, pages 2864–2869, 1998.

[11] J. P. Desai, J. P. Ostrowski, and V. Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 12 2001.

[12] J. P. Desai, J. P. Ostrowski, and V. kumar. Modelling and control of formation of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, December 2001.

[13] D. Dudenhoeffer and M. Jones. A formation behavior for large scale micro-robot force deployment. *2000 Winter Simulation Conference*, (WSC '00):972–982, Dec 2000.

[14] Y. Elmaliach. Single operator control of coordinated robot teams. Master's thesis, Bar Ilan University, 2004.

[15] R. Fierro, A. K. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. *IEEE International Conference on Robotics and Automation*, 2001.

[16] J. Fredslund and M. J. Mataric. A general algorithm for robot formations using local sensing and minimal communications. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 10 2002.

[17] G. Inalhan, F. Busse, and J. How. Precise formation flying control of multiple spacecraft using carrier-phase differential gps. In *Proceedings of AAS/AIAA Space Flight Mechanics*, 1 2000.

[18] G. A. Kaminka and M. Bowling. Robust teams with many agents. In *AAMAS 2002*, pages 729–736.

[19] M. Lemay, F. Michaud, D. Létourneau, and J.-M. Valin. Autonomous initialization of robot formations. *IEEE International Conference on Robotics and Automation*, pages 3018–3023, 2004.

[20] F. Michaud, D. Létourneau, M. Gilbert, and J.-M. Valin. Dynamic robot formations using directional visual perception. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[21] D. J. Naffin and G. S. Sukhatme. Negotiated formations. *International Conference on Intelligent Autonomous Systems*, pages 181–190, March 2004.

[22] L. E. Parker. Designing control laws for cooperative agent teams. *IEEE Robotics and Automation Conference*, pages 582–587, May 1993.

[23] H. Tanner, G. Pappas, and V. Kumar. Leader-to-formation stability. *IEEE International Conference on Robotics and Automation*, 2003.