Technical Report IRIS-01-403
Institute for Robotics and Intelligent Systems, University of Southern California, March 2001.

Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination

Brian P. Gerkey Maja J Matarić
Computer Science Dept.
University of Southern California
Los Angeles, CA 90089-0781, USA
{bgerkey|mataric}@cs.usc.edu

Abstract

We present a distributed planar object manipulation algorithm inspired by human behavior. The system, which we call `pusher-watcher`*, enables the cooperative manipulation of large objects by teams of autonomous mobile robots. The robots are not equipped with gripping devices, but instead move objects by pushing against them. The* `pusher` *robots have no global positioning information, and cannot see over the object; thus a* `watcher` *robot has the responsibility for leading the team (and object) to the goal, which only it can perceive. The system is entirely distributed, with each robot under local control. Through the use of* MURDOCH*, a resource-centric general purpose task-allocation framework, roles in the team are automatically assigned in an efficient manner. Further, robot failures are easily tolerated and, when possible, automatically recovered. We present results and analysis from a battery of experiments with* `pusher-watcher` *implemented on a group of three Pioneer 2 mobile robots.*

1 Introduction

Multi-robot coordination is a complex control problem, especially in *tightly-coupled* tasks, which involve a mutual dependence of the robots on each other's performance. The problem may be made even more complex through the use of heterogeneous robots, with different physical and/or behavioral capabilities. How can groups of such heterogeneous robots coordinate their behavior so as to execute tightly-coupled tasks?

In previous work [5], we proposed a partial answer to this question in the form of MURDOCH, a general-purpose task-allocation system. MURDOCH was designed for use on physically embodied robots living and working in noisy, dynamic environments in which they have little information and even less control. In this paper, we apply MURDOCH to the particularly difficult problem of multi-robot box-pushing. Using MURDOCH, we have implemented a distributed control system, called `pusher-watcher`, that enables a team of three heterogeneous robots to cooperatively relocate a large box to a specified goal, despite having no global position information and no detailed model of the box or its physical properties. We evaluate the system in four sets of experiments and present quantitative results and analysis.

2 Related Work

Box-pushing has long been one of the canonical task domains for robot researchers. In [10], Mason presents pioneering work in the analysis and planning of pushing operations, albeit in the context of fixed manipulators. As for pushing by mobile robots, several (but not many) systems have been demonstrated; we highlight the relevant ones here.

At one extreme, [7] and [8] describe a swarm-like method for moving a large box with many small, locally controlled robots; the system could fairly be described as *emergent*. In stark contrast is the planner-based master-slave pushing system described in [12]. A similarly deliberative approach is taken in [3]; the authors focus on the analysis of various two-robot pushing protocols with regard to information requirements. Some middle ground is found by the two-robot behavior-based approach presented in [11], with an emphasis on the robots' learning policies to enable effective cooperation. In [13], a fault-tolerant two-robot box-pushing system is developed, and proof-of-concept demonstrations are given. More recently, a method for single-robot box-pushing through an obstacle field (in the context of robot soccer) is given in [4].

With regard to the pushing control system itself, the work we present in this paper is most similar to the `pusher-steerer` protocol in [3] and, to a lesser extent, the `pusher-supervisor` system in [12]. However, neither of these architectures made any provision for robot failures. Of the other three multi-robot systems, only [11] is goal-directed, and in that case, both pushing robots could directly perceive the goal, somewhat reducing the need for cooperation.

A great deal of work has also been reported on multi-agent coordination systems, though the agents themselves are seldom physically embodied (e.g., [9], [15]). Notable exceptions are BLE [16], and the ALLIANCE architecture [14], both of which have been applied to a multi-target tracking task with groups of robots. ALLIANCE was also applied to a box-pushing task [13], but the system runs open-loop, whereas we have closed the control loop through the use of our `watcher`, with the side-effect of adding new opportunities for cooperation. Further, while ALLIANCE relies on an expertly-built structure of interconnected *motivational behaviors*, MURDOCH [5] provides a general-purpose, resource-centric, fitness-based task-allocation system that is independent of a robot's internal control system (e.g., MURDOCH does not require the use of behavior-based methods for internal robot control).

3 Algorithm

The task we address is cooperatively moving a box, large relative to the size of the robots, from some initial location to some observable goal location. In solving this problem, we take inspiration from human coordination behavior commonly observed when people move large pieces of furniture. If the people who are pushing or carrying the piece of furniture cannot see where they are going, another person stands between the carried object and the goal and periodically directs them. This "watcher" can see both the current position of the object and the goal, and thus can compute the error signal, perhaps in the form of a correction angle, that can be communicated to the "pushers"[1].

In the cooperative mobile robot domain, we formally define this problem with a set of constraints. First, both the box and the goal are observable, and

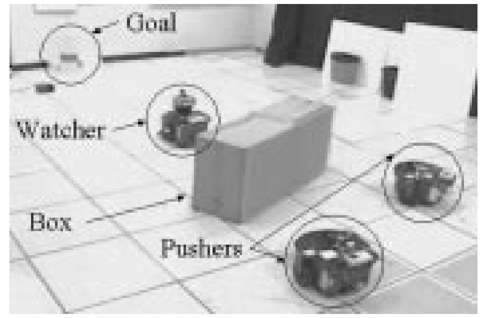

Figure 1: *Our experimental box-pushing setup. The task is for the* `pushers` *to move the box to the goal with the help of the* `watcher`. *(Image 1 of 3 taken from an experimental trial; see Figures 3 & 4.)*

there is an obstacle-free path between them that is wide enough for the box and robots to pass (i.e., we do not consider negotiating obstacles in a coordinated fashion, only as part of individual low-level control). Second, the box is large compared to the robots[2]. Third, the robots can only move the box by pushing through frictional contact. Finally, the pushing robots cannot directly perceive the goal due to the size of the box.

Given these constraints, we have implemented a multi-robot control system, called `pusher-watcher`, that is similar to the common human solution for this task. As shown in Figure 1, two robots act as `pushers`, and a third performs the `watcher` role. The `pushers` can see the box, and the `watcher` can see the goal. In addition, the `watcher`, while servoing on the goal, can accurately perceive (using a scanning laser range-finder) the angular error of the box's orientation with respect to the path from the box to the goal. Our aim, then, is to rotate the box until that angular error is zero (i.e., the box is orthogonal to the path to the goal), while simultaneously translating it toward the goal.

In order to implement this algorithm, we must first determine the `pushers`' velocities. To this end, as shown in Figure 2, we model the box and `watcher` together as a rigid body; we attach an imaginary link between the center of the watcher, C, and the center of the side of the box, orthogonal to the box. The box and link together rotate freely about C. The `pushers` are assumed to be point velocities that act on this rigid body (for simplicity, we disregard mass and acceleration). At each point in time, the `watcher` is rotated away from the normal to the box by an an-

[1]Actually, the watcher likely will not communicate the raw angle, but rather some higher-level command, such as "push more on the right"; we do the same (see Section 4).

[2]Specifically, the intended contact surfaces should allow at least two robots to be pushing simultaneously.

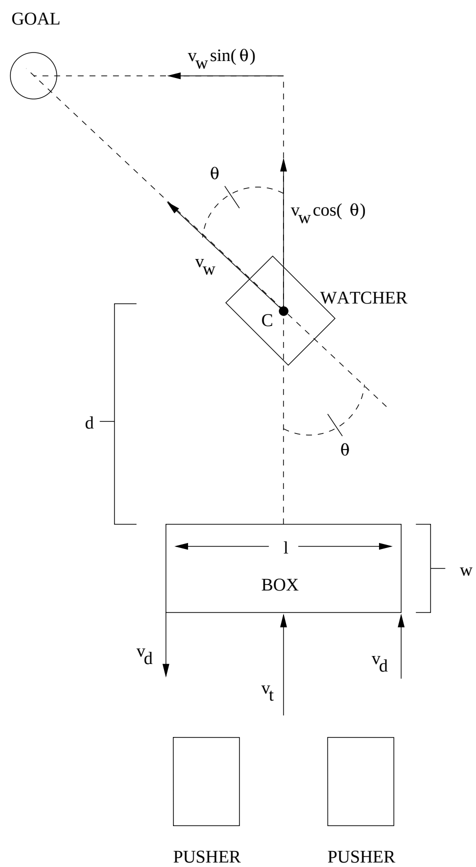

Figure 2: *The model used to derive the pushing velocities for moving the box along the desired trajectory.*

gle θ, and is moving toward the goal with a velocity v_w. These translational and angular velocities of the box will then be governed, respectively, by two simple equations:

$$v_t = v_w \cos\theta$$

$$(d + w)\dot{\theta} = v_w \sin\theta$$

After solving for $\dot{\theta}$, we can distribute it differentially to the two pushing points:

$$v_d = \frac{l}{2}\frac{v_w}{(d + w)}\sin\theta$$

Now, we compute the two `pusher` velocities:

$$v_p = v_t \pm v_d$$

The velocities given by v_p, if applied continuously at either end of the box, will ensure that a constant distance is maintained between the box and the `watcher` and that the angular error θ tends toward zero. The actual trajectory will be a curve that approaches the path to the goal with the box's orientation tending toward orthogonality with respect to that path.

The astute reader will note that in certain situations (e.g., when θ is large), this control law can yield negative pushing velocities; such implementation-specific issues are addressed in Section 5.

4 Communication Model

As previously mentioned, we have implemented the `pusher-watcher` system using the task-allocation facilities provided by MURDOCH. We will give a brief overview of MURDOCH and how it is applied to the multi-robot box-pushing problem. For a more complete discussion of MURDOCH itself, see [5].

MURDOCH is a general-purpose task-allocation system designed for use in dynamic environments in which many robots may come and go at any time. In order to support this kind of transience, communication in MURDOCH is fundamentally anonymous, and we treat the entire collective as a pool of resources that can be applied to tasks that we want done. We achieve anonymity through the use of the increasingly popular *publish/subscribe* paradigm [2],[1]: messages are not addressed to individual robots, but rather are tagged with a descriptive *subject* and are *published* onto the network for anyone to hear. A robot registers interest in a particular subject by *subscribing* to it; that robot will receive a published message if the message's subject(s)[3] match the robot's subscription list.

Since we are concerned with the allocation of resources to tasks, we use subjects in MURDOCH to represent resources. Resources can be physical devices, such as a camera or a microphone. They can also be more abstract representations of a robot's capabilities or current state, such as the possession of a map of the building, or having sufficient energy reserves. Each robot is always subscribed to the set of subjects representing its currently available resources. So, to send a message to every robot that has a laser and camera, we address the message as: {`laser camera`}.

On top of this resource-centric addressing scheme we implement an efficient auction-based negotiation protocol that is used to allocate tasks. This protocol is best explained by way of example, so, in the

[3]We generalize the basic single-subject matching criterion to include subset-matching; a message is published to an unordered set of subjects, and will be received by those robots for whom that set forms a proper subset of their subscription list.

interest of brevity, we will simultaneously explain the general protocol and its specific instantiation for use in `pusher-watcher`.

At the start, we have two robots with cameras, and a third with both a camera and a laser range-finder. We, as users, pose a `relocate-box` task to MURDOCH; this task is hierarchical and is in fact composed of a `watch-box` task that has two children `push-box` tasks. The `watch-box` task is published as a *task announcement*, and is addressed to {`mobile laser camera`}. The one robot with those resources responds, claiming the task and becoming our `watcher`. The `watcher` begins executing the `watch-box` task, which consists of: finding the goal, determining the angular error of the box, evaluating the control equations given in Section 3, and announcing new pushing tasks.

Each `push-box` task is announced to {`mobile camera`}, and is accompanied by a *metric* that potential pushers can use to score themselves as to their fitness for the task. In general, metrics can involve any arbitrary computation and take as input any part of the robot's state; in this case, the metric is a measure of how well-positioned the robot is for pushing on a certain end of the box. For example, when the task is to push on the right end, the metric will reflect whether the box is offset to the left in the robot's camera image. Each candidate executes the metric and publishes its score back to the others, and so everyone immediately knows which robot was the winner (the robots are honest, and tie-breaking mechanisms are built-in). The `watcher`, as auctioneer, awards the winner a time-limited task contract, then enters the monitoring phase. Left and right pushing tasks are allocated in pairs, parameterized with appropriate velocities, based on the orientation of the box.

We use time-limited contracts both because we cannot be sure of a robot's ability to complete a given task and because we may soon want to assign a different task. In our box-pushing domain, each `push-box` task lasts 3 seconds. During those 3 seconds, the `watcher` can, if it sees fit, *renew* either or both pending contracts; alternatively, the contracts can expire, and new, more appropriate tasks can be assigned.

In a typical run of `pusher-watcher`, the `watcher` initially announces left and right `push-box` tasks with proper velocities, and lets them push until the box's orientation changes sufficiently to warrant different pushing velocities and thus new tasks. At that point, the current contracts are allowed to expire, and new ones are formed. This reactivity to world conditions is the feature that enables MURDOCH to dynamically reassign tasks in the face of robot failure. For example, when only a single robot is available (see Figure 3), the `watcher` will actually try to allocate two pushing tasks as usual, but only one (the one with the higher velocity and thus higher priority) will be claimed. That single contract is renewed and the robot pushes on one end of the box until the orientation changes enough that it is more important to push the other end, at which point the robot will simply switch sides. When another robot is introduced, it will claim the next available pushing task and the two robots will work together at pushing the box.

5 Robotic Platform

We tested the `pusher-watcher` system on a group of ActivMedia Pioneer 2-DX mobile robots. These robots are non-holonomic, achieving locomotion through differential steering of two front drive wheels, with a passive caster in back. Many sensor configurations are possible with the Pioneers; for these experiments, each robot is equipped with a front sonar ring, a color camera, and a vision system that performs real-time color segmentation. The *watcher* robot is additionally equipped with a SICK laser range-finder, used to determine the relative orientation of the object being pushed.

Internally, each robot houses a Pentium-based computer running Linux, which executes its control program. Also onboard is Player [6], a freely available device server developed at USC that handles low-level sensor and actuator control. Inter-robot communication is provided by way of wireless Ethernet; the topology is such that every robot on the network can communicate freely with every other robot.

As is the case with any choice of robots, our decision to implement `pusher-watcher` on this particular platform added extra constraints to the problem. We account for these constraints by implementing not the exact algorithm given in Section 3, but rather a suitable approximation. For example, although the algorithm can result in negative pushing velocities, the robots can only push, not pull, the box. Thus we bound the pushing velocities below by zero, which has the effect of increasing the minimum turning radius of the box (see Section 7). Further, it turns out that some pairs of pushing velocities, especially those with large differences, are extremely difficult to execute robustly in practice. This difficultly is due mostly to the non-holonomicity of our robots, which cannot move laterally; if a robot slips off the end of the box, it cannot re-acquire it without performing a sort of "parallel-parking" maneuver, which is challenging in

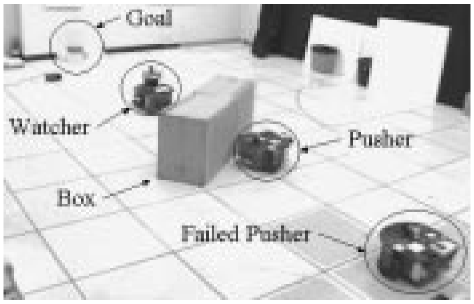

Figure 3: *Fault-tolerance in action: after we induced a single robot failure, the remaining robot is left to push on its own. (Image 2 of 3 taken from an experimental trial; see Figures 1 & 4.)*

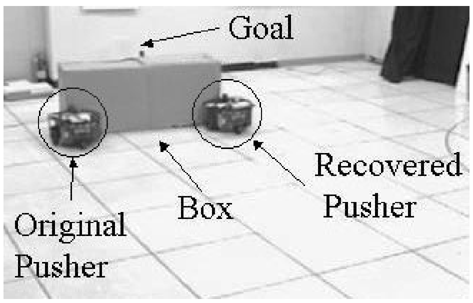

Figure 4: *We "revived" the failed robot, it was reintegrated into the team, and they completed the task together. (Image 3 of 3 taken from an experimental trial; see Figures 1 & 3.)*

a dynamic environment. For this reason, we discretize the box's orientation space into bins (currently five) for which the resultant pushing velocities are practical.

6 Experiments

In order to evaluate the `pusher-watcher` system, we performed four sets of experiments on our group of Pioneer robots, as described below. Video footage of all of these experiments is available at: `http://fnord.usc.edu/~gerkey/videos`. During the experiments, we measured two quantities: success/failure and elapsed time. We define success as the situation in which the `watcher` declares that the task is terminated, and the center of the box is positioned within 0.25 meters of the target location (see Figure 4); we do not specify a target orientation for the box. Conversely, a trial is a failure if either the `watcher` declares termination when the box is not close enough to the goal, or the box is rotated so far that the `watcher` can no longer perceive it using its laser rangefinder (this threshold is approximately 70°).

Experiment	μ	σ
No failure	31.22	0.44
Pusher failure	132.75	26.94
Pusher failure & recovery	116.44	37.72

Table 1: *Mean (μ) and standard deviation (σ) of the elapsed time (in seconds) for the successful pushing trials in each of the three experiments.*

In Experiment Set 1, as a control, two `pusher` robots had to move the box approximately 3 meters along a straight-line path (90% of the length of our lab). In Experiment Set 2, we tested the system's tolerance to individual robot failure. The setup is the same as in Experiment Set 1, with two `pushers`, but, after they pushed the box approximately 1.2 meters, we simulated a robot failure by seizing one `pusher` and shutting it off. As a result, the remaining `pusher` was left to push the box by itself, alternating sides under the direction of the `watcher` (see Figure 3). In Experiment Set 3, we tested the system's dynamic response by inducing both failure and recovery. We first let them both push approximately 0.6 meters, then seized one `pusher` to simulate failure. After the remaining `pusher` had single-handedly pushed the box another 1.2 meters or so, we reintroduced the failed `pusher`, at which point they had to finish the task together (see Figure 4). In Experiment Set 4, we tested the ability of the system to execute curved trajectories by placing the goal marker off to one side. In order to follow this non-straight path, the robots had to behave in a tightly coordinated fashion, making a series of rotational and translational adjustments.

7 Results

We performed 10 trials each from Experiments Sets 1–3. In the 30 trials, there were a total of three failures, one occurring in each set. Two failures were due to over-rotation of the box, and the third was due to premature termination on the part of the `watcher`, presumably because of sensor noise. With 27 successes in 30 trials, the two-sided 95% binomial confidence interval for the overall success rate of the system is: $p \in (0.73, 0.98)$

We also analyzed the time elapsed during the successful trials, as a measure of relative efficiency among the different experiments. The results are shown in Table 1. Using a t-test, we find that the system's behavior in the base-case differs significantly (with $\alpha = .005$) from the other two experiments; this is not surprising, given that two robots pushing the box together is much more effective than one on its own. By the same test, the two experiments involving robot failure do not differ from each other. The fact that the standard deviation increases monotonically is intuitive, since, as the complexity of the situation grows, the exact behavior of the system quickly becomes less repeatable, due to the numerous interacting dynamic processes (e.g., variable torque output from motors, friction between the box and the floor).

We note that, in Experiment Set 3, the two `pushers` switched sides when appropriate, which was in half the trials. The appropriateness of switching was determined by the configuration of the box and the remaining `pusher` at the time of reintroduction, and this configuration was in turn a result of the complex system dynamics mentioned above. However, the fact that the pushers automatically switched sides at the right times, with no detriment to the performance, demonstrates that our task-allocation system performs as specified.

From trials in Experiment Set 4 we found that there is an envelope of initial offset angles θ (see Figure 2) for which the system can reliably execute the curved trajectory. This envelope, which effectively limits the turning radius of `pusher-watcher` in this implementation, is due to the physical constraints discussed in Section 5. We derived empirically that the reliable range at present is $\{-30^\circ < \theta < 30^\circ\}$, insufficient to runs laps in our lab, which is one of our goals. We are investigating system modifications that can extend this range.

8 Conclusion

We have described the design and implementation of `pusher-watcher`, a novel distributed control system for box-pushing by teams of mobile robots. Based on the negotiation-style task-allocation facilities provided by MURDOCH, `pusher-watcher` is tolerant to robot failures, and efficient in its use of resources. We have demonstrated this system through a series of experiments on a group of Pioneer mobile robots. The results from these experiments are encouraging, and we are currently exploring other directions of this work, including different control laws, better task metrics, and more robots (both `pushers` and `watchers`). We plan to use this system to push large objects through more complex environments, such as building corridors and obstacle fields.

References

[1] Guruduth Banavar et al. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the Intl. Conf. on Distributed Computing Systems*, Austin, Texas, June 1999.

[2] Arvola Chan. Transactional publish/subscribe: The proactive multicast of database changes. In *Proceedings of ACM SIGMOD Conf. on Management of Data*, page 521, Seattle, WA, June 1998.

[3] Bruce Donald, Jim Jennings, and Daniela Rus. Information invariants for distributed manipulation. *The Intl. Journal of Robotics Research*, 16(5):673–702, October 1997.

[4] Rosemary Emery and Tucker Balch. Behavior-based control of a non-holonomic robot in pushing tasks. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Seoul, South Korea, 2001.

[5] Brian P. Gerkey and Maja J Matarić. Principled communication for dynamic multi-robot task allocation. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, Waikiki, Hawaii, December 2000.

[6] Brian P. Gerkey, Kasper Støy, and Richard T. Vaughan. Player robot server. Technical Report IRIS-00-392, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, November 2000.

[7] C. Ronald Kube and Hong Zhang. Collective robotic intelligence. In *Proceedings of the Intl. Conf. on Simulation of Adaptive Behavior (SAB)*, pages 460–468, 1992.

[8] C. Ronald Kube and Hong Zhang. The use of perceptual cues in multi-robot box-pushing. In *Proceedings of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2085–2090, Minneapolis, Minnesota, April 1996.

[9] David L. Martin, Adam J. Cheyer, and Douglas B. Moran. The open agent architecture: A framework for building distributed software system. *Applied Artificial Intelligence*, 13(1):91–128, Jan–Mar 1999.

[10] Matthew T. Mason. Mechanics and planning of manipulator pushing operations. *The Intl. Journal of Robotics Research*, 5(3):53–71, 1986.

[11] Maja J Matarić, Martin Nilsson, and Kristian T. Simsarian. Cooperative multi-robot box-pushing. In *Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 556–561, Pittsburgh, Pennsylvania, August 1995.

[12] Fabrice R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *The Intl. Journal of Robotics Research*, 12(1):79–98, 1993.

[13] Lynne E. Parker. ALLIANCE: An architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *Proceedings of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 776–783, Munich, Germany, September 1994.

[14] Lynne E. Parker. ALLIANCE: An architecture for fault-tolerant multi-robot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[15] Katia Sycara, Keith Decker, Anandeep Pannu, Mike Williamson, and Dajun Zeng. Distributed intelligent agents. *IEEE Expert*, 11(6):36–46, December 1996.

[16] Barry Brian Werger and Maja J Matarić. Broadcast of local eligibilty for multi-target observation. In *Proceedings of the Intl. Symp. on Distributed Autonomous Robotic Systems (DARS)*, Knoxville, Tennessee, October 2000.