Bar-Ilan University
Department of Computer Science

Optimal Construction of Control Graphs in Multi-Robot Systems

by

Ilan Lupu

Submitted to the Senate of Bar-Ilan University

Ramat-Gan, Israel
October 2015

Contents

Abstract

Control graphs are used in multi-robot systems to maintain information about which robot senses another robot, and at what position. On the basis of such graphs, it is possible to compute a shared coordinate system, localize relative to others, and maintain stable formations. While existing work shows how to utilize control graphs for these tasks, it makes two critical assumptions. First, it assumes edge weights of control graphs are single deterministic scalars. However in reality there are many stochastic factors (e.g, latency, resource costs, or position errors), that affect optimality of control graphs. Second, it assumes that a single robot is given, to serve as the global anchor for the robots' relative positioning and location estimates. However, optimal selection of this robot is an open problem. In this work, we generalize control graphs to distinguish different stochastic sensing factors that may be represented by control graphs, beyond existing work, and discuss risk-based policies for their treatment. We show that existing work in coordinate frame alignment and formation maintenance may be recast as graph-theoretic algorithms inducing control graphs for more general representation of the sensing capabilities of robots. We then formulate the problem of optimal selection of an anchor, and present a centralized algorithm for solving it. We evaluate use of these algorithm on physical and simulated robots equipped with depth and image sensors (RGB-D cameras), and show they very significantly improve on existing work.

1 Introduction

Control graphs are used in multi-robot systems to maintain information about which robot senses another robot, and at what position. In such control graphs, nodes represent robots in given positions. Weighted edges represent sensing capabilities; an edge from node A to node B, with weight w, represents the fact that robot A can sense robot B, with preference w (typically, smaller weight indicates stronger preference). On the basis of such graphs, it is possible to build a shared coordinate system (e.g., [24]), compute message passing paths in ad-hoc networks, and maintain stable formations (e.g., [8, 10, 17]).

Existing work utilizing control graphs raises several open challenges. First, it offers no systematic treatment of the edge weights, how they are determined, and how they should be utilized in the computation of optimal control graphs. Different tasks (e.g., building a coordinate system versus formation maintenance) utilizes the edge weights differently. Second, it makes the assumption that a single robot is given, chosen to serve as the *global anchor* of the shared coordinate system, *leader* of the formation, or origin of a message whose position is taken as the basis for the robots' relative positioning and location estimates. Third, it ignores uncertainty in the weights of edges, such that, for instance, if the edge weight denotes a distance, it assumes the distance is known with certainty, despite the inherent uncertainty that exists in real-world sensing. In this work, we tackle these open challenges.

First, we synthesize from existing work, and then generalize the notion of control graphs and their uses. We begin by refining the definition of *monitoring multi-graphs* [17], which distinguish between different sensing configurations of robots. We show how existing techniques (e.g., for computing shared coordinate systems) can be optimized by re-casting them in terms of graph-theoretic algorithms for inducing directed trees from the multi-graphs, such that the trees optimize for a given criteria (e.g., team costs, individual position error). Each such tree is an *optimal control graph* for a given task (e.g., message passing, formation maintenance).

Second, on the basis of this more general understanding of how control graphs are generated from monitoring multi-graphs, we formulate the problem of optimal selection of *leader* or *global anchor* in a given monitoring multi-graph. A leader robot serves as the root of the control graph (tree) generated from it. We present a centralized algorithm that efficiently determines the optimal leader for a given task, as well as the resulting control

graph.

We evaluate use of the novel algorithms on physical and simulated robots equipped with depth and image sensors (RGB-D cameras), and contrast them with results obtained from existing work. The results show very significant improvements from using these algorithms for coordinate frame alignment, in both simulated and real robots, in static and dynamic settings.

2 Related Work

The use of the mathematical notion of a graph for reasoning about roles of robots in cooperative multi-robot tasks has a long history. We survey below only the most related, recent work.

Formation maintenance. In the problem of formation maintenance, a team of robots should move while maintaining a shape, dictated by their relative positions (e.g., column, diamond, arrow). Some of the literature focuses on aspects of stability, motion efficiency and feasibility for different types of robots, all of which we do not address here [2, 3, 5, 25, 26, 28].

Balch et al. [4] developed behaviors for formation maintenance in multi robot systems. The formation behaviors were implemented as a reactive navigational strategy, where every robot had some high level behavioral intentions and the sum of all responses that served every intention created the output for execution by each robot. Balch et al. introduced the behavior *maintain-formation* which generates a movement vector towards the correct robot location in the formation. Three techniques for positioning in the formation were introduced: Unit center reference, leader reference and neighbor reference. In this work we focus on the robots positioning and how to choose the correct leader that optimize the robots locations. We do not interfere with the robots movement or try to improve the formation stability.

Desai et al. [8–10] defined a *control graph* as an unweighted directed graph (digraph) whose vertices are the robots in the formation. An edge from A to B represents that robot A monitors robot B's position. They utilize this graph to discuss formation maintenance tasks, in which the shape of the formation (and thus the graph) changes to accommodate the terrain and obstacles. Moreover, they show that a formation can be stably maintained if the control graph implies each robot (except a single *leader*) maintains its bearing (angle) and separation (distance) with respect to one other robot (*target*). This type of formation control is known as SBC (Separation-Bearing Control). Without referring to control graphs, Fredslund and Matarić [13] propose a distributed algorithm for generating SBC monitoring rules (i.e., which robot monitors whom) given a target placement of the robots and the leader, by their identification (IDs). This restriction on placement, and additional constraints on the formations, maintain connectivity of the underlying graph. In contrast, we consider weighted edges in our control graphs, and show how to optimally induce (select) control graphs for different tasks (not just formations). We also address the question of leader selection. However,

our algorithms here are centralized, to avoid these restrictions.

Kaminka et al. [17] generalized on these works. They defined a monitoring multi-graph, which compactly represents all possible SBC control graphs for a given placement of robots. Each edge represents a possible configuration of the follower robot by which it can sense a target robot (local leader). Graph edges are weighted, where the weights indicate costs of sensor usage in the given configuration. They present a centralized algorithm for inducing a specific control graph, which optimizes the selection of targets, given a chosen leader. In particular, their algorithm optimizes individual sensor usage costs, which accumulate over edges. We show that their representation and algorithm is a special case of a broader definition of monitoring multi-graphs, and of a family of useful algorithms which can optimize for criteria other than individual accumulating costs. Moreover, we address the question of leader selection, which they leave open.

Lemay et al. [18] and Michaud et al. [23], present a distributed method of assigning robots to formation positions. The computation relies on a cost function that considers distances and angles to the teammates; it outputs the lowest-cost assignment of robots to positions. Additionally, a leader robot is determined, which minimizes costs over all possible assignments. In contrast, we begin with robots already assigned, and only then select a leader and SBC targets. Each robot in a formation determines a cost for assigning its teammates to positions in the given formation, assuming it is the leader (which they refer to as *conductor*). Then the best (minimal cost) assignment of roles to robots (including the leader) is made. Our approach complements this work. We use computation of sensing costs *after* robots have already been assigned to their positions, to determine the sensor configuration used by each robot to monitor its target. Furthermore, the technique we present allows dynamic switching of control graphs within the same formation, where the work by Lemay et al. and Michaud et al. do not allows switching of the formation shapes.

Shared Coordinate Systems (Coordinate Frame Alignment). Another common task is that of multiple robots agreeing on a common coordinate system (axes and origin), e.g., as the basis for relative localization to the surrounding environment and multi-robot mapping [1, 11, 14]. There are several studies regarding the construction and alignment of coordinate systems (e.g., [15, 21, 27, 30]). Briefly, the task here is for robots to identify their alignment (translation and rotation) with respect to each other (typically one of the robots serves as a global anchor). As not all robots can

sense the global anchors, they may instead localize via *anchor chains*, i.e., localize with respect to local anchors, who sense other anchors, etc. This is also referred to as coordinate frame alignment.

Most such work focuses on the filtering mechanisms able to cope with the uncertainty inherent to this process, and with various types of errors (e.g., receiving only range information).

Piovan et al. [27], introduced the frame localization problem in a connected network and provided an algorithm for positioning the nodes based on a bearing sensor (distance and angle readings). Some assumptions were made on the network, such as that it is bi-directional in the sense that if node i can sense node j then node j can sense node i and that the network is static. The algorithm aligned the network frames and established position for all the nodes. In our work we remove the bidirectional and static assumptions and focus on choosing the optimal node to align with. We use the same technique for aligning the frame after the optimization has been finished.

Recently, Nagavalli et al. [24] presented a distributed method for improving the accuracy of such alignments, by utilizing a breadth-first search (BFS) to minimize the number of anchors in anchor chains, all beginning with a selected global anchor. In this work we present a centralized algorithm for selecting an optimal global anchor in this task, and show that this further improves (significantly) the position estimates of the robots. Moreover, this works with anchors that are not part of the team, such as objects in the team surroundings that the robots can identify.

3 Optimal Construction of Control Graphs

We begin with robots placed in fixed relative positions, and no leader assigned. In Section 3.1 we show how to compactly represent all the different possibilities for robots to sense each other in their positions, using a refined definition of *monitoring multi-graphs*, originally presented in [17]. Then, in Section 3.2 we show how existing work can be re-cast in terms of graph-theoretical algorithms, properly extended to run on monitoring multi-graphs. Existing work leaves open the question of optimal leader selection, which we address in Section 3.4.

3.1 Monitoring Multi-Graphs

A monitoring multigraph captures all the potential control graphs for a group of robots in fixed positions. As defined in [17], it is a directed, weighted multigraph $G = \langle V, E \rangle$, where V is a set of vertices representing robots, and E is *a bag* (multi-set) of weighted edges between vertices.

Each $v_i \in V$ represents a unique robot i, identified by its index, and having a specific pose in space. The function $pos : V \mapsto \Re^n$ identifies the unique pose of each robot $v \in V$ (typically, $n = 3$, with the pose determined by the position and orientation of the robot v).

Let $v_i, v_j \in V$ be two robots. Suppose v_i can use a specific configuration of its sensors to sense v_j, i.e., v_i computes an estimate of $pos(v_j)$, denoted by $\hat{pos}(v_j)$. Denote the specific configuration by x. For instance, it may refer to a specific pan of a camera or Lidar, combined with a specific sensor processing algorithms (e.g., visual marking recognition, depth perception), a specific choice of resolution or focus, etc. [17] propose using scalar costs to indicate the robot manufactures' preferences for the use of the sensor in this configuration, based on distance, field of view, and pan range. Sensor reliability based on these factors, i.e., the quality of $\hat{pos}(v_j)$, is supposed to be the basis for setting this preference. The scalar cost values are combined, using a weighted-sum function, into a single scalar value c_{ij}^x to be used as the edge weight.

We depart from this definition as presented in [17] in two ways. First, we distinguish between *directly measurable resource costs* (such as expenditure of power, computation time, or sensor processing latency), and errors in the estimate $\hat{pos}(v_j)$, which are given in terms of deviations from the ground truth. Second, we accept that realistically, costs and errors can only be

estimated with uncertainty. Thus we model them—and their translation into edge weights—as random variables, with a known probability distribution function.

More precisely, with each measurable cost factor k in the operation of the sensor, and each component of error m resulting from it in $\hat{pos}(v_j)$, we associate a known probability distribution $C_{ij}^{x,k}$ ($R_{ij}^{x,m}$, respectively), explicitly or parametrically represented. For instance, if the perception latency l is known to be uniformly distributed in the range 20ms–30ms, this may be explicitly represented by setting $C_{ij}^{x,l} \equiv \mathcal{U}(20, 30)$. If the distance from v_i to v_j is d, measured by a Lidar with a 3%, we may set $R_{ij}^{x,d} \equiv \mathcal{U}(-0.015d, +0.015d)$. As v_i only approximates the true position of v_j with $\hat{pos}(v_j))$, we use an approximate d and update it as additional measurements are made. The overall costs associated with the edge e_{ij} are then drawn from the joint distribution of all $C_{ij}^{x,k}$, denote $\overline{C_{ij}^{x}}$. Likewise, we denote the errors by $\overline{R_{ij}^{x}}$.

Given these definitions, we define the edges in E as follows. An edge $e_{ij}^{x} \in E$ is a tuple $e_{ij}^{x} = \langle v_i, v_j, \overline{C_{ij}^{x}}, \overline{R_{ij}^{x}} \rangle$. When clear from the context, we omit the superscript ^{x}. This definition departs from [17] in that we add the representation of errors, and distinguish multiple components in costs and errors. We also depart from [17] in that we assume that the sensing robot can identify the sensed robot id and contrast the graph with the existing edges without assuming all possible edges can exist and eliminating edges that are occluded by other robots. Alternative configurations may result in improved costs or lower errors; often a robot may trade these off, e.g., by spending more computation time or more energy to improve its position estimate of the other robot. Given $|X|$ configurations for robot v_i to monitor v_j, the exist edges $e_{ij}^{1}, e_{ij}^{2}, \ldots, e_{ij}^{|X|} \in E$.

3.2 Inducing Control Graphs (for a Given Robot)

Monitoring multigraphs compactly represent all potential ways in which robots could monitor each other in their positions. Given a task which requires robots to monitor each other's positions (e.g., formation maintenance), we want to induce a *control graph*: a subset of the monitoring graph, which specifies for each robot which sensor configuration to use, and what other robot(s) to monitor, in order to improve task performance.

Many previous methods for tasks such as formation maintenance and distributed localization can be recast as a process of such control graph induc-

tion, if we ignore the distributions, and instead assume a single deterministic scalar value is associated with each edge. To see this, we distinguish between tasks that accumulate edge costs (or errors) and tasks that do not. The distinction is between tasks, not between types of costs or errors.

For example, consider the use of errors. In *relative* multi-robot localization [15, 24], robots build ego-centric coordinate systems where their relative positions are known; each robot can determine its position with respect to others (anchors), who determine their position with respect to other anchors, etc. A single robot can be selected as an agreed upon origin for a *shared* coordinate system. Position estimates with respect to this origin accumulate errors with every such hop from one anchor to the next. Thus minimizing the length of these anchor chains, as a heuristic, reduces these accumulating errors. On the other hand, in many *flocking* algorithms, robots need to maintain their distances to others within certain bounds, so as to form amorphous clusters (in contrast to specific positions, in formation maintenance tasks). Here, position errors do not accumulate over links, as chains of robots that use each other to anchors do not relate to a single origin point or target positions.

Likewise, for costs. In formation maintenance, the latency in identifying motion in observed robots results in the familiar "traffic-light effect", where there is a noticeable, accumulating delay between the time the leader of the formation moves, and the time the last robot moves. Longer anchor chains (e.g., in convoys) aggravate this effect. On the other hand, when costs measure power usage when monitoring other robots, costs are not accumulated. If the task is to maintain visibility, the power spent by a robot to maintain its target robot in sight is not an accumulation of the power spent by the target robot itself. These examples are summarized in Table 1.

	Costs	Errors
Accumulating Factor Task	Motion detection latency Formation-maintenance e.g., [13, 17]	Position error Relative Localization e.g., [24]
Non-accumlating Factor Task	Power usage Maintain in sight e.g., [32]	Position error Aggregate e.g., [31]

Table 1: Example tasks relying on accumulating and non-accumulating costs or errors.

Indeed, we can now begin to recast existing work in terms of the algorithms used to induce a control graph from a monitoring multigraph, whether explicitly represented or not. We focus here on accumulating factors (Table 2 summarizes).

In the column marked "No leader selection, no optimal control graph" we list previous works which utilize heuristic algorithms for deciding, for each robot, which robot it should monitor, i.e., heuristic methods for constructing control graphs, which are not guaranteed to be optimal (in the sense of reducing accumulating errors or costs). In the next column, marked "no leader selection, optimal control graphs", we list investigations which, for a given leader, generate an optimal control graphs minimizing accumulating errors or costs (assuming scalar edge weights).

	No leader selection, No optimal control graph	No leader selection, Optimal control graph	Leader selection, Optimal control graph	Leader selection, Uncertainty
Algorithm type Algorithm type	Heuristic	Dijkstra's	All Pairs Shortest Path	All Pairs Shortest Path
Formation maintenance Relative Localization	[13] [15, 30]	[17] [24]	This Work	This Work

Table 2: Related work utilizing accumulating factors, re-cast by type of algorithm and problem settings. [17] uses costs to represent errors. [24] assumes uniform errors, allowing use of BFS instead of Dijkstra's algorithm.

In general, a variant of Dijkstra's single-source shortest path (S3P) algorithm [7] is optimal for all cases in the "no leader selection, optimal control graph" column, as long as edge weights are given in scalar terms. As described in [17], once a leader (source) is given, and by reversing the direction of all edges, Dijkstra's algorithm can in principle compute a control graph where the accumulating factors are minimized for a path from the leader to any robot. If all edge weights are the same, Dijkstra's algorithm reduces to a breadth-first search (BFS).

For example, for the task of formation maintenance, Fredslund and Matarić develop an algorithm for inducing formation-maintenance SBC rules using a heuristic distributed algorithm. The algorithm does not explicitly construct a monitoring multigraph from which it extracts a control graph. However, in effect it acts to build a control graph by considering observable target robots.

The leader is selected heuristically; the control graph is not guaranteed to minimize accumulated errors. To address this, Kaminka et al. present a method, based on Dijkstra's algorithm, which minimizes path length to a single target robot [17], using scalar edge weights. Thus once a leader is selected, an optimal control graph is generated for it.

Several challenges remain open: First, how to address the uncertainty inherent—and explicitly represented—in the stochastic edge weight composed of $\overline{C_{ij}^x}, \overline{R_{ij}^x}$. Second, the challenge remains of determining the optimal leader (i.e., one that whose associated control graph is superior to those of other leaders). We address these in the next two sections.

We note in passing that for non-accumulating factors, whether errors or costs, Dijkstra's algorithm will not yield an optimal control graph. Instead, the strategy here would be for every robot to select its own minimally-weighted edge, as long as every robot is reachable. This, in particular, is essentially equivalent to determining a minimum-spanning tree over a directed graph, e.g., using ChuLiu/Edmonds' algorithm [6, 12]. Here again, a leader robot must often be chosen, as the direction of edges must be considered in the construction of the tree.

For each robot $v_i \in V$ there is a set of edges $(v_i, v_j) \in E$ than robot v_i can use in order to measure its relative position. When talking about global costs, the strategy to obtain minimal graph will be for every robot to individually choose the minimal edge (v_i, v_j) and set v_j as his local anchor v_i^A. Two problems can occur, first, this strategy does not promise a connected graph so not all robots in the team will be aligned on the same coordination system. Second, it is possible that by choosing for robot v_i a sub optimal edge, the total cost for the team will decrease.

3.3 Inducing Control Graphs with Uncertainty: Managing Risk

Following [19], we refer to a multigraph with random-variable weights as a stochastic multigraph. Our task here is to determine the shortest path in the stochastic graph, from a given root vertex (the leader) to every other vertex (more accurately, in the reverse direction).

The length of a path in a stochastic graph is a function of random events characterized by the probability distributions associated with the cost along the path. We therefore have to decide how we would like to deal with the

uncertainty. The common approach to dealing with uncertainty is by considering the risk involved in the decision. Standard policies include *risk-aversion* (hoping to reduce risk, even at higher cost, i.e., minimize the expected maximal cost/error); *risk-seeking* (hoping to reduce costs, even at higher risk, i.e., maximize the expected minimal cost/error); and risk-neutrality (perfectly balancing risk and costs, i.e., the mean cost/error). A decision may also be bounded by either constant cost or risk.

In the risk averse policy, the shortest path selected will be the one that minimize the cost in the worst case scenario. There are few algorithms dealing with this problem, in general, we should choose a risk level α, between 0 to 1, which will determine the likelihood of scenarios we want to take into account. In the case of Global costs, the probability of each weight will be bound by α, and the cost value C will be calculated by: Select C such that $P(x < C) = \alpha$ for every edge.

In the risk-seeking policy we are trying to maximize the probability of choosing the shortest path, where the aim is to get the lowest possible cost. In this case the lowest cost available in every edge distribution will be chosen as the edge cost. This strategy can be moderated by choosing different costs with higher probability for every edge.

Different decision strategies can lead to different shortest path selections in a given control graph. For instance, supposed we are given the following monitoring multigraph (Figure 1). Here, robot S has to choose a minimal path to the leader G, based on accumulating positioning errors (measured in cm; the lower the better) that are normally distributed in the edges. Using a risk averse policy with risk level $\alpha = 95\%$, S will choose the minimal path $S \to C \to G$ with accumulated error of 33.3cm, as the other path is expected to be 35.4cm in length under the same risk policy. However, in case of using the risk neutral policy, the path $S \to A \to G$ will be chosen as minimal path with expected value of 28cm, while the other path value is 30cm.

In general, variants on Dijkstra's algorithm, such as those used in previous work, must be modified to consider risk-dependent policies. Several such algorithms appear elsewhere [16, 19], and are outside the scope of this paper. However, it has been shown that risk-neutral selection both works correctly [19], and is safe, in the sense that it minimizes notions of regret [29]. For the remainder of the work, and in the experiments, we therefore used the risk-neutral policy, by using the expected (mean) value of the distributions $E[\overline{C_{ij}^x}]$ (or, as needed, $E[\overline{R_{ij}^x}]$) as the edge weights. Here $E[P]$ is the expected (mean) value of the probability distribution P.

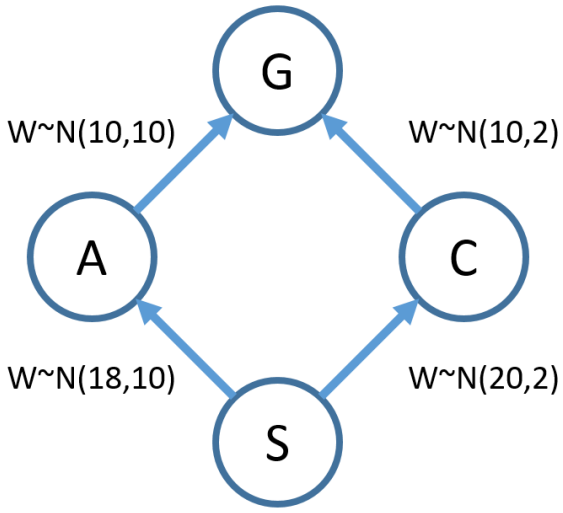


Figure 1: An example stochastic graph, with weight drawn from the normal distributions shown.

3.4 Inducing Control Graphs with Optimal Global Anchor

Having decided on a policy for handling risk resulting in the distributions of errors and costs, we now turn to the second challenge raised by the state of the art. Our task here is to select a single robot which will serve as the origin point (global anchor) for an agreed-upon shared coordinate system, or as leader of a formation. In the previous sections, we have discussed how, given this selection, it is possible to compute optimal control graphs which minimize costs or errors. In this section, we discuss how to efficiently select a robot whose associated optimal control graph is superior to all other control graphs, for all other leaders. We focus concretely on the task of relative localization and construction of a shared coordinate system, and will therefore optimize the leader selection and associated control graph to reduce the errors $\overline{R_{ij}^x}$. However, we emphasize that the same algorithm can be used to optimize costs.

3.4.1 Problem Formulation

K robots are positioned in space. Each robot is equipped with sensors, allowing it to identify (some) other robots in its vicinity, and to estimate their position with respect to itself (i.e., their position in its own ego-centric coordinate frames). Furthermore, we assume robots are able to communicate with their peers, at least with those they are able to observe. The settings are captured by a monitoring multi-graph G_K. The task is to extract a control graph where the coordinate frame of a single robot (global anchor) is used as the origin, and all robots align their coordinate frames to it. Because not all robots can directly sense the global anchor, each robot can decide to align its coordinate system with respect to one other robot (called local anchor), who aligns itself to the global anchor, or to another local anchor. Thus a coordinate frame alignment control graph has the following properties:

- The vertex representing the global anchor has an out-degree of 0.
- All other vertices (robots) have an out-degree of 1.
- There exist a path from every vertex (robot) to the vertex representing the global anchor.

A coordinate frame alignment control graph is optimal with respect to the selected global anchor v_A if it minimizes the errors in position estimates of the robots. Suppose we have a robot v_0. Its position estimate in the shared coordinate system accumulates errors with every local anchor it uses on a path from itself to the global anchor in the control graph. It thus seeks to minimize the sum of expected errors $\sum_{e_{ij}} E[\overline{R_{ij}}]$ where e_{ij} is an edge on the path from v_0 to v_A. The question is how to choose v_A. If we assume all robots have the same position estimate errors, then the control graph is the directed breadth-first search tree resulting from applying BFS to the monitoring multigraph, beginning with the global anchor [24]. Otherwise, a version of Dijkstra's algorithm may be used to induce an optimal control graph for the global anchor [17].

3.4.2 Optimal Global Anchor Selection

A global anchor v_A is called optimal, if its associated control graph is superior to the control graphs associated with any other potential global anchor. We

consider two different ways a control graph may be superior to another: It may reduce the average position error for the group (a societal view of errors), or it may reduce the maximal position error (an individual view of errors). Our task here is to determine the optimal global anchor for both definitions.

The process includes the following steps (see details next).

1. Transform the stochastic monitoring multigraph G_K into an intermediate representation, G'_K, which is a deterministically-weighted regular digraph (embedding errors, and reversing direction of edges). This step is carried out in time $\mathcal{O}(|E|)$, where E is the bag of edges in G_K.

2. Apply an All Pairs Shortest Path (APSP) algorithm to the graph G'_K. The time needed depends on the algorithm chosen, but is generally $\mathcal{O}(|V|^3)$, where V is the set of vertices in G'_K (normally, $|V| = K$).

3. Determine for each robot $v \in V$ the set of shortest paths leading from it P_v. For each such set P_v, determine the sum of the path lengths S_v, or the maximal path length M_v, depending on the global anchor selection criteria. This is carried out in time $\mathcal{O}(|V|^2)$.

4. The global anchor v_A is one that minimizes S_{v_A} or M_{v_A}. This is determined in time $\mathcal{O}(|V|)$.

Transformation of G_K into G'_K. This step is carried out to transform the stochastic directed monitoring multigraph into a deterministic graph, which embeds the necessary information, yet amenable to the execution of familiar graph-theoretic algorithm. The graph $G'_K = \langle V', E' \rangle$ is built as follows.

First, we set $V' \leftarrow V$. Then, for each pair of vertices $v_i, v_j \in V$, we do the following.

1. If an edge e^x_{ij} exists, with error distribution $\overline{R^x_{ij}}$, then create a temporary reversed edge, e'^x_{ji}, with scalar weight $r^x_{ji} = E[\overline{R^x_{ij}}]$.

2. Among all edges e'^x_{ji}, select the one with minimum r^x_{ji}, i.e., $e_{ji} = \arg\min_{e'^x_{ji}}(r^x_{ji})$.

3. Add e_{ji} to E'

The result is a directed graph, with scalar deterministic edge weights, in which all errors have been folded into the edge weights using the risk-neutral

policy, redundant edges in the multigraph removed, and edge direction reversed. This allows us to now run the shortest path algorithm in the direction from a selected vertex to others, representing the reverse direction to the robots' monitoring of the selected robot. For ease of presentation, we described the process as if it examines all pairs of vertices, but it could be rewritten to work enumerating the edges.

All Pairs Shortest Paths. We now run an algorithm for determining the shortest paths for all pairs of vertices. In our implementation we utilized Johnson's algorithm [7]. Given the size of V' is the number of robots K, the algorithm runs in $\mathcal{O}(K^2 \log K + K|E|)$. The result is often represented in a matrix L, such that matrix cell l_{ji} contains the length of the shortest path from vertex j to vertex i (or ∞ if none exists). As edges are revered in direction compared to the sensing direction, l_{ji} is the accumulating error in position estimates, from robot v_i to robot v_j, where $v_i, v_j \in V$.

Determine S_v and/or M_v. We propose two different criteria for selecting a global anchor that, if used as the origin for a shared coordinate system, would result in smaller position estimate errors for the team of K robots. One possible criterion is to minimize the mean position error of all K robots. This is a societal criterion, as it balances the errors across all robots. An alternative criterion is to minimize the worst-case error of any single robot, possibly resulting in some robots accepting a larger error than individually needed, in order to reduce the error got others.

We examine the matrix L. Let S, M be vectors of dimension K. We denote S_v the component of S associated with a given v (and similarly, M_v). For all $v \in V$, $S_v = \frac{1}{K} \sum_{i=1}^{K} l_{vi}$, i.e., the sum of all cells in row v divided by K, or more intuitively, the mean length of shortest paths from all robots i to robot v. As these shortest path represent smallest errors, this is the mean smallest error in position estimates, if v is selected as global anchor. Similarly, for all $v \in V$, $M_v = \max_{i=1}^{K} l_{vi}$, i.e., the maximal smallest error in position estimate for any robot i, if v is the global anchor.

Determine global anchor v_A. Finally, a new global anchor can be chosen, by setting $v_A = \arg\min v \in V' S_v$, if we prefer a global anchor that minimizes the average position error, or $v_A = \arg\min v \in V' M_v$, if we prefer to minimize

Robot ID	Max number of hops	AVG number of hops
1	3	$13/9 = 1.4$
2	2	$18/9 = 2$

Table 3: Global anchor selection changes based on criteria.

the maximal error instead. If there are ties, they can be broken by preferring according to the other criterion, or arbitrarily.

As an example of the difference in selection criteria, consider the graph in Figure 2. It shows 9 robots and their control graph. Let us assume all position errors are the same, thus minimizing position errors is correctly approximated by minimizing the number of edges in a path. If we select robot 1 as the global anchor, the average number of hops reaching all robots is the sum of all path lengths (13), divided by the number of robots (9), i.e., 1.4. The maximum path length for any single robot, however, is 3 (from the rightmost robot to robot 1). On the other hand, if we select robot 2 as the global anchor, the average number of hops will be $18/9 = 2$, and the maximum path length will be 2. Thus in case, the average error will increase, but the worst-case error is reduced.

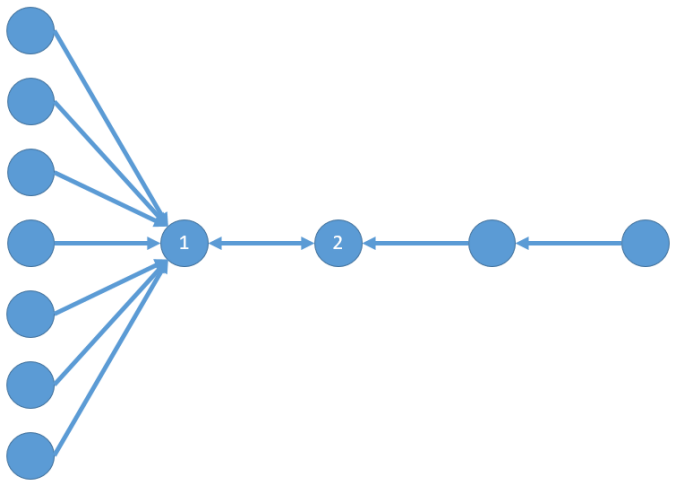


Figure 2: Example graph for global anchor selection.

4 Aligning Frames

In the previous section we presented a method for finding the optimal anchor for a given team of robots, given an optimization criterion. In this section we present in details the method used in order to align the robots' frames to the optimal anchor frame. This will produce a single axis system, with origin on the optimal anchor, and a set of positions that represents every robot in the team relative to the new origin.

4.1 Defining a single axis system

In order to create a single axis system on which every root knows its relative place to all other robots, the team needs to define one axis system, and then every robot needs to align its coordination frame relative to the best anchor. Each robot needs to calculate its rotation relative to the axis system and its translation relative to the axis system origin (the optimal anchor).

There are different ways to define one axis system that will be agreed by all the team. Moreover depending on the given team of robots and the given task the team is planning to preform, there is a different relevant axis system (1D, 2D or 3D). For example, a team of robots that are patrolling on a straight line, will need only 1D axis system in order to align them selves. A team of UGVs that preform a coverage task will need 2D axis system, but a team of UAVs that can move in 3D and are not bounded to move on the ground, will need a 3D axis system in order to align themselves.

There are some cases were even though some of the robots in the team can maneuver in 3D, the team needs only a 2D axis system in order to align themselves. For example, if the team is composed of a given number of UGVs and one UAV, it is enough to have a 2D axis system that all the team is aligned accordingly, and only the UAV will have the dimension of height.

The axis system can be determined using vision. In this case, all robots in the team needs to see a number of anchors, and create the axis system accordingly. That means that for a 2D axis system, all robots in the team need to decide on the positive direction of the 'X' axis and the positive direction of the 'Y' axis. After the axis have been defined, rotation can be calculated for all the robots in the team.

In order to agree on the axis system, a team of robots can use other resources other than vision. The most basic sensor that is widely used by robots, and can help in defining an axis system, is the compass. Using the

North-East convention for 2D axis system, or the North-East-Down convention for 3D system, can ease the construction of the axis system, and ease the stage of calculating the rotation for every robot.

In the experiments we preformed in the simulator and with real robots we used the compass sensor in order to construct the axis system. All robots calculated their rotation relative to the North and later calculated their translation from the origin.

4.2 Aligning the team frames

Given the selection of the global anchor v_A, we can now proceed to the construction of the shared coordinate system by frame alignment. After each robot decided on which anchor to align its frame to, the process of aligning one robot frame to its local anchor frame is a set of affine transformation (rotation and translation) which after, the robot's location is relative to the anchor coordination system. This process is described in detail in [24], The next section will describe the algorithm for iteratively aligning all robot in the team with v_A.

Briefly, robot v_A sets itself as the origin point, and broadcasts its own identifier as such. Robots monitoring v_A then align their coordinate frames with respect to v_A, and inform their peers that they are thus aligned and can serve as local anchors. Robots further down the anchor chains, repeat this again and again, until all robots are aligned to the origin in v_A.

4.2.1 A Centralized Algorithm For Aligning Frames

In previous sections we presented the method of choosing v_A that minimizes the errors. The algorithm presented here is a centralized algorithm for aligning all team frames. It is centralized in the sense that it considers the entire visibility graph, and can act on any edge or vertex arbitrarily. To simplify the algorithm we assumes that if edge $(v_i, v_j) \in E$ exists, its cost is 1, which means that the visibility graph is equal to the cost graph. This assumption can be removed by using another graph in the algorithm.

The algorithm that we present is implemented as a framework and can be easily applied to any multi-robot system. The robots in the system should have the capability to sense their location relative to other teammates in the group and should provide an implementation for checking if robot v_i can sense robot v_j ($bool\ isConnected(i, j)$).

The algorithm has 5 main parts:

1. Initialization (line 2)
2. Updating v_A (line 4)
3. Alignment based on local anchor (line 7)
4. Preforming robots' tasks (line 21)
5. Updating the connectivity matrix (line 22)

In the first part the algorithm sets value to two matrices. $ConMat_{old}$ is initialized to *null* and is being used to track changes in the connectivity between the robots in the team. The second matrix $ConMat_{new}$ is initialized with the current connectivity matrix. $GetConnectivityMatrix()$ in line 3 construct the connectivity matrix by using $isConnected(i, j)$ function. It iterates over all robots and checks for every robot if it is connected to all others (implementation can be seen in line 1)

Since the second part of the algorithm is computationally expensive and takes long time to calculate, it is being preformed only if there is a change in the connectivity matrix. If the team keeps its formation or standing still, and the connectivity is not being changed, this part will be preformed only once, in the beginning of the algorithm.

In the second part of the algorithm we calculate all the shortest paths that exists between all the robots based on the connectivity matrix. Any implantation of the algorithm family of $AllPairsShortestPath$ can be used to calculate the shortest path connectivity matrix ($ConMat_{sp}$). In our experiments we implemented the Johnson's algorithm.

After calculating $ConMat_{sp}$, we use this data to find v_A as shown in previous section.

In the third part of the algorithm, the alignment of the coordinate system is preformed and the location estimate is calculated. But before, each robot needs to find its optimal local anchor to align according to it, and it needs to make sure that its local anchor is already aligned according to v_A. This can be achieved by iterating over the team members by their number of hops from v_A, and make sure all the local anchors will be aligned according to v_A.

In line 7, i represents the current number of hops from v_A and in every iteration, j is used to find all robots in the connectivity matrix with i number of hops from v_A (line 11). After finding robot j with i hops from v_A, robot

j needs to find its local anchor. This is being preformed by traversing row number j in the shortest path matrix and looking for a cell with the value 1. From all robots that are connected directly to j, the one with minimal cost to v_A is chosen as the local anchor of robot j. After choosing the local anchor, robot j aligns its coordinate system with its local anchor and calculate its estimate location based on relative data gain from its image sensor. Robot j has its one coordination frame, its local anchor frame and both has a given axis system. Based on this data, rotation and translation can be calculated.

The fourth part (line 21) of the algorithm is where the robots preform their individual task. As this algorithm is a framework for multi-robots systems, in this part all robots already has estimated locations relative to one globe anchor and can start preforming their tasks. This part should not take long time or block the main loop of the algorithm, because it can affect the accuracy of the estimations.

The last part (line 22) is used for updating the connectivity matrix. After preforming some task or moving, some of the edges in the connectivity graph may change, and this change can affect the selection of v_A. We keep the current matrix in $ConMat_{old}$ and setting the new matrix to $ConMat_{new}$ using $GetConnectivityGraph()$.

This entire process of selecting a global anchor and re-aligning may need to be repeated as robots move or their sensing of their peers changes. A weakness of the algorithm above is that it is centralized, so changes requiring re-computation must be announced and the operation of the team must be paused until a new global anchor is computed. We conjecture that there is a method for making this process run faster than the initial process, by propagating edge weight changes only where and as needed. This was done for the more limited Dijkstra variant [17]. We leave this for future work.

Algorithm 1 Centralized Coordination Frame Alignment

```
1: procedure AlignFrames())
2:     ConMat_old ← null
3:     ConMat_new ← GetConnectivityMatrix()

4:     if ConMat_new! = ConMat_old then
5:         ConMat_sp = AllPairsShortestPath(ConMat_new)
6:         bestAnchor = GetBestAnchor(ConMat_sp)

7:     i = 0
8:     for i < sizeOfTeam do
9:         j = 0
10:        for j < sizeOfTeam do
11:            if ConMat_sp[bestAnchor][j] == i then
12:                localAnchorID = null
13:                localAnchorDepth = sizeOfTeam
14:                k = 0
15:                for k < sizeOfTeam do
16:                    if ConMat_sp[j][k] == 1 then
17:                        if ConMat_sp[bestAnchor][k] < localAnchorDepth
   then
18:                            localAnchorID = k
19:                            localAnchorDepth = ConMat_sp[bestAnchor][k]
20:                AlignCoordinateFrames(localAnchorID)

21:    DoWork()

22:    ConMat_old ← ConMat_new
23:    ConMat_new ← GetConnectivityGraph()
24:    goto 4
```

Algorithm 2 Get Connectivity Graph Function

```
1: function GetConnectivityMatrix
2:     i = 0
3:     for i < sizeOfTeam do
4:         j = 0
5:         for j < sizeOfTeam do
6:             if i == j then
7:                 ConMat[i][j] = 0
8:             else if isConnected(i, j) then
9:                 ConMat[i][j] = 1
10:            else
11:                ConMat[i][j] = ∞
12:    return ConMat
```

5 Evaluation

To evaluate the effects of using the techniques presented in this work, we implemented the algorithms for optimal global-anchor selection and coordinate frame alignment in ROS (Robot Operating System[1]), to be used on Gazebo[2]-simulated and real RoboTICan Lizi[3] robots (shown in Figure 4b) and the hector quad-rotor for simulated experiments on UAV-UGV formations. All robots in the team were marked with unique visual markers identifying each robot. Using image and depth data from an RGB-D sensor (a Kinect), each robot identified its neighbors and measured their relative position in its reference frame. The performance of the algorithm was tested on a variety of formations.

We have carried out experiments in three types of settings: robots standing still (Section 5.2), robots moving while maintaining a static formation (Section 5.3), and robots moving while changing formation (Section 5.4). In the first two settings, the relative positions of the robots are maintained—by definition in the first setting, and using feedback control in the second. In the third setting, moving robots changed their initial formation, requiring them to select a new global anchor. In the experiments, the error measurements R_{ij} were synthesized from a sensor model that was estimated for the robots'

[1] http://www.ros.org/
[2] http://gazebosim.org/
[3] http://www.robotican.net/

RGB-D sensors from calibration data (Section 5.1). Results summery can be seen in table 4.

5.1 Sensor Model

In our algorithm we assume that the robots can assign cost or error to every edge in the control graph that is relevant to their node. There are two separate ways by which we can get the cost and error distributions for a sensor, or a combination of sensors. The first is by learning/estimating the distribution from calibration. The second is by relying on the specs of the sensors from the manufacturer.

We wanted to test and see if the cost and error can be predicted based on the robot's sensor. We placed the Kinect sensor in front of a marker of size 10×10 centimeters with a distance of 1.5 meters and measured the difference between the real distance from the marker and the Kinect measurements for 1 minute. We did the same measurement for every 10 centimeters, and calculated the average difference (error) for every distance. The results are shown in Figure 3. The graph represents the average error for every measured distance. For example, when the sensor was placed at a distance of 150 centimeter, the average error was 0.055 meters. That mean that the average measurement was 1.445 meters. The error bars show the standard deviation of the measurements, that means that some of the measurements were 150 ± 2 and some were 150 ± 9.

The results show that the sensor has an optimal range that produces minimal error in range measurement, this is the readings range between 50 to 70 centimeters. Above this range and also below it, the error is growing. Based on this data the algorithm can assign predictable error/cost to the edges. The second interesting result that can be seen in the Kinect sensor model graph is that even though there is a distribution of the measurement error it is not around the real measured value, that means that for the Kinect, we can try to improve the accuracy of the measurement by averaging more than one result, but it still going to produce an error.

We conducted the same evaluation for other sensors that can be used in multi robots team in order to measure relative location such as a simple USB camera with marker recognition software, the *Hokuyo URG-04LX-UG01* and the *Hokuyo UTM-30LX* laser sensors. We did not repeat the same experiment for the Hokuyo lasers but instead we evaluated the errors, costs and working distance from the manufacturer specification sheet. For the USB camera we

Type	Experiment	Robot ID	Arbitrary v_A Error in meters	**Optimal** v_A Error in meters	Significance p value (one-tailed t-test)
Standing	3-line (simulation)	3	0.058 (0.102)	0.036 (0.009)	7.12×10^{-15}
	3-line (real robots)	3	0.107 (0.019)	0.049 (0.001)	0 (below excel limit)
	6 zigzag (simulation)	2	0.031 (0.021)	0.014 (0.006)	2.62×10^{-78}
		4	0.073 (0.142)	0.030 (0.005)	4.12×10^{-15}
		5	0.086 (0.181)	0.032 (0.005)	4.90×10^{-15}
		6	0.134 (0.239)	0.061 (0.033)	7.49×10^{-16}
Moving	Simulation 4 center	3	0.036 (0.014)	0.019 (0.018)	$1.72 \times 10{-}98$
		4	0.032 (0.017)	0.013 (0.012)	5.92×10^{-143}
	Real moving 4 center	3	0.155 (0.076)	0.095 (0.009)	8.31×10^{-6}
		4	0.140 (0.105)	0.084 (0.038)	0.00056

Table 4: All experiment results, including mean errors in meters (standard deviations), and t-test significance testing. Robot ID is shown for robots not acting as global anchor v_A in either setting. The optimal global anchor column shows significant improvement in *all experiments*.

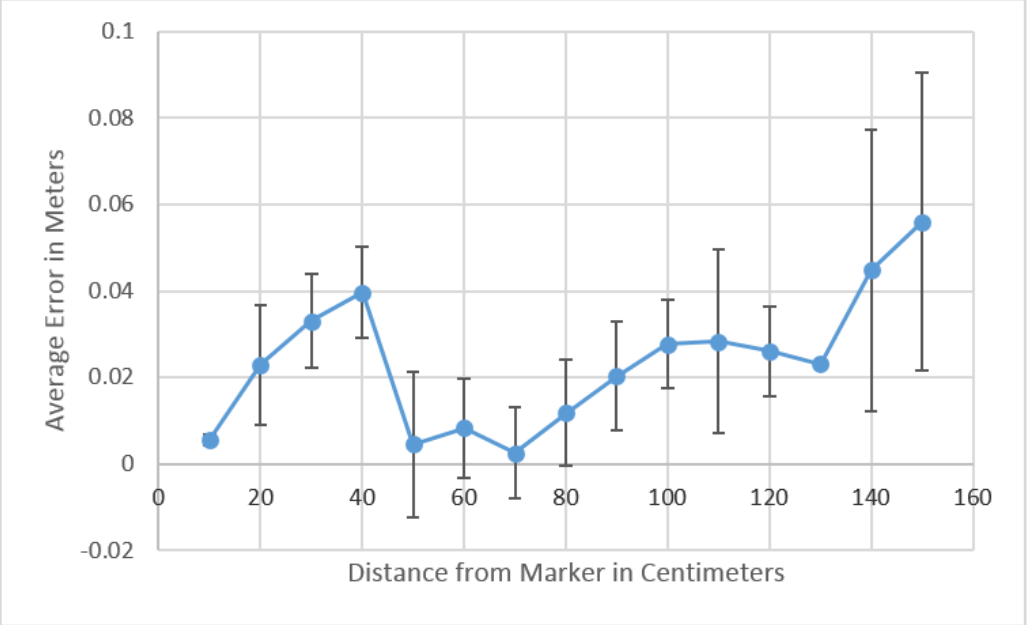


Figure 3: Kinect sensor model

used the manufacturer sheet for costs and distances and for estimating the error we based our prediction on the results shown in [20].

Table 5 summarizes the cost and capabilities for all the sensors. Every row in the table represents a different sensor and its capabilities. The first column notes the electrical cost for using the sensor in ampere (Taken from the manufacturer manuals). If a robot decides to use the 'Hokuyo 30' over the 'Hokuyo 04' it will use more battery power on the sensor. The second column notes the estimated error in range measurement. For example, as seen in Figure 3, the average error for the Kinect sensor depends on the distance from the measured object, that means that for an object that is located in front of the sensor with distance of $10 - 150$ centimeters, the error can vary between an average of 1 centimeter for the optimal distance measure and an average of 6 centimeters for the worst. In the case of the Hokuyo sensors, based on the manufacturer specification sheet, the error is calculated as a percentage of the distance, which mean that for a distance of 150 centimeters the error can be up to 4.5 centimeters. The last column notes the maximum distance that the sensor can measure.

Sensor Type	Costs (Electricity)	Errors	Max Range
Kinect	1A (RGB & Depth)	10–60mm depends on distance to leader	1.5m (For 10×10cm marker)
USB Camera	0.3A (RGB only)	30–150mm depends on distance to leader	1.5m (For 10×10cm marker)
Hokuyo URG-04LX	0.5A	0.06 to 1m : 30mm, 1 to 4m : 3% of measurement	4m
Hokuyo UTM-30LX	0.7A	0.1 to 10m : 30mm, 10 to 30m : 50mm	30m

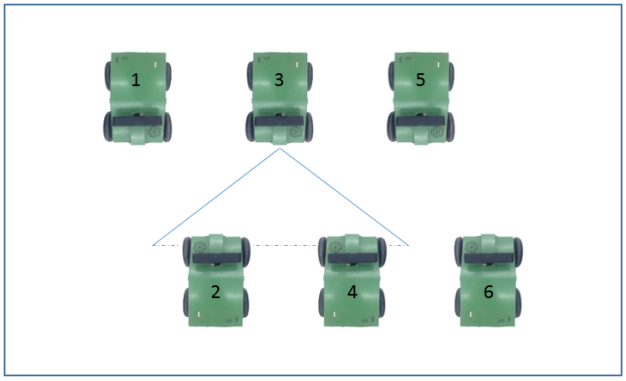
Table 5: Cost and capabilities for different types of sensors

5.2 Robots standing still

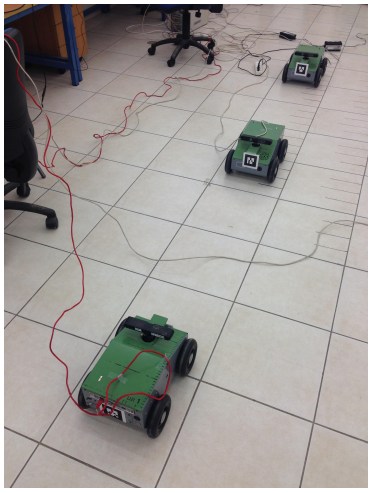
We begin with experiments in simulation. Our first experiment here recreates an experiment in [24]. Six Lizi robots are placed as shown in Figure 4a. All robots are static, and align their coordinate system with respect to the selected global anchor. We use two different selection algorithms. The control experiments utilized the robot with minimal ID (an arbitrary selection), as in [24]. The treatment experiments used the optimal global anchor selection as presented above (specifically, minimizing average error). Each setting was run 5 times, each trial for two minutes, for a total of almost 1000 measurements in each setting. We then compare the global position errors of the robots using the two global anchors, based on ground truth measured externally.

We conducted similar experiments, placing three robots as shown in Figure 4b. These were conducted both in simulation, as well as in real robots. Robot 1 (bottom of the image) could monitor robot 2 (center) and vice versa; robot 3 could see robot 2. We again collected position measurement data for 10 minutes in each settings, in simulation, and then again in the real robots.

The results, summarized in the upper part of Table 4, show a very significant improvement in the position estimates of the robots, in the shared coordinate system. For example, in the experiment with six standing robots, when using the minimal robot ID as a global anchor the farthest robot (#6)



(a) Six simulated robots.

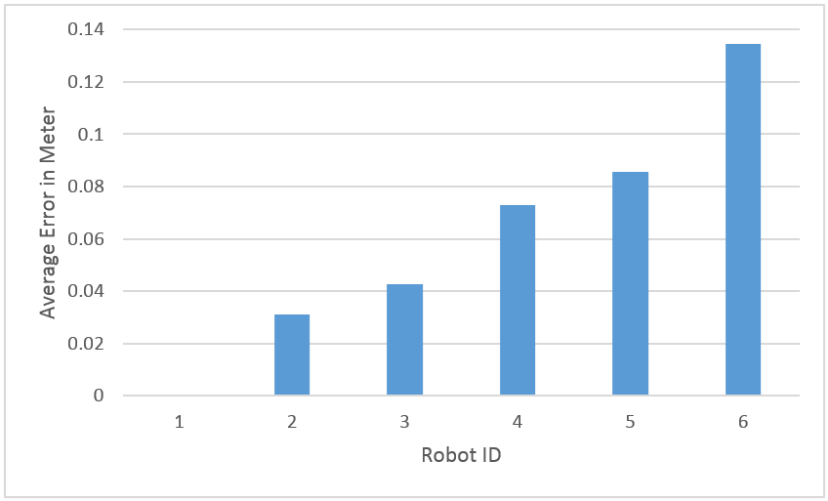


(b) Three real robots. Simulated robots placed likewise.

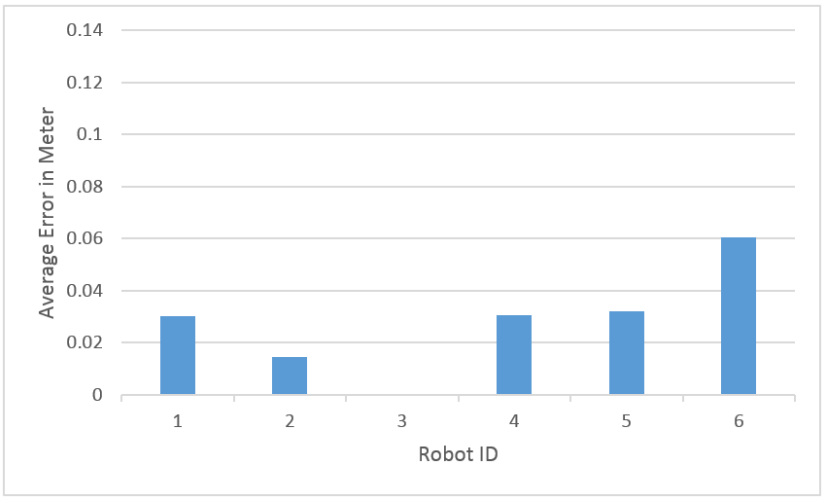
Figure 4: Formation in static experiments.

was located five hops away, and accumulated approximately 13cm in error. However, using the global anchor selected by our algorithm, the average error for the same robot, now located 3 hops away, drops to 6cm. This improvement is statistically significant (one tailed t-test, $p < 7.49 \times 10^{-16}$). Similar improvements can be seen in the case of three robots standing in line, both simulated and real. Over all trials, these results are over approximately 5000 measurements in each settings, for each robot. Figures 5 and 6 shows the results of the simulated experiment while Figure 6 shows the comparison between minimal and optimal v_A average error for a given number of hops in simulated experiments.

Results in Figure 7 show the improvement in location accuracy for the real robots experiment. When using the minimal robot ID as a global anchor the farthest robot (#3) was located two hops away, and accumulated approximately 10cm in error. However, using the global anchor selected by our algorithm, the average error for the same robot, now located one hop away, drops to 5cm.



(a) Minimal robot ID as v_A



(b) Optimal robot ID as v_A

Figure 5: Simulated experiment with six static robots

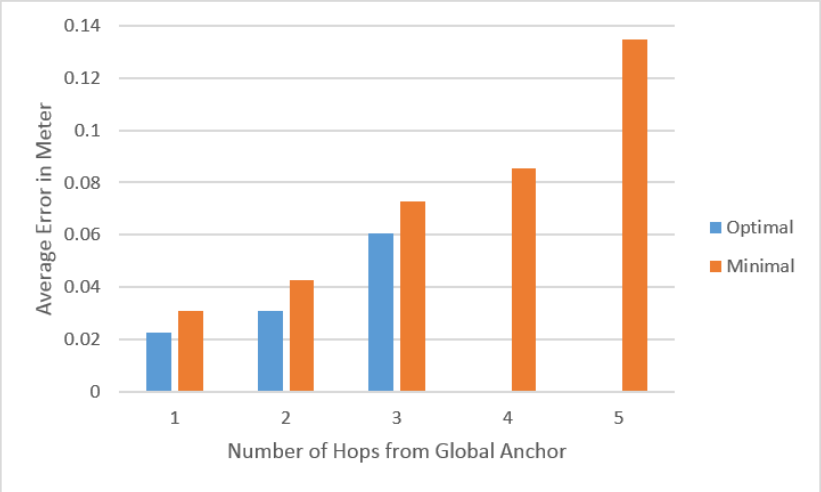


Figure 6: Simulated experiment with six static robots, Optimal Vs. Minimal

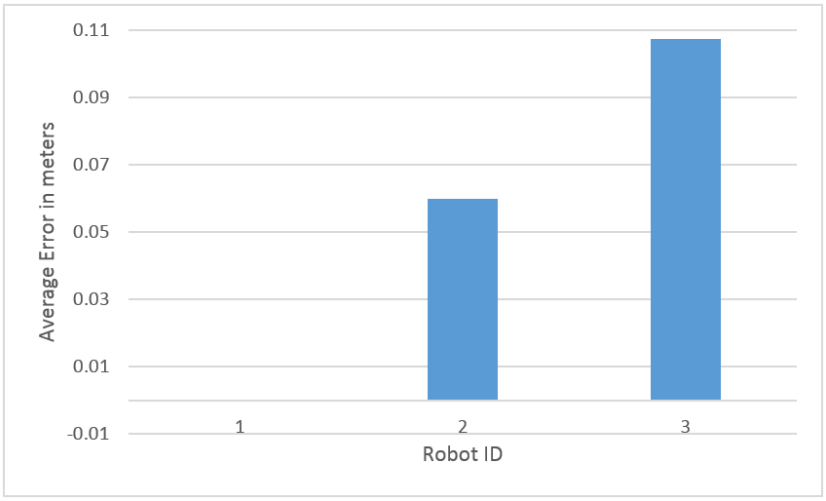
5.3 Moving in Static Formation

We now turn to experiments where robots moved while continually estimating their position based on a shared coordinate system, with the origin at the selected global anchor. We placed four robots in the formation shown in Figure 8a, again both in simulation as well as in the lab. Robot 1 (front of the formation) could monitor robot 2 (center) and vice versa, robots 3 and 4 (side by side, bottom) could monitor robot 2. Figure 8b shows the real robots in one of the trials.

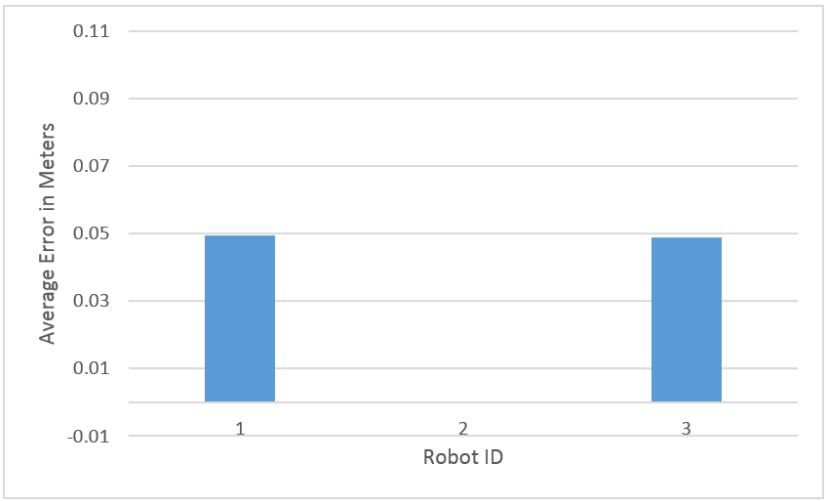
We conducted 5 runs of the formation in each of the settings (using the minimum ID for the global anchor v_A, and using the global anchor selected by our algorithm). Each run was for two minutes, allowing us to collect just under 1000 position measurements in each settings, *for each robot*. In the minimum ID settings, robot 1 was selected as the global anchor. In the optimal settings, our algorithm chose robot 2 as the global anchor.

While in the simulated experiment the location error was measured relative the ground truth, in the experiments with real robots we did not have a accurate location of the robots in the lab. We got the real robots relative location from measurements that where done during the experiment with a roller and measured the errors based on the chosen leader.

The results of this experiment are summarized in the lower part of Table 4. We see very significant improvements in the accuracy of the position

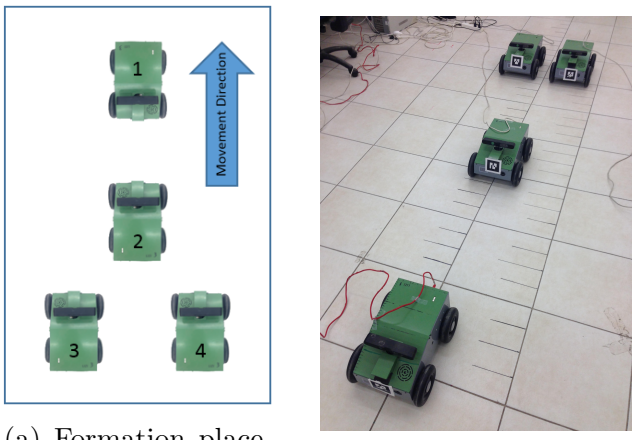


(a) Minimal ID as v_A



(b) Optimal ID as v_A

Figure 7: Running static experiments on real robots



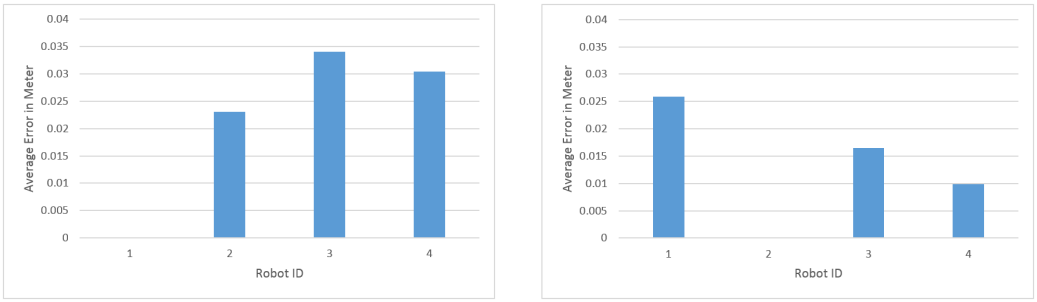
(a) Formation placement.

(b) Real robots.

Figure 8: Formation maintained while moving.

estimates of robots 3 and 4, which are trailing behind. For instance, in the real robots, position estimate errors dropped from around 13cm to around 9cm. Figure 9 and 10 show the results for the simulated and real experiments.

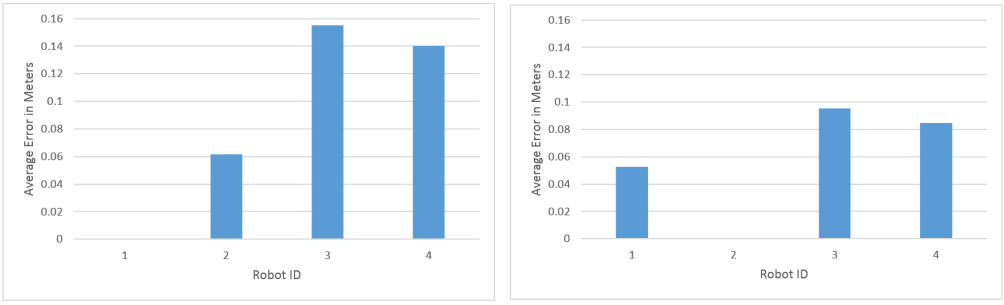
An interesting observation can be made from this experiment, regarding the difference in tasks between formation maintenance and stability, and coordinate frame alignment. This experiment showed that the first robot in a formation may be optimal as a leader in the sense of showing the path to all robots but may not be optimal as global anchor to the shared coordinate system.



(a) Minimal ID as v_A

(b) Optimal ID as v_A

Figure 9: Simulated experiment with four moving robots



(a) Minimal ID as v_A (b) Optimal ID as v_A

Figure 10: Real experiment with four moving robots

5.4 Moving and switching global anchor v_A

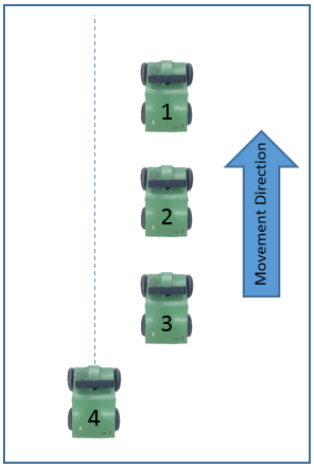


Figure 11: Dynamic formation. Robot #4 overtaking others.

As a final experiment, we tested the ability of the algorithm to adjust the global anchor while moving, when the relative position of robots is changed. Four simulated robots were placed as shown in Figure 11. All robots moved forward; robots 1–3 at constant speed, and robot 4 three time faster, along the dotted path shown in the figure, and until it pulled ahead of everyone else. While moving, the robots continually checked and recomputed the global anchor appropriate to their current settings.

We ran 5 trials of this experiment, each taking approximately two and a half minutes. At the beginning of each run, robot 1 was chosen as global

anchor v_A, and the algorithm chose local anchors for all other robots: robot 4 monitored 3, which monitored 2, which monitored 1. However, as robot 4 begins to overtake it peers, its local anchor changes from 3 to 2, then to 1, until finally it overtakes robot 1, at which point it becomes the global anchor, and root 1 switches to monitor it.

Figure 12 shows the mean error (error bars indicate standard deviation) of robot 4 during the experiment. It shows that between 0.1 minutes and 0.5 minutes into a trial, when robot 4's local anchor is robot 3, the error in position (in the shared coordinate system where robot 1 is the origin) is around 40cm. After passing robot 3, robot 4 changes local anchor based on the optimal selection, first to robot 2 and then to robot 1. Approximately 0.95 minutes into the run, and until 1.15 minutes in it, robot 4's local anchor is robot 1 which is still the global anchor v_A. We see a corresponding decrease in robot 4's position error as it now monitors the global anchor directly. After 1.15 minutes, robot 4 cannot see any other robot and its error increases due to moving and assuming location in its last position. With real robot it is possible to change the localization method to less accurate one such as GPS in this situation. After robot 4 enters robot 1's field of view, the algorithm sets robot 4 to serve as v_A.

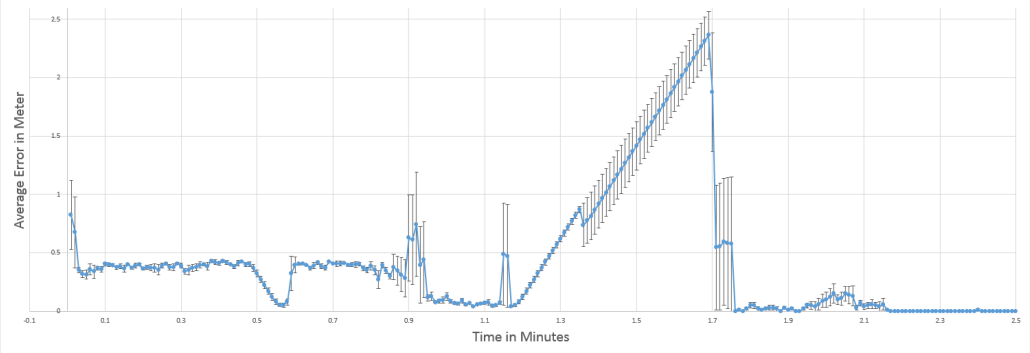


Figure 12: Changing control graph in real time

5.5 UAV-UGV Formations

In order to evaluate our algorithm with heterogeneous robots and to evaluate the affect of different camera to marker angles on the location estimate, we

created a simulated experiment with two robots: one UAV and one ground robot. Sections 5.5.1 and 5.5.2 describe the experiment setup and results.

5.5.1 UAV-UGV Experiment Setup

In this experiment we used two simulated robots, a UGV and a UAV. For the UAV model we used the *hector_quadrotor* ROS package [22], we have used the basic quadrotor model with one fixed camera in the front of the UAV. For the UGV model we created a new omni directional robot shaped as a cube. We placed five different markers on the robot, one marker on each cube face and one on the top. We used the first five markers in the *ar_pose* ROS package implementation.

In the experiment presented here, the UAV flies in a circle around the UGV while maintaining the marker placed on top of the cube in its sight. We ran our algorithm to set the optimal v_A (in this case there is only one possibility) and measured the errors for the UAV estimated location.

5.5.2 UAV-UGV Experiment results

In this experiment we found that the angle of the UAV relative to the marker on the ground changes the accuracy of the distance measurement, thus effects the location estimate error. We saw the same behavior in the Lizi experiments and also with two different implementations of markers in ROS: *ar_pose* and *ar_track_alvar*. Figure 13 shows the UAV location error while it circled around the UGV and estimated its location based on one marker placed on top of the UGV. We can see that in some bearing angles to the marker the UAV has better accuracy in its location. Those findings can be taken in consideration when choosing v_A, a better understanding of the causes to the errors can help improving the location estimations.

6 Discussion and Extensions

The algorithm presented (and evaluated) in the previous sections can be used for different multi robot teams with a variety of tasks, in many environments. In this section we discuss some extensions that can be made in order to improve the capabilities of the algorithm and in order to improve its evaluation.

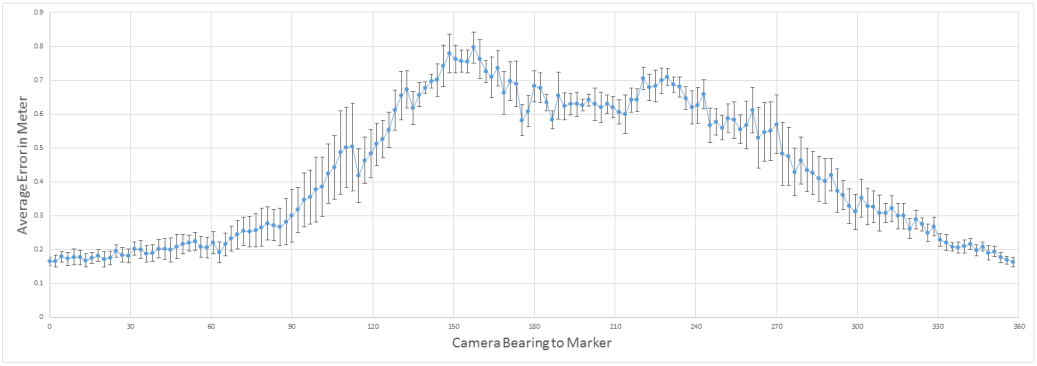


Figure 13: Average UAV location error given different angles to marker

6.1 Static Landmarks

It is possible that for some tasks the team of robots will prefer to align their coordinate system on static landmarks that the team can identify. Those landmarks can be objects in the surrounding area or even a malfunctioning robot. In this case the team needs to take in consideration that the object or robot can serve as a global anchor for the team but it should not be considered as an robot that needs to align its coordinate system or to serve as a local anchor. To deal with this case we extended the algorithm showed above to take this option of choosing a global anchor that is not from the team.

The changes that needs to be done are as follow:

1. In the monitoring multi-graph, vertices that represent the static landmarks need to be added, and all edges from all robots that can identify the landmark are added with their error distribution.

2. Run the *All Pairs Shortest Paths* algorithm.

3. Because that the vertices that represents the static landmark in the monitoring multigraph have no out-going edges, when calculating S_v or M_v there will be no change to the sum, because there are no paths between the static landmarks to the robots in the team.

4. The rest of the algorithm has no changes and the output with introducing static landmarks can improve the accuracy of the coordination system.

6.2 Optimal Control Graphs for Heterogeneous Teams

In section 5.5 we briefly started evaluating our algorithm on heterogeneous teams. In this section we would like to extend this evaluation on heterogeneous teams by using simulated graph with three types of robots and different possible types of sensors. The data regarding the sensors capabilities is based on the sensor models that were presented in section 5.1 and on the sensors manufacture specification. Table 6 summarizes the capabilities of our three robots used in the simulated graphs. The sensors we use are the Kinect sensor with two options of usage: RGB & Depth or RGB only that acts as a USB camera and two types of Hokuyo laser, URG-04LX and UTM-30LX. Despite the fact that the laser sensors can not be used in order to identify the other robots identification number, we use their specifications in order to show the differences in robots capabilities and leader selection.

In the simulated graphs, for a given formation of the robots, we show the different leader selection made by the robots in different optimization criteria (cost and number of hops). We also present the different leader selection for different distribution of robots types in the formation. Previous works (e.g., [17]) also included the tilt of the sensor and the range of the local leader in the cost function. The cost can be defined by many parameters that are relevant to the specific multi robot team. In this experiment we use the electricity consumption of the sensor in order to define the cost of using the sensor.

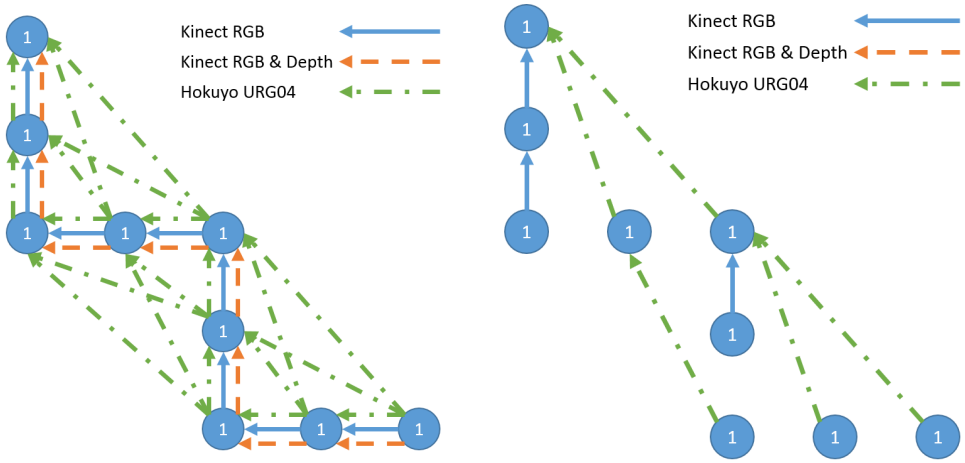
We present here two static formations, for each one we present two possible distributions of robots. One with only robots of "Type 1" and the second with a mix of robots of "Type 2" and "Type 3". In the example figures (15, 14, 17, 16), the type of robot is written on each circle that represents a robot. In all of our examples the distance of every two adjacent robots is 1.5 meter and the capabilities to identify a robot is based on this distance.

We also included in this section an example for the different leader selection in a simulated graph when changing the selection criteria for V_a. the formation presented in Figure 17.

Figure 14 shows a formation of nine robots all of the same type. The arrows present the capabilities of each robot in the team to identify and measure it relative location to other robots. Figure 14a presents the full multigraph that emerge for this example. Figure 14b presents the selected global and local anchors and the selected sensor that is optimal to use when trying to minimize the number of hops.

Robot type	Sensors Mounted	Costs (Electricity)	Errors	Max Range
Type 1:	Kinect	1A (RGB & Depth)	10–60mm	1.5m
		0.3A (RGB only)	30–150mm	1.5m
	Hokuyo URG-04LX	0.5A	0.06 to 1m : 30mm, 1 to 4m : 3% of measurement	4m
Type 2:	Hokuyo URG-04LX	0.5A	0.06 to 1m : 30mm, 1 to 4m : 3% of measurement	4m
	Hokuyo UTM-30LX	0.7A	0.1 to 10m : 30mm, 10 to 30m : 50mm	30m
Type 3:	Kinect	1A (RGB & Depth)	10–60mm	1.5m
		0.3A (RGB only)	30–150mm	1.5m
	Hokuyo UTM-30LX	0.7A	0.1 to 10m : 30mm, 10 to 30m : 50mm	30m

Table 6: Three types of robots that differ in the sensors mounted on them.



(a) Full multigraph

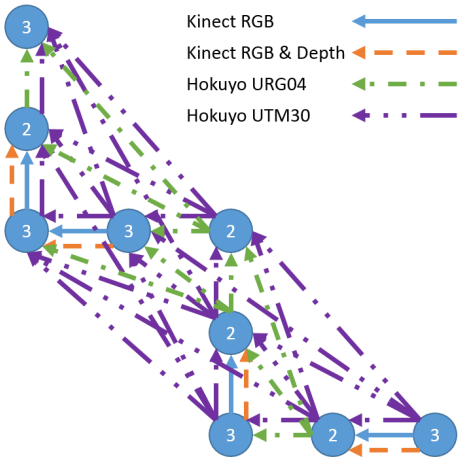
(b) Minimizing the number of hops

Figure 14: Nine robots formation of "Type 1"

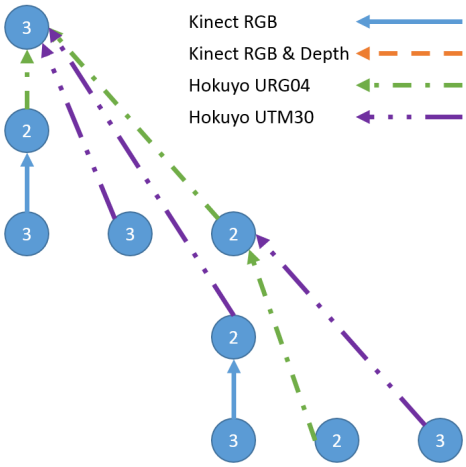
In Figure 15 we use the same formation as in the previous example, but the type of robots is changed. Figure 15a presents the new emerged multigraph and Figures 15b and 15c presents the selected edges when minimizing the number of hops to the global anchor and when minimizing the cost of using the sensors of the robots. The results show that for different types of robots, different formations can emerge. Even thought the global anchor in this formation is given (as it is the only one how can serve as global anchor), when trying to optimize different features the robots choose different local anchors.

Figures 16 and 17 present a formation of seven robots arranged in a shape of a square. In this formation the upper left robot and upper central robot can see each other. Due to this, both of these robots can serve as global anchors. In the next examples we can see that not only the local leaders are changing but also the global leader can be changed due to different optimization or different distribution of robots.

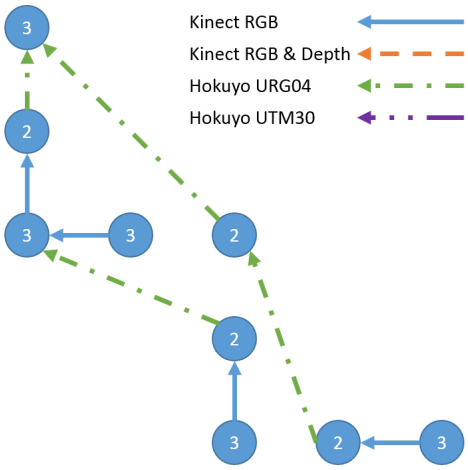
Figure 16a present the full multigraph that emerge from the square formation when all the robots are of "Type 1" while Figure 16b present the selection of global leader and local leaders when optimizing the number of hops. Even thought some of the robots can detect other robots with lower cost by using a different sensor, when trying to optimize the number of hops,



(a) Full multigraph



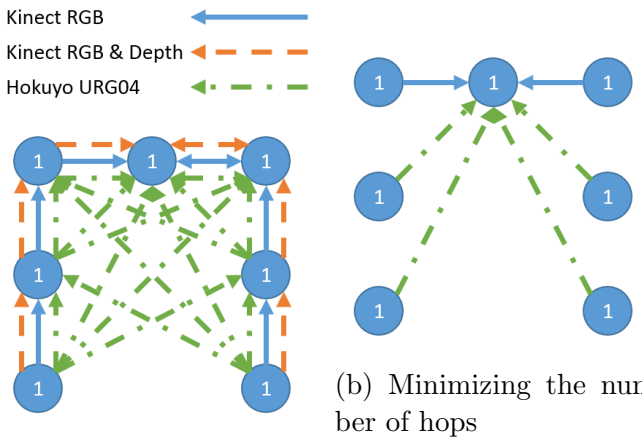
(b) Minimizing the number of hops



(c) Minimizing the cost

Figure 15: Nine robots formation of mix types

sometimes the individual robot in the team will "pay" more then the minimal cost his could.



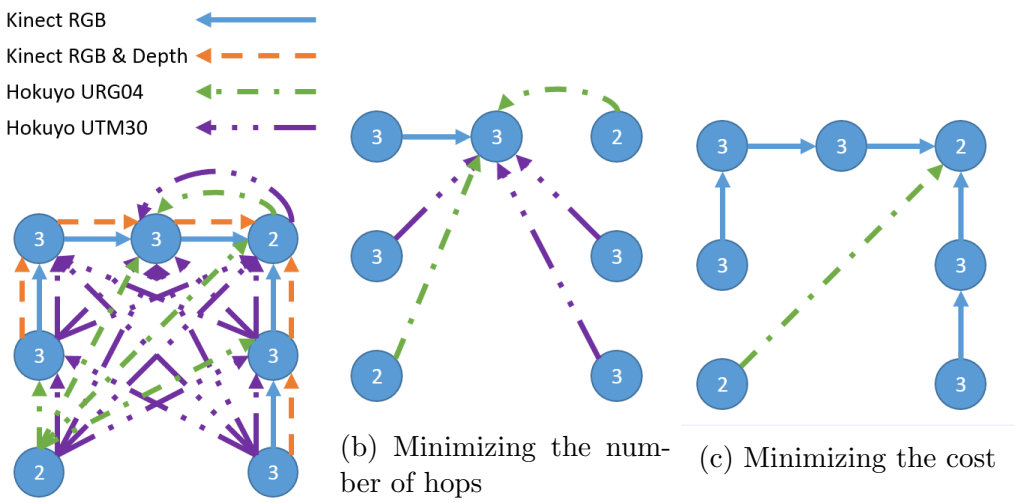
(a) Full multigraph

(b) Minimizing the number of hops

Figure 16: Square formation of "Type 1"

For the last example for heterogeneous robots team, we used the same team of seven robots holding a formation of a square, only that we changed the robots' type to "Type 2" and "Type 3". Figure 17a presents the multigraph for this formation and Figures 17b and 17c the formation control graph after the optimization. The results in these two control graphs show that for different optimizations the global anchor may change even thought the distribution of robots stayed the same.

The last example with simulated graphs, presents how changing the selection criteria of v_A, also changes the selected control graph. Figure 18a presents the full multi graph for the given formation. Figures 18b and 18c presents the selected control graph for the given team formation while choosing the Mean selection criteria and the Min/Max selection criteria. We can see that in this formation the selected v_A changed based on the criteria used to choose it. This is not true for all team formations, in the examples shown above, the leader selection and the selected control graph would be the same for both v_A selection criteria.



(a) Full multigraph

(b) Minimizing the number of hops

(c) Minimizing the cost

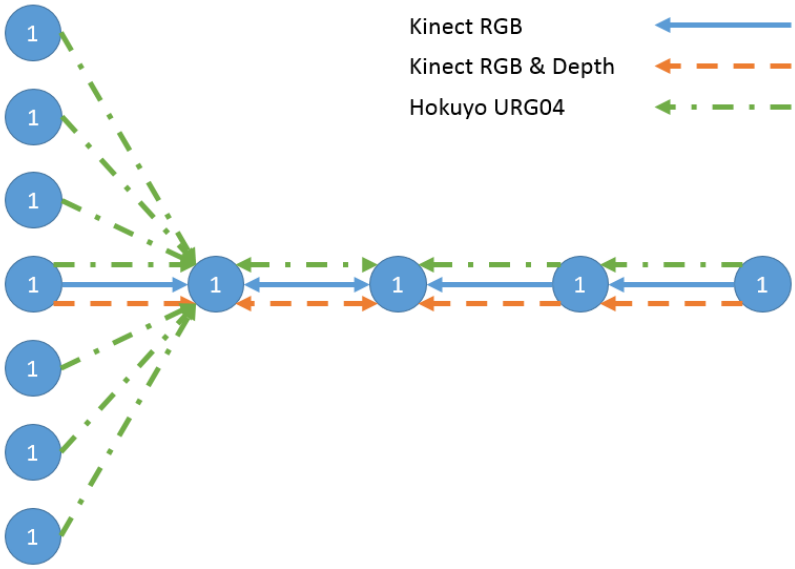
Figure 17: Square formation of mix types

7 Conclusion And Future Work

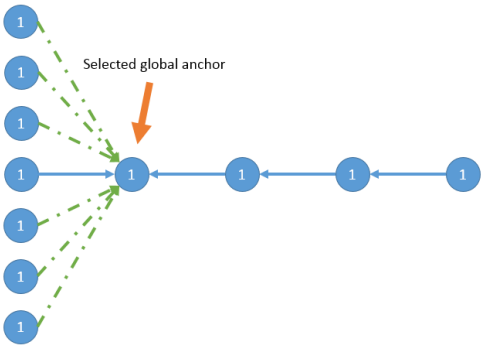
Control graphs are used in multi-robot systems to maintain information about which robot senses another robot, and at what position. On the basis of such graphs, it is possible to compute a shared coordinate system, localize relative to others, and maintain stable formations. To compactly represent all possible control graphs, a monitoring multigraph construct was proposed in [17], with the idea that an optimal control graph could be induced from such a multigraph.

In this work, we argued that while existing work shows how to induce and utilize control graphs for these different tasks, it makes two critical assumptions. First, it assumes that a single robot is given, to serve as formation leader or global anchor and origin point for coordinate frame alignment. Second, it induces control graph heuristically (often) and on the basis of unrealistic deterministic weights which qualitatively correspond to the robot builders belief in sensor reliability.

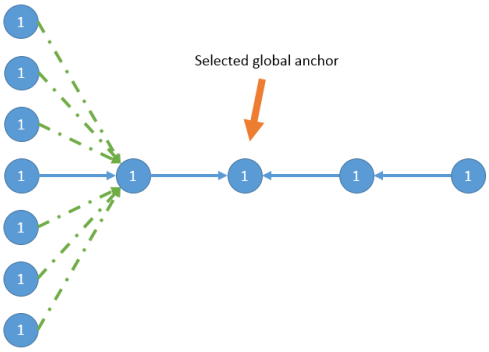
We address both of these assumptions. First, we extended the definition of *monitoring multigraphs*, a construct intended to compactly represent all possible control graphs, in several ways. We distinguished the cost and error factors which need to be taken into account, and argued for the ex-



(a) Full multigraph



(b) Mean selection criteria



(c) Min/Max selection criteria

Figure 18: Eleven robots of type one, optimized with two different v_A selection criteria

plicit representation of distributions of these factors, so as to allow managing risk involved in control graph induction. We focused on risk-neutral decision policy, which allows us to replace the stochastic edge weights with the deterministic expected value of the distributions (i.e., their mean). Second, we demonstrated that an All Pairs Shortest Path algorithm can be utilized, on the extended monitoring multi-graph, through some transformations. This facilitates the automatic determination of an optimal robot to lead a formation or serve as a global anchor. We conducted extensive experiments in real and simulated robots; these show very significant improvement to the robots' position estimates. In future work, we hope to examine alternative methods for dealing with decision policies that are risk-averse, or risk-seeking.

References

[1] R. Aragues, J. Cortes, and C. Sagues. Distributed consensus on robot networks for dynamically merging feature-based maps. *IEEE Transactions on Robotics,*, 28(4):840–854, 2012.

[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 M. Hybinette. Social potentials for scalable multirobot formations. In *Proceedings of IEEE International Conference on robotics and automation (ICRA)*, 2000.

[4] T. R. Balch and R. C. Arkin. Motor schema-based formation control for multiagent robot teams. In *Proceedings of the international conference on multiagents systems (ICMAS-95)*, pages 10–16, 1995.

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

[6] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica 14*, pages 1396–1400, 1965.

[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. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 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. P. Ostrowski, and V. Kumar. Modeling and control of formation of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, 2001.

[11] M. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation,*, 17(3):229–241, 2001.

[12] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards-B. Mathematics and Mathematical Physics*, 71B(4):233–240, 1967.

[13] 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, 2002.

[14] A. Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[15] A. Howard, M. J. Matarić, and G. S. Sukhatme. Putting the 'I' in 'Team': an ego-centric approach to cooperative localization. In *In Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 868–874. IEEE, 2003.

[16] L. K. Hwang. Stochastic shortest path algorithm based on lagrangian relaxation. Master's thesis, University of Illinois at Urbana-Champaign, 2010.

[17] G. A. Kaminka, R. Schechter-Glick, and V. Sadov. Using sensor morphology for multi-robot formations. *IEEE Transactions on Robotics*, pages 271–282, 2008.

[18] M. Lemay, F. Michaud, D. Létourneau, and J.-M. Valin. Autonomous initialization of robot formations. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[19] R. P. Loui. Optimal paths in graphs with stochastic or multidimensional weights. Technical Report TR115, Computer Science Department, University of Rochester, 1982.

[20] P. Malbezin, W. Piekarski, and B. H. Thomas. Measuring artoolkit accuracy in long distance tracking experiments. In *The First IEEE International Workshop on Augmented Reality Toolkit*, 2002.

[21] A. Martinelli, F. Pont, and R. Siegwart. Multi-robot localization using relative observations. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2797–2802. IEEE, 2005.

[22] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk. Comprehensive simulation of quadrotor uavs using ros and gazebo. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 400–411. Springer, 2012.

[23] F. Michaud, D. Letourneau, M. Guilbert, and J.-M. Valin. Dynamic robot formations using directional visual perception. In *Proceedings of International Conference on Intelligent Robots and Systems, IEEE/RSJ*, volume 3, pages 2740–2745. IEEE, 2002.

[24] S. Nagavalli, A. Lybarger, L. Luo, N. Chakraborty, and K. Sycara. Aligning coordinate frames in multi-robot systems with relative sensing information. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 388–395. IEEE, 2014.

[25] P. Ogren and N. E. Leonard. Obstacle avoidance in formation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 2492–2497. IEEE, 2003.

[26] L. E. Parker. Designing control laws for cooperative agent teams. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 582–587, Atlanta, GA, 1993.

[27] G. Piovan, I. Shames, B. Fidan, F. Bullo, and B. D. O. Anderson. On frame and orientation localization for relative sensing networks. *Automatica*, 49(1):206–213, 2013.

[28] E. C. S. Y Chiem. Vision-based robot formations with bézier trajectories. In *Proceedings of the Eighth Conference on Intelligent Autonomous Systems (IAS-8)*, volume IOS Press, 2004.

[29] M. Traub, G. A. Kaminka, and N. Agmon. Who goes *there*? using social regret to select a robot to reach a goal. In *Proceedings of the Tenth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2011.

[30] N. Trawny, X. S. Zhou, K. Zhou, and S. I. Roumeliotis. Interrobot transformations in 3D. *IEEE Transactions on Robotics*, 26(2):226–243, April 2010.

[31] A. E. Turgut, H. Çelikkanat, F. Gökçe, and E. Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2-4):97–120, 2008.

[32] R. Wattenhofer, L. Li, P. Bahl, and Y.-M. Wang. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. In *In Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1388–1397. IEEE, 2001.