USC Robotics Research Labs
University of Southern California
Los Angeles, USA

LOST: Localization-Space Trails for Robot Teams

Richard T. Vaughan, Kasper Støy, Gaurav S. Sukhatme, Maja J. Matarić

vaughan@robotics.usc.edu

Technical Report IRIS-01-401
http://iris.usc.edu/irislib

April 3, 2001

Contents

Abstract

We describe an algorithm that enables a team of robots to navigate between places of interest in an initially unknown environment using a trail of landmarks. The landmarks are not physical; they are waypoint coordinates generated on-line by each robot and shared with team-mates. Waypoints are specified in each robot's local coordinate system, and contain references to features in the world that are relevant to the team's task and common to all robots. Using these task-level references, robots can share waypoints without maintaining a global coordinate system.

The algorithm is tested in a series of real-world multi-robot experiments. The results demonstrate that the method (i) copes with accumulating odometry error; (ii) is robust to the failure of individual robots; (iii) converges to the best route discovered by any robot in the team. In one experiment, a team of four autonomous mobile robots performs a resource transportation task in our uninstrumented office building. Despite significant divergence of their local coordinate systems, the robots are able to share waypoints, forming and following a common trail between two fixed locations for more than 3 hours, traveling a total of 8.2km (5.1 miles) before running out of power.

1 Introduction

Ants form supply columns to relocate valuable items through complex, dynamic environments. Their remarkable effectiveness is due to a highly robust strategy of local interactions among a large number of autonomous agents. The chemical trails formed by ants along their supply routes are robust with respect to changes in the environment and to the 'failure' of many individual ants [19, 32, 17]. Self-organized in this way, the ants are more effective and efficient than they would be as individuals; an ant trail is 'more than the sum of its parts'.

These properties are attractive to agent designers in general and to robot builders in particular who can see immediate applications for robot supply columns in hazardous or tedious environments. We employ ant-inspired trail following as a means to perform cooperative path-finding in general, and resource transportation in particular.

In this paper we describe a method for generating and using trails of landmarks to navigate between places of interest. The landmarks are not physical; they are waypoint coordinates generated on-line by each robot and shared with team-mates. Waypoints are specified with reference to features in the world that are relevant to the team's task and common to all robots. Using such task-level features as landmarks avoids the need for recognizing and naming an external set of objects that are otherwise irrelevant to the task. Using these common landmarks, each robot can transform waypoint coordinates into its local reference frame, avoiding the cost of maintaining a fixed global coordinate system. Trails converge on 'good' routes, where 'good' is specified by a distance metric which can be defined to suit the application.

We present simple data structures that combine landmarks into useful trails, and algorithms for generating, sharing and using the trails. We call the method 'Localization-Space Trails', abbreviated to 'LOST'. The means for generating and using trails are loosely analogous to ant trail following. Instead of directly modifying their physical environment like ants, or performing an information-transmitting dance like bees [45], our robots communicate localization space landmarks over a wireless network. This communication requires very little bandwidth and is well within the capabilities of current networks.

The method is applied to an example 'resource transportation' task, in which multiple autonomous robots find and repeatedly traverse a path in an unknown environment between a known 'home' and a supply of resource at an initially unknown position. This is not a toy or contrived system; our experiments are performed with teams of three and four real robots in an unaltered, uninstrumented office environment. The robots use standard sensors, onboard computation and have no prior information about the world. Running the final version of LOST described below, the robots shuttle between home and goal for hours without human intervention.

1.1 Task

A team of robots works to transport resource in an unknown environment. Robots start from a home position and search for a supply of resources. On reaching the source, they receive a unit of resource and must return home with it, then return to fetch more resource repeatedly for the length of a trial.

Achieving this task reliably with robots would meet a real-world need. For example a factory may require a supply of widgets manufactured at position A to be transferred by robot to position B. In a Flexible Manufacturing System, the layout of the factory floor is expected to change over time. There will also be occasional robot breakdowns, perhaps blocking the supply route. There may be considerable benefit from a team of robots that can automatically find a new route without an up-to-date map, or exploit several routes in parallel. There is also a military need for supplies (medicine, food, ammunition) to be transported over hazardous and uncertain terrain. The start location and the existence of one or more goal locations may be the only known features, and these may be a few meters or several kilometers apart. Perhaps a future colony on Mars might want to to retrieve resources from a distant autonomous mining robot. A single future nano-robot could move only a tiny payload on its own; like ants, large populations of robots will be required to move a significant amount of resource in a reasonable time.

In general, establishing a reliable robot supply column robust with respect to loss of individual robots could be very valuable. We chose this task as an ideal example of the application of multiple mobile robots.

1.2 Contributions

This paper describes Localization-Space trails for robot teams. The main contributions of this work are:

1. We show that a team of robots can reliably create, share and follow virtual trails, specified as lists of waypoints in localization-space coordinates, between two locations in the real world. The method generalizes to N locations, and provides an alternative to mapping and planning strategies.

2. By referring to task-level features common to all robots, waypoints can be shared between unrelated coordinate systems. The robots require no external frame of reference other than the events that characterize their task. By sharing and manipulating a common minimal model of the world, anchored by task features, we exploit feedback through the model in a kind of simulated stigmergy that produces coherent group behavior.

3. This work is an example of our methodology of Constructive Robot Ethology; we take a functional view of an animal solution to a hard general problem and engineer a related robot solution.

4. Long-duration experiments with teams of real autonomous teams provide a case study of a realized, robust, cooperative multi-robot system.

1.3 Structure of this paper

After reviewing related work, we describe our previous experiments in cooperative trail following (Section 2). The central part of the paper describes a generalized trail-following strategy (Section 3). We then describe a real robot team that implements the trail following algorithm to perform the transportation task (Section 4). We demonstrate in a series of experiments that (i) the method copes with unbounded accumulation of localization errors (Section 4.6), (ii) uses the shortest discovered path (Section 4.7). A subtle problem with the original controller is discussed and a solution is provided in Section 4.7.2. Section 4.8 shows that a suitably modified system works robustly for long periods. The findings and implications of the work are then discussed in Section 5. At various points in the paper, we identify and describe a LOST operating constraint. These are listed and summarised at the end of the Discussion.

2 Previous and related work

2.1 Ant algorithms

The effectiveness of ant foraging has inspired solutions to a variety of optimization problems expressible as path-finding, for example load balancing and routing in communications networks [12, 11]. Ant algorithms are totally distributed, with no centralized controller. The outcome of the system is thus an emergent property of the interaction of the agents with the world. Such algorithms converge on the goal state by means of feedback loops mediated either by direct communication between agents, or by indirect interaction by repeated sensing and modification of the environment, known as *stigmergy*. Holland *et al* exploit stigmergy in a foraging task with real robots [27]. A robot is used to model non-trail ant navigation in [29]. Ant algorithms have previously been suggested for a version of the transportation task [37].

Chemical trail laying and following has been demonstrated in robots [35, 34]. However it is often impractical and sometimes undesirable for robots to physically mark their environment. Trail following in localization space is not true stigmergy because the external world is not modified. We aim to reproduce stigmergy-like advantages by maintaining a minimal world model which is jointly manipulated; this *simulates stigmergy*. Mindful of the advice of [6], we seek to maintain as little state as possible in the world model, and sense the world directly for the most part.

2.2 Cooperative group robotics

A useful review of cooperative multi-robot systems is given in [7]. Cooperative robot systems are typically behavior-based [2, 3]. For example, [5] demonstrates control of formations of lab robots and outdoor vehicles

using reactive behaviors. A variety of group behaviors were constructed from a common set of basis behaviors in [26]. Formations without global localization have been demonstrated by [1].

Communication is expected to enhance the capabilities of multi-robot systems. Communication on various levels in multi-robot systems was studied by [4], who found that simple, low-cost communication was almost as effective as their more complex strategies. A distinction was made in [31] between two types of knowledge shared in robot teams; Type 1 is *global goals* where the agents are informed of a global goal that each must work to achieve; Type 2 is *global knowledge*; data that is shared about the common environment that may help an individual achieve its goals. LOST shares only Type 2 global data.

Our communication strategy is informed by the work of [18, 28], who examined the emergence of coherent group behavior by means of minimal signaling and listening strategies.

2.3 Localization and mapping

The dual problems of localization and mapping have been extensively studied in robotics. They are duals because solving one greatly simplifies the solution to the other. Given a map, a robot can localize itself. Given accurate localization estimates, a robot can build a map. Most approaches to mapping and localization (whether tackling one or both simultaneously) have focused on the generation of accurate maps of the environment with metric support. In particular, metric localization has been studied in two variants: as the tracking problem (where the initial pose of the robot is known) [33], and as the kidnapped robot problem (where the initial pose of the robot is unknown) [23, 15, 20]. Metric mapping with imprecise localization has been studied as the simultaneous localization and mapping problem [24, 40, 9]. Metric approaches to localization and mapping while accurate are computationally intensive. On the contrary, topological approaches [22, 25, 36, 13] to mapping are computationally less intensive and scale to multiple robots [10] rather more easily. However, the resulting maps (and location estimates derived from them) are coarse.

Mapping techniques could be applied to the transportation task. Once a map is generated and the goal found by exploring the environment, conventional path-planning could be used to find the optimal (or, more tractably, a near-optimal) route through the environment. Subsequent planning can also be used to generate trajectories for the robots.

Trail following schemes like ours are proposed as an alternative to conventional mapping and path planning for solving path-finding problems under certain constraints. An important advantage of trail following is that it scales well; indeed it works better as the population size increases. We will return to this in Section 5 below.

2.4 Trail following in a global but noisy localization space

We define localization space as any consistent spatial or topological representation of position. Such a space is *shared* if there is some (probably imperfect) correlation between the representations maintained by two or more individuals. A prime example is the Global Positioning System (GPS). Two systems equipped with GPS share a metric localization space in planetary coordinates. Similarly two robots that start out with known positions in the same coordinate system and maintain a position estimate via odometry share a localization space. In both examples each robot has only an *estimate* of its position in the true space, but the true space is common to both. More abstract localization spaces can be considered, such as the location of a data byte in a hierarchical database or a URL on the Internet. In these cases too, there can be some uncertainty in position; for example if position is described by a fuzzy matching rule or an ambiguous data query. Our intuition is that trail following may be possible in such spaces. For now, we examine only metric spaces.

A first version of our trail following strategy was described in [44]. It was inspired loosely by the foraging mechanisms of ants and bees [19, 45]. In our simulator 'Arena' (now renamed 'Stage' [41]), a population of robots shuttled between pre-specified 'Home' and 'Goal' regions of a large (60x60 robot-body-lengths) environment sparsely filled with obstacles.

Trail-following was found to compare favorably with an algorithm that used global knowledge of the environment. Robots were localized using a GPS-like global coordinate system with a noise model imposed. The method was shown to be robust with respect to significant localization error; indicating that it should be suitable for use in the real world.

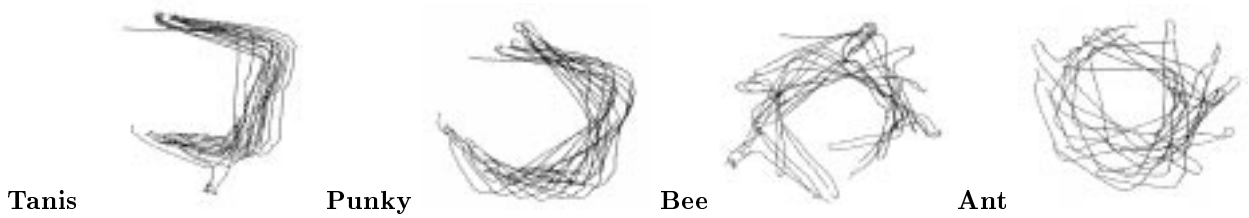

Figure 1: The trails announced by each robot over the 30 minutes trial, illustrating the accumulating odometry errors and the large variation between robots.

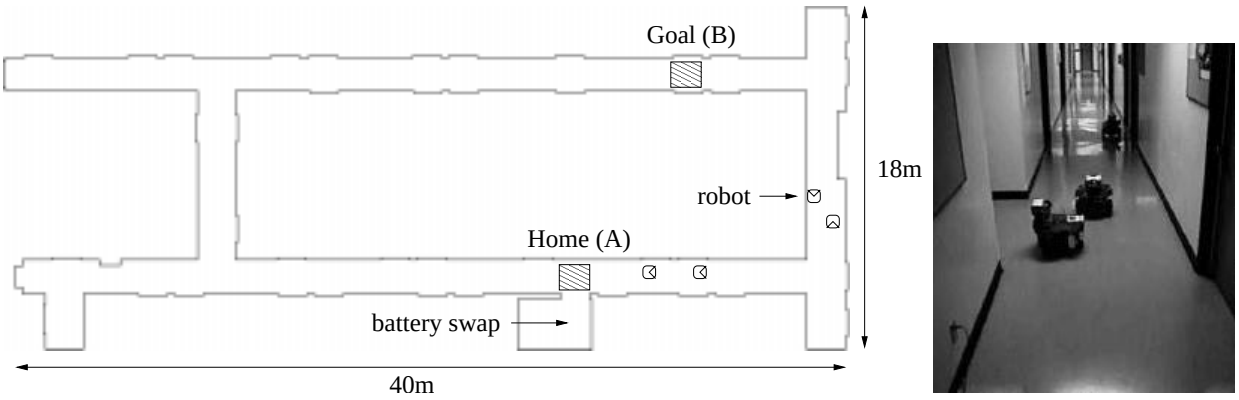

Figure 2: Diagram of the experimental environment; part of our office building; robots are drawn approximately to scale (left). Photograph of the environment (right).

2.5 Trail following in global localization space using odometry

Following these encouraging results, the method was implemented on real robots [42]. We performed experiments on a team of robots which localized by integrating their odometry. All robots are started from the same position so their coordinate systems are initially aligned. The correlation between the spaces was observed to deteriorate over time as the error in each individual's localization estimate increases. Figure 1 shows plots of the localization estimates of four robots as they work in the hallways of our building, shuttling between 'Home' and 'Goal' positions (Points A and B in Figure 2.

By sharing waypoints specified in their local coordinate systems, the robots were able to successfully travel from home to goal for 28 minutes, at which time the system broke down. After this much time, the robot's coordinate systems had diverged so much that waypoints could not be meaningfully shared between them.

In spite of this, our original trail following algorithm has some attractive properties: it finds good paths between locations, is reliable, is independent of the localization method used, is adaptive to dynamic environments, and requires modest computation and communication resources. It is reasonably robust to localization error, but it requires a global localization space. Maintaining an accurate global position estimate is often complex and always costly [14, 39, 38].

2.6 Abandoning global localization space

The most readily available sensors for localization are rate-measuring devices; odometry, accelerometers, gyroscopes, which have no fixed frame of reference. The rates must be integrated over time to obtain position estimates. The magnetic compass and Global Positioning System (GPS) are counter-examples with fixed reference frames, but these are restricted to certain (magnetically neutral, satellite line-of-sight) environments, so we do not consider them here. Rate-integrating methods suffer unbounded drift [33] and

are therefore nonstationary; this means that a population of robots which start from known global positions and update their position estimates by integration will have their coordinate systems diverge over time.

Our thesis is that many tasks, including cooperative resource transportation, can be achieved without computing full, map-like representations of the environment, thus saving the time and power required for the computation.

For reasons of generality, simplicity, efficiency and robustness we aim to design robot systems that can work with these local, nonstationary coordinate systems in novel environments [constraint 1]. In multi-robot systems with symbolic communication this becomes a problem of referring to places in the world without maintaining a common coordinate system.

Our previous work assumed that there were two places of interest in the environment (referred to as 'Home' and 'Goal') connected by a single trail. The trail was followed forwards towards the 'goal' and backwards towards 'home'. We have since generalized the method to handle N places by recording N trails (though we only present results here for an N=2 experiment).

3 Generalized Localization-Space Trail Following

This section presents the details of our generalized trail-following method, including data structures and algorithms for their manipulation. We then describe an implementation on a team of robots and show how the trail-following method can be straightforwardly integrated with a low-level navigation controller that is specific to a particular robot and environment.

3.1 Events

We define an Event as a task-relevant occurrence that is perceived by a robot. For example, in a transportation task that requires acorns to be collected from a source pile and delivered to a bucket, the relevant Events would be 'pick-up-acorn-from-pile' and 'drop-acorn-in-bucket'. A robot must be able to recognize these events in order to switch between acorn-seeking behavior to bucket-seeking behavior. Typically, Events are goals within the system; a robot transporting acorns would seek to experience Event 'pick-up-acorn-from-pile', followed by 'drop-acorn-in-bucket', and so on. It is a requirement of a goal-directed system that it is able to recognize goals in this way.

LOST requires that Events be recognized and given common names by multiple robots in the population; for information to be shared between two individuals, they must recognize at least two Events in common [constraint 2]. In a team of homogeneous robots engaged in a single group task, we expect that all robots would recognize the same set of named Events. A more diverse population might have robots that recognize different subsets of all the possible Events.

We assume that the localization error is relatively small on a single traverse, i.e. error accumulates slowly relative to the frequency of experiencing events [constraint 3].

For now we assume that all Events of the same type always happen at the same place [constraint 4] (though this constraint will be relaxed in future work as we exploit more of the dynamic properties of LOST).

When a robot detects an Event, it records its current location estimate and associates it with that Event. A robot can then express information about the world relative to location of Events. Other robots that have position estimates for the same Events can interpret the coordinates in their own local frame of reference.

LOST uses Events to exploit the regularity of the population's shared task(s), allowing individuals to share references to places in the world. The rest of this section describes the machinery required to maintain the necessary estimates, perform the required calculations, and interface with the robots' control system.

3.2 Trails, Places and Crumbs

The purpose of the Trail-following method is to guide the robot to an Event of interest, somewhere in the environment. It provides two useful pieces of information; (i) the *heading-hint* θ_T is the direction in which to travel to reach the current goal; (ii) the *distance-hint* d_T is the estimated time required to reach the goal.

The rest of this section describes how trails are created, manipulated and used to generate the Heading-hint and Time-hint.

The method's representation consists of *Crumbs* (the waypoints) and *Places* (the task-level common landmarks) collected into a *Trail*. Information is shared by combining Trails according to certain rules. Each of these elements is described below.

3.2.1 *Places:* global task-level landmarks

The main aim of this work is to remove the requirement that a group of robots maintain a conventional global coordinate system or map in order to communicate information about places of mutual interest. The method is based on the commonality of Events.

In our transportation task illustrated by Figure 2, the source (labeled B) and sink (labeled A) of resources are common to all robots. At the source the robot finds a unit of resource (Event B); at the sink it its relieved of the unit (Event A). The coordinate at which this happens is different for each robot as each has an independent frame of reference; it even changes over time as the local coordinate frame drifts. References to another Event C can be made relative to the current location estimates for A and B. Any member of the population that maintains estimates for A and B can interpret a reference to C in local coordinates in terms of A and B. The accuracy of the transformation is determined by the accuracy of estimates A and B held by both parties.

Places are represented as tuples containing the name E of the event (e.g. A,B), and a position L in localization space. The localization space can have any number of dimensions; L must fully specify a single point in this space. We consider only a two dimensional localization space, so here $L = (x, y)$.

```
Place -> [E, L]      #in general
Place -> [E, (x,y)]  #for points on the plane
```

3.2.2 *Crumbs:* local waypoints

Named for the trails of Hansel and Gretel, our trail waypoints are called Crumbs. Our Crumbs are data structures that describe the distance (by some metric) to a Place from a particular location in the local coordinate system.

A Crumb is composed of the name of the Place P to which it refers, a localization space position L, an estimate d of the distance from L to P, and the time t when the Crumb was created, to serve as a timestamp:

```
Crumb -> [P, L, d, t]
```

The distance d is does not have to be in any particular dimension, as long as all robots agree. For our experiment, we used time as the distance value. Our robots moved at an approximately constant speed of $0.25ms^{-1}$, and a Crumb was generated every 2 seconds, so a distance value of 1 (in seconds) corresponds to a physical distance of approximately 0.5m.

3.2.3 *Trails:* shared local data

A Trail is a set containing any number of Crumbs and Places. Each robot starts with an empty trail which is built up over time according to the trail-laying algorithm. A very simple trail might look like this:

```
[A, (0,0)]          #Place
[B, (10 5)]         #Place
[A, (3,4), 20, 201] #Crumb
```

This could be expressed in English as: in the coordinate system in which Place A is at (0,0) and Place B is at (10,5), A was 20 distance units away from (3,4) at t=201 seconds.

3.2.4 Putting Places into Trails

Events either cause a new Place to be added to the Trail, or an existing Place to be modified, according to the algorithm:

$$\mathcal{P} = \{P = [E, L] \in T_i \mid E_g = E\}$$
$$T_{i+1} = \{T_i \setminus \mathcal{P}\} \cup \{[E_g, L_g]\}$$

(Let $\mathcal{P}$ be the set of all Places in the Trail that locate Event E. The trail at the next timestep is equal to the current Trail with P replaced by a new Place $[E_r, L_r]$).

It follows that a Trail can contain only one Place per Event. For example in a resource transportation task with a single source and a single sink, there are two Places; one for receive-resource Event, and one for the drop-resource Event.

3.2.5 Combining trails

Two Trails can be combined if they have two Places in common. If Trail α contains Places at locations A_α and B_α and Trail β contains places A_β and B_β, there is a unique transformation γ that maps the vector from A_α to B_α onto the vector from A_β to B_β:

$$\gamma(\vec{A_\beta B_\beta}) = \vec{A_\alpha B_\alpha}$$

where γ is a linear transform consisting of a translation, scaling and rotation.

Once γ is determined using the Places common to both Trails the same transformation can be applied to map the coordinates of any location L, whether in a Crumb or a Place, from one Trail to the other. To combine the current Trail T_i with a new Trail T:

$$T_{i+1} = T_i \cup \{f(X) \mid X \in T\}$$

i.e., The trail at the next timestep equals the current trail plus all the members of the incoming trail transformed through f, where f is a function that transforms the location element of the vector X, whether a Crumb or a Place, though γ.

The resulting trail T_{i+1} contains all of T's Crumbs and Places, transformed into T_i's coordinate system.

If two trails do not have two Places in common, two different actions can be taken depending on an adopted policy. If we adopt a *pessimistic* policy, the coordinate systems are assumed to be different, so we set $f(X) = 0$ and the trails are not combined. If we adopt an *optimistic* policy the coordinate systems are assumed to be the same and the trails are combined as-is, by simply making $f(X) = X$ and directly copying Crumbs and Places from one Trail to the other. If the Trails have one Place in common and the optimistic policy holds, orientation and scale are assumed to be similar to both Trails, and the Crumbs and Places can be transformed by translation alone into the local coordinate system.

3.3 Trail decay

The trail is continually scanned and any Crumbs with timestamp older than the age threshold a (we used a=6 minutes) are destroyed. This allows for dynamic update of the trail, as out-of-date information is deleted. The dynamic response of the trail to changing environments is a function of a.

3.4 Trail-laying algorithm

To implement LOST, each robot maintains an individual Trail. The trail is initially empty. A robot will add members to its Trail for two different reasons:

1. The robot experiences an Event E. If this happens the robot updates the Trail by adding a Place $[E, L]$ where L is the robot's current location estimate. The Place is incorporated into the Trail according to the rules in Section 3.2.4. This is how a robot gathers information about the locations of task-relevant events for itself as it explores.

2. If a robot receives a trail on the network, the received trail is combined with the local Trail according to the rules in Section 3.2.5. In this way Crumbs and Places generated by individual robots are incorporated into the Trails of all robots on the network.

   In our robot implementation, the received trail is transformed into the local coordinate system. Another possibility is to transform the contents of the local Trail to match the received trail; this will cause all the Trails to converge to a common coordinate system. However, as the incoming trails are very small, and the local trails are relatively large, this is computationally expensive. As we do not seek a global representation, we choose to do the least expensive transformation.

Trails appear on the network because they are periodically generated and broadcast by robots. If a robot is moving, it creates a temporary Trail T_{temp} every S seconds (we used $S = 4$). The temporary trail consists of copies of all the Places in the robot's main Trail T_i, plus a single Crumb filled in with the current location, the name of the last Event E, the distance d from the Event (we used elapsed time as a distance metric), and the current time t, i.e.

$$T_{temp} = \{P \in T_i\} \cup \{[E, L, d, t]\}$$

where P denotes a Place. [Alternatively, a Crumb can be generated for each Event in the Trail rather than just the most recent Event; we will examine this option in future work].

This temporary trail is broadcast on the network, then deleted. We refer to the act of broadcasting a single-Crumb Trail as 'dropping a Crumb'.

In this way, as they explore the environment, experiencing Events, periodically dropping Crumbs, and (more frequently) receiving Trails on the network, each robot builds a Trail structure containing Places and Crumbs that encode information about Events, and distances to Events.

3.5 Trail-using algorithm

Suppose a robot has a certain Event E_g as its goal, such as E_g = 'drop-acorn-in-bucket'. The robot interrogates the Trail to discover which way to move to decrease its distance from the goal Event.

Given the robot's current location L_r and goal Place P_g, the Trail can recommend a heading θ_T and provide an estimate of the distance to the goal d_T. These are found by the following algorithm:

$$\begin{aligned} \mathcal{C} &= \{C_i = [E_i, L_i, d_i, t_i] \in T \mid E_i = E_g \wedge \\ &\quad (\widehat{L_i L_r} < r) \wedge \\ &\quad (d_i \leq d_j, \forall C_j = [E_j, L_j, d_j, t_j] \in T)\} \\ d_T &= d_i \\ \theta_T &= \angle L_r L_k \end{aligned}$$

where L_k is the location of any Crumb in $\mathcal{C}$. If $\mathcal{C}$ is empty, then $d_T = -1$, indicating that the Trail has no information about Event E within r meters of L_r.

That is, find the set of Crumbs $\mathcal{C}$ in the Trail whose Event E_i equals E_g and whose location L_i lies within a fixed radius r of the robot's current location L_r. The distance-hint d_T is equal to the smallest distance-to-goal estimate d_i in this set of Crumbs. The heading-hint θ_T is the angle from L_r to the location L_k of any Crumb in $\mathcal{C}$.

The Crumb reading algorithm is illustrated in Figure 3, in which a robot navigates to a goal Event by following a trail of Crumbs with decreasing distance estimates. The lowest distance-to-goal estimate of the Crumbs within the robot's sense radius r is 7; the robot steers towards the Crumb, and therefore towards the goal. The robot will take the shortest route so far discovered from that location. By following the Crumbs dropped by the whole population, each robot benefits from the others' exploration; robots will find a reasonable route much more quickly than they would alone. The larger the population size, the greater the probability of finding an efficient route and the more quickly a good route is found.

If the Crumb sensing radius r is larger than the distance between dropped crumbs (the dropping frequency f multiplied by the robot's translation speed s), as in Figure 3, then the robot can follow a continuous trail of crumbs to the goal.

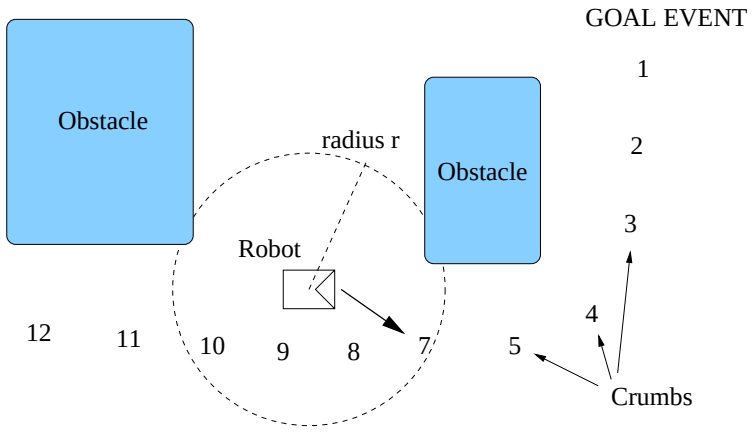

Figure 3: The Trail-reading algorithm; a robot moves towards the Crumb (represented by numbers) within its sense radius r that has the lowest estimate of distance-to-goal.

4 Experiments

The following experiments were performed to test the effectiveness of LOST in a simple version of the resource transportation task. We describe three experiments that demonstrate different aspects of the approach.

4.1 Environment

The system was tested in our unmodified office building. The floors are smooth and hard, offering good conditions for odometry. The corridors are just wide enough to allow two robots to pass each other. The experiments were performed in the evening but the corridors were open to other building users, and people passed by the robots repeatedly during each trial. The robot navigation controller was designed to avoid obstacles such as trashcans and lockers, and to ignore passers-by (including people and other robots) unless they directly impeded the robot's progress.

The floor layout is shown in Figure 2(left) and is the same as used in [42]. Figure 2(right) shows the robots navigating in the test environment. Home and Goal are separated by at least 31m of corridor and two corners in every case.

4.2 Robots

These experiments were performed with teams of three and four robots drawn from our pool of nine identical ActivMedia Pioneer 2DXs. We refer to the robots by their names (Bug, Ant, Bee, Fly, Punky and Tanis) in the results below. These are small (50x50x40cm) mobile robot bases with two independently driven wheels and a rear castor. Ours are fitted with PC104+ Pentium II computers running Linux. Each robot has front and rear 8-sensor sonar rings and wheel encoders as its basic sensors. A SICK LMS 200 scanning laser rangefinder was fitted to the front of each robot, providing good quality range readings over a 180^o field of view. In its default configuration the SICK gives two samples per degree and a range accuracy of a few millimeters to most surfaces. 802.11 wireless Ethernet connected the robots and our workstations with an effective bandwidth of 1Mbit/sec (1.9Mbit/sec on sustained large-file transfers).

Each robot was controlled by a single program running on its internal computer, using an early version of our networked robot interface *Player* [16]. The controllers are described below. Performance logs were kept locally on each robot to minimize network load, then downloaded after each experiment. Visualization and debugging software ran on a central workstation.

We chose to abstract away the handling of units of resource. Rather than picking up and carrying real objects, the robots listen on the network for messages indicating they have reached a Place. The messages are generated by the experimenters pressing a button on a GUI when the robot is within a preset distance from the Place; Places are marked on the floor with tape. For verification and as a visual aid, the experimenters

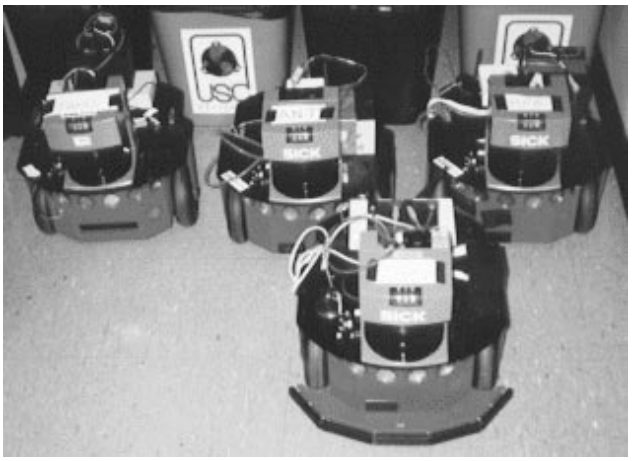

Figure 4: The Pioneer 2DX robots used in the experiments.

also attached sticky notes to each robot's laser turret when reaching Place B to indicate picking up resource, then removed it when reaching place A. The sticky notes at A were counted after each trial to verify the logs kept by each robot.

Simulating the handling of objects in this way minimizes the complexity of our robots without changing the nature of the problem. We assume that the trail-following system can be independent of the sensing and handling mechanisms that would be required to manipulate real objects.

Localization estimates are made by integrating odometry from the wheel encoders alone. Our experience with the Pioneers has shown that the accuracy of an individual's odometry estimate varies from robot to robot and from trial to trial, so we do not attempt to characterize its accuracy here. Rather the Pioneer is taken to be representative of a typical research robot. Occasionally the Pioneers suffer dramatic odometry errors; the position estimate will jump several meters or not change for several seconds despite a constant actual translation. This shortcoming of the Pioneers increases the difficulty of our experiments and severely tests the robustness of our algorithms.

We have previously demonstrated significant improvement of the Pioneer's localization estimate by combining its odometric position estimate with an that of inertial sensor package by means of a Kalman filter [30]. To emphasize the usefulness of the trail-following method, we chose to use only the most simple odometric localization scheme, available 'out-of-the-box' to most robot users.

4.3 Robot Behavior

All robots run an identical controller, designed in the behavior-based paradigm [2]. Navigation and obstacle avoidance are provided by three high-level 'behaviors': *Navigate*, *Stop* and *Panic*.

Navigate drives the robot around the environment following walls, turning down corridors and following Crumb trails. Navigate is described in detail below.

Stop monitors the laser sensor; if any obstacle is sensed within a threshold distance of the robot, the wheels are immediately stopped. Stop has the highest priority; it subsumes (disables the outputs of) all other behaviors.

Panic monitors the robot's wheel speeds; if the robot has not moved for a short time (we used 2 seconds), Panic subsumes Navigate and turns the robot on the spot in a random direction until it detects free space, then moves forward. After a latch interval (we used 6 seconds) Panic times out, re-enables Navigate and returns to monitoring the wheels.

Stop and Panic work together to handle situations that confound *navigate*. They prevent collisions and effectively 'unstick' the robot from obstacles, dead ends and traffic jams. We have found Randomized behaviors (like Panic's choice of turn direction) to be a powerful mechanism for avoiding deadlock and cyclic behavior and are often a better solution than a more 'thoughtful' behavior.

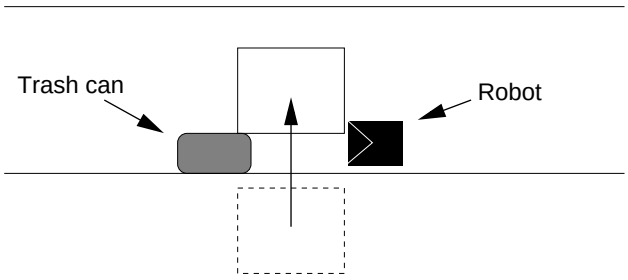

Figure 5: Principle of the sliding box algorithm. The free-space box is initially placed to the left of the robot and is slid in the direction of the arrow until there are no obstacles detected inside the box.

4.4 The Navigate behavior

Navigate exploits the accuracy of the SICK laser scanner to move through the environment, closely following walls and turning into open corridors. Whenever a Trail-hint is available, Navigate uses it to decide which way to turn.

The Navigate behavior is more sophisticated than Stop and Panic. It performs one of these discreet, mutually exclusive actions:

1. Follow walls
2. Turn left
3. Turn right
4. Turn around

The action taken is determined by the following algorithm, in which Booleans C_{left} and C_{right} are true iff a corridor is detected to the left and right, otherwise they are false. The '$\approx$' operator returns true if its arguments differ by less than $\frac{\pi}{4}$ radians. If no Crumb is within range, $\theta_T = -1$. Otherwise θ_T is the heading from the robot to the Crumb with lowest time-to-goal estimate. E is the heading error, i.e. the difference between the current heading θ_r and the Trail-suggested heading θ_T.

- MAIN_LOOP:
- IF($\theta_T > 0$) EXPLORE
- ELSE FOLLOW_TRAIL

- EXPLORE:
- IF(C_{left}) turn left with probability 0.5
- ELSE IF(C_{right}) turn right with probability 0.5
- ELSE follow wall

- FOLLOW_TRAIL:
- E = $(\theta_r - \theta_T)$ mod 2π
- IF(E $\approx \pi$) turn around
- ELSE IF(C_{left} AND E $\approx \frac{\pi}{2}$) turn left
- ELSE IF(C_{right} AND E $\approx \frac{-\pi}{2}$) turn right

The actions do the following things:

1. *Follow walls:* wall following is implemented using a sliding box algorithm. A virtual box a little bigger than the robot is placed to the left in the laser scan (see Figure 5). The box is moved left to right in front of the robot until the laser scan shows no obstacles within the box. The robot moves forwards

at a constant speed, turning towards the center of this box. If there are no obstacles, the box slides along the wall as the robot moves forward. If an obstacle appears, such as the trash can in Figure 5), the box slides out into free space and the robot heads out from the wall to avoid the object.

This approach is a simple way to provide some 'look-ahead' and adjust the path to avoid approaching obstacles. The width of the box is adjusted to determine the distance kept to the wall; its length determines the distance at which obstacles influence the robot's heading.

If there are no obstacles in the first box that is placed it is assumed that the robot is in a big open space and therefore moves forward. If no open space is found the wall-following system turns the robot in the opposite direction of the nearest obstacle.

2. *Turn left/right*: While wall following the robot is continually processing the laser range data for large openings to the left and right. A corridor is identified if the extreme left or rightmost laser samples show a nearby obstacle followed by a large discontinuity and an opening wide enough to accommodate the robot.

   The robot performs a sharp 90^oleft turn into corridors on the left, and wide 90^oright turns into corridors on the right. This minimizes the chance of interference with other robots in the junction, and leaves the robot at approximately the correct following distance from the left-hand wall.

3. *Turn around:* The robot turns 180^o, traveling a right-handed semicircle of diameter a little narrower than the corridor. This makes a smooth transition from following the left-hand wall, to following the other wall in the opposite direction.

This navigation controller was designed to make the most of the limited space available in our environment. Navigate will keep control of the robot unless the Stop behavior detects an obstacle critically close to the front of the robot. These behaviors make it possible for the robots to perform their task in small rooms and narrow corridors, passing within as little as 20cm to the side and 40cm to the front of each other before interference occurs.

When no Trail heading is available (for example at the beginning of a trial), Navigate explores the environment, turning into available corridors at random. When Trail headings are available, Navigate causes the robot to follow the Trail.

4.5 Search strategy

A random exploration strategy is chosen because it is simple, requires no *a priori* knowledge of the environment and will always find a path if one exists, given sufficient time. Thus if a path to the resource exists, a robot will eventually reach it. The first robot to reach the resource must have used the most time efficient path yet discovered and, by the trail broadcasting mechanism, all robots now have a list of waypoints describing this path. Random search finds a path more quickly with larger populations, but we found that a population of four was large enough to perform well in our relatively constrained environment.

4.6 Experiment 1: Transforming local coordinate frames

This experiment was performed to verify that LOST will enable real robots to share Trails between diverging reference frames. Combined with the low-level robot controller described above, the system performs the resource transportation task. Three robots were used with the controller described in the previous section. Figure 6 shows the environmental set-up for this experiment.

4.6.1 Procedure and data gathered

The robots are started one at a time from the same spot at the home position A, with the same orientation. This position is taken as the origin of their odometric localization space. On start-up each robot is told it is at Home/A, so it creates a Place with name A at the origin. The robots' local coordinate systems are therefore initially aligned, so we adopt the *optimistic* strategy for receiving trails as described above. This allows a robot to use received trails before it has visited both Places A and B. This speeds up the

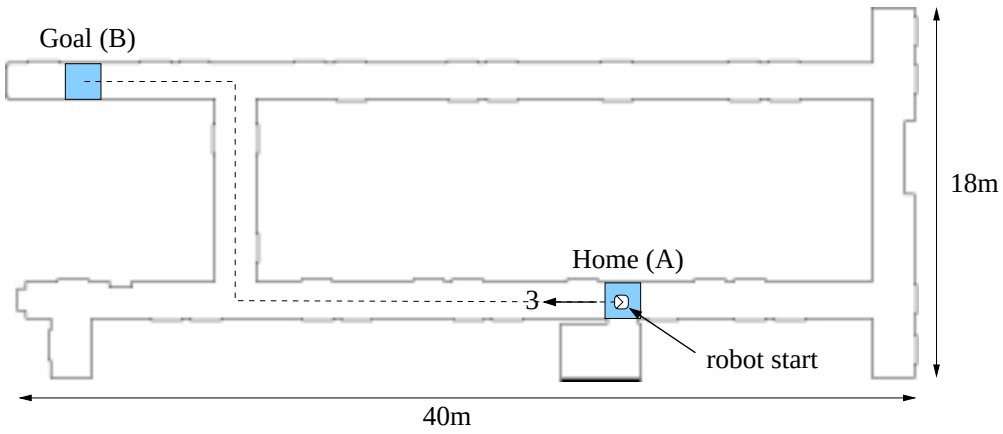

Figure 6: Diagram of experiment 1; 3 robots start at A facing to the left and must find a route to B.

	Punky	Fly	Ant	Total
Trips	14	19	19	52
Distance (m)	483	553	568	1605
Lifetime (min)	32	40	41	113

Table 1: Experiment 1: Transforming coordinates - Summary of results

convergence to a common trail, but makes the system vulnerable to failure if coordinate systems diverge very quickly. The optimistic strategy was found to be acceptable in practice with our implementation.

After initialization, the controller is run and the robot starts to explore the environment. When a robot comes within 1m of the home or resource position the experimenter 'gives' or 'takes' a unit of resource by sending the robot a message via the network. The robots are fully autonomous for the entire trial. The trial was stopped when three out of the four robots had stopped working.

Every time a robot completed an A-B or B-A trip the time and the name of the robot is noted. A log is also kept of all 1-Crumb Trails broadcast by each robot.

4.6.2 Results

Table 1 summarizes the results of the experiment, showing the number of trips, distance and lifetime of each robot. They made a total of 52 trips of ≈31m each over 41 minutes, traveling a total of 1.6km (0.99 miles) before the experiment was stopped.

At 34 minutes, Punky started to circle around a point approximately 1m short of the goal zone, repeatedly following the wall then turning around following the opposite wall. The experimenters assumed a bug in the code (we had previously ironed out several problems), disabled the robot's wheel motors and pushed it out of the path of the other robots. Fly and Punky continued for another 7 minutes, until Fly's battery was exhausted and the experiment was stopped.

Figure 7 shows the time that each robot reached Places A and B, where a visit to Place A is represented by a point plotted low, and Place B by a point plotted high. The regular interval between points in each case shows that the robots were consistently following the trail between the Places. The interval between Places is similar for all robots, indicating similar performance of each robot while running.

Figure 8(top) shows plots of the localization estimates of the three robots over the length of the experiment. The 3-dimensional plots 8(bottom) are of the localization estimate (x,y) plotted against time (z) to show how the path evolves.

Punky's failure can be seen as the tight circling at (16,2) and 34 minutes in Figure 8(right). As the Pioneer's odometry estimate degrades rapidly when turning, Punky's orientation estimate became poor

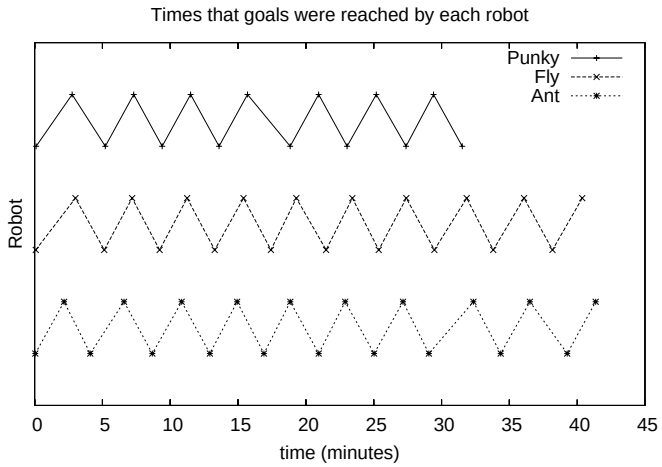

Figure 7: Experiment 1: Transforming coordinates - Events

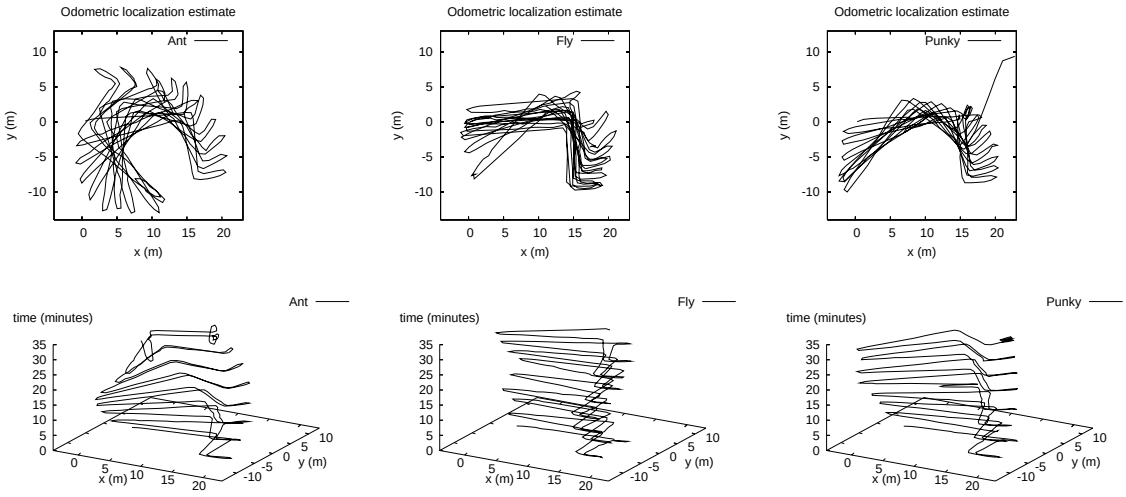

Figure 8: Experiment 1: Transforming coordinates - Paths

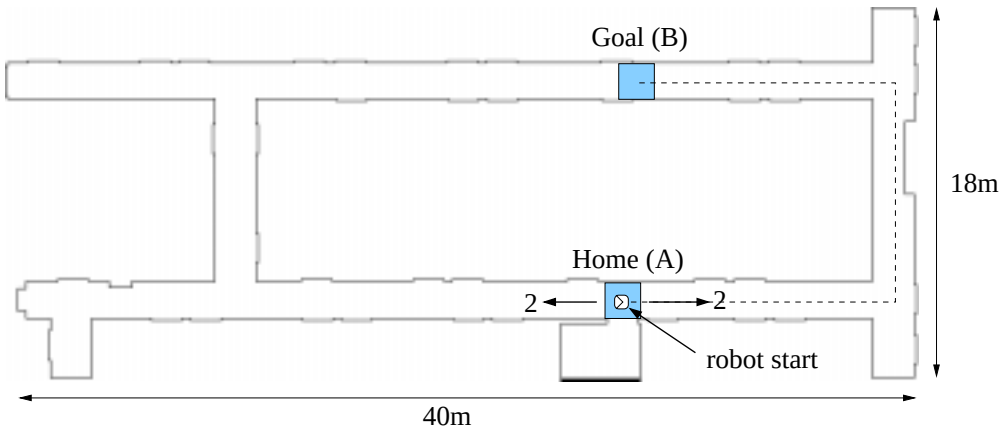

Figure 9: Diagram of experiment 2; four robots start at A. Two face left, two face right. They must find a route to B.

	Tanis	Bug	Fly	Ant	Total
Trips	18	16	17	16	67
Distance (m)	516	486	475	449	1926
Lifetime (min)	36	37	32	26	131

Table 2: Experiment 2: shortest path - Summary of results

during the cycling behavior. The single line to the top right of the 2-D plot shows Punky's position estimate as we pushed it down the corridor.

It can be seen that Ant's coordinate system Figure 8 (left) rotated more quickly than that of Fly (middle) or Punky (right). After 30 minutes, Ant's coordinate system has rotated more than 90^o relative to that of both Fly and Ant. Despite the relative rotation of their coordinate systems, Fly and Ant continued to successfully follow the trail with zero turning errors until the end of the trial.

This demonstrates that the three robots were able to transform Crumbs between their coordinate systems, despite the divergence of their coordinate systems due to accumulating error in their odometric localization estimates.

4.7 Experiment 2: Shortest path convergence

This experiment was performed to verify that the method causes robots to converge to the shortest path discovered.

The experimental set up is shown in Figure 9. Four robots are started at A. The robots use identical controllers to the previous experiment. Two start facing left (Bug and Tanis) and two face to the right (Ant and Fly). They are initially co-localized, i.e. they are all started from approximately the same point, which is the origin in their localization space and the two right-facing robots have their initial orientation set to 180^o.

The robots are started and begin to explore the environment. The same statistics are kept as for the previous experiment. Again, the experiment ends when only one robot is left working.

4.7.1 Results

Table 2 summarizes the results of the trial. The robots made a total of 67 trips of ≈28m each over 36 minutes, traveling a total of 1.9km (1.2 miles) before the experiment was stopped. Figure 10 shows the times of arrival at Places A and B. From the plot we see that Fly and Ant first arrive at around 2 minutes, return to A at around 4 minutes, then shuttle between the Places at approximately 4 minutes per round trip. Tanis

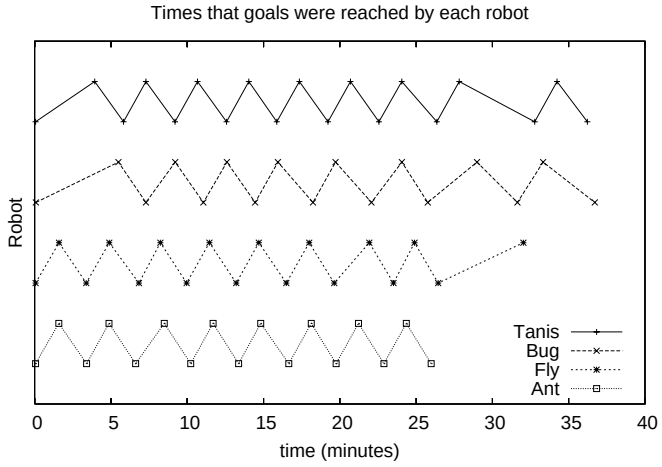

Figure 10: Experiment 2: shortest path - Events

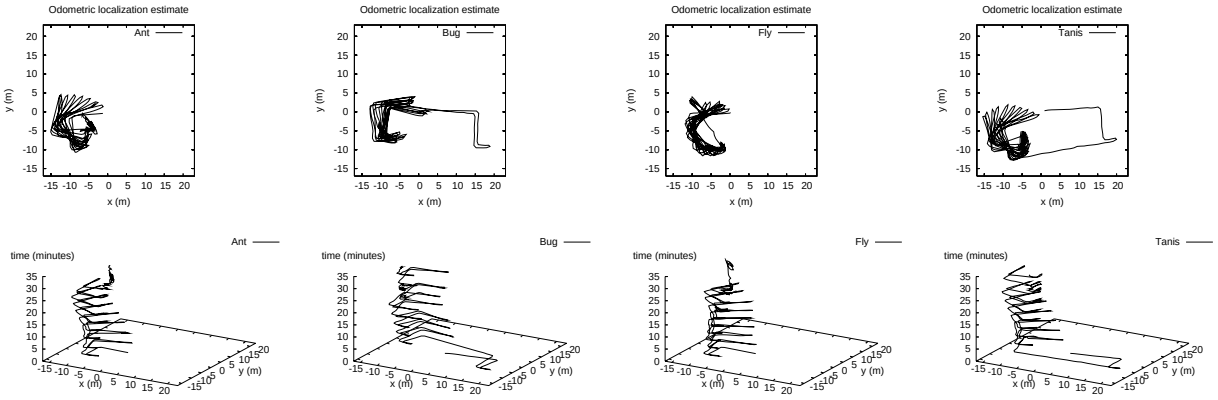

Figure 11: Experiment 2: shortest path convergence - Paths

and Bug take around 4.5 and 5 minutes respectively to reach B for the first time, then settles into a similar 4 minute round trip time.

The localization plots in Figure 11 show that Bug and Tanis initially headed away from the goal. Bug turned two corners at random, as it traveled from (0,0) to approximately (19,-10) then turned around and, by chance, took the same route back towards the origin. When it nears the origin, it can calculate a Trail heading from the data received from Fly and Ant. It takes the route discovered by Fly and Ant and shuttles between A and B for the rest of the trial.

From its furthest point from the origin, Tanis does not take the same route back, but by chance follows the corridor along to reach B from the opposite direction to Fly and Ant. Tanis' Trail now contains Crumbs describing two routes between A and B; the one it took itself, and the shorter one discovered by Fly and Ant. It takes the shorter route to A. The longer route is never used. As the Crumbs that mark the longer route reach their maximum age, they are deleted and the long route is forgotten. This saves storage space and processing time as the Trail is scanned.

The localization estimate plots again indicate that the robots' coordinate systems diverged over time. Tanis' estimates drifted by 100^oover 37 minutes; Ant's drifted by 85^o. Bug's odometry shows drift in both directions, initially clockwise until around 15 minutes, then anti-clockwise. This illustrates the variability of the Pioneer's odometry.

Ant, Fly and Tanis all failed in the same way as Punky in the previous experiment; they failed to reach Place B, circling just short of it until they were switched off. Bug exhibited the same behavior during the trial at 20 minutes and 24 minutes, but recovered both times and continued to drive between the Places.

This experiment demonstrates that the robot team will converge to use the shortest known path. However, the system suffered from a problem that makes it unreliable and could not be traced to software error.

4.7.2 Diagnosis and solution

Consideration of these results leads to the conclusion that the problem lies in an accumulating shortening of the trail.

The circling is inevitable if a robot reaches the end of the Trail (i.e. the Crumb with the lowest time-to-goal estimate) before reaching the Place. Recall that the controller makes the robot follow walls in the direction of the lowest-valued Crumb. When the lowest-valued Crumb is reached, the robot will continually servo to it, causing the observed circling behavior.

If the trail is too long, i.e. the last Crumb lies beyond the Place, the robot detects the Place as it drives by it and adjusts its estimate accordingly. If the trail is too short, the robot may never reach the Place, never have its estimate reset, and cannot recover.

By abstracting away the objects to be transported, with the goal of demonstrating a less complex, more robust system, that did not need to servo to a target at the end of a trail, we actually made the trail following more fragile. However, this was useful as it makes explicit an important constraint to the system; in order to recover from errors in the Trail (inevitable due to localization error), a robot must search for the goal after reaching the last Crumb [constraint 5].

If a physical object (such as a non-virtual unit of resource) is present at the goal, the robot should sense the object and servo towards it. The Trail becomes a device to get the robot *close enough* to the goal object to recognize it with sensors.

As we have no physically sensible Places, we solve the problem by using random exploration. When the robot reaches a Crumb with time-to-goal estimate less than or equal to the Crumb dropping frequency (2 seconds in these experiments), it stops using the Trail and instead explores at random until it finds the Place. In practice the Trail always ends very near its Place. In our environment, the robot drives past the last Crumb and reaches the Place almost immediately.

We made this modification and tested the robots again.

4.8 Experiment 3: Trail-end Event searching

This experiment was performed to verify the solution to the trail-shortening problem proposed in the previous section. The experimental set up and procedure were identical to the previous experiment, except for the addition of a battery hotswap. As the results below show, this trial ran for a long time. As each robot's

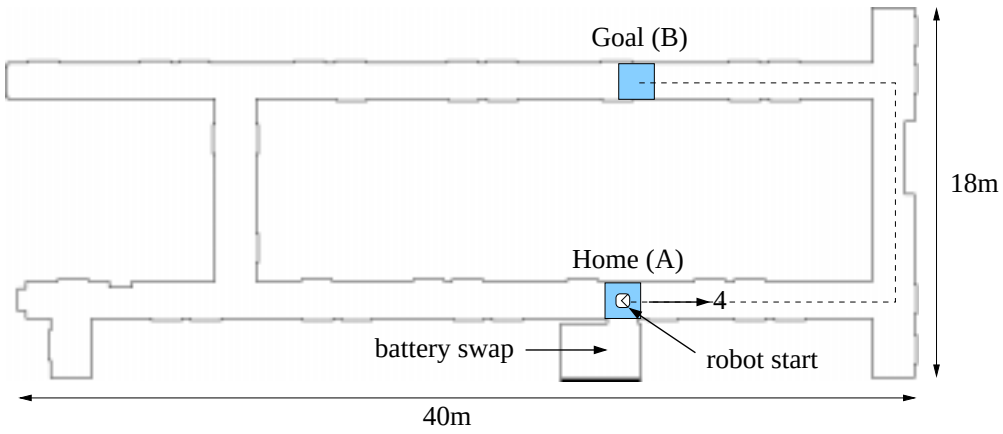

Figure 12: Diagram of experiment 3; 4 robots start at A facing to the right and must find a route to B.

	Tanis	Bug	Fly	Ant	Total
Trips	36	59	78	86	262
Distance (m)	1102	2102	2414	2592	8211
Lifetime (min)	81.3	138.1	176.6	188.4	583.6

Table 3: Experiment 3: Event searching - Summary of results

batteries were drained, we disabled the wheel motors, towed the robot into a side room and swapped its batteries. The controller was still running, so the robot was able to maintain localization estimates during the swap. Once the fresh batteries were installed the robot was towed back into the corridor and its wheel motors enabled. In this way we were able to prolong the duration of the experiment past the battery life time of the robots. Mechanical handling of the robot was observed to add error to the localization estimate; adding a further challenge to the Trail following method.

4.8.1 Results

Table 3 summarizes the results of the experiment, showing the number of trips, distance and lifetime of each robot. They made a total of 262 trips of ≈31m each over 3 hours, 8 minutes, traveling a total of 8.2km (5.1 miles) before the experiment was stopped. Two of the three robots that failed ran out of power; the other suffered catastrophic odometry failure - an occasional problem with the Pioneer robot.

Figure 13(top) shows the time that each robot reached Places A and B, where a visit to Place A is represented by a point plotted low, and Place B by a point plotted high. The regular interval between points in each case shows that the robots were consistently following the trail between the Places. Where there are larger gaps between Places (Fly at 45min, Ant at 160min) the robot was paused to have its battery hot-swapped. The long gap for Bug at 140min was a symptom of its odometry failure - it made two more trips by chance before it failed completely. The interval between Places is similar for all robots, indicating similar performance of each robot while running.

The average time between Places was $(9h43m40s/262) = 2m13s$. This gives an average speed of $0.23ms^{-1}$. The Navigate behavior drives the robots at $0.25ms^{-1}$ when cruising freely. This is the highest speed output by the controller, so we conclude that the team was working at approximately 92% of maximum efficiency, i.e. the robots were following walls in free space in the right direction 92% of their operating time.

Figure 14 shows plots of the Crumbs generated by each robot, both on the plane (top) and extended over time in 3D (bottom). The large variation in the plots indicates that their coordinate systems drifted independently. For example, Bug's coordinate system rotated clockwise; the other robots' systems drifted

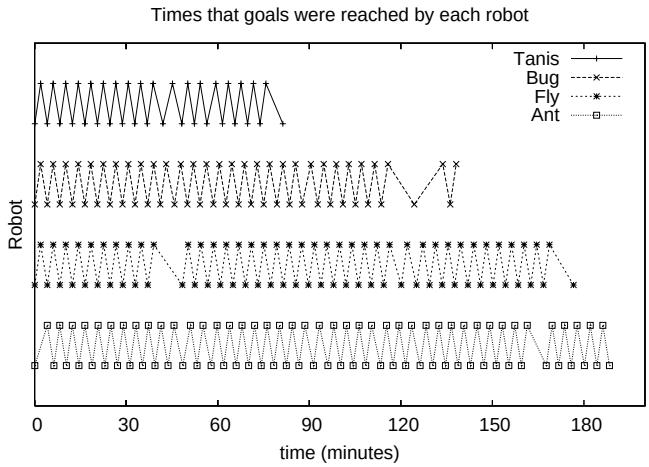

Figure 13: Experiment 3: Event searching - Events

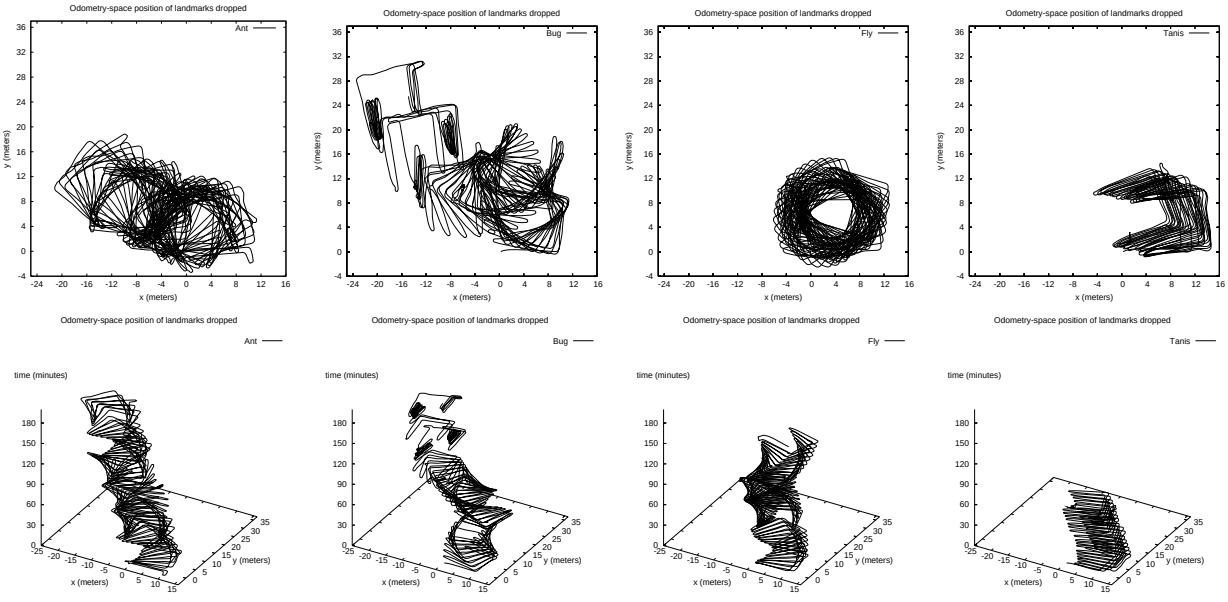

Figure 14: Experiment 3: Event searching - Paths

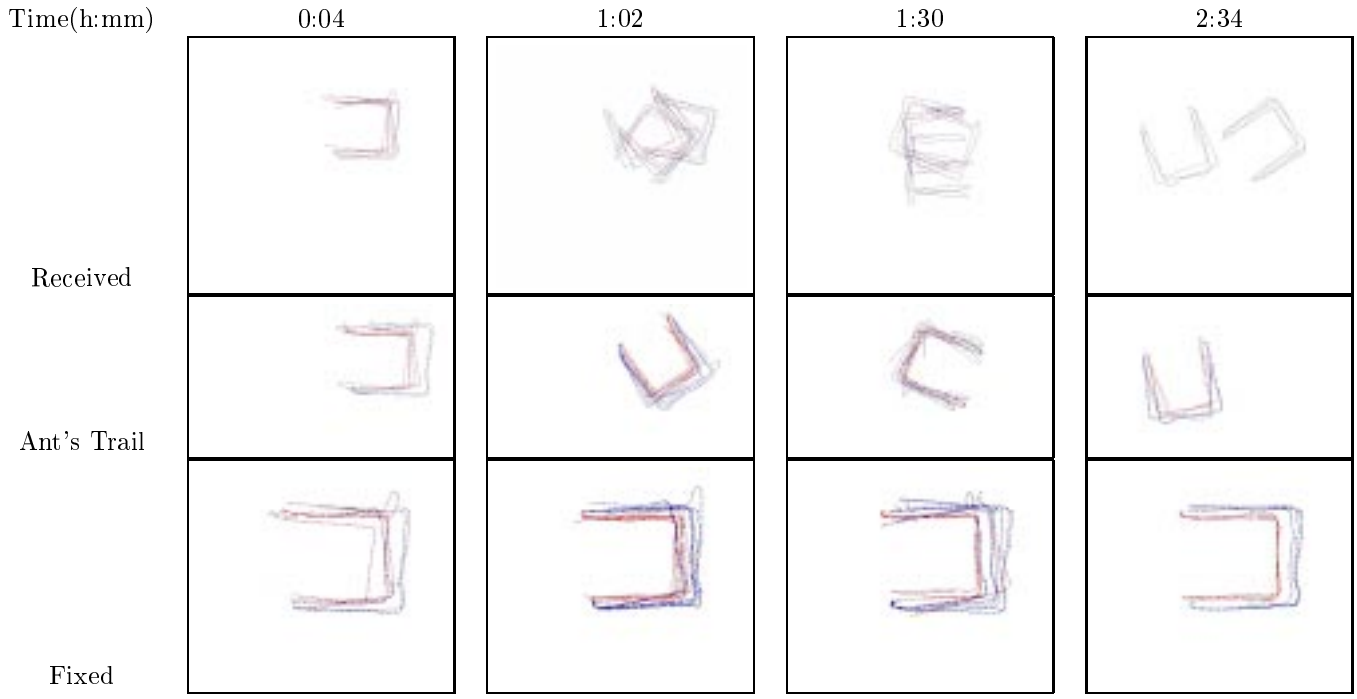

Figure 15: Trail snapshots over time from three points of view: Unprocessed (top); as incorporated into Ant's Trail (middle); as incorporated into a fixed reference frame (bottom). [Scales are different in each case].

anti-clockwise. Tanis' system rotated only by around 0.35^o per minute, while Fly's rotated at 2.00^o per minute. Despite the divergence of the robots' coordinate systems, they were able to share trails throughout the trial.

Figure 16 shows a plot of all 34,743 Crumbs received by Ant, plotted in a fixed global coordinate system; i.e., in which Places A and B are held at fixed points. The trail is noisy, with occasional rotated sections, but the noise is well within the tolerance of the trail following algorithm. The plot shows all 188 minutes worth of Crumbs. At any time during the run, Ant's actual Trail contained many fewer Crumbs, as Crumbs were deleted at $a = 6$ minutes after generation.

The plots in Figures 14 and 16 show all the Crumbs in the Trail for the length of the trial. Due to the deletion of old Crumbs, however, at any point in the time, the robot has only T seconds worth of Crumbs in its Trail. As described above, this prevents the robot using out-of-date data. Old Crumbs become useless as the robot's coordinate frame drifts over time, and if the environment changes and a new route must be discovered (dynamic environments are discussed further below).

Figure 15 shows plots of the Crumbs stored in Ant's Trail at various times during the experiment. The top plot shows the Crumbs received by Ant during the last T seconds before they are incorporated into Ant's Trail. Early in the experiment the Crumbs from all robots overlap, as their coordinate frames are very similar. Over time the coordinate systems can be seen to diverge; the Crumbs no longer overlap. The middle plot shows Ant's Trail; the Crumbs from all robots have been transformed into Ant's coordinate frame. At any time during the experiment all Crumbs are combined into a coherent path from A to B. As Ant's reference frame drifts with respect to the real world, the positions of A and B change. The bottom plot shows how the Trail looks if the positions of A and B are held constant. Incoming Crumbs are transformed into the fixed coordinate frame and the Trail always shows a coherent path.

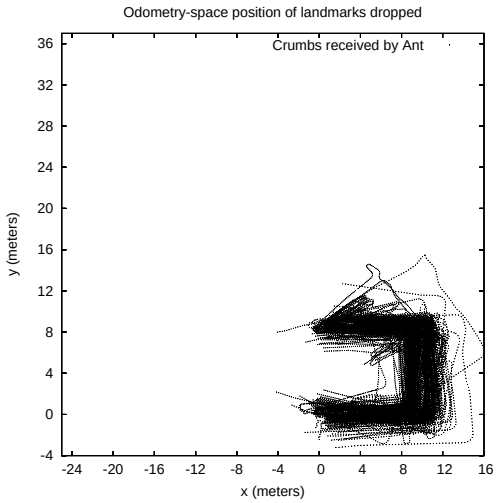

Figure 16: Experiment D: goal searching - paths

5 Discussion

This section discusses the reasons for our design and implementation decisions, and suggests some alternatives.

5.1 Parameters

Any implementation of LOST must set the values of variables r, s and a from the Trail reading, writing and decay algorithms respectively. These are the only variables intrinsic to LOST; several more are required to tune the low-level navigation controller, but these are specific to the robots and environment and not essential for this discussion.

In setting the Crumb sense radius r, there is a trade-off between the amount of information available (the larger the value of r, the greater the probability of a useful Crumb being available), and its relevance (the further away a Crumb is, the less likely it is to give you locally useful information). For example, consider a trail of Crumbs that follows around the perimeter of an obstacle. If r is smaller than the size of the obstacle, the robot cannot sense the Crumbs behind the obstacle, and follows the trail around the perimeter. If r is larger than the obstacle, the robot may attempt to drive directly towards a Crumb on the other side and fail to follow the perimeter.

The size of r, combined with the robots' (and hence the Crumbs') localization errors determines the resolution of information read from the trail. We set $r = 1$m, which proved to be satisfactory for our experiments.

The Crumb dropping rate s is determined by the speed of the robots and sense-radius r. We want Crumbs to be close enough together that at least one Crumb is visible at any time while a robot is following a trail, i.e. the gap between Crumbs must be $< 2r$. Our robots moved at $0.25ms^{-1}$ and $r = 1$m, so we set $s = 4$ seconds, so that a Crumb would be dropped approximately every 1m.

The Crumb age limit a determines how long Crums survive and thus determines the dynamic response of the Trail. In choosing a value for a, we trade off the desire to respond quickly to changes in the environment, against the cost of losing information that may still be pertinent. We chose to set $a = 6$ minutes, according to a conservative heuristic: our robots visit each Place approximately every four minutes; therefore a population of one robot could still maintain a working Trail that contained four minutes worth of Crumbs. Four plus a 50% safety margin gives us $a = 6$. Future work in more dynamic environments may require more sophisticated strategies for setting a.

5.2 Distance estimates

The Crumb's distance estimate d is the mechanism that decides what the trail optimises. If d is spatial distance, the robot follows the shortest known path. If d estimates time-to-goal, the robot follows the quickest path. Alternative metrics for d could be chosen to optimize paths for another property. For example if d was energy-to-goal, flat routes would be favored over hilly routes.

Non-distance values for d can be used, i.e. metrics that do not provide a continuous gradient, such as an estimate of remaining resources or safety, by adding a direction indicator to the Crumb data structure. The trail-reading and writing algorithms are easily changed to use the heading instead of the location of the Crumb. The early versions of LOST described in [44, 42] use this technique.

5.3 Data structures

Our Trail implementation used a simple linked list of Crumbs. Searching the list for matching Crumbs is a relatively expensive operation. A more efficient Trail implementation could use a content-addressable data structure to significantly reduce overhead when searching the set of Crumbs.

Our Trail contained an arbitrary number of Crumbs. Another possible efficiency improvement would be to maintain a queue of fixed length. A queue always contains the most recent data; so this removes the need to scan for old Crumbs and delete them. If the population size and Crumb dropping rate are known, the length of the queue determines the maximum age of a Crumb.

5.4 Scalability

LOST was designed to exploit the physical distribution of a population of robots. The environment is searched in parallel; once a good route is found, the system converges to that route. While the real-robot experiments in this paper used a small population for logistical reasons, our earlier experiments in simulation [44] successfully demonstrated larger populations.

For the following reasons, we believe that the LOST will scale to large populations of real robot, and may even be more efficient as population size increases:

- As the population size increases, the amount of the environment searched in parallel increases. Thus the probability of finding good routes increases and the initial search time decreases. Increased interference may work against this, however.
- The number of Crumbs required to create effective trails in the environment is a function of the size and complexity of the environment; it is independent of the population size. Thus the number of calculations required per robot need not increase as the population grows.
- As the population size increases, robots are more likely to be distributed over a wider area (unless extreme interference causes traffic-jams). This means that the probability that the last N Crumbs received are distributed along the length of the trail increases. Conversely, a smaller N is required to encode the whole trail with high probability than for a smaller population. We suggest that in larger populations, it maybe possible for the total number of Crumbs in the system to actually be *less* than for small populations, reducing the amount of computation per robot.

  If we choose not to decrease the Crumb dropping frequency, we can instead use a fixed-size queue of length N to hold Crumbs in a Trail.

5.5 Automatic constraint satisfaction

As trails are continually updated, LOST provides an automatic method for reducing interference. If the Crumb's distance metric is time-to-goal, trails are optimized for shortest time. A route that is heavily congested with robots will take a long time to traverse due to interference; a longer route may be more time efficient. Similarly, a short-distance route that passes through a small opening may be suitable for a small population, as the chance that several robots will meet at the opening is small. With a large population, the route is not optimal due to a traffic jam at the opening. Robots will select the longer but faster route. This

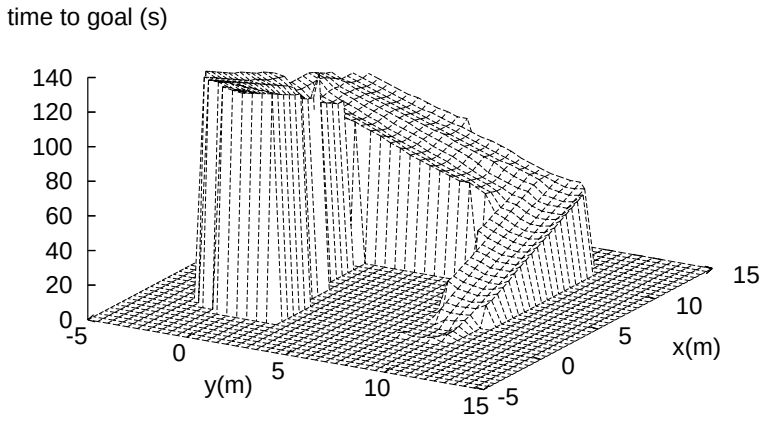

Figure 17: Surface plot of Ant's Place-B Crumb list at 900 seconds. For each position, the lowest time-to-goal of the Crumbs within sense-radius is plotted. Where no Crumb was found, 0 seconds is plotted. Notice the steady gradient from A (the origin) to B, and the curved sides of the gradient 'chute'.

effect can clearly be seen in the simulation experiments in [44]. This interesting property will be investigated further.

5.6 Comparison with mapping and planning

The Trail data structure is something between a dynamic map and a plan. It combines metric data about the positions of objects in the world with instructions on how to get from one place to another.

Figure 17 shows a visualization of a Trail stored by Ant during Experiment 3. The plot shows the lowest value of the distance-to-goal estimate d of the Crumbs within Ant's sense radius r (i.e. the distance-hint d_T) for a grid of points in localization space. The resulting surface can be interpreted as a map - all points where $z > 0$ are free space - or as a plan - move downhill whenever possible. It resembles the work space of a potential field path planner with no local minima [21, 8].

5.7 Summary of constraints

We have identified and described various constraints throughout the paper which must be satisfied for LOST to work. These are:

1. Robots must maintain a localization estimate.
2. Localization error must be small relative to the length of a traverse.
3. Robots can only share Trails about Places they mutually recognize. This is usually reasonable, but might not be true in heterogeneous or learning systems.
4. Events occur in fixed places [this is a provisional constraint: we have not yet investigated the behavior of the system with moving Events. We suspect it will work if Events move slowly relative to the time between occurrences; this more relaxed constraint may replace this provisional one.]
5. Robots must have some search strategy when reaching the end of the trail to recover from accumulated trail-shortening.
6. Routes found are not guaranteed to be optimal.

A further constraint is imposed by our choice of implementation; the Crumb-decay algorithm requires that robots have approximately synchronised clocks. This constraint is removed if a fixed-sized queue is used to contain Crumbs.

In summary, LOST is applicable to a class of trail-finding and following problems in which localization estimates can be maintained, and optimality is not essential.

6 Conclusions and further work

In this paper we have presented the novel algorithm Localization-Space Trails (LOST), for finding and traversing routes through unknown environments with a team of robots. We have demonstrated that trail-laying behaviors can be usefully implemented in a shared localization space, rather than directly in the real world. Waypoints can be shared between divergent localization spaces by specifying them with reference to landmarks common to all robots. Unlike previous landmark-based localization strategies, we use task-level features that are necessarily already known to and named by all robots.

A straightforward robot implementation was described and a series of experiments demonstrated that:

(i) LOST can guide robots between places of interest in the environment; (ii) Robots will take the best known path; (iii) Path information can be shared between coordinate systems using task-relevant features as common reference points; (iv) The system is robust to individual robot failure.

Our robot team performs a useful general task without supervision for long periods. Four robots operated autonomously in an unmodified office building, with a total robot run time of 9 hours 45 minutes and a distance traveled of 8.2km before being defeated by hardware failure. In addition to coping with highly-diverged frames of reference, the system was shown to be robust to individual robot failure and battery hot-swaps. We believe this technique will be of use in a variety of path-finding scenarios, and its simplicity and low computational load make it an attractive alternative to instrumented environments, map building and other techniques that require an explicit global coordinate system.

Having assessed the method applied to a simple task, it should be tested with more complex tasks and environments. We aim to examine the system's ability to adapt to dynamically changing environments, particularly its ability to cope with blocked and newly-available routes, and to automatically reduce interference. The existing robot team will be applied to the transportation task with N sources and M sinks. The effects and opportunities provided by large populations will be examined in simulation studies.

The system's performance was degraded somewhat by near-collisions between robots. The phenomenon of interference is an important problem for multi-robot systems, particularly in constrained spaces such as home and office environments. In these experiments we deliberately worked only in the corridors, where robots could always pass each other. We are extending our controllers to cope with co-localization conflicts, such as two robots passing in opposite directions through a narrow doorway. Results of our initial simulation work have been published elsewhere [43] and a similar approach will be implemented on the real robots in the near future.

We have not presented a framework or methodology; rather we offer a solution to a realistic problem. The application we have studied is ideally suitable for multiple robots, as is our solution. The method is completely decentralised, highly scalable and directly exploits the physical distribution of the robots that make up the population.

Like conventional path-planning and mapping solutions, LOST is applicable to a class of problems and is independent of the underlying robot controller. Unlike typical planning and mapping, our system leverages its embodiment and distribution so that it requires relatively modest computation for online, any-time path finding in the real world.

We hope that the LOST method will be directly useful for building robust path-finding robot teams. We also hope that this paper will be of value as a case study of a fully-realized cooperative multi-robot system.

Acknowledgments

The authors gratefully acknowledge the assistance of Esben Østergård, Jakob Fredslund and Brian Gerkey.

This work is supported by DARPA grant DABT63-99-1-0015, contract DAAE07-98-C-L028, NSF grants ANI-9979457 and ANI-0082498 and ONR grant N00014-00-1-0140.

References

[1] R. Alur, A. Das, J. Esposito, R. Fierro, G. Grudic, Y. Hur, V. Kumar, I. Lee, J. Ostrowski, G. Pappas, B. Southall, J. Spletzer, and C. J. Taylor. A framework and architecture for multirobot coordination. In *International Symposium on Experimental Robotics (ISER 2000)*, Hawaii, December 10-13 2000.

[2] Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[3] Ronald C. Arkin and Tucker Balch. Cooperative multiagent robotic systems. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT Press, 1998.

[4] Tucker Balch and Ronald Arkin. Communication in reactive multiagent robotic systems. *Autonomous Robots*, 1(1):27–52, 1994.

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

[6] R. A. Brooks. New approaches to robotics. *Science*, 253:1227–1232, 1991.

[7] Y. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng. Cooperative mobile robotics: Antecedents and directions. In *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS'95)*, 1995.

[8] Christopher I. Connelly and Roderic A. Grupen. On the applications of harmonic functions to robotics. *Journal of Robotic Systems*, 10(7):931–946, October 1993.

[9] M. Deans and M. Hebert. Invariant filtering for simultaneous localization and mapping. In *IEEE International Conference on Robotics and Automation*, pages 1042–7, April 2000.

[10] G. Dedeoglu and G.S. Sukhatme. Landmark-based matching algorithm for cooperative mapping by autonomous robots. In L.E. Parker, J. Barhen, and G.A.Bekey, editors, *Proceedings of the 5th International Symposium on Distributed Autonomous Robotic Systems*, volume 4, pages 251–260. Springer-Verlag, October 2000.

[11] Jean L. Deneubourg, G. Theraulaz, and R. Beckers. Swarm-made architectures. In F. Varela and P. Bourgine, editors, *Proc. European Conference on Artificial Life*, pages 123–133. MIT Press, 1992.

[12] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, pages 26(1):29–41, 1996.

[13] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Map validation and robot self-location in a graph-like world. *Robotics and Autonomous Systems*, 22:159–178, 1997.

[14] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environment. *Artificial Intelligence*, (11):391–427, 1999.

[15] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[16] Brian P. Gerkey. Player robot server. Technical report, Institute for Robotics and Intelligent Systems Technical Report IRIS-00-391, University of Southern California, 2000.

[17] Deborah Gordon. *Ants at Work*. Norton, 1999.

[18] O. Holland and C. Melhuish. Chorusing and controlled clustering for minimal mobile agents. In *Proc. European Conference on Artificial Life*, Brighton, UK, 1997.

[19] B. Holldopler and E.O. Wilson. *The Ants*. Springer-Verlag, 1990.

[20] Patric Jensfelt and Steen Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 2001 (to appear).

[21] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 500–505. IEEE, March 1985.

[22] B. J. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[23] J. J. Leonard and H. F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.

[24] J. J. Leonard and H. F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1442–1447, 1991.

[25] Maja J. Matarić. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[26] Maja J Matarić. Designing and understanding adaptive group behavior. *Adaptive Behavior*, 4(1):50–81, December 1995.

[27] C. Melhuish, O. Holland, and S. Hoddell. Collective sorting and segregation in robots with minimal sensing. In *Proc. 5th Int. Conf. Simulation of Adaptive Behaviour*, 1998.

[28] C. Melhuish, O. Holland, and S. Hoddell. Using chorusing for the formation of travelling groups of minimal agents. In *Proc. 5th Int. Conf. Intelligent Autonomous Systems*, 1998.

[29] Ralf Moller, Dimitrios Lambrinos, Rolf Pfeifer, Thomas Labhart, and Rudiger Wehner. Modeling ant navigation with autonomous agents. In *Proc. 5th Int. Conf. Simulation of Adaptive Behavior*, pages 185–195, 1998.

[30] Goel P., Roumeliotis S.I., and Sukhatme G.S. Robust localization using relative and absolute position estimates. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyongju, Korea, 1999.

[31] Lynne E. Parker. Designing control laws for cooperative agent teams. In *IEEE International Conference on Robotics and Automation (ICRA-1993)*, pages 582–587, 1993.

[32] O.W. Richards. *The Social Insects*, page 113. Harper Torchbook, 1970.

[33] S.I. Roumeliotis, G.S. Sukhatme, and G.A. Bekey. Circumventing dynamic modeling: Evaluation of the error-state kalman filter applied to mobile robot localization. In *Proc. 1999 IEEE International Conference in Robotics and Automation*, May 1999.

[34] R.A. Russell. Ant trails - an example for robots to follow? In *Proc. 1999 IEEE International Conference on Robotics and Automation*, pages 2698–2703, 1999.

[35] Titus Sharpe and Barbara Webb. Simulated and situated models of chemical trail following in ants. In *Proc. 5th Int. Conf. Simulation of Adaptive Behavior*, pages 195–204, 1998.

[36] Hagit Shatkay and Leslie Pack Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 920–929, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.

[37] D.J. Stilwell and J. S. Bay. "toward the development of a material transport system using swarms of ant-like robots". In *Proc. 1993 IEEE International Conference on Robotics and Automation*, pages 766–771, 1993.

[38] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning 31*, (31):29–53, 1998.

[39] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 2001 (to appear).

[40] Sebastian Thrun, W. Burgard, and Dieter Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots (joint issue)*, 31/5:29–53 / 253–271, 1998.

[41] Richard T. Vaughan. Stage: a multiple robot simulator. Technical report, Institute for Robotics and Intelligent Systems Technical Report IRIS-00-393, University of Southern California, 2000.

[42] Richard T. Vaughan, Kasper Støy, Gaurav S. Sukhatme, and Maja J. Matarić. Blazing a trail: insect-inspired resource transportation by a robot team. In L. E. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robot Sytems 4*, pages 111–120. Springer, 2000.

[43] Richard T. Vaughan, Kasper Støy, Gaurav S. Sukhatme, and Maja J. Matarić. Go ahead, make my day: Robot conflict resolution by aggressive competition. In *Proc. 6th Int. Conf. Simulation of Adaptive Behaviour*, pages 491–500, 2000.

[44] Richard T. Vaughan, Kasper Støy, Gaurav S. Sukhatme, and Maja J. Matarić. Whistling in the dark: cooperative trail following in uncertain localization space. In *Proc. Fourth Int. Conf. Autonomous Agents*, pages 187–194, 2000.

[45] K. von Frisch. *Bees. Their Vision, Chemical Senses, and Language.* Cornell University Press, Ithaca N.Y., 1950.