Exploiting Task Regularities to Transform Between Reference Frames in Robot Teams

Richard T. Vaughan

HRL Laboratories LLC, Malibu, California 90265, USA

Kasper Støy, Gaurav S. Sukhatme, Maja J. Matarić

University of Southern California, Los Angeles, California 90089, USA.

Abstract— We describe a team of robots that uses a trail of landmarks to navigate between places of interest. The landmarks are not physical; they are waypoint coordinates generated online by each robot and shared with team-mates over the network. 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 to sense and name physical landmarks. 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. The algorithm is tested in an experiment in which a team of 4 autonomous mobile robots run in our office building for more than 3 hours, travelling a total of 8.2km (5.1 miles). Despite significant divergence of their local coordinate systems, they are able to share waypoints, forming and following a common trail between two fixed locations.

I. INTRODUCTION

WE are interested in cooperative strategies for teams of autonomous robots as a means to improve overall performance. Cooperation is a means whereby a whole system can be 'more than the sum of its parts'. In this paper we describe a method for teams of robots to generate and use trails of waypoints for navigating between places of common interest. Waypoints are specified with reference to features in the world that are relevant to the team's task and common to all robots. Using these task-intrinsic landmarks, each robot can transform waypoint coordinates into its local reference frame, avoiding the cost of maintaining a fixed global coordinate system and/or identifying non-task-essential landmarks.

The method is applied to a realistic '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.

An earlier version of our trail following strategy was described in [1] and developed in [2]. It was inspired loosely by the foraging mechanisms of ants and bees [3], [4]. The effectiveness of ant foraging has inspired

solutions to a variety of optimization problems expressible as pathfinding [5], [6]. Chemical trail laying and following has been demonstrated in robots [7], [8], as have other applications of stigmergic communication [9]. Further useful references can be found in [10] and our previous papers.

Our 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. However, because it is impractical for our robots to leave physical trails, our trails are purely informational; they are lists of waypoints in localization space, shared over a wireless network. This is an important drawback compared to the physical trail marking of ants as it requires that a position estimate is maintained by each robot. Maintaining an accurate position estimate in a global coordinate frame is often complex and always costly, with the cost rising with the accuracy required.

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 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.

Some methods for sharing information between agents are more tolerant to misaligned frames than others; our trail following scheme was designed to be robust to significant error, both in terms of variance [1] and in mean [2]. However, once two agents' localization spaces have diverged past a certain point, they can no longer share position information. In an ex-

periment with real robots localized by odometry, the system broke down after 28 minutes as the coordinate systems diverged drastically [ibid.].

The obvious solution to this problem is for all agents in a team to share a common global coordinate system. Unfortunately, global localization is a general complex problem. Some of the most successful approaches [11] are computationally intensive and many use a map of the environment, either supplied as a prerequisite, or generated as a by-product or an explicit system goal [12].

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 in these local, nonstationary coordinate systems in novel environments. In multi-robot systems with symbolic communication this becomes a problem of referring to places in the world without maintaining a common coordinate system.

A. Contributions

This paper describes an extended version of our basic method which overcomes some earlier limitations and constraints. The main features are:

1. By referring to task-level features common to all robots, waypoints can now be shared between unrelated coordinate systems. This removes the earlier requirement for a common frame of reference within the localization system.

2. The method is generalized to N source/sink locations.

3. Waypoints are labelled with an evaluation property. The system produces path which are optimized by this property. We use an estimate of time-to-goal, which produces shortest-time paths.

With these additions our real robot implementation becomes highly robust. The extended algorithm is described formally below (Section 2). The experiment of [2] is repeated; this time the algorithm is shown to work indefinitely in a team of four robots (Section 3).

II. TASK AND APPROACH

A team of robots works to transport resource in an unknown environment. Robots start from a home position and search for a source 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. Each robot records its movements as a sequence of waypoints in localization space and shares this path with its team mates.

Thus each robot in the team has access to information about parts of the environment it has never visited, improving the performance of the system compared to non-communicating robots.

We present a novel algorithm, named *Localization-Space Trails* or *LOST*, which guides the robots between home and goal, or more generally, between any places of common interest. It is independent of the method used to drive the robot's wheels; rather it can provide a hint of a 'good' heading to pursue from the robot's current location. The details of efficiently navigating through the environment and avoiding obstacles are left to a low-level controller. We view the independence of high level navigation from the detailed control of the robot as a significant advantage of this method.

A. Localization-Space Trails

We define an *Event* as a task-relevant occurence that is perceived by a robot. For example, in a 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.

In our transportation task illustrated by Figure 1, the source (labelled B) and sink (labelled A) of resources are common to all robots. At the source the robot finds a unit of resource (Event B); at the sink it is 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.

If we assume (i) that the local coordinate system drift is small over the interval between occurrences of any event; and (ii) that the system can tolerate some position error when using shared coordinates, then robots can share information even when their natural coordinate systems are otherwise unrelated. The most simple way to maintain estimates of positions A and B is to maintain a current position estimate, and when an event (such as finding resource) occurs, to set the
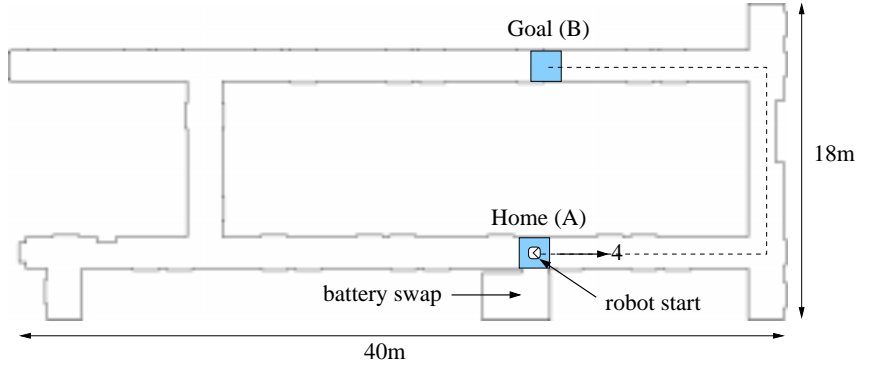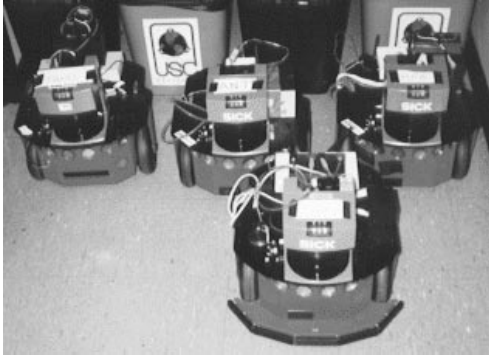
Fig. 1. Four Pioneer-2DX robots (left) start at A facing to the right and must find a route to B (right).

event position estimate equal to the current position. As the goal of the transportation task is to shuttle between A and B, their position estimates are updated frequently.

This is essentially a common landmark method; unique environmental features can be distinguished and the population agrees on a naming convention (A, B) for each feature. The advantage of using task-related events (finding/dropping resource) instead of conventional landmarks (the Empire State Building, the North Pole) is that (i) *the agents already agree on what the landmarks will be*; (ii) *no overhead is required to detect the landmark*; (iii) *landmarks are independent of the environment.* The ability to detect the states the **receive-resource** and **drop-resource** is already required by the system. Thus any robot that is capable of doing the task has sufficient information to share coordinate systems with any other. The user does not need to implement a separate sensing/processing landmark-recognizing-and-naming subsystem on each robot. Indeed, we assume nothing about the sensing and internal architecture of any member of the population. All that is required is that each robot maintain estimates of the positions of common task-related events.

A.1 *Places:* global task-level landmarks

Places are tuples containing the name E of an Event and a point L in localization space. The localization space can have any number of dimensions; L must fully specify a single point in this space. We will 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
```

A.2 *Crumbs:* local waypoints

A *Crumb* is a tuple containing the name of the Place P to which it refers, a localization space position L, an estimate d of the distance (by some user-defined

metric) from L to P, and the time t when the Crumb was created:

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

A.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.

A.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 .

A.5 Combining trails

Two Trails can be combined if they have two Places in common. If Trail α contains Places at locations $A\alpha$ and $B\alpha$ and Trail β contains places $A\beta$ and $B\beta$, 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:

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

where γ is a simple linear coordinate 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, we do:

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

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 and the trails can not be combined by the algorithm above. If we adopt an *optmistic* policy the coordinate systems are assumed to be the same and the trails are combined as-is, by simply copying Crumbs and Places from one Trail to the other. If the Trails have one Place in common and the optimistic policy obtains, then 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.

B. Trail decay

The trail is continually scanned and any Crumbs with timestamp older than the age threshold a (we used $a=4$ 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.

C. Trail-laying algorithm

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 II-A.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 II-A.5 above. 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 peridically 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.

D. 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 sugessted heading θ_t and an estimate of the distance to the goal d_T are found by the following algorithm:

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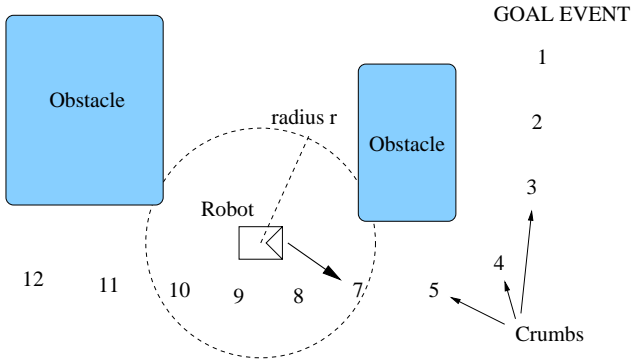& (d_i \leq d_j, \forall C_j = [E_j, L_j, d_j, t_j] \in T)\}
\end{aligned}
$$

Fig. 2. 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..

$$d_T = d_i$$
$$\theta_T = \angle L_r L_k$$

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.

I.e. Find the set of Crumbs $\mathcal{C}$ in the Trail whose Event E_i equals E_g and whose location L_i lies within the 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 2, in which a robot navigates to a 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 2, then the robot can follow a continuous trail of crumbs to the goal.

III. Implementation and Experiment

This section describes an implementation on a team of robots, showing how LOST can be straightforwardly integrated with a low-level navigation controller specific to a particular robot and environment. The implementation was evaluated in a simple version of the resource transportation task.

A. Robots

The platform for this experiment was four identical ActivMedia Pioneer 2DX robots (named Bug, Ant, Fly, and Tanis). These are small (50x50x40cm), differential-steering mobile robot bases, fitted with PC104+ Pentium II computers running Linux. Each robot has front and rear sonar rings and wheel encoders as its basic sensors. For these experiments a SICK LMS scanning laser rangefinder was fitted to the front of each robot, providing good quality range readings over a 180º field of view. 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. The robots run the *Player* networked device interface [13][http://robotics.usc.edu/player].

B. Environment

The system was tested in our unmodified office building, with the resource separated from the home position by two corners and 31m of corridor. To follow the optimal route to the resource, the robots must make two correct turning decisions based on the hint given by the crumb trail. The layout is shown in Figure 1 and is the same as used in our previous work, so the results can be directly compared.

C. Robot Behaviour

All robots run an identical controller, designed in the behaviour-based paradigm [14]. Navigation and obstacle avoidance are provided by two high-level 'behaviors': Navigate and Panic.

Navigate drives the robot around the environment following walls, turning down corridors and following crumb trails. The navigate behaviour is fairly sophisticated and contains several sub-behaviours, described in detail in [2]. It exploits the accuracy of the SICK laser scanner to make precise movements, turning tight corners and closely following walls.

Panic is designed to cope with situations that confound Navigate. It 'unsticks' the robot from obstacles, dead ends and traffic jams. Panic runs in parallel with Navigate, monitoring the laser scanner; if the robot gets too close to any obstacle, Panic overrides ('subsumes') Navigate, taking control of the robot until it finds enough space to Navigate again.

The controller is carefully designed to make the most of the limited space available in our environment. The robots successfully navigate small rooms and narrow corridors, passing within as little as 20cm of each other before interference occurs.

	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 I

RESULTS: NUMBER OF TRIPS PER ROBOT

D. Initial search strategy

In this implementation the robots initially search the environment to find goal B by random exploration. They follow walls and turn at random at intersections. This was 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.

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 all robots now have a list of waypoints describing this path.

E. 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 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 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 (in practice the robot is informed of the state change by a network message). The robots are fully autonomous for the entire trial. The trial was stopped when 3 out of the 4 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.

F. Results

Table I 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, travelling a total of 8.2km (5.1 miles) before the experiment was stopped. Two of the
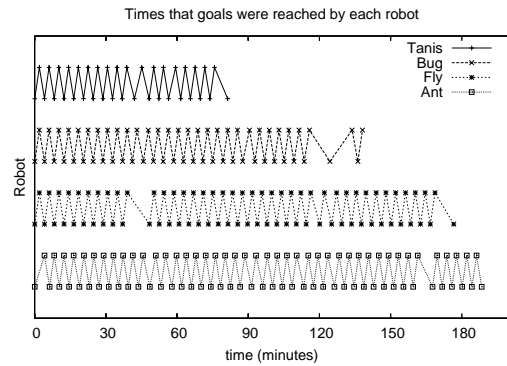

Fig. 3. Results: Event times

three robots that failed ran out of power; the other suffered catastrophic odometry failure; an occasional problem with the Pioneer robot.

Figure 3 shows the time that each robot reached Places A and B, where a visit to A is represented by a point plotted low, and 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. Bug made two more trips by chance before it failed completely.

The average time between Places was (9h43m40s/262) = 2m13s. This gives an average speed of $0:23\mathrm{ms}^{-1}$. The Navigate behavior drives the robots at $0:25\mathrm{ms}^{-1}$ 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 4 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 anti-clockwise. Tanis' system rotated only by around 0.35^o per minute, while Fly's rotated at 2° per minute. Despite the divergence of the robots' coordinate systems, they were able to share trails throughout the trial.

IV. CONCLUSIONS AND FURTHER WORK

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
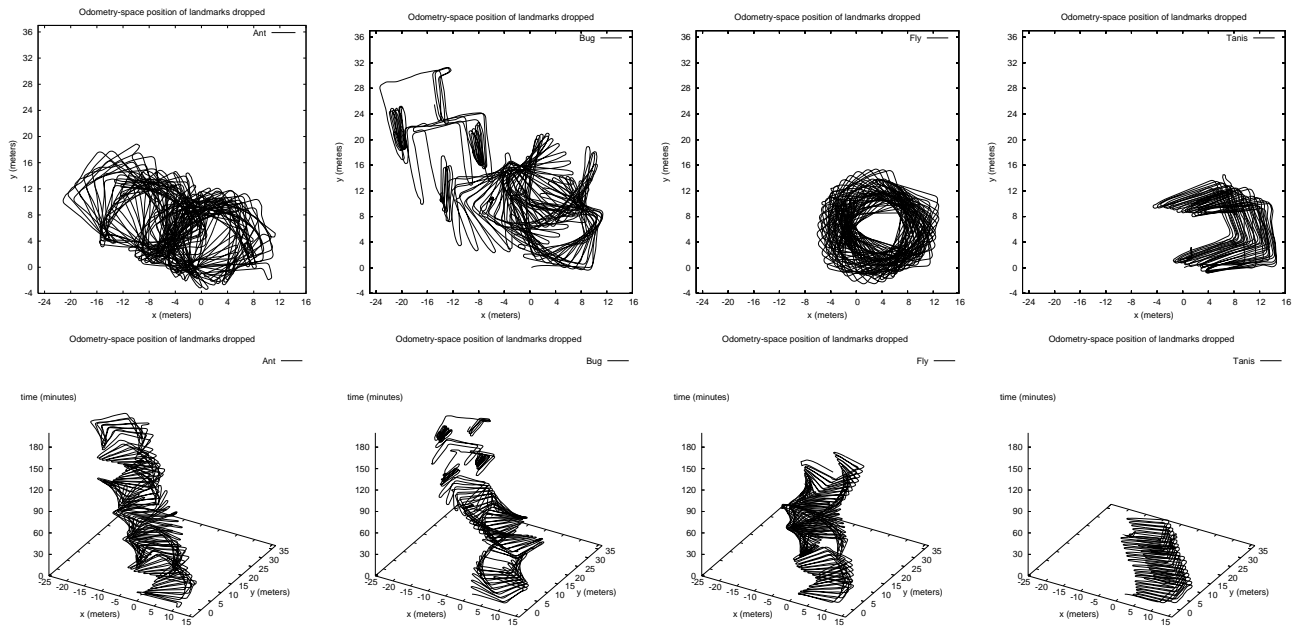
Fig. 4. Results: paths in localization space

common to all robots. Unlike previous landmark-based localization strategies, we use task level features that are necessarily common to all robots. This avoids the cost of sensing and naming physical landmarks.

The Trail 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.

Our robot team performs a useful general task without supervision for long periods. The system operated autonomously in an unmodified office building, with a total robot run time of 9 hours 45 minutes before being defeated by hardware failure. The trail following method was shown to be robust to individual robot failure, battery hot-swaps and highly-diverged frames of reference. 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.

ACKNOWLEDGEMENTS

REFERENCES

[1] Richard T. Vaughan, Kasper Stoy, 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*, 2000, pp. 187–194.

[2] Richard T. Vaughan, Kasper Stoy, Gaurav S. Sukhatme, and Maja J. Matarić, "Blazing a trail: insect-inspired resource transportation by a robot team," in *Distributed Au-*

tonomous Robot Sytems 4, L. E. Parker, G. Bekey, and J. Barhen, Eds. 2000, pp. 111–120, Springer.

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

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

[5] 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*, pp. 26(1):29–41, 1996.

[6] J. L. Deneubourg, S. Aron, S. Goss, J. Pasteels, and G. Duerinck, "The dynamics of collective sorting: Robot-like ants and ant-like robots," in *Proc. 1st Int. Conf. Simulation of Adaptive Behaviour*, 1990, pp. 356–363.

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

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

[9] 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.

[10] Y. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng, "Co-operative mobile robotics: Antecedents and directions," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS'95)*, 1995.

[11] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research*, vol. 11, 1999.

[12] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Machine Learning*, vol. 31, no. 1-3, pp. 29–53, 1998.

[13] Brian P. Gerkey, Richard T. Vaughan, Kasper Stoy, Andrew Howard, Gaurav Sukhatme, and Maja J. Matarić, "Most valuable player: A robot device server for distributed control," in *Proc. Int. Conf. Intelligent Robotic Systems*. 2001, IEEE.

[14] Ronald C. Arkin, *Behavior-Based Robotics*, The MIT Press, 1998.