Global Reconfiguration of a Team of Networked Mobile Robots Among Obstacles

Robert Fitch[1], Alen Alempijevic[2], and Thierry Peynot[1,3]

[1] Australian Centre for Field Robotics, The University of Sydney, Sydney, Australia, rfitch@acfr.usyd.edu.au
[2] University of Technology, Sydney, Australia, Alen.Alempijevic@uts.edu.au
[3] School of Electrical Engineering and Computer Science, Queensland University of Technology, Brisbane, Australia, t.peynot@qut.edu.au

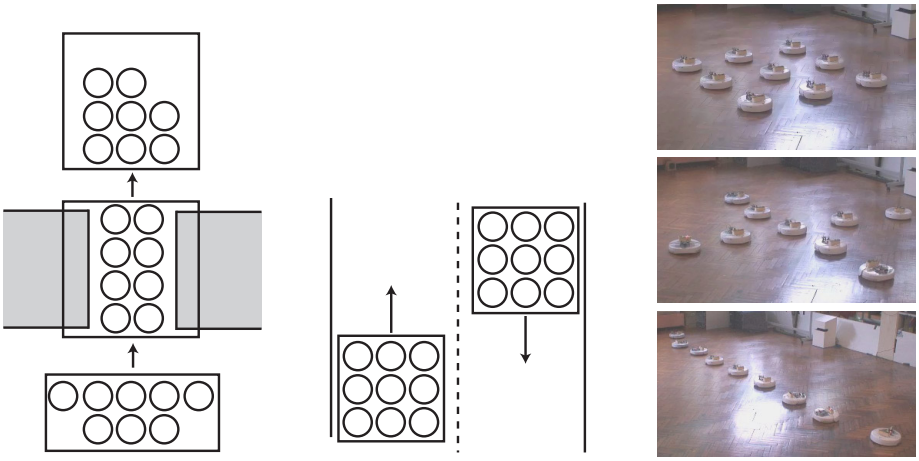
Abstract. This paper presents a full system demonstration of dynamic sensor-based reconfiguration of a networked robot team. Robots sense obstacles in their environment locally and dynamically adapt their global geometric configuration to conform to an abstract goal shape. We present a novel two-layer planning and control algorithm for team reconfiguration that is decentralised and assumes local (neighbour-to-neighbour) communication only. The approach is designed to be resource-efficient and we show experiments using a team of nine mobile robots with modest computation, communication, and sensing. The robots use acoustic beacons for localisation and can sense obstacles in their local neighbourhood using IR sensors. Our results demonstrate globally-specified reconfiguration from local information in a real robot network, and highlight limitations of standard mesh networks in implementing decentralised algorithms.

Keywords: team reconfiguration, networked robots, multi-robot systems, decentralised navigation function

1 Introduction

Large teams of ground or aerial robots pose interesting challenges in motion planning and control. We are interested in the problem of dynamic *team reconfiguration*, where the relative positions of robots in a team are adapted to a specific environment or task. We focus on the case where robots may only communicate with local neighbours and where the team must adapt to obstacles in the environment that are only sensed locally at relatively short range. The goal formation either can be defined loosely as a geometric bounding region (such as a convex polygon), or can be defined exactly by specifying a list of robot positions. During reconfiguration, the team must avoid collisions with obstacles and adapt the resulting goal shape accordingly. Here we present experiments with a system of nine modest mobile robots with onboard localisation, computation, communication, and sensing that demonstrates dynamic reconfiguration.

Reconfiguring teams are interesting for several reasons. One is that this capability can be viewed as a subtask of a variety of multi-robot planning and control problems such as distributed assembly/construction [1] and flocking [2]. In these problems, the goal is to achieve global coordination using local information and environmental sensing, which is a long-standing goal in multi-robot systems research inspired in part by



(a) Squeezing through obstacles

(b) Obstacles may be virtual

(c) Robots reconfiguring from box to line shape

Fig. 1. Representative examples of team reconfiguration. 1a and 1b show application to obstacle avoidance and lane keeping. 1c shows nine mobile robots reconfiguring from a box shape to a line shape (one intermediate configuration shown).

biological control architectures [3]. The formation can dynamically change in response to obstacles, such as to squeeze through a small gap. Another motivation is to implement virtual obstacles that constrain the motion of the team to designated areas such as lanes on a roadway or virtual lanes in a construction, manufacturing or cargo handling situation. Alternatively it could be useful to specify the shape of a formation for a specific task, such as a sensing task, without specifying the position of any individual robot. Finally, team reconfiguration can be used for group navigation. This is particularly useful in the case of many small obstacles or in the human-robot interface context where it is desirable for one user to semiautonomously command n robots. Figure 1 shows diagrammatic examples of two of these motivating applications, along with snapshots of the mobile robots that we consider.

In order to implement reconfiguration for a team of robots with local communication and sensing, we adopt a decentralised *navigation function* approach inspired by earlier work in highly scalable self-reconfiguration in modular robots [4], where collision constraints and global planning are handled by a high-level discrete layer that relies on message-passing through a wireless communication network. Low-level continuous control is simplified and can be implemented using basic steering controllers. The two main algorithmic challenges in this two-layer approach are connectivity preservation and motion planning. Since real communication networks are subject to bandwidth constraints and sometimes unpredictable latency, another challenge is how to cope with potentially high-latency message delivery.

This paper presents a novel adaptation of the algorithm in [4] to the mobile robot case where mechanical inter-module connections are replaced by communication links

between mobile robots, and low-level steering controllers are added to form the two-layer architecture. The main assumptions are that robot positions are structured as a 2D lattice, the goal shape is known to all robots (or can be communicated via message-passing), relative localisation is available, neighbour-to-neighbour communication is available, and robots have short-range obstacle sensing capability.

The novelty of our approach is its computational and communications efficiency and suitability for implementation in a wireless network; asynchronous dynamic programming does not require low-latency message passing. This benefit is significant because existing approaches such as [5] rely on continuous global state estimation that can easily overwhelm the capacity of non-infrastructure-based communication networks.

Our approach is also beneficial in its efficiency in planning around many small obstacles, squeezing through small gaps, and for splitting or merging operations. Recent approaches based on task allocation perform reconfiguration by explicitly solving the costly graph bipartite matching problem [6]. Our approach solves this problem through implicit coordination at lower computational cost.

In this paper, we present a full system demonstration of dynamic reconfiguration. We present results from experimental validation with nine mobile robots that operate in an indoor environment, localise using an extended Kalman filter, and sense nearby obstacles with IR sensors. Our experiments validate the approach and show that hardware implementation is straightforward with minimal computation, memory, and sensing resources.

Another important contribution of the paper is to experimentally highlight the performance limitations of standard mesh networks in implementing decentralised algorithms. When multiple robots communicate simultaneously, mutual interference occurs. This effect is evident in our experimental analysis and has significant implications for networked multi-robot systems. In particular the total number of robots, even with neighbour-to-neighbour communication, is limited by the mutual interference properties of the underlying communication system.

2 Related Work

Many algorithms have been proposed that address the related *reconfiguration planning* problem for self-reconfiguring modular robots [7–9]. The algorithm we present in this paper is based on the algorithm presented in [4], but is extended to the mobile robot case with the addition of the low-level control layer as well as hardware experiments that demonstrate its implementation in a team of modest robots.

Team reconfiguration as a control problem is defined in [10]. A decentralised solution based on a square-lattice representation is presented by Miklic et al. [11], where robots are assigned to discrete target positions and use a collision-avoidance scheme to prevent inter-robot collisions. In contrast, our approach plans collision-free paths in the discrete layer and eliminates the need for collision avoidance within the continuous controllers. Other work that uses a variable geometric representation of the goal shape includes [5], using distributed control laws with provable guarantees. The key difference is that our approach does not rely on any centralised process to track the current global shape of the formation. Our algorithm never explicitly computes the global shape

of the formation and instead uses decentralised dynamic programming for parallel path planning.

There is a large volume of research in control theory for distributed formation control of teams of mobile robots. A good survey is presented by Murray [12]. Other representative overviews can be found in [13] and [14]. The approach of using a virtual (fixed) structure is presented in [15]. Decentralised navigation functions are investigated in [16] for the problem of stabilising small (three- or four-robot) regular formations.

Our approach, however, is derived from a planning perspective and is more closely related to recent work in formal methods, where a discrete plan is implemented by continuous controllers [17]. Although we do not use formal methods to build discrete plans, we do have a two-layer control structure. Dynamic programming is used as the high-level discrete planner and simple steering controllers implement the low-level continuous control.

3 Dynamic Reconfiguration Algorithm

We formulate the dynamic reconfiguration problem for robot teams in terms of graph reconfiguration. The problem is to transform one formation into another through a sequence of primitive moves. A *formation* is defined as a graph where each node represents a robot and nodes are embedded in a 2D lattice (i.e. nodes represent physical positions constrained to lattice positions). Edges connect adjacent nodes (i.e. a 4-connected grid). A *goal* formation is represented geometrically as a bounding region such as a bounding box, where the goal is considered reached when all robot positions lie within the bounding region. Other representations are possible, including a complete list of node positions. The goal is generated manually by a user or autonomously by a separate decentralised process. A *move* is a three-step process whereby a robot 1) removes its node from the graph, 2) physically moves to a new lattice position, and 3) re-inserts itself into the graph. The graph must remain connected following node removal. The main assumptions of our approach are that node position is restricted to a planar lattice and goal shapes do not have holes.

Robot location is specified with respect to a formation-fixed coordinate frame. The origin of this frame is assigned arbitrarily to some lattice position. As robots move through the lattice, their positions in the formation-fixed coordinate frame change. This coordinate frame is considered to be distinct from the world frame. This formalism allows for the formation itself to rotate and translate. In other words, if the robots move as a group *without* reconfiguring, then the formation frame is considered to move with respect to the world frame. The two coordinate frames are illustrated in Fig. 2a. The work in this paper concentrates on reconfiguration, and does not involve any cases of moving without translating; this situation is left for future work.

A fixed set of motion primitives is defined over lattice positions. These primitives are shown in Fig. 2. There are twelve possible motions corresponding to an 8-connected grid: up, down, left, and right lateral translations, plus two forms of diagonal translations. The diagonal translations are defined for both “elbow-up” and “elbow-down” versions in order to allow a robot to achieve the diagonal translation without colliding with a lattice neighbour. At any given time, only a subset of moves will be valid

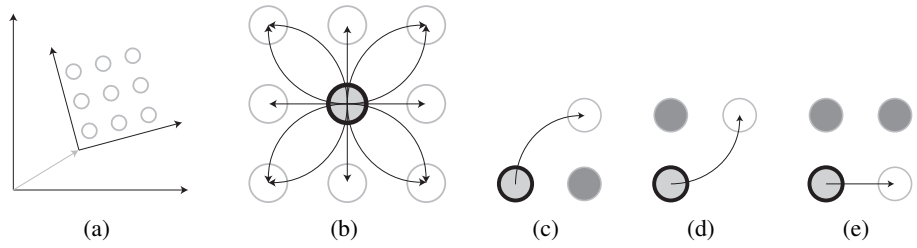


Fig. 2. Algorithm setup. (a) The coordinate frame attached to the formation is separate from the world frame. (b) The full set of motion primitives. Motion primitives are derived from two canonical cases, convex translation (c,d) and lateral translation (e). Unfilled circles represent free positions, grey circles represent positions occupied by robots or obstacles. Arrows represent motion of a robot into a free position.

for a particular robot determined by which adjacent lattice positions are occupied by other robots and obstacles. A move is only valid if it can be achieved in a collision-free manner, which is determined by examining the local neighbourhood. The planning algorithm ensures that the appropriate neighbour robots remain static during a move, described in the following subsections.

3.1 Algorithm Overview

The complete algorithm comprises three distinct parts: 1) a connectivity preserving algorithm, 2) a motion planning algorithm, and 3) a top-level state machine. Connectivity and motion planning are decentralised algorithms based on the message-passing model. The state machine coordinates the other two algorithms and runs locally on each robot. The algorithm can be viewed as a two-layer control structure, where these three components implement the high-level discrete layer and a low-level continuous controller executes the commanded motion primitives. Note that all components execute locally on each robot, and that each robot only has access to information derived from local sensing, communications from neighbours, and its own localisation system.

Reconfiguration begins when a goal configuration is supplied to the system. The goal is specified as a bounding region, represented in this paper as a bounding box. The goal is communicated to all robots through local message-passing. When a goal is received, the state machine initiates motion planning. All robots initiate this in parallel. Immediately, all robots whose positions are outside the goal region attempt to move. In order to move, a robot must first guarantee connectivity preservation by executing the connectivity algorithm. If this algorithm succeeds, the robot then decides on a move based on information computed by the motion planning algorithm, moves, and iterates. All robots execute asynchronously and in parallel until all are within the goal.

3.2 Connectivity

The *connectivity check* algorithm is presented in previous work and fully described in [4, 18]. We provide a summary here for convenience. The algorithm guarantees that

the robot graph remains connected during simultaneous moves. For any given node, the approach is to search (using message passing) for a set of paths in the graph that connect each pair of neighbours. Nodes along these paths are then locked. By definition, this condition is sufficient for preserving graph connectivity. All nodes execute their search asynchronously and in parallel. Locks are non-exclusive and can be shared between multiple moving nodes. Locks are cleared after a node is reinserted in the graph. Deadlock can occur if two nodes attempt to lock each other, but this situation is easily prevented with arbitrary prioritisation such as module ID.

3.3 Motion Planning

The motion planning algorithm was originally presented and analysed in [4] in the context of modular robots. Here we extend this algorithm for the case of mobile robots. The main extension is to substitute connection/disconnection actions with the motion primitives described earlier in this section. We summarise the algorithm here for completeness and describe the extensions.

Our motion planning algorithm uses decentralised dynamic programming (DP) to compute a navigation function [19]. This function encodes shortest paths from each robot to its closest position in the goal formation. Because the graph structure continually changes, the navigation function is updated in response to each robot move. This replanning is similar in character to the idea of *prioritized sweeping* in reinforcement learning.

Figure 3 shows two example configurations with the value function included. Values are assigned to empty lattice positions adjacent to robots. It is important to note that each *robot* stores values for its *adjacent lattice positions*. Values are interpreted as follows. If a robot were to occupy a given lattice position, the assigned value is a measure of how "good" it is to occupy this position. Since this is a motion planning application, the value indicates distance to goal measured in number of primitive moves. It is possible to compute other distance functions that consider other quantities of interest but in this case we are interested only in minimising move count.

The basic operation in dynamic programming is to update the value of one state based on the values of reachable states. Two states (lattice positions) are reachable if there exists a motion primitive that connects these states. We implement these dynamic programming updates with message-passing between neighbour robots. This follows from the definition of our motion primitives. Reachable states are either stored within the memory of a single robot (in the convex translation case) or adjacent robots (in the lateral translation case).

When a robot moves, the value function must be maintained. Figure 4 shows an example. After a move, the value of states that have changed are reinitialised to a pessimistic value (negative infinity). Then the mobile robot sends update messages to its neighbours. Updates propagate until values no longer change; if a robot receives an update message and does not change any values, the message is not propagated further. Eventually, the value function converges to a global optimum.

When a robot is ready to move, it must query the value function. The query operation involves sending a message to a neighbour robot. For convex translations, the neighbour can return the value directly. For lateral translations, the neighbour robot

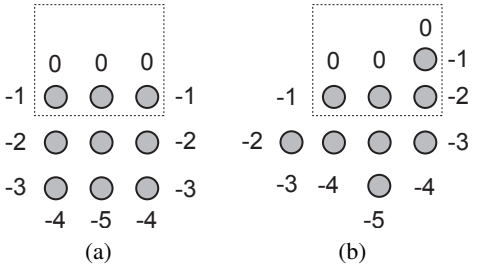


Fig. 3. The value function computed by dynamic programming. Two examples shown. Values are assigned to states, where a state is an empty lattice position adjacent to a robot, and its value is stored in the memory of the adjacent robot. Only those states adjacent to robots are considered valid. Value 0 is assigned to the goal region.

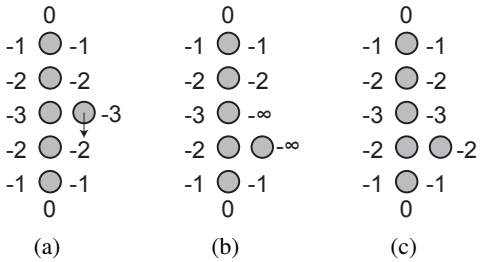


Fig. 4. An example of updating the value function. After a move, unknown values are reinitialised to negative infinity. The decentralised dynamic programming algorithm then computes the correct values via local message passing, and the global navigation function is thus maintained.

must then query its neighbour. This process is illustrated in Fig. 5. All neighbours are queried in this manner and the best value is chosen. Ties are broken by choosing randomly from the set of equal-valued moves.

3.4 Analysis

Correctness of the connectivity checking algorithm is proved in [4, 18] but we provide a proof sketch here for convenience. Correctness follows from the definition of connectivity in an undirected graph. The algorithm succeeds when all pairs of neighbours are connected by a path. Therefore, the robot that initiates the algorithm is free to move. The graph remains connected because for each pair of nodes, a connecting path exists. Running time can be analysed by counting the number of messages passed. The time to process a single message is proportional to the number of neighbours and can be considered constant. The number of messages is bounded by the maximum depth of iterative deepening search. In the worst case (a loop configuration), this depth is equal to the total number of robots. However in the special case where a robot has only a single neighbour, the number of messages is constant. In practice this algorithm has been observed to exhibit near-constant time performance for large (million module) systems [4].

Likewise, properties of the motion planning algorithm are given in [4]. We again provide a proof sketch for convenience. Dynamic programming is known to converge

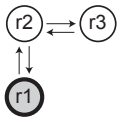


Fig. 5. Querying the value function. Robot $r1$ sends a message to $r2$, which then must query $r3$. The result is sent back to $r1$. If $r3$ was not present, $r2$ could return the maximum value directly.

in polynomial time in the number of states [20], even with asynchronous updates. Our decentralised formulation corresponds to the case of asynchronous updates. The number of states in our case is linear in the number of robots since each robot is responsible for a constant number of adjacent lattice positions. This shows that if a path exists for a robot to enter the goal, the algorithm will find this path efficiently. Such a path is guaranteed to exist and has been shown previously [21]. Therefore, each robot eventually finds a path to a goal position and the system is guaranteed to converge to the goal shape. The time for a robot to choose an action, once connectivity is established, is constant since this operation queries the navigation function using a constant-time table lookup.

4 Experimental system

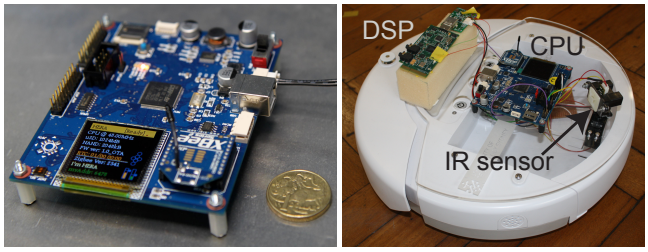
This section describes the robots used in our system and our implementation of algorithms from Sec. 3. We implemented our team reconfiguration algorithm using an experimental system consisting of nine *iRobot Create* mobile bases controlled by custom electronics, shown in Fig. 6, and a system of three acoustic beacons used for localisation. The Create is a differential-drive base with onboard power and reasonable-quality wheel encoders.

4.1 Mobile Platform

The Create is connected via a serial interface to our custom electronics. The embedded platform design is based on the ST Microelectronics ARM Cortex-M3 CPU, a 32-bit processor with a clock speed of 72MHz, 512kB flash memory, 64kB of RAM, and 12-bit ADC. A micro-SD card slot is included and is intended for storage of large files (currently up to 16GB) such as data logs. For debugging purposes, the platform includes a 1.5”, full-color, 128x128 pixel OLED (Organic Light Emitting Diode) display. Wireless communication is implemented with off-the-shelf, 2.4GHz ZigBee hardware modules from Digi.

The software system uses a real-time operating system (RTOS), the Crossworks Tasking Library (CTL). CTL is a pre-emptive multitasking RTOS with concurrency support. We developed a custom over-the-air update system that allows us to reprogram software using the wireless communication link.

Odometry-based localisation is performed by integrating wheel encoder data provided by the Create. The system also supports acoustic localisation through a peripheral electronic subsystem that consists of a MEMS microphone (Knowles Acoustics



(a) (b)

Fig. 6. Robot hardware used in our experiments: (a) custom electronics subsystems, (b) iRobot Create base with added components for sensing, communication, localisation, and control.

SPM0404HE5H) and digital signal processor (Texas Instruments TMS-320VC5505). The Microphone signal is amplified 20x and fed into the ADC of the DSP at 8-bit resolution. An MLS sequence acquired from speakers is correlated on the DSP in real time to provide localisation with an accuracy of 4.25cm (Update rate is 2Hz). The DSP computes position estimates to be sent out to the host via serial port. Odometry and acoustic localisation are fused into a single EKF estimate.

Range sensing is supported by a scanning infrared sensor. An emitter/detector pair is mounted on a stepper motor controlled by the main processor.

4.2 Non-Holonomic Control (Motion Primitives)

Motion primitives (defined earlier in Fig.2) are implemented using simple steering controllers. Note that all motion primitives are derived from two canonical cases (lateral translation and convex translation). During execution, relevant lattice positions are guaranteed (by the algorithm) to be free of obstacles (and other robots). Therefore a steering controller (as opposed to an obstacle avoidance algorithm) is sufficient.

4.3 Beacon Hardware

In order to localise the platform we have placed three standard audio speakers in the environment. A Microcontroller Board (STM32) and the speaker driver hardware generate a Maximum Length Sequence (MLS) with length = 341/speaker, M=11 at 8KHz.

4.4 Extended Kalman Filter Localization

The localisation of each robot is performed using onboard computation only by observing beacons placed in the environment. An extended Kalman filter (EKF) utilises two time difference of arrival (TDOA) measurements as observations acquired from speakers situated in known locations in the environment. In this section, the EKF localisation algorithm is presented in order to clearly describe our experimental results.

The state vector is denoted as $X = (x_r, y_r, \phi)$. Variables x_r, y_r denote the global robot positions and ϕ is the robot orientation.

Suppose at time k, after the update, the estimate of the state vector is $\hat{X}(k|k)$ and the covariance matrix of the estimation error is $P(k|k)$. The prediction step is given by

$$\begin{aligned} \hat{X}(k+1|k) &= F(\hat{X}(k|k), u(k), k) \\ P(k+1|k) &= \nabla F_X P(k|k) \nabla F_X^T + \nabla F_U Q \nabla F_U^T, \end{aligned} \tag{1}$$

where F is non-linear state transition function at time k, and $u(k)$ is the system input at time k.

The robot process model considered in this paper is

$$\begin{bmatrix} x_r(k+1) \\ y_r(k+1) \\ \phi(k+1) \end{bmatrix} = \begin{bmatrix} x_r(k) + d\cos[\phi(k+1)] \\ y_r(k) + d\sin[\phi(k+1)] \\ \phi(k) + \theta \end{bmatrix}, \tag{2}$$

where d, θ are the system inputs $u(k)$ (the distance travelled and the change in angle between successive readings).

For the system described by equation (2) ∇F_X and ∇F_U are Jacobians of F in (2) with respect to the robot pose X and the control (d, θ), respectively. They are evaluated at the current estimate $\hat{X}(k|k)$ while Q is covariance matrix of the measurement noise. We assume the measurement noise is proportional to the control inputs d and θ.

$$\nabla F_X = \begin{bmatrix} 1 & 0 & -d\sin\phi \\ 0 & 1 & d\cos\phi \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$\nabla F_U = \begin{bmatrix} \cos\phi & -d\sin\phi \\ \sin\phi & d\cos\phi \\ 0 & 1 \end{bmatrix} \tag{4}$$

$$Q = \begin{bmatrix} (d\delta_d)^2 & 0 \\ 0 & (\theta\delta_\theta)^2 \end{bmatrix} \tag{5}$$

At time $k+1$, the TDOA measurements from the beacon/microphone system (detailed in section 4) are used to calculate the differences in radius between the platform and the speakers $R_{21} = R_2 - R_1$ and $R_{31} = R_3 - R_1$. The global location of the speakers is known *a priori*, thus, the microphone location $X_s = (x_s, y_s)$ is calculated based on the intersections of hyperbolic curves [22]. This system of quadratic equations (refer Eq. 10 from [22]) provides two possible solutions, restricting the receiver to lie within the bounds of the speakers provides a unique solution.

The observation model $z_i(k+1) = H_i(X(k+1))$ can be written as

$$z_i(k+1) = \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} x_r(k+1) + R_{off} cos(\theta_i) \\ y_r(k+1) + R_{off} sin(\theta_i) \end{bmatrix} \tag{6}$$

where $\theta_i = \phi(k+1) + \phi_{off}$ and R_{off},ϕ_{off} are the distance and angular offset between the microphone and center of the platform. ∇H_i is the Jacobian of function H_i evaluated at the current estimate $\hat{X}(k+1|k)$:

$$\nabla H_i = \begin{bmatrix} 1 & 0 & -R_{off} sin(\theta_i) \\ 0 & 1 & R_{off} cos(\theta_i) \end{bmatrix}. \tag{7}$$

The innovation $\mu(k+1)$ and innovation covariance $S(k+1)$ can be calculated as

$$\begin{aligned} \mu(k+1) &= z_i(k+1) - \hat{X}(k+1|k) \\ S(k+1) &= \nabla H_i P(k+1|k) \nabla H_i^T + R_{x_i y_i}. \end{aligned} \tag{8}$$

The validity of observations is checked using the Mahalanobis distance γ, in which the measurement is considered acceptable if it falls within a 95% χ^2 level of confidence

$$\gamma = \mu(k+1) S^{-1} \mu^T(k+1). \tag{9}$$

The estimate of the state vector can now be updated using

$$\hat{X}(k+1|k+1) = \hat{X}(k+1|k) + W(k+1)\mu(k+1), \tag{10}$$

where the Kalman gain matrix $W(k+1)$ is

$$W(k+1) = P(k+1|k) \nabla H_i^T S^{-1}(k+1). \tag{11}$$

The state covariance after the observation update is

$$P(k+1|k+1) = P(k+1|k) - W(k+1) S^T(k+1) W^T(k+1). \tag{12}$$

5 Experiments

We performed experiments both with and without obstacles. The two goals of these experiments were to: 1) validate assumptions on computation, communication, and memory requirements, and 2) validate the algorithm in a complete hardware system. We implemented the motion primitives using a simple steering algorithm that uses proportional control on range and bearing to destination lattice position. At the completion of a move, the robot scans its local neighbourhood to detect obstacles. Our test arena is 7 m long and 5 m wide. We tested a total of five scenarios, three without obstacles and two with a single obstacle. These scenarios are defined in Fig 7. The two scenarios with

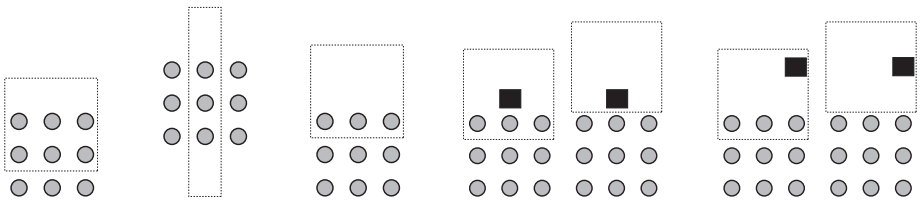


Fig. 7. Five reconfiguration scenarios considered. The wireframe box shows the goal configuration, and the circles indicate initial robot positions. The last two scenarios require two reconfigurations each.

obstacles required two reconfigurations each. In these cases, a user manually initiated the next goal when the first reconfiguration was complete.

In our experimental implementation, the formation-fixed coordinate frame was defined by choosing an arbitrary robot as the origin and initialising the position of other robots using (flood-fill) message passing. Goal positions were communicated to an arbitrary robot (by the user) and communicated to other robots likewise.

5.1 Increasing the Number of Robots

The first experiment is the SINGLE-SQUARE scenario. We tested this scenario with teams of four, six, eight, and nine robots and performed five trials each. Because our algorithms are implemented as simple message handling routines, the performance measure of interest is the total number of messages and the time taken to transmit them. We report average message count and latency (clock time) in Fig. 8. The total message count averaged across the five trials in all cases is less than 100 for planning (DP) and less than 50 for connectivity maintenance. The dip in message latency for nine robots shows that the performance of the connectivity algorithm improves with denser configurations (maximum search depth is shorter in dense configurations).

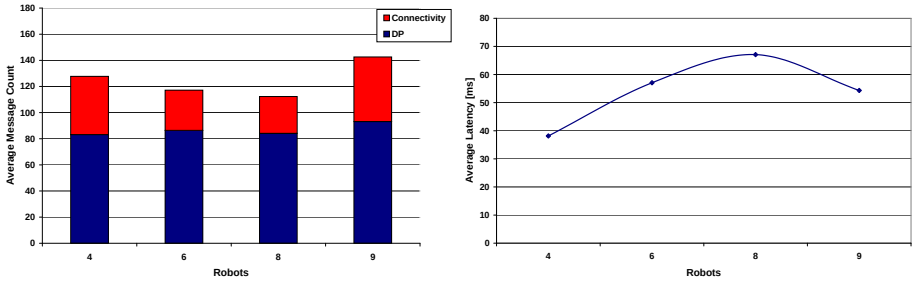
These results illustrate the computational and communications efficiency of our approach. The algorithm requires relatively few messages as compared to techniques that require continuous tracking of the global shape of the team. Our approach avoids this by considering only the local neighbourhood around each robot, yet plans globally through decentralised dynamic programming. The computational requirement to handle each message is very low (as discussed in Sec. 3) and is easily implemented using the modest resources onboard our experimental platforms.

5.2 No Obstacles

We tested two other reconfiguration scenarios without obstacles, NINE-TO-LINE and DOUBLE-SQUARE. Figure 9 shows the path taken by each robot for both scenarios. Paths shown are derived from EKF-estimated location. These paths show the expected behaviour, where each robot takes a minimal number of primitive moves to reach a position in the goal configuration while avoiding other robots. Figures 10a and 10b evaluate localisation performance for one randomly selected robot in the NINE-TO-LINE scenario, and show the EKF-estimate compared to the dead-reckoning estimate.

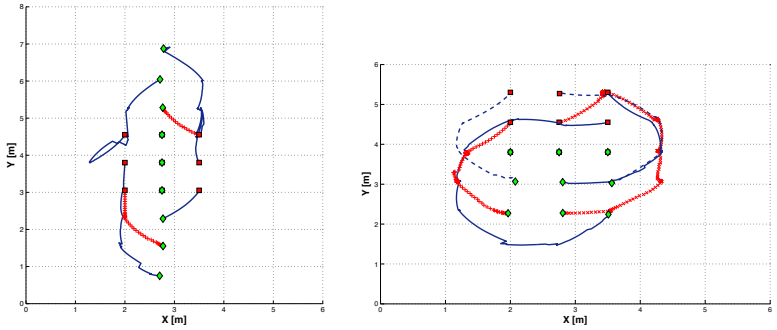
5.3 Obstacles

We performed two experiments with a box-shaped obstacle with dimensions 300 mm x 250 mm x 130 mm. Figures 11 and 12 show video snapshots from these experiments, and Fig. 13 shows message counts and latencies as compared to obstacle-free scenarios. Because the obstacle experiments are longer in terms of total number of moves, the total message count increases. However, the average number of messages per move decreases because these experiments exploit the best-case performance of the algorithms. This arises when a single robot executes a series of moves in sequence. At each move,



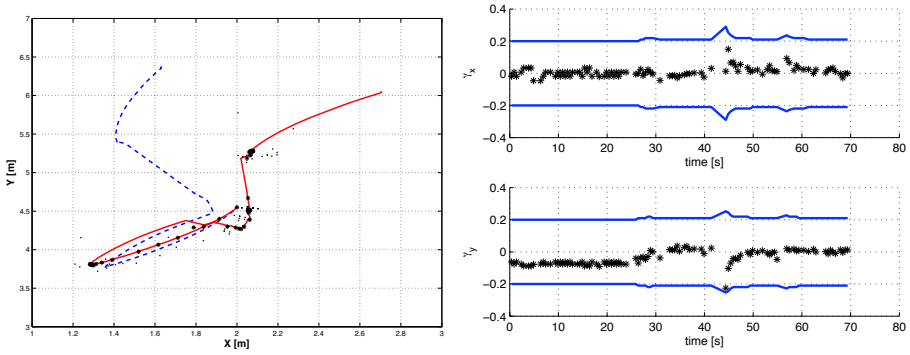
(a) Average message count (b) Average message latency

Fig. 8. Average message count by message type (a) and average message latency (b) for a varied number of robots in the SINGLE-SQUARE scenario.



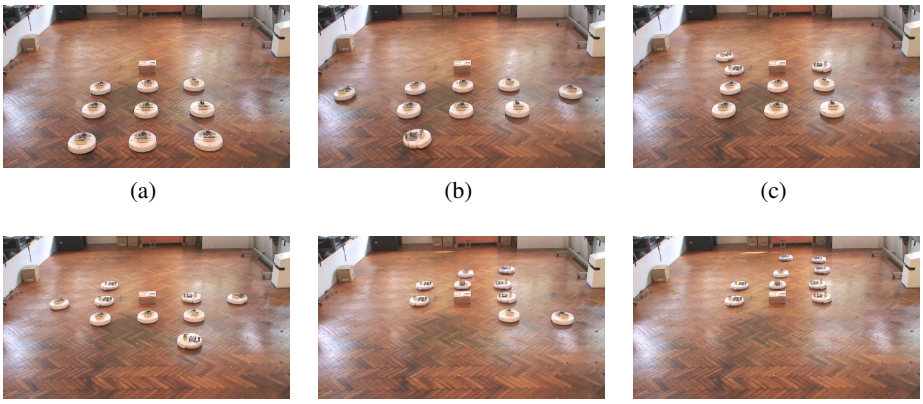
(a) (b)

Fig. 9. Paths taken by nine robots in the NINE-TO-LINE (a) and DOUBLE-SQUARE (b) scenarios. Red squares indicate start positions, green diamonds indicate end positions, and lines (solid, dotted, line with crosses) show complete paths.



(a) (b)

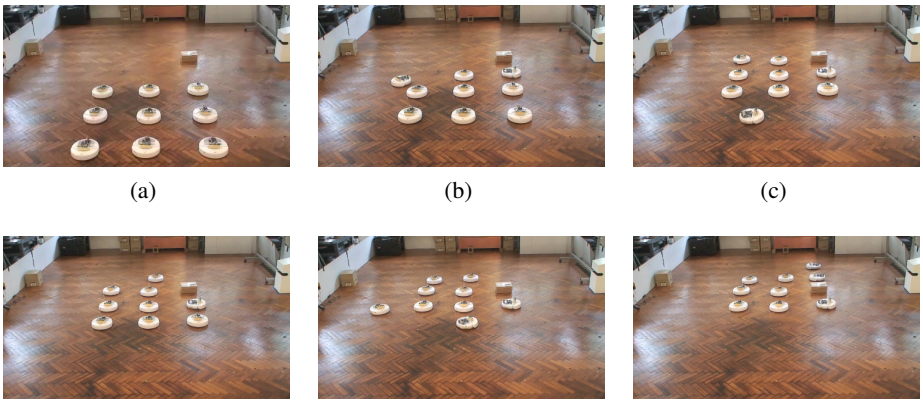
Fig. 10. (a) Localisation of a single robot from the nine robots in the NINE-TO-LINE scenario. The dashed line shows the dead reckoning result. The full line shows the EKF solution. Black dots indicate observations from the acoustic localisation system. Black stars show valid observations used in the EKF estimator. (b) Localisation of a single robot. Black stars indicate the Mahalanobis distance of the EKF localisation estimates that fall within 2 σ bounds shown in blue.



(a) (b) (c)

(d) (e) (f)

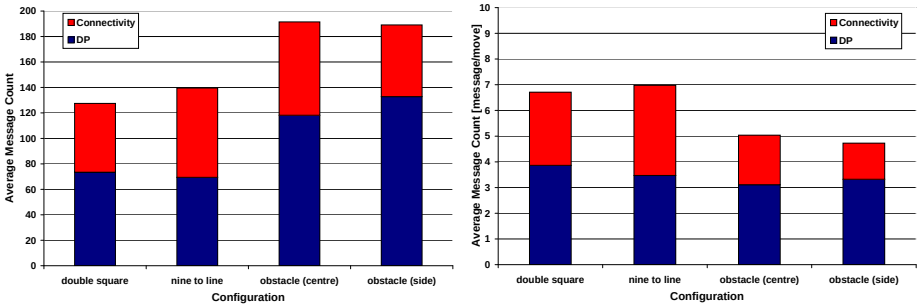
Fig. 11. Video snapshots from the OBSTACLE-1 (CENTER) reconfiguration experiment.



(a) (b) (c)

(d) (e) (f)

Fig. 12. Video snapshots from the OBSTACLE-2 (SIDE) reconfiguration experiment.



(a) Average message count (b) Average message count per move

Fig. 13. Average message count and latency for various reconfiguration scenarios.

the robot has a single neighbour and therefore connectivity messages are minimal. DP messages are also minimal because few values need to change. Average message count per move in all cases is less than 10.

6 Discussion and Future Work

We have experimentally demonstrated dynamic reconfiguration with a nine-robot team and relatively low-bandwidth mesh network. Our algorithms are efficient and have previously been shown to scale to very large modular systems (millions of modules). Here we provide a full system implementation and demonstration in the context of mobile robots. This system shows that reconfiguration into a class of globally defined goal shapes can be implemented using local communication and sensing for robots with modest resources.

One important lesson learned is that neighbour-to-neighbour communication must consider properties of the underlying communication hardware. Mesh networks are fundamentally limited in their ability to support large teams, even with efficient algorithms, due to mutual interference that arises when many nodes attempt to transmit simultaneously. This problem can be addressed by a new type of multi-radio multi-channel network that is immune from mutual interference in neighbour-to-neighbour communication [23]. Our algorithm could exploit such a network for larger teams because all communication is local.

Working with a full hardware system allows us to make interesting observations about the interaction between planning and communication. The communication requirement of our algorithm, although large in terms of number of small messages, is asynchronous. Asynchrony means that the algorithm does not require low-latency communication. Because dynamic programming planning is designed to handle stochasticity, robots can make a number of moves with a "stale" navigation function. Non-optimal moves can simply be viewed as stochastic actions.

The experiments in this paper validate only the most basic capabilities of our approach. Important questions for future work include fully integrating team reconfiguration with navigation, where the team coordinate frame translates simultaneously with reconfiguration operations.

Acknowledgments

This work was supported in part by the Australian Centre for Field Robotics (ACFR) and the NSW State Government.

References

1. Stewart, R.L., Russell, R.A.: A distributed feedback mechanism to regulate wall construction by a robotic swarm. Adapt. Behav. **14**(1) (2006) 21–51
2. Ferrante, E., Turgut, A.E., Huepe, C., Stranieri, A., Pinciroli, C., Dorigo, M.: Self-organized flocking with a mobile robot swarm: a novel motion control method. Adapt. Behav. **20**(6) (2012) 460–477

3. Theraulaz, G., Bonabeau, E.: Modelling the collective building of complex architectures in social insects with lattice swarms. J. Theor. Biol. **177**(4) (1995) 381–400
4. Fitch, R., Butler, Z.: Million module march: Scalable locomotion for large self-reconfiguring robots. Int. J. Rob. Res. **27**(3-4) (2008) 331–343
5. Michael, N., Kumar, V.: Planning and control of ensembles of robots with nonholonomic constraints. Int. J. Rob. Res. **28**(8) (August 2009) 962–975
6. Liu, L., Shell, D.A.: Physically routing robots in a multi-robot network: Flexibility through a three-dimensional matching graph. Int. J. Rob. Res. **32**(12) (2013) 1475–1494
7. Yim, M., Shen, W.M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., Chirikjian, G.S.: Modular self-reconfigurable robot systems: Challenges and opportunties for the future. IEEE Robot. Automat. Mag. **14**(1) (March 2007) 43–52
8. Fitch, R., Rus, D.: Self-reconfiguring robots in the USA. Journal of the Robotics Society of Japan **21**(8) (Nov. 2003) 4–10
9. Butler, Z., Fitch, R., Rus, D.: Experiments in distributed control for modular robots. In: Experimental Robotics VIII. Springer (2003) 307–316
10. Zelinski, S., Koo, T.J., Sastry, S.: Optimization-based formation reconfiguration planning for autonomous vehicles. In: Proc. of IEEE/ICRA. (2003) 3758–3763
11. Miklic, D., Bogdan, S., Fierro, R.: Decentralized grid-based algorithms for formation reconfiguration and synchronization. In: Proc. of IEEE/ICRA. (2010) 4463–4468
12. Murray, R.: Recent research in cooperative control of multivehicle systems. J. Dyn. Sys., Meas., Control **129**(5) (2007) 571–583
13. Balch, T., Arkin, R.: Behavior-based formation control for multirobot teams. IEEE Trans. Robot. Automat. **14**(6) (Dec 1998) 926–939
14. Das, A.K., Fierro, R., Kumar, V., Ostrowski, J.P., Spletzer, J., Taylor, C.J.: A vision-based formation control framework. IEEE Trans. Robot. Automat. **18**(5) (2002) 813–825
15. Olfati-Saber, R., Murray, R.: Distributed cooperative control of multiple vehicle formations using structural potential functions. In: Proc. of IFAC World Congress. (2002)
16. Tanner, H., Kumar, A.: Formation stabilization of multiple agents using decentralized navigation functions. In: Proc. of Robotics: Science and Systems. (2005)
17. Kress-Gazit, H., Wongpiromsarn, T., Topcu, U.: Correct, reactive, high-level robot control. IEEE Rob. Autom. Mag. **18**(3) (2011) 65–74
18. Fitch, R., Lal, R.: Experiments with a ZigBee wireless communication system for self-reconfiguring modular robots. In: Proc. of IEEE ICRA. (2009) 1947–1952
19. Sutton, R.S., Barto, A.G.: Introduction to reinforcement learning. MIT Press (1998)
20. Littman, M., Dean, T., Kaelbling, L.P.: On the complexity of solving markov decision problems. In: Proc. of UAI. (1995) 394–402
21. Dumitrescu, A., Pach, J.: Pushing squares around. In: Proc. of the Symposium on Computational Geometry. (2004) 116–123
22. Chan, Y.T., Ho, K.C.: A simple and efficient estimator for hyperbolic location. IEEE Trans. Signal Processing **42**(8) (1994) 1905 –1915
23. Kuo, V.: Enabling Parallel Wireless Communication in Mobile Robot Teams. PhD thesis, The University of Sydney (2013)