Swarm Intelligence manuscript No.
(will be inserted by the editor)

Cache Consensus: Rapid Object Sorting by a Robotic Swarm

Andrew Vardy · Gregory Vorobyev · Wolfgang Banzhaf

Received: date / Accepted: date

Abstract We present a new method which allows a swarm of robots to sort arbitrarily arranged objects into homogeneous clusters. In the ideal case, a distributed robotic sorting method should establish a single homogeneous cluster for each object type. This can be achieved with existing methods, but the rate of convergence is considered too slow for real-world application. Previous research on distributed robotic sorting is typified by randomized movement with a pick-up/deposit behaviour that is a probabilistic function of local object density. We investigate whether the ability of each robot to localize and return to remembered places can improve distributed sorting performance. In our method, each robot maintains a cache point for each object type. Upon collecting an object, it returns to add this object to the cluster surrounding the cache point. Similar to previous biologically-inspired work on distributed sorting, no explicit communication between robots is implemented. However, the robots can still come to a consensus on the best cache for each object type by observing clusters and comparing their sizes with remembered cache sizes. We refer to this method as *cache consensus*. Our results indicate that incorporating this localization capability enables a significant improvement in the rate of convergence. We present experimental results using a realistic simulation of our targeted robotic platform. A subset of these experiments is also validated on physical robots.

1 Introduction

Social insects such as ants, wasps, termites and bees act collectively to bring order to their environments. In addition to the marvellous sophistication of their nest construction, they also organize resources and waste materials in particular ways.

A. Vardy
Department of Computer Science / Faculty of Engineering & Applied Science
Memorial University of Newfoundland, St. John's, Canada
E-mail: av@mun.ca

G. Vorobyev and W. Banzhaf
Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada
E-mail: gvorobyev@mun.ca, banzhaf@mun.ca

Honeybees distribute honey, pollen, and brood in a concentric pattern (Beekman et al 2008). Ants also organize their brood in a concentric pattern according to size (Sendova-Franks et al 2004). Another interesting ant behaviour is their tendency to cluster waste material (e.g., dead ants) into groups (Deneubourg et al 1990). The models suggested by biologists to explain these various phenomena assume decentralized control and local sensing. Many researchers have been inspired to design robotic systems with the same characteristics in the hopes of inheriting advantages such as the flexible and adaptive allocation of workers to tasks exhibited in social insect colonies (Gordon 1996). In this paper we introduce a new mechanism to allow a swarm of robots to sort objects such that a single cluster for each object type is quickly formed and persists as a stable configuration.

The ability to sort distributed materials has a wide range of potential applications. Sorting mixed materials is central to household recycling and waste management. In the context of manufacturing the gathering together of related parts is an important precursor to the assembly process. Even when there is only one type of object, aggregating those objects together has applications such as household cleaning and gardening (e.g., raking leaves into piles).

Deneubourg et al (1990) proposed a model for the general capability of insects to cluster like materials into larger and larger groups. The individual agents in this model pick up and drop objects with probabilities governed by the local object density. Isolated objects have a high probability of being picked up by unladen agents, while agents already carrying objects have a high probability of depositing them in high density areas. Over time this process aggregates the objects into larger and larger clusters. Various authors have taken inspiration from Deneubourg et al. or have proposed algorithms that apply a similar methodology (Beckers et al 1994; Maris and Boeckhorst 1996; Martinoli et al 1999; Kazadi et al 2002). In their seminal paper Deneubourg et al (1990) also adapted their model to the problem of segregating objects by type. Melhuish et al (1998) described this as *patch sorting*: "grouping two or more classes of objects so that each is both clustered and segregated, and each lies outside the boundary of the other". Patch sorting has been studied by Melhuish and colleagues (Melhuish et al 1998, 2001, 2006) as well as Zhang and colleagues (Wang and Zhang 2003; Verret et al 2004). These researchers have continued in the tradition of Deneubourg et al in developing minimalistic agents.

In the majority of these papers the motion of the agents (simulated or physical robots) is essentially a random walk with avoidance, deposit, or pick-up behaviours triggered by colliding with an object such as another robot, a cluster of pucks, or an isolated puck. Verret et al (2004) simulated a more extended view for each robot using an overhead camera. This allowed the robots to make a decision (still via a probabilistic rule) to approach an isolated puck for pick-up or a cluster at which to deposit the carried puck. Verret et al found that sorting was accelerated with increased sensing range and with inter-robot communication.

In previous work we demonstrated accelerated sorting in comparison to a variant of Deneubourg et al's model by considering clusters within the view of the robot's camera as potential targets for pick-up or deposit (Vardy 2012). The rule employed was quite straightforward. If the robot is not carrying a puck it should seek the smallest cluster in view. If already carrying and seeking a place to deposit, it should seek the largest cluster in view. The general philosophy is to make the

robots more proactive in seeking out isolated pucks for pick-up and large clusters at which to deposit those pucks.

In this paper we consider the benefits of incorporating an ability for robots to localize and return to remembered places. That is, we assume they can select a position in space as a reference and then localize with respect to that position. Localization is an extremely well-studied problem and many solutions are available. Examples include satellite-based positioning for outdoor systems (e.g., GPS), overhead tracking systems, map-based probabilistic localization (Thrun et al 2005), and visual homing (Vardy and Möller 2005; Möller et al 2010).

Using our technique, known as *cache consensus*, the ability to localize enables a dramatic acceleration in sorting performance. Each robot maintains a set of caches—one per object type. In our experiments the objects consist of coloured pucks. A cache is represented as a cache point which may have a cluster of pucks growing around it. A robot that is not carrying a puck will wander randomly, examining visible clusters and considering each as a potential target for pick-up. A Deneubourg et al. style probabilistic rule determines whether a cluster is targeted and an individual puck is further selected for collection. Up to this point the method is quite similar to that of Verret et al (2004) or our previous work (Vardy 2012). However, once a puck has been collected, the robot homes towards the cache point corresponding to the carried object type and then deposits at that cache's cluster. The qualitative behaviour of such a system is fairly clear—some cache clusters may grow at the expense of others, but since each robot defines its own caches there may be no clear trend towards global convergence (one homogeneous cluster per object type). The innovation lies in cache reassignment. A robot observes clusters and can choose to select the centroid of a cluster as its new cache for that object type. We consider three different methods for cache reassignment. Even when starting with an arbitrary distribution of pucks and caches, these methods yield an implicit consensus between robots such that they all come to share the same caches. Once cache consensus is achieved, global convergence to a single homogeneous cluster per type quickly follows.

In addition to the inspiration provided by insects which cluster and sort resources in their environment, we are also inspired by insects which make collective decisions that require consensus. Seeley provides an elegant and comprehensive account of the process used by honeybees to identify a nest site for a new colony that has just separated off from its home colony (Seeley 2010). Scout bees fly out to inspect potential sites and then return to the swarm to inform their peers about the location and quality of these sites by an adapted form of waggle dancing. Each returning scout bee promotes the site they have returned from by performing a number of waggle dances that is related to their perceived quality of that site. A strongly promoted site will attract additional scouts who provide independent assessments about the site's quality. In the vast majority of cases a consensus on one site is reached. Further, Seeley demonstrates that the chosen site is usually optimal with respect to known nest quality criteria. In the cache consensus algorithm there are also a number of options that are considered. In this case, those options are potential sites for the final cluster of each object type. There is no explicit communication between robots and therefore no promotion of one cluster versus another. However, the robots each observe the clusters built by their peers and can choose to adopt them as their own. In our experiments the typical final result is unanimous agreement on one homogeneous cluster for each object type.

1.1 Localization

Cache consensus differs from all known distributed robotic sorting algorithms in its use of localization. We can classify localization systems based on the type of sensors they employ:

Infrastructure Localization information is broadcast or downloaded from an external sensor network that has been installed into the environment.

Allocentric Robots are equipped with allocentric sensors such as cameras or laser-range finders which can be used to determine their pose[1] with respect to a stored sensory snapshot or map.

Egocentric Robots are equipped with egocentric sensors such as wheel encoders or inertial measurement devices which determine their pose with cumulative error that must intermittently be zeroed by some means.

An infrastructure-based localization system might consist of a network of tracking cameras which views the robots from above and communicates their position wirelessly. Another possibility is satellite-based localization such as provided by the Global Positioning System (GPS). The camera network could be installed anywhere, but at potentially significant cost, whereas satellite-based localization is limited to outdoor environments but is generally freely accessible. For the experiments described in this paper a single overhead camera is used to track unique markers on top of each robot (see Section 3.1). The use of this infrastructure imposes undesirable constraints on our system which we hope to alleviate in future implementations by utilizing one of the following strategies.

Allocentric localization covers a broad range of sensing technologies and localization solutions. If a map of the environment is available or can be built autonomously then the robot's current sensory snapshot can be compared with the expected sensory stimuli across all possible poses within the map. Incorporating self-motion information via recursive Bayesian filtering allows the pose to be estimated with much more accuracy and efficiency. A multitude of possible approaches can be found in recent robotics texts (Thrun et al 2005; Siegwart et al 2011; Dudek and Jenkin 2010). Even without a map, allocentric sensors can be used for localization. In visual homing, a snapshot image is captured at the reference position. After some displacement from the reference position has occurred and the robot needs to return to it, the current image is compared with the snapshot to yield a movement vector (see a recent review of visual homing techniques in (Möller et al 2010)). The constraint of visual homing is that there must exist sufficient commonality between images, meaning that the current position must lie within the catchment area defined by the home position. Visual homing has been postulated to explain the ability of insects such as bees to return to food sources (Cartwright and Collett 1983). The representation of places by stored images appears to be a key tool in insect navigation (Collett and Collett 2002).

Egocentric localization relies only upon the integration of measured velocity and/or double integration of acceleration to determine the robot's pose. Wheel encoders are a popular technology for this purpose in mobile robotics, although the advent of MEMS accelerometers and gyros has provided another low-cost

[1] The term 'pose' implies the specification of both position and orientation with respect to some reference frame.

alternative which does not suffer from wheel slippage errors. Egocentric localization is subject to cumulative error that typically increases with distance travelled but can be combined with allocentric localization to eliminate this error.

In the next section we present our benchmark method and the details of the *cache consensus* method. Section 3 describes our experimental methodology and section 4 provides experimental results. Discussion, future work, and conclusions follow in sections 5 and 7.

2 Methods

In this section we present two different algorithms for distributed sorting to provide benchmarks on the performance of our proposed algorithm. We also present the cache consensus algorithm in several variations that differ in how cache points are reassigned. Prior to discussing these methods we discuss physical requirements and the perceptual system common to all tested methods.

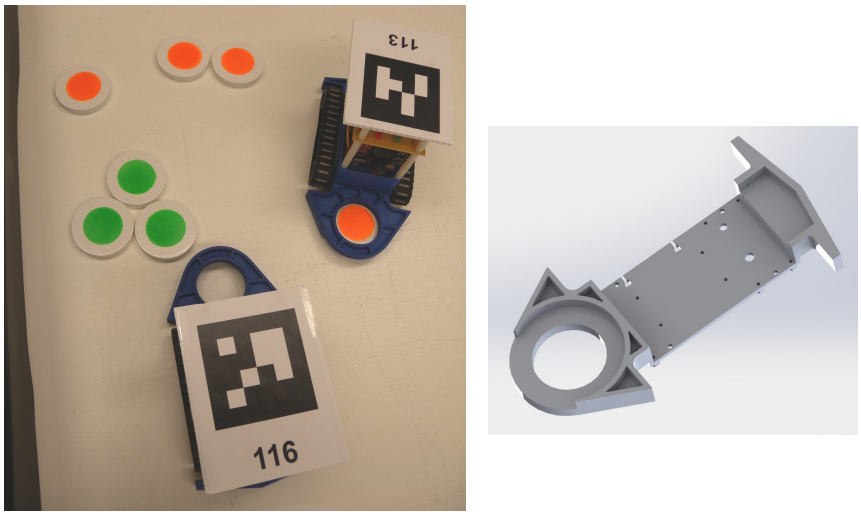
2.1 Requirements

The requirements for robotic implementation of all of the algorithms described below include a forward-facing camera, some means of localization with respect to the home point (for cache consensus only), and a passive gripper mechanism allowing one puck to be carried. Figure 1(a) shows an image of our robots in operation—robot '113' is currently carrying a puck within its passive gripper, while '116' is unladen. The use of a passive gripper was inspired by Beckers et al. who employed a C-shaped gripper which could hold up to three pucks (Beckers et al 1994). Our robots are similar in that they employ a C-shaped gripper, although our grippers hold only one puck. Depositing a carried puck is achieved simply by backing up and then moving away, leaving the puck behind. One constraint this imposes is that a puck wedged into a corner of the environment would be difficult to extract. For this reason we utilize a rectangular test environment with rounded corners. All of the methods described could easily be adapted to use active grippers.

The robot platform used in our experiments and modelled by our simulator is the SRV-1[2] which is a differentially steered tracked vehicle measuring 12.5×10.8 cm^2. The standard camera lens has been replaced with a fisheye lens which provides a horizontal field of view of 187° along the image equator. A customized body for the robot has been fabricated which includes a passive gripper mechanism capable of holding a single puck (see Figure 1(b)).

The objects sorted here are wooden pucks, painted white with a coloured circle on top. As discussed below, pucks are visually identified as distinctly coloured blobs. However, our algorithms could be adapted to use other strategies such as visual markers (Olson 2011) or classification of 'natural' objects using computer vision.

[2] `http://www.surveyor.com`



(a) (b)

Fig. 1 **(a)** Two of our modified SRV-1 robots in operation. Views from robot '116' are presented in Figure 2(a). **(b)** The underside of the robot's housing, showing the shape of the passive gripper. (Colour figure available online)

2.2 Perception

The only perceptual input to our agents is an image captured from a forward-mounted camera. The camera systems on our robots are equipped with a fisheye lens to maximize their field of view. Pixels in the image are classified by their colour as obstacles (black), pucks (red, green, etc...), or other robots (blue). After colour segmentation, a morphological erosion operation is applied to remove small blobs which are likely to be noise (Gonzalez and Woods 2002). The calibration data for this camera and its fixed geometry with respect to the ground-plane allow us to relate image pixels with 2-D position on the ground-plane in the robot's reference frame (details are provided in section 3.3). This allows the formation of an occupancy grid, where each cell is filled with an integer label that corresponds to the classification of the source pixel. Figure 2(a) shows the view from robot '116' as depicted in Figure 1(a), the colour segmented image, and the mapping of coloured pixels onto the occupancy grid.

Imposed on the occupancy grid is additional information. Pucks are identified by applying connected components labelling (Gonzalez and Woods 2002) for each possible puck colour. This process yields a set of blobs (interconnected pixels). The centroid of each such blob is projected onto the occupancy grid and indicated by circles in Figure 2. This process yields a list of puck locations. We then determine connectivity between pucks by representing each perceived puck as a node in a graph. If the distance between any pair of pucks is less than 1.5 times the puck diameter then an edge is created between them. The connected components of the resulting graph correspond to clusters and we can easily determine the size and centroid of each cluster. Note that clusters are extracted independently for

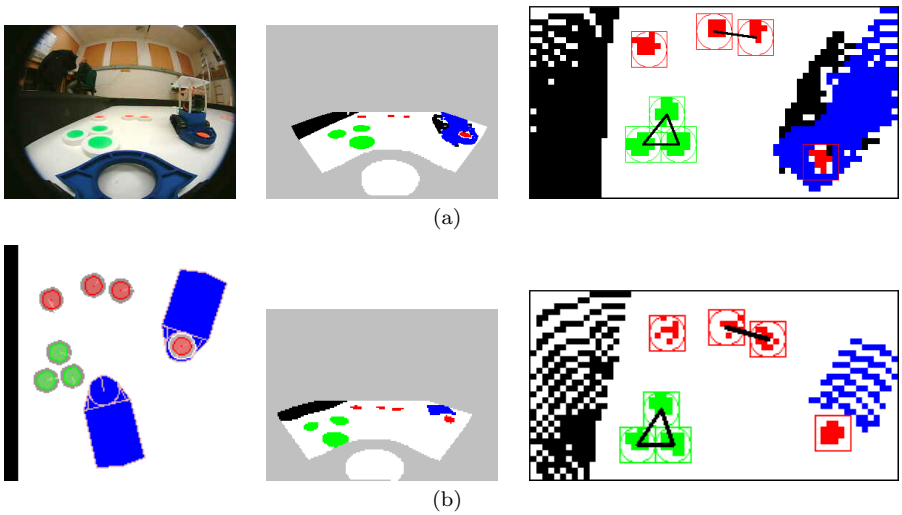


Fig. 2 This figure shows the view from one of our SRV-1 robots **(a)** and a simulated robot **(b)**. In **(a)** the robot's raw view, colour segmented image, and local map are shown from left-to-right. In **(b)** an overhead view of the simulator is shown, the simulated robot's raw view (no colour segmentation is required), and the local map. (Colour figure available online)

each colour and therefore consist of only one colour by definition. We refer to the occupancy grid, list of visible pucks of each colour, and list of clusters of each colour as the *local map*.

Some additional processing is applied to the information stored in the local map. Each robot can see within its own gripper and may also be able to see within the grippers of other nearby robots. To prevent one robot from attempting to collect a puck already held by another, we remove from the list of pucks any that lie within a threshold distance in the image plane of another robot. The effect of this rule is visible in Figure 2(a) in that the red cells within the gripper of the other robot are not extracted as a puck (they are not circled). It is also apparent from this figure that the other robot is partially classified as a robot (blue), but also partially classified as an obstacle (black). This is simply because parts of the robot are difficult to paint, such as the black rubber treads. Also, we take special note of puck colours within the robot's own gripper. If the fraction of puck colours within the gripper is high enough then we assert that a puck is being carried. A puck at the position of the gripper is added to the local map in this case. The carried puck, just like other pucks, can belong to a cluster. If so, we know the robot has made contact with this cluster. This is a significant event for the sorting methods discussed below.

Clusters extracted from the local map are denoted as $G_j^{(i)}$ where i is the index of the cluster and j is the object type. These clusters are graphs $G_j^{(i)} = (V_j^{(i)}, E_j^{(i)})$ where $V_j^{(i)}$ corresponds to the node set (i.e., the perceived pucks) and $E_j^{(i)}$ is the set of edges between nodes. The algorithms described below respond to the number of pucks in a cluster. Therefore, we define the size of a cluster as the number of nodes: $\text{size}(G_j^{(i)}) = |V_j^{(i)}|$. The smallest and largest clusters of type j are denoted

as S_j and L_j,

$$S_j = \arg\min_{G_j^{(i)}} \{\text{size}(G_j^{(i)})\} \tag{1}$$

$$L_j = \arg\max_{G_j^{(i)}} \{\text{size}(G_j^{(i)})\} \tag{2}$$

It is important to note that single pucks are considered full-fledged clusters. Therefore, $\text{size}(G_j^{(i)})$ lies in the range $[1, n_j]$, where n_j is the overall number of pucks of type j. Since clusters are extracted independently for each type, each cluster is homogeneous by definition.

Removing the subscript from S_j or L_j is meant to indicate the smallest and largest clusters regardless of type,

$$S = \arg\min_{S_j} \{\text{size}(S_j)\} \tag{3}$$

$$L = \arg\max_{L_j} \{\text{size}(L_j)\}. \tag{4}$$

2.3 Probabilistic Pick-up / Deposit

For two of the sorting methods presented below (ProbSeek and CacheCons) the decision as to whether a puck should be picked up or deposited is a probabilistic function of the size of the candidate cluster, denoted as *size*. We utilize the functions shown in Figure 3 from Deneubourg et al (1990). If an unladen robot encounters a cluster and is in the appropriate state, it will draw a random number in the range $[0, 1]$ and if this number is less than $p_{pu}(size)$ then the pick-up state will be activated. Similarly, a robot carrying a puck will initiate a deposit with probability $p_{de}(size)$. The pick-up threshold function $p_{pu}(size)$ is a decreasing function of cluster size, implying that smaller clusters will be preferentially targeted. Similarly, the deposit threshold function $p_{de}(size)$ is an increasing function of cluster size, implying that the formation of larger clusters is always preferred. These functions were introduced by Deneubourg et al (1990) and further studied by Kazadi et al (2002) who derived a required condition on the ratio of the pick-up and deposit probabilities which these functions satisfy. However, our algorithms are more complex and we cannot assume that this result holds. It may be the case that other functional forms (e.g., the same functions but with a different exponent) would be just as effective for the original probabilistic clustering algorithm or for our approach.

2.4 Beckers, Holland, and Deneubourg (BHD) Clustering

Our first benchmark method is intended to be faithful in principle to the clustering robots proposed by Beckers, Holland, and Deneubourg (Beckers et al 1994) and is therefore referred to as BHD. The robots developed by Beckers et al. were equipped with a passive gripper, a microswitch to detect the force of pushing three or more pucks, and infrared sensors to detect other robots or obstacles. The behaviour is to move forwards until either the microswitch or the infrared sensors are triggered.

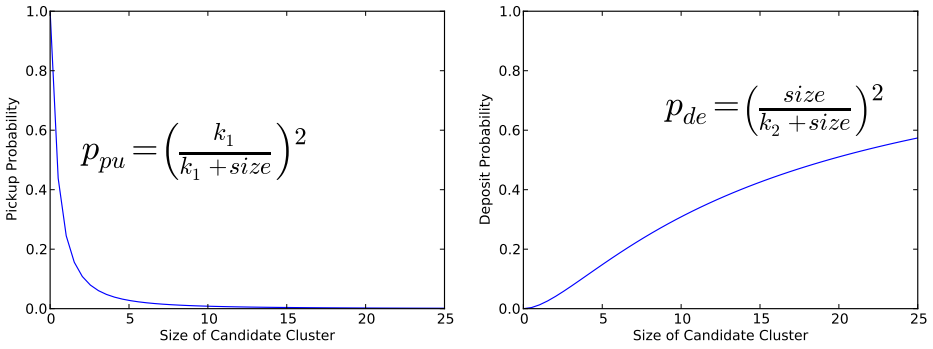


Fig. 3 The pick-up (left) and deposit (right) probabilities as functions of cluster size. The dashed lines indicate the particular choices of parameters utilized in our experiments ($k_1 = 1$, $k_2 = 8$).

If the microswitch is triggered then the robot backs up, turns by a random angle, then continues straight. If the infrared sensor is triggered then the robot turns by a random angle, then continues straight.

In our implementation the event of triggering the microswitch is replaced by the event of contacting a cluster. A cluster is contacted when the gripper contains a puck and that puck is within the threshold distance (1.5 puck diameters) to any other puck. This can occur even if the robot is not already carrying a puck, but pushes into an existing cluster such that one of the cluster's pucks passes into the gripper. In place of the infrared sensors we apply the VFH+ obstacle avoidance algorithm (to be discussed in more detail below) to determine whether an obstacle lies immediately ahead. If so a random turn is triggered. Note that pucks are rendered invisible to VFH+ for BHD. In other words, there is no attempt to avoid or steer around clusters.

All of the methods described in this paper are implemented as finite state machines (FSM) as we believe they provide a simple and easily reproducible description of behaviour. The FSM for BHD is shown in Figure 4. The robot's action in the `FORWARD` state is simply to move straight ahead. The `PUSH` state is activated upon contact with a cluster. While in `PUSH` the robot moves forward for a fixed number of iterations (1 in our experiments) to embed the carried puck into the cluster. `BACKUP` is then entered which causes the robot to move backwards for a fixed number of iterations (5). `TURN` is always entered after `BACKUP` and is also triggered by detection of an obstacle while in `FORWARD`. In the `TURN` state the robot rotates on the spot by a random angle in the range $[f\frac{\pi}{2}, f\pi]$, where f is set to either -1 or 1 depending upon which side of the occupancy grid contains fewer obstacles.

2.5 ProbSeek

This method is referred to as ProbSeek because it utilizes both Deneubourg et al's original probabilistic rules and the seeking behaviour described in (Vardy 2012). It shares a common core of functionality with the cache consensus method

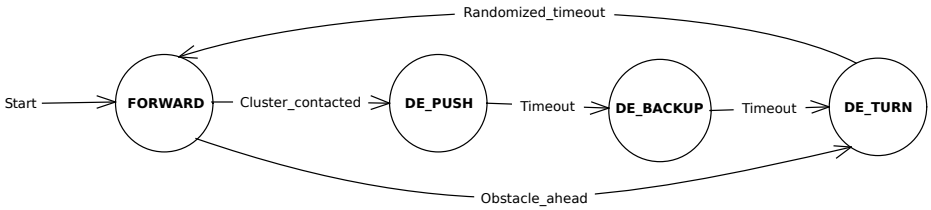


Fig. 4 Finite state machine for BHD.

presented below and therefore serves as a useful benchmark. It is essentially the same algorithm without the notion of caches.

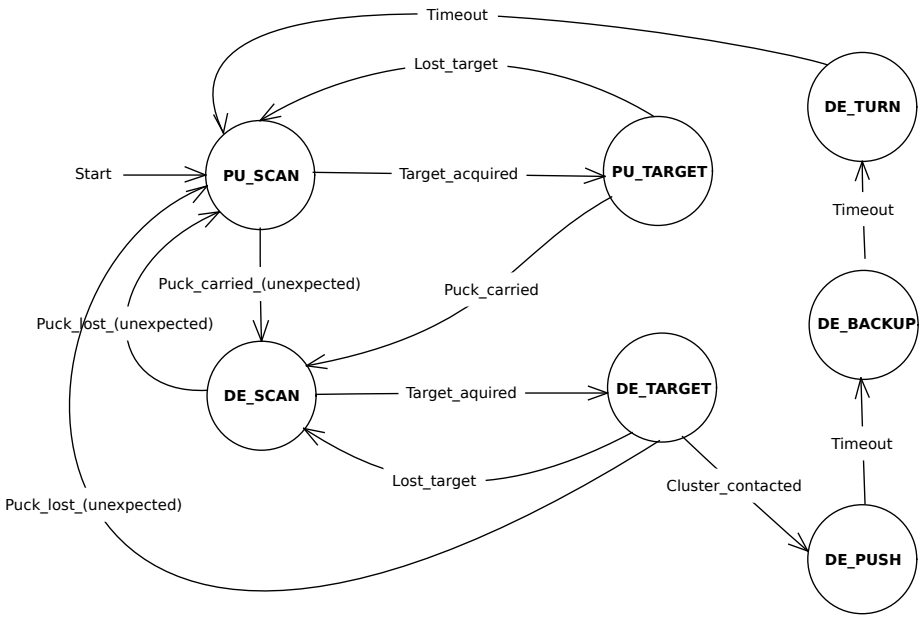
In Deneubourg et al's experiments the agents operate within a two-dimensional grid with idealized interactions between robots and pucks. However, the robots in our experiments move on a planar surface shared with the objects to be clustered—coloured pucks. An obstacle avoidance strategy is required to mitigate interactions with other robots, the boundary, and clusters of pucks that are not being explicitly targeted for pick-up or deposit. The VFH+ algorithm is used for this purpose (Ulrich and Borenstein 1998). VFH+ computes polar histograms over the robot's possible movement directions and attempts to determine the best turn angle so as to approach the goal, while avoiding imminent collisions. Siegwart et al (2011) surveyed a wide variety of obstacle avoidance algorithms and found that, although it is subject to local minima traps, VFH+ was among the simplest and most computationally efficient.

ProbSeek is implemented as the finite state machine shown in Figure 5(a). The behaviours for each state are given below. States prefixed with 'PU' are concerned with picking up pucks, while 'DE' states are concerned with depositing carried pucks.

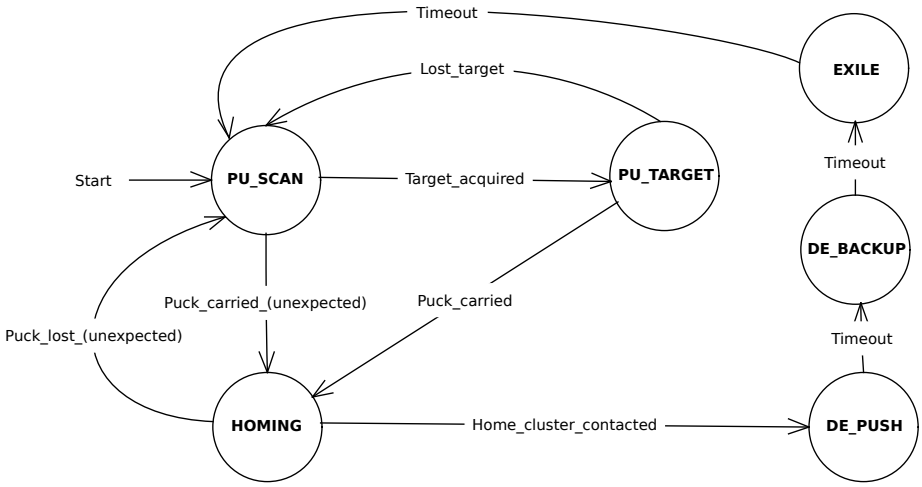
`PU_SCAN` The smallest visible cluster, S, is considered as a target for pick-up. The probability p_{pu} of targeting this cluster for pick-up is given by the equation in Figure 3 (left) with $k_1 = 1$. At each iteration in which `PU_SCAN` is active, a random number is drawn and if it is less than p_{pu} one of the pucks lying in this cluster is targeted (the left or right-most, depending upon which has the lower degree and is therefore less strongly connected to the rest of the cluster) and the state transitions to `PU_TARGET`. The robot executes a wandering behaviour while in `PU_SCAN`. A random forward angle is selected as the goal for VFH+, which alters this heading as needed to avoid collisions.

`PU_TARGET` The puck closest to the previously selected target position is identified. If the distance between this puck and the previously selected target exceeds a threshold then the target is considered lost and the robot reverts to `PU_SCAN`. A successful pick-up occurs when a puck appears in the robot's gripper, which leads to a transition to `DE_SCAN`. While in `PU_TARGET` the robot simply steers towards the targeted puck without engaging VFH+.

`DE_SCAN` This state serves an analogous function to `PU_SCAN` except that it is concerned with finding clusters for deposit, rather than pick-up. If the carried puck is of type j then the largest visible cluster of that type, L_j, is considered as a potential target. This cluster is targeted with probability p_{de} as shown in



(a) ProbSeek finite state machine.



(b) CacheCons finite state machine.

Fig. 5 Finite state machines for ProbSeek and CacheCons.

Figure 3 (right, with $k_2 = 8$). Similar to `PU_SCAN` we draw a random number at each iteration and if it falls below p_{de} then the cluster is targeted and the state transitions to `DE_TARGET`. The same wandering behaviour as in `PU_SCAN` is applied in this state.

`DE_TARGET` Behaviour in `DE_TARGET` is identical to that described for `PU_TARGET` with the exception of the success condition. In this case success is indicated when the carried puck becomes part of a cluster. Due to the way in which

clusters are extracted, two pucks can only belong to the same cluster if they are of the same type. If this occurs the state transitions to `DE_PUSH`.

States `DE_PUSH`, `DE_BACKUP`, and `DE_TURN` are functionally identical to the states `PUSH`, `BACKUP`, and `TURN` defined for BHD.

ProbSeek shares several features with the method presented in our previous work (Vardy 2012), in particular targeting the smallest cluster in view for possible pick-up and the largest cluster for possible deposit. This feature was found to accelerate clustering performance over the method of Beckers et al (1994). The main difference with the previous method is the incorporation of Deneubourg et al's probabilistic heuristics. For an unladen robot the previous method would always target the smallest cluster in view regardless of the size of that cluster. Incorporating Deneubourg et al's heuristic means that small clusters will be targeted for pick-up more readily than larger clusters. This is consistent with the goal of convergence to one cluster. Similarly in the case of deposit, ProbSeek is more likely to target a larger cluster than a smaller one. The other significant difference is the usage of the finite state machine formalism to implement ProbSeek, whereas the previous method was implemented within a simplified subsumption architecture (Brooks 1986).

2.6 CacheCons

The cache consensus algorithm will be referred to as CacheCons. CacheCons inherits the probabilistic pick-up behaviour of ProbSeek, but differs in how carried pucks are handled. In brief, whenever a puck is collected, it is immediately delivered to the robot's cache point for the carried object type. 'Cache point' refers to the position in space chosen to represent a cache.

The state machine for CacheCons is given in Figure 5(b) and is identical to ProbSeek in many respects. The primary difference is the absence of the `DE_SCAN` and `DE_TARGET` states. Instead, when a puck is collected the `HOMING` state is entered. There is no need to identify potential targets for deposit since the cache point for the carried object type serves as the only goal. While in `HOMING` the robot will steer towards this cache point, with the actual turn angle mitigated by VFH+ to avoid collisions. The assistance of VFH+ is important here as it means the robot is 'respectful' of other clusters of pucks and will not disturb them unless boxed in. `HOMING` will terminate if the cache cluster is contacted (i.e., the carried puck becomes part of this cluster).

The pick-up behaviour in CacheCons differs from ProbSeek's in one small way. If a puck is being considered for pick-up we check if it is visibly connected to the cache cluster for its type. If so, the pick-up attempt is abandoned. This reduces the frequency at which a robot will collect pucks that already lie within its cache. However, if the robot's field of view does not encompass the cache point then we cannot determine whether a cluster belongs to the cache. In this situation it is possible for pucks to be picked-up from the cache cluster and then re-deposited.

Another difference between CacheCons and ProbSeek is in the behaviour after a puck is deposited. For ProbSeek a deposited puck is first pushed in (`DE_PUSH`), then the robot backs up (`DE_BACKUP`), and finally turns by a random angle (`DE_TURN`). In CacheCons the `DE_TURN` state is replaced in this sequence

with the `EXILE` state. After completing a deposit the robot should seek a puck to collect. As mentioned above, we wish to avoid the situation of a robot depositing a puck at a cache cluster then turning around to collect another puck that is already part of that cluster. Therefore, instead of employing a random turn, the `EXILE` state drives the robot away from its home point for a fixed number of time steps to encourage the collection of 'foreign' pucks.

2.6.1 Cache Point Assignment

On every step when the robot processes its input according to the CacheCons finite state machine described above, it also considers L, the largest cluster in view of any type, and determines whether the centroid of this cluster should be selected as a new cache point. If size(L) exceeds the remembered size of the cache point for the corresponding object type then its centroid is selected as the new cache point for that object type. The memory of cache sizes, m, is initialized as an array of zeros. Whenever the current cache cluster of type j is in view, we update the memory as follows,

$$m'_j = \max(m_j, \text{size}(C_j)) \tag{5}$$

where m_j is the remembered size of the cache cluster for type j and C_j indicates the visible cluster lying closest to the cache point for type j within a threshold distance. That is, C_j represents the robot's current view of the cache cluster. The max function is used because the robot's field-of-view may not completely encompass the cluster which implies that the current perceived size may be an underestimate.

2.6.2 Cache Separation

The perceptual system described above extracts lists of pucks and clusters independently for each object type. Thus, there is no need to detect the homogeneity of a cluster since all clusters are homogeneous by definition. However, it remains possible for clusters of different types to grow in close proximity to each other. This may be undesirable or not depending on the application, but it was found to reduce overall performance in our initial experiments because it increases the amount of time the robots spend avoiding obstacles (to a robot carrying a red puck, green pucks are obstacles).

This situation can be managed by specifying a minimum threshold distance between cache points. This distance is set to 50 cm in our experiments which are run within a 187×187 cm^2 environment. However, a choice needs to be made when a new potential cache lies within the threshold distance of an existing cache of a different type. The new cache candidate is the largest visible cluster, L. This candidate is checked against all existing cache points to determine those that are in conflict. We then compare the size of the candidate with the existing conflicting caches and only adopt it if it is the largest. If the candidate is adopted, the conflicting caches are removed from memory.

3 Experimental Setup

3.1 Physical Setup

Experiments on our SRV-1 robots were conducted within a square arena measuring 187×187 cm with rounded corners (radius 15 cm). The arena's floor is painted white and its walls are made of black vinyl baseboard material. The pucks to be sorted have a diameter of 6.3 cm and a height of 1 cm. Each puck is painted white with an approximately 4 cm diameter coloured circle painted on top. At the start of each trial the pucks and robots are manually positioned with the aid of a projector which displays start positions for all pucks and robots, as drawn from a uniform distribution. A camera is mounted in the ceiling overhead to capture the progress of experiments and to track visual markers on the top of each robot. The AprilTag visual marker system is employed to allow the position, orientation, and identity of each robot to be determined (Olson 2011). Figure 1(a) shows part of the arena, the pucks, and two SRV-1 robots with AprilTag markers.

The SRV-1 robots are capable of on-board image processing, but we found it more convenient for experimental and debugging purposes to execute the image processing and finite state machine updating on a desktop computer (a dual-core Pentium with a clock cycle of 2.93 GHz and 8 GB of RAM). This computer maintains network connections to each robot using a Wi-Fi router. The control software iterates through the list of connected robots, downloads their current images at 320×240 resolution, processes, then uploads a movement command to be executed for 75 ms. The overhead camera is also connected to this computer and the AprilTag marker tracking is integrated with the robot control software so that CacheCons can localize.

3.2 Simulation

The physical setup described above was replicated as closely as possible in a custom simulation environment. The simulator utilizes the JBox2D[3] physics library to provide realistic interactions between the robots and pucks. Some aspects of the physical setup were not modelled such as localization, since position information can be provided to the robots directly.

3.3 Calibration

Calibration is necessary in order to relate image coordinates with the position of relevant objects (e.g., pucks, other robots) in the plane. This relationship is required for the formation of the occupancy grid (see section 2.2) as well as for generating synthetic images in our simulation. Camera parameters can be separated into intrinsic parameters such as focal length, and extrinsic parameters which describe the camera's position and orientation in world coordinates. Standard calibration techniques exist to estimate some or all of these parameters (Trucco and Verri 1998). In initial experiments we utilized the OCamCalib calibration toolbox

[3] `http://jbox2d.org`

(a) (b)

Fig. 6 Views of the calibration grid from overhead **(a)** and from the robot **(b)**. (Colour figure available online)

(Scaramuzza et al 2006) which is suitable for the fisheye lenses on our robots. However, we lacked a reliable procedure for estimating the camera's position with respect to the ground plane. Also missing was a technique to identify those parts of the image which corresponded to rays that would actually intersect the ground plane.

The calibration procedure adopted makes no attempt to estimate the camera's intrinsic and extrinsic parameters but instead finds the relationship between image pixels and points on the ground plane directly. In the first phase of calibration pucks are arranged in a 4×7 grid with the robot's gripper centred over the central puck on the bottom row. Views of the calibration grid are shown in Figure 6. The positions of all pucks along with two points on the robot are manually selected. Thus, the relationship between image pixels and points in the ground plane is obtained for these selected points. We interpolate between selected points using the python library, matplotlib[4]. See Figure 2(a) for an example where the calibration data is used to develop the local map from a robot-captured image. Figure 2(b) provides an example of both a synthetic image generated for the simulation and the local map that is further generated from it.

3.4 Performance Metrics

A number of different metrics have been proposed to assess distributed clustering and sorting algorithms. These include number of clusters (Beckers et al 1994; Maris and Boeckhorst 1996), size of the largest cluster (Beckers et al 1994), mean cluster size (Maris and Boeckhorst 1996; Martinoli et al 1999), and spatial entropy (Bonabeau et al 1999). Melhuish et al (1998) introduced a metric for sorting algorithms whereby completion is declared when some fixed percentage of all pucks lie within the largest cluster of their particular colour. We utilize percentage completion as our base metric.

The number of objects (pucks) of type j is n_j. Thus, the total number of pucks is $n = \sum_j n_j$. Equation 2 defined L_j as the largest cluster of type j in the robot's

[4] See `http://matplotlib.org/`. The `griddata` function is used for interpolation.

local map. For analytical purposes we assume access to the global map which describes all clusters with respect to world coordinates. Clusters are defined in the global map in the exact same manner as for a robot's local map. The largest cluster of type j in the global map is denoted as Λ_j and percentage completion (PC) is defined as follows.

$$PC = 100\% \cdot \frac{\sum_j \text{size}(\Lambda_j)}{n} \tag{6}$$

One important metric derived from percentage completion is the number of steps required to reach a particular level of completion. We track the first time the level of percentage completion reaches a target threshold. Since 100% completion is not reached by all tested methods, we use a less ambitious threshold of 50% completion. We refer to this measure of steps-to-completion as STC. It is possible for a sorting method to reach the target completion threshold, but then to degrade in performance. To capture this phenomenon we use a time-weighted sum over percentage completion—also known as time-weighted completion or TWC.

$$TWC = \frac{1}{100} \frac{\sum_{t=0}^{t=n_t} t \cdot PC}{\sum_{t=0}^{t=n_t} t} \tag{7}$$

where n_t is the number of time steps for the experiment.

4 Results

The purpose of our experiments is to contrast the performance of CacheCons with benchmark methods (BHD and ProbSeek). The first set of experiments described below are carried out in simulation, but we validate the performance of CacheCons and ProbSeek in the standard configuration on physical robots. In addition to the results below, an online supplementary material paper has been prepared to study the performance of CacheCons and ProbSeek when key task parameters are varied and also to consider the impact of external disturbances which may shift or randomize the positions of pucks.

Experiments conducted in our simulation consist of a block of trials, with each trial lasting 100,000 time steps and starting from a different uniform distribution of pucks and robots. Trials were executed on a variety of modern PC's that comprise our computing cluster with an average duration of 36 minutes. For setting parameters we used blocks of 5 trials (as in the next section), but for comparisons between methods we used blocks of 20 trials.

4.1 Standard Configuration

The standard configuration for our experiments has the following characteristics:

- 40 pucks of 2 types (20 red, 20 green)
- Environment at original scale (scale factor 1)
- 4 robots

The design choices for all three sorting algorithms were optimized for the standard configuration using independent parameter searches. All parameters which are common to the three methods are set to the same values (e.g., all three spend five iterations in their respective `BACKUP` states). An important parameter for ProbSeek is k_2 which governs the probability of targeting a cluster for deposit. We tested values in the set $\{1, 2, 4, 8, 16\}$. The highest average time-weighted completion (TWC) occurred for $k_2 = 8$ which was therefore selected for subsequent experiments.

4.2 Number of Object Types

The number of object types is clearly an important dimension in the patch sorting problem. In the standard configuration the 40 pucks are divided into 2 types. We tested 1, 2, 4, and 8 different object types, with the total number of pucks maintained at 40 and the number of each type evenly divided. Figure 7 shows the percentage completion averaged over 20 trials.

4.2.1 One Object Type

When there is only one object type, the sorting problem becomes identical to the clustering problem. In this case the methods tested included BHD, ProbSeek, and CacheCons. The upper left plot in Figure 7 suggests a substantial difference in performance between these three methods. Snapshots from the simulation shown in Figure 8 further support the observation that CacheCons very quickly reaches and maintains convergence, while ProbSeek and BHD are unable to reach the same level of completion within the allotted time. This observation is further confirmed with statistical tests. First, we considered the two metrics, TWC and STC. Figure 9 shows the distribution of TWC and STC values. The D'Agostino & Pearson normality test indicates that the TWC data for ProbSeek and CacheCons do not follow a normal distribution, however the STC data for all three methods does appear to follow a normal distribution. Focusing our analysis on STC allows us to use the standard parametric tests which have greater statistical power than non-parametric tests. We performed a repeated measures ANOVA which indicated that the mean STC values are all drawn from different distributions ($p < 0.0001$). Applying Tukey's multiple comparisons test revealed that all differences were significant. That is, the mean STC for BHD is significantly greater than for ProbSeek and CacheCons and the mean STC for ProbSeek is significantly greater than for CacheCons (in all cases $p < 0.0001$).

4.2.2 Multiple Object Types

Figure 7 also shows the percentage completion averaged over 20 trials for 2, 4, and 8 object types. Figure 10 shows snapshots for 4 object types. Note that BHD is not included in these results since it is only applicable for a single object type. Paired t-tests on the STC results from each trial indicate that the number of steps before 50% completion is significantly greater for ProbSeek than for CacheCons for 2, 4, and 8 object types ($p < 0.0001$, $p = 0.0003$, and $p < 0.0001$).

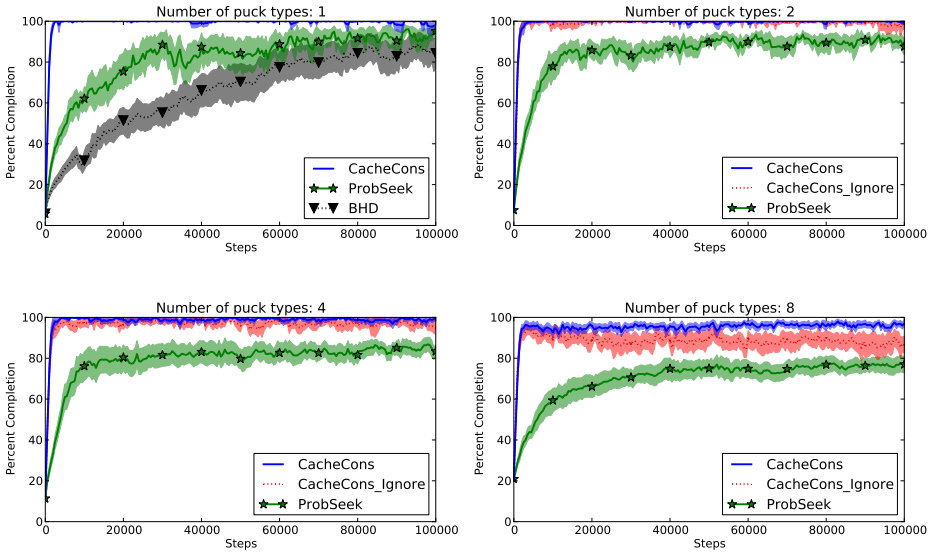


Fig. 7 Plots of percentage completion versus time step while varying the number of puck types. The mean value for each data set is indicated by a heavy trace, surrounded by a shaded region. The extent of the shaded region is ±1.96 standard errors. Thus, these shaded regions correspond to 95% confidence intervals for the mean. (Colour figure available online)

To illustrate the effect of incorporating a mechanism for cache separation we include results for CacheCons when cache separation is ignored, indicated as `CacheCons_Ignore` in the figure. CacheCons exhibits excellent performance for 2 and 4 object types, but some degradation is apparent with 8 object types. However, if not for the cache separation strategy the performance would be worse. As the number of object types increases, so does the amount of spatial interference. This interference often takes the form of 'unintentional' destruction of clusters—for example, when a robot is backing up after completing a deposit, or when two robots are attempting to avoid each other. Incorporating a cache separation strategy reduces this interference, but cannot eliminate it entirely. Figure 8 presents an example of spatial interference with only one object type. Notice the second trial for CacheCons which is shown in the bottom-most row of the figure. What had been a single stable cluster has been divided in two by time step 100,000. At 50,000 time steps this cluster was whole, but concave on its left side. The VFH+ obstacle avoidance algorithm may drive a robot into such a concavity. Since we do not permit the robots to reverse course (because this would cause a robot to drop its puck) there is no option but to move forward and break the cluster. Such clusters will eventually be reformed into one, although in this case the trial reaches its conclusion before this can occur.

4.2.3 Convergence of Caches

The experiments presented so far have focused on the pucks, but for CacheCons we can also look at the cache points and how they converge over time. In Figure 8 we see that all cache points have already converged by 1000 time steps. In

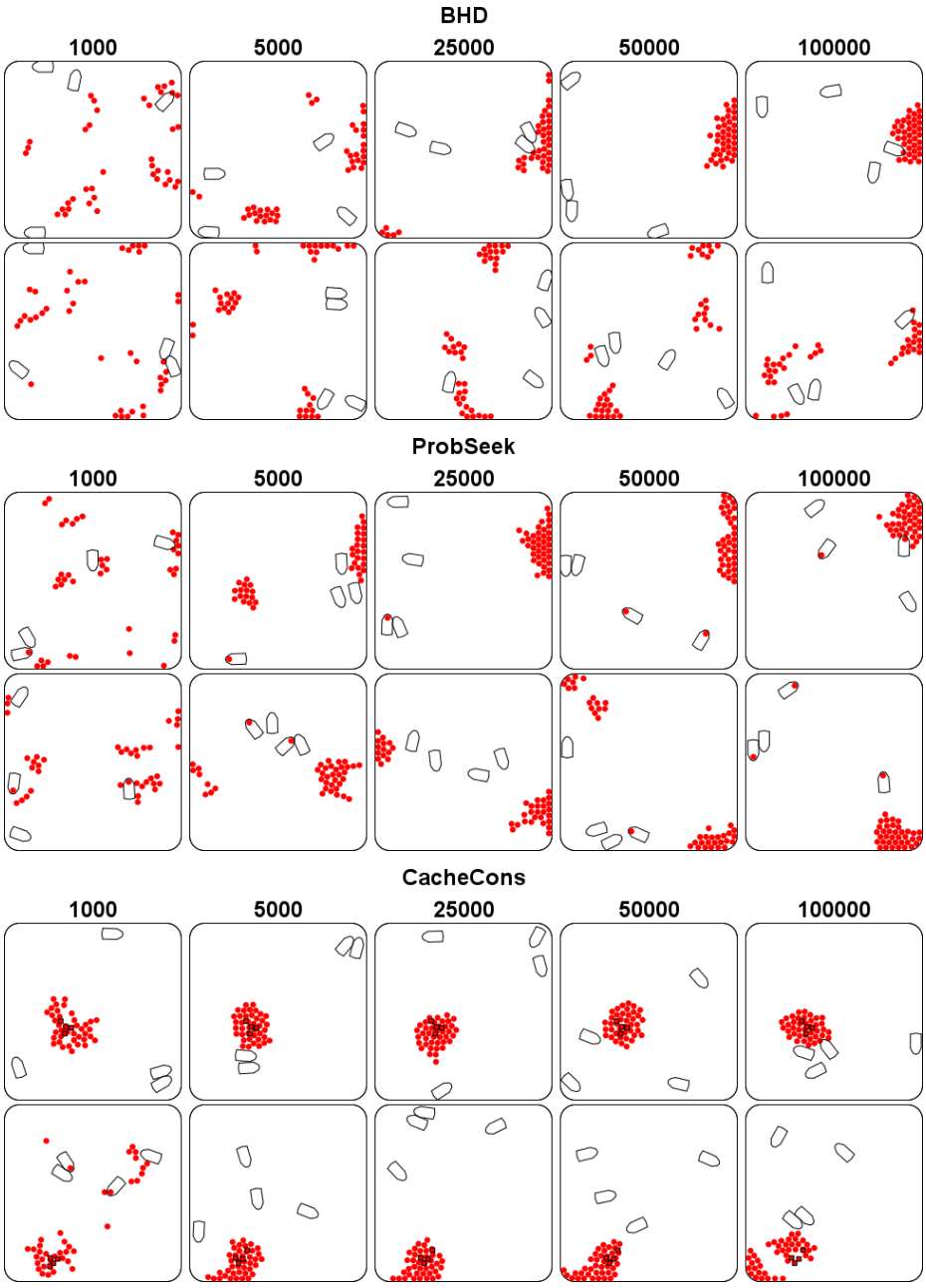
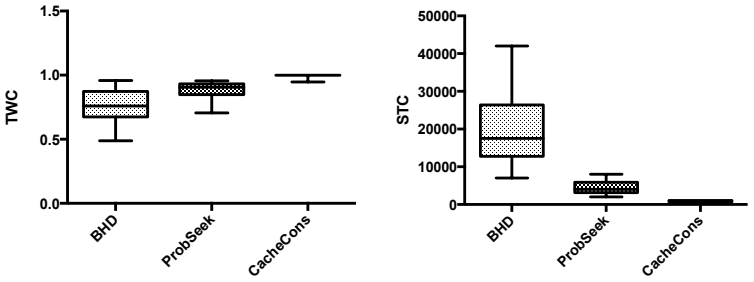


Fig. 8 Snapshots of the performance of BHD, ProbSeek, and CacheCons operating on 40 pucks of one type. Snapshots from the first two of 20 trials are shown in each row. Cache points for each robot are shown as filled squares. The position of the robots at the time of each snapshot is also indicated. Note that in some cases the robots are carrying pucks. (Colour figure available online)



(a) (b)

Fig. 9 Box and whisker plots showing the distribution of **(a)** TWC and **(b)** STC for 20 trials of BHD, ProbSeek, and CacheCons on a single object type. The boxes extend from the 25^{th} to the 75^{th} percentiles with whiskers extending to the smallest and largest values.

Figure 10 for `CacheCons_Ignore` (middle of the figure) we have the same result, the cache points for each object type have converged by 1000 time steps. However, for `CacheCons` (bottom) convergence occurs somewhere between 1000 and 5000 time steps.

To measure the convergence of cache points for each object type we compute the cache point centroid, then determine the distance of each cache point from that centroid. The variance of these distances is our measure of convergence for a single object type. Figure 11 shows the progression of this variance, averaged over all object types and across all 20 trials in the standard configuration. After a very brief period of growth, the variance decreases almost monotonically, then stabilizes.

Another perspective on the convergence of cache points is obtained by comparing the performance of CacheCons with an informed variant of the algorithm with preset cache points. For the informed variant, referred to as CacheCons_Informed, the cache points for all robots are initialized to the same positions and are never reassigned. Figures 12(a), (c), and (e) plot the percentage completion achieved by CacheCons against the informed variant for 2, 4, and 8 puck types. In each case the preset cache points are taken along a circle with a diameter 6/8 of the arena width. Figures 12(b), (d), and (f) show snapshots of the performance of both CacheCons and CacheCons_Informed at 2,000 time steps. Paired t-tests on the STC results from each trial indicate that the number of steps before 50% completion is significantly greater for CacheCons than for CacheCons_Informed for 2 and 4 object types ($p = 0.001724$, $p = 0.00252$) but not for 8 object types ($p = 0.1649$). For 8 object types there is substantial spatial interference between clusters, in that delivering a puck of one type requires avoiding the 7 (or more) other clusters as well as other robots. Thus, it may be the case that a non-circular arrangement might lead to better performance for CacheCons_Informed. While the reduced time for completion is observable (if not statistically significant) in all three cases, it is notable that the performance gap is relatively narrow.

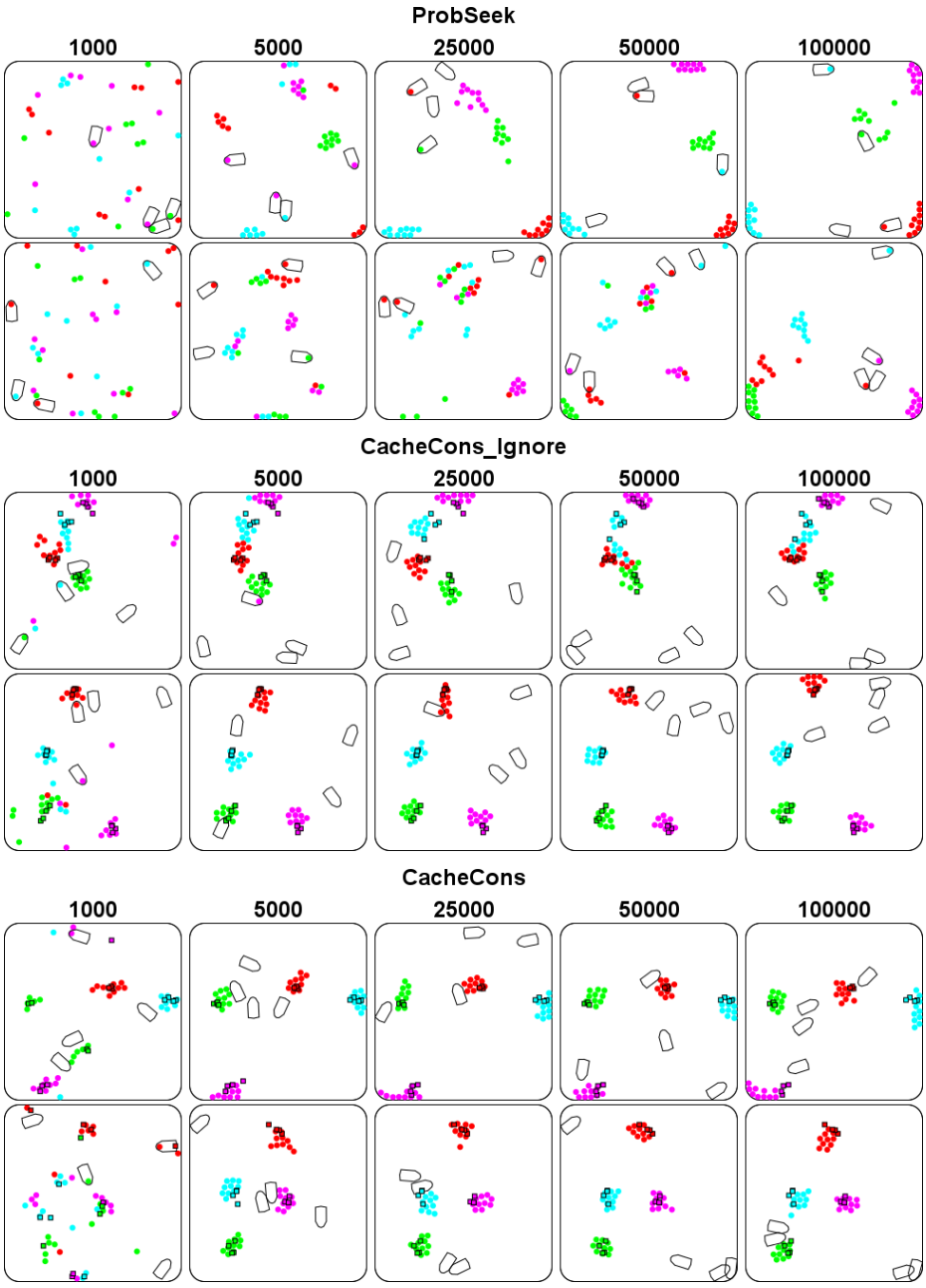


Fig. 10 Snapshots of the performance of ProbSeek and CacheCons operating on 4 types of pucks. Note in the bottom row that two robots have become entangled. We have worked to eliminate this possibility in our simulation but it still occurs in certain rare conditions. (Colour figure available online)

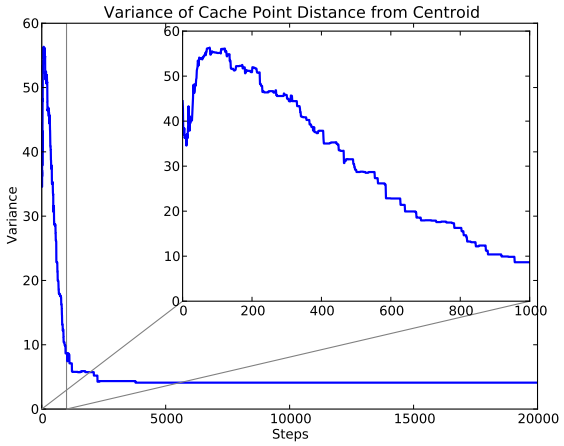


Fig. 11 Variance of cache point distance from the centroid of all cache points of the same object type, plotted for CacheCons in the standard configuration and averaged over both object types.

4.3 Validation on Physical Robots

We have tested the performance of ProbSeek and CacheCons on our SRV-1 robots to provide some confidence that the main ideas underlying these algorithms are applicable in a real-world context. While we have worked to eliminate significant discrepancies between the simulation and physical setup, there are some important differences:

Localization: Localization is achieved by tracking AprilTag visual fiducials (Olson 2011) from an overhead camera.

Image resolution: A resolution of 160×120 is used for the simulation, but more clarity was required for real-world imaging so a resolution of 320×240 was adopted.

Length of trials: Each trial for our physical experiments lasted 5000 time steps, as opposed to 100,000 in simulation. This corresponded to an average trial length of 2 hours, 5 minutes.

Number of trials: Three trials were executed for our physical experiments, as opposed to 20 in simulation.

One reason for the reduced length and number of trials were the technical difficulties encountered in communication with the robots. The SRV-1 robots were executing their standard firmware which provides a set of commands that can be sent using Wi-Fi (e.g., commands to set motor speeds and to download the current image). Unfortunately, we experienced occasional glitches which caused the robots to become unresponsive. For both methods two problem-free trials were completed along with one trial in which two out of four robots became unresponsive. Nevertheless, the operation of the other robots continued and the impact on performance of these failures appears to be mild.

Figure 13 shows the percentage completion of ProbSeek and CacheCons averaged over three trials. Although we lack the data for a full statistical analysis

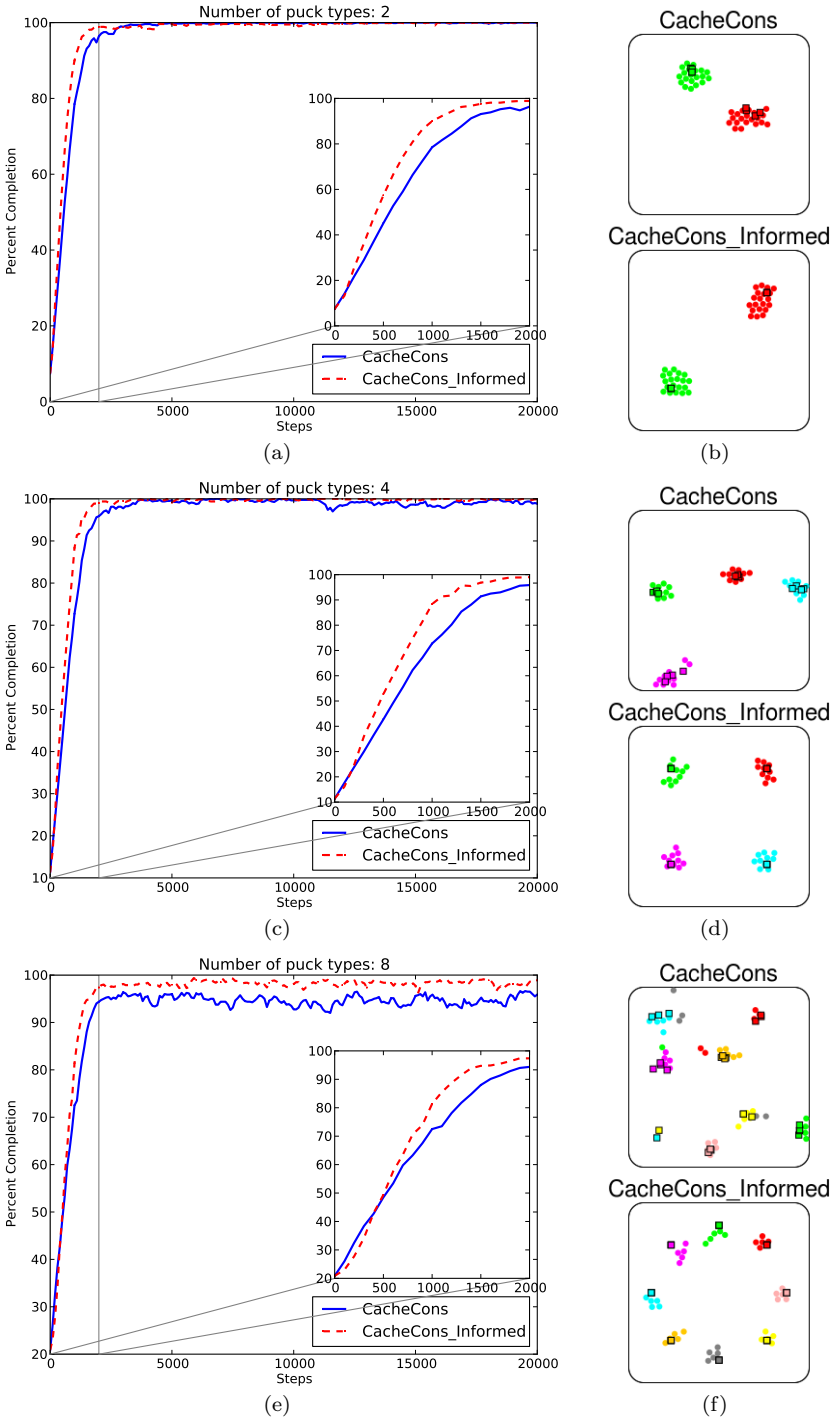


Fig. 12 (a), **(c)**, and **(e)**: Percentage completion plots for CacheCons versus CacheCons_Informed for 2, 4, and 8 object types. **(b)**, **(d)**, and **(f)**: Snapshots captured at 2,000 time steps. (Colour figure available online)

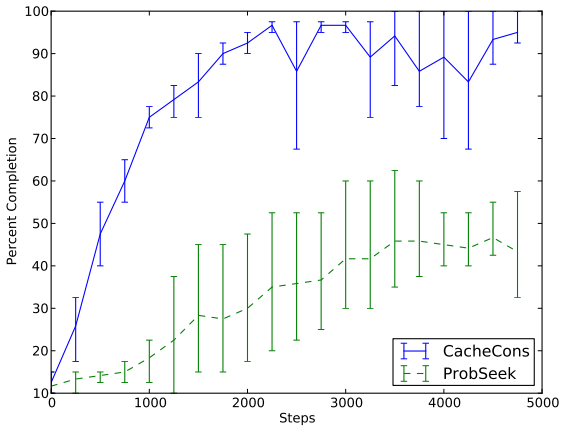


Fig. 13 Percentage completion of ProbSeek and CacheCons averaged over three trials. The error bars show minimum and maximum percentage completion.

there is a clear distinction in performance between these two algorithms. Figures 14 and 15 provide snapshots for all three trials of ProbSeek and CacheCons, respectively. The results on ProbSeek demonstrate gradual sorting, but the algorithm is unable to reach the desired state of one homogeneous cluster per object type within 5,000 time steps. For CacheCons this ideal of 100% completion is reached in one trial (shown on the bottom row of Figure 15). 92.5% completion is reached in the other two trials. Note that Figure 15 is annotated with the cache points established by each robot. For two out of three trials the cache points have converged by 1000 time steps.

Overhead videos showing all live trials are available as supplementary online material.

4.4 Supplementary Results

The online supplementary material paper provides additional results on the performance of CacheCons and ProbSeek when key task parameters are varied. In the first experiment the number of pucks is varied. The gap in performance between CacheCons and ProbSeek appears to widen as the number of pucks is increased. Increasing the size of the environment also appears to impact ProbSeek more profoundly. In another experiment we modified the number of robots and identified the number at which peak performance was achieved.

We also studied the impact of external disturbances of two types: shifts of established clusters, or re-insertion of all pucks at random positions. These disturbances are intended to model situations where a swarm of sorting robots is interfaced with other agents that operate on the same materials (for example, in the context of recycling). ProbSeek was found to be unaffected by cluster shifts since it maintains no memory of cluster locations. In its standard form CacheCons exhibits a drop in performance in response to these shifts. However, we introduced a variant which simply erases its own memory periodically and this variant proved to be more robust to shift disturbances. For random re-insertion disturbances,

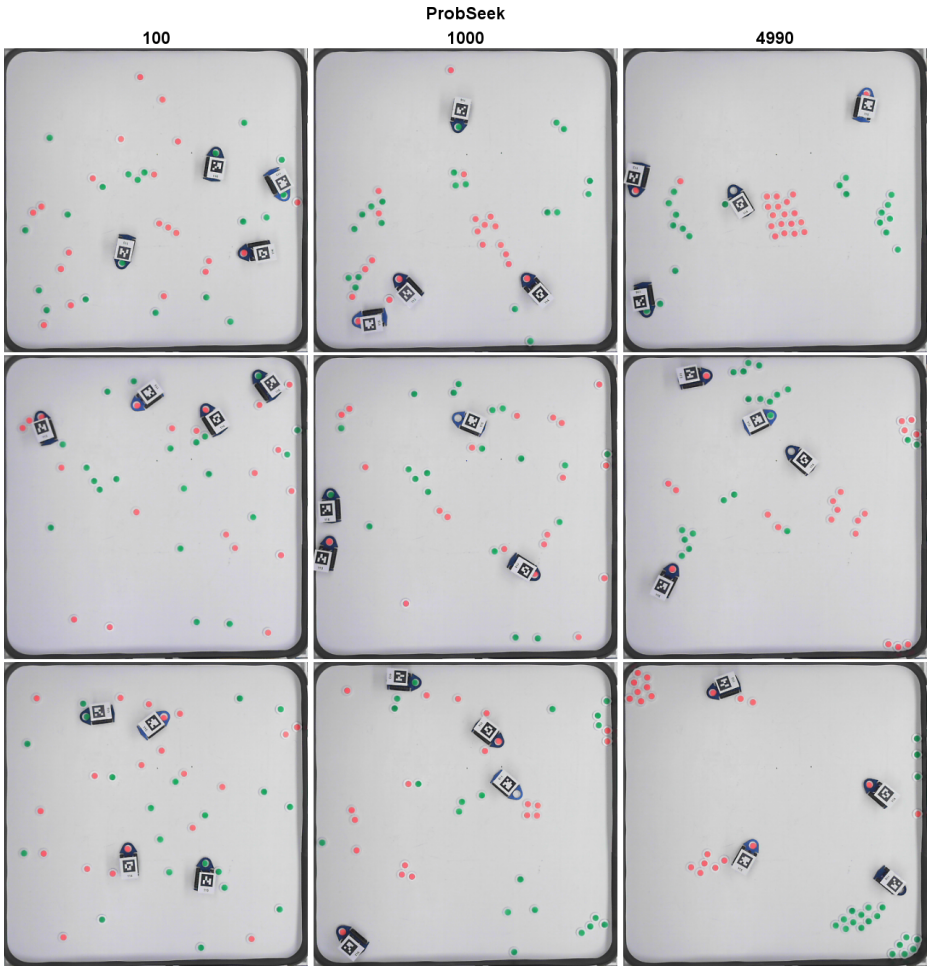


Fig. 14 Snapshots from all three live trials of ProbSeek. (Colour figure available online)

all methods showed a similar impact on performance, although both CacheCons variants could recover much more quickly than ProbSeek.

5 Discussion

It has not yet been demonstrated that swarm robotic techniques can be applied in industrial applications. One possible obstacle (or at least, perceived obstacle) is their rate of convergence. For example, Beckers et al (1994) reported convergence times that often extended to several hours for robot swarms of size 1-5 clustering 81 pucks in a 250×250 cm^2 arena. Others have reported roughly similar convergence times using robots controlled in a similar manner (Maris and Boeckhorst 1996; Melhuish et al 1998; Martinoli et al 1999). Requirements on completion times will differ from one application to the next, but we suspect that faster convergence will

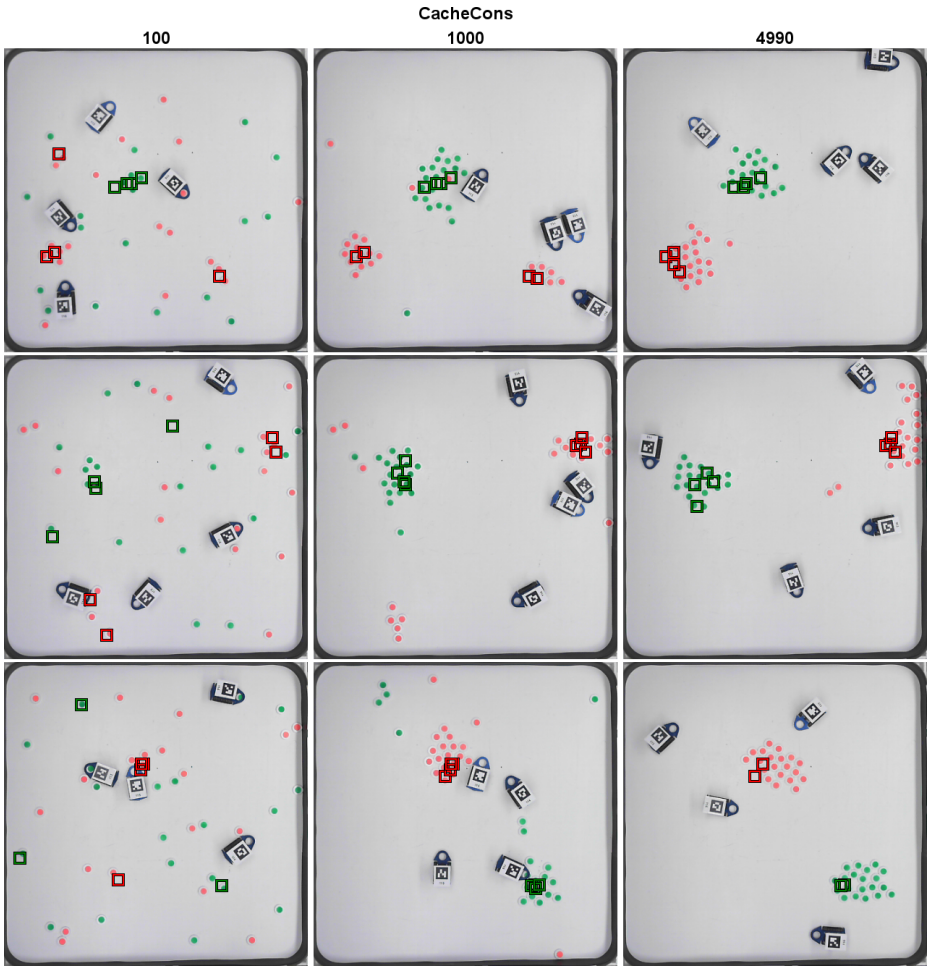


Fig. 15 Snapshots from all three live trials of CacheCons. Coloured squares indicate the cache points of each robot. (Colour figure available online)

generally be required. Indeed, even strong proponents of swarm intelligence have expressed scepticism about the applicability of these ideas to robotics:

> Although designing robots that implement the process of distributed clustering may not be particularly useful in itself in the context of engineering, several groups have done it [...] because it is *conceptually* useful. (Bonabeau et al 1999, chapter 4).

The question remains whether swarm robotics can go beyond conceptual utility. Our physical experiments lasted 125 minutes on average. However, we believe dramatic improvements in speed are well within reach. In our case, an overhead image was captured and processed; images were downloaded from each robot in turn in a crowded Wi-Fi environment; the image processing was conducted on one computer (taking approximately 20-30 ms per robot); and motor commands were

executed with built-in delays during which no other processing was possible. The estimated forward speed of our robots during this stop-and-go process is only 2.5 cm / sec which is well below the published speeds achievable for similar robots.

We are therefore encouraged that future implementations of the cache consensus algorithm will exhibit sufficiently rapid convergence for practical applications. We have clearly demonstrated superior performance to the BHD and ProbSeek algorithms which exemplify the approaches described so far in the literature. The disadvantage of our method is that it requires some means of localization. Thus this method will be unsuitable for robots below a certain cost and complexity threshold. However, we believe this threshold is rather low. As mentioned in Section 1.1 there are a wide array of technologies and associated algorithms to support localization. In the next section we discuss our intention to test cache consensus using laser-based and egocentric localization strategies.

6 Future Work

We see a number of directions for future work. We are currently developing new robots which move beyond the main limitations of the SRV-1's used here. There are also a number of interesting experiments that remain to be conducted to probe the parameter space of our algorithm, as well as possible enhancements to pursue. Finally, we are investigating the recycling industry as a possible area of application.

The robots currently under development will utilize laser range finders for perception of other robots and the walls of the environment. With circular robots and planar walls we expect a relatively straightforward segmentation which will allow other robots to be avoided and the walls to be used for map-based localization. Algorithms for laser-based localization have become quite mature and have been incorporated into popular robotics software packages such as ROS (Quigley et al 2009). The objects to be sorted would lie below the level of the laser range finder and would be imaged using a standard camera. For future implementations we are considering the use of depth cameras to perceive and classify real-world objects.

Once a more suitable swarm of robots has been developed we plan to assess the impacts of localization error and field-of-view limitations. With regards to localization error, we are particularly interested in whether the cache consensus idea can be applied on relatively inexpensive robots that use only egocentric sensors such as wheel encoders and inertial measurement units. Since the robots frequently return to their caches it should be feasible to correct for odometric drift by periodically resetting the caches' position estimates. Communication between robots may also prove useful in allowing position estimates to be collaboratively refined as in the *social odometry* framework (Gutiérrez et al 2010).

The limited field-of-view of most optical sensors places a limit on the maximum cluster size that can be perceived. It would be very interesting to modulate field-of-view as an experimental parameter and determine whether the system can still converge to a single cluster. We suspect that as cluster size perception saturates, the overall operation of the system would begin to resemble the original Deneubourg et al (1990) model, with the exception that all remaining clusters would be at least as large as the maximum perceivable cluster size. Alternatively, we could investigate techniques to estimate the size of even very large clusters by circumnavigating those clusters and calculating their areas.

In section 1 as well as in the online supplementary material paper we discussed possible applications to the recycling industry. As mentioned, mechanisms for interfacing a distributed robot sorting algorithm with a recycling plant's input and output streams would be required. Another necessary component would be the ability to classify objects. We consider visual classification the most promising technique due to the low cost and weight of optical sensors and the wealth of existing research on this topic. Separation of polycoat and PET (Polyethylene Teraphthalate) plastics has been demonstrated using machine vision techniques (House et al 2011). In general, visual classification is possible if sufficient reference images are available. Classification becomes difficult when objects appear together and when one considers the transparency of some recyclable materials. Distinct objects which are mechanically coupled can be segregated by partitioning the scene into rigid members with estimated kinematic joints between them (Katz et al 2010).

7 Conclusions

We have presented and explored a new algorithm for distributed robotic sorting which builds upon earlier biologically-inspired clustering and sorting methods but achieves a new level of performance by incorporating localization.

The promise of swarm robotics is that adaptive and potentially useful behaviour may be achieved by a collective of simple individuals. It is helpful to question how minimalistic the capabilities of these agents should be (Sharkey 2007). We have shown here that the addition of localization capability can significantly enhance distributed sorting performance. Since spatial navigation appears to be of profound importance to the social insects, it seems logical to investigate its use in distributed robotic systems that are already designed according to biological inspiration.

Acknowledgements Thanks to Paul Gillard for helpful discussions and support in diagnosing myriad hardware problems. WB gratefully acknowledges funding from NSERC under Discovery Grant RGPIN 283304-2012

References

Beckers R, Holland O, Deneubourg JL (1994) From local actions to global tasks: Stigmergy and collective robotics. In: Artificial Life IV, MIT Press, Cambridge, MA, pp 181–189

Beekman M, Sword GA, Simpson SJ (2008) Biological foundations of swarm intelligence. In: Blum C, Merkle D (eds) Swarm Intelligence, Natural Computing Series, Springer Berlin / Heidelberg, pp 3–41

Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York, NY

Brooks R (1986) A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2(1):14–23

Cartwright B, Collett T (1983) Landmark learning in bees. Journal of Comparative Physiology A 151:521–543

Collett T, Collett M (2002) Memory use in insect visual navigation. Nature Reviews Neuroscience 3:542–552

Deneubourg JL, Goss S, Franks N, Sendova-Franks A, Detrain C, Chrétien L (1990) The dynamics of collective sorting robot-like ants and ant-like robots. In: First Int. Conf. on the Simulation of Adaptive Behaviour, MIT Press, Cambridge, MA, pp 356–363

Dudek G, Jenkin M (2010) Computational Principles of Mobile Robotics, 2nd edn. Cambridge University Press, New York, NY, USA

Gonzalez R, Woods R (2002) Digital Image Processing, 2nd edn. Prentice Hall, Upper Saddle River, NJ

Gordon D (1996) The organization of work in social insect colonies. Nature 380:121–124

Gutiérrez A, Campo A, Monasterio-Huelin F, Magdalena L, Dorigo M (2010) Collective decision-making based on social odometry. Neural Computing and Applications 19(6):807–823, DOI 10.1007/s00521-010-0380-x, URL http://dx.doi.org/10.1007/s00521-010-0380-x

House B, Capson D, Schuurman D (2011) Towards real-time sorting of recyclable goods using support vector machines. In: Sustainable Systems and Technology (ISSST), 2011 IEEE International Symposium on, IEEE Xplore, pp 1–6, DOI 10.1109/ISSST.2011.5936845

Katz D, Orthey A, Brock O (2010) Interactive perception of articulated objects. In: 12th International Symposium of Experimental Robotics, Springer Berlin / Heidelberg, pp 1–15

Kazadi S, Abdul-Khaliq A, Goodman R (2002) On the convergence of puck clustering systems. Robotics and Autonomous Systems 38(2):93–117

Maris M, Boeckhorst R (1996) Exploiting physical constraints: heap formation through behavioral error in a group of robots. In: IEEE/RSJ International Conference on Robots and Systems (IROS), IEEE Xplore, vol 3, pp 1655–1660

Martinoli A, Ijspeert A, Gambardella L (1999) A probabilistic model for understanding and comparing collective aggregation mechanisms. In: Advances in Artificial Life, Lecture Notes in Computer Science, vol 1674, Springer Berlin / Heidelberg, pp 575–584

Melhuish C, Holland O, Hoddell S (1998) Collective sorting and segregation in robots with minimal sensing. In: 5th Int. Conf. on the Simulation of Adaptive Behaviour, MIT Press, Cambridge, MA

Melhuish C, Wilson M, Sendova-Franks AB (2001) Patch sorting: Multi-object clustering using minimalist robots. In: Advances in Artificial Life - Proceedings of the 6th European Conference on Artificial Life (ECAL), Springer

Melhuish C, Sendova-Franks AB, Scholes S, Horsfield I, Welsby F (2006) Ant-inspired sorting by robots: the importance of initial clustering. Journal of the Royal Society: Interface 3(7):235–242

Möller R, Krzykawski M, Gerstmayr L (2010) Three 2D–warping schemes for visual robot navigation. Autonomous Robots 29(3):253–291

Olson E (2011) AprilTag: A robust and flexible visual fiducial system. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), IEEE Xplore, pp 3400–3407

Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng AY (2009) ROS: An open-source robot operating system. In: ICRA workshop on open source software

Scaramuzza D, Martinelli A, Siegwart R (2006) A toolbox for easily calibrating omnidirectional cameras. In: IEEE/RSJ International Conference on Robots and Systems (IROS), IEEE Xplore, pp 5695–5701

Seeley TD (2010) Honeybee Democracy. Princeton University Press, Princeton, NJ

Sendova-Franks AB, Scholes SR, Franks NR, Melhuish C (2004) Brood sorting by ants: two phases and differential diffusion. Animal Behavior 68:1095–1106

Sharkey AJ (2007) Swarm robotics and minimalism. Connection Science 19(3):245–260

Siegwart R, Nourbakhsh I, Scaramuzza D (2011) Introduction to Autonomous Mobile Robots, 2nd edn. MIT Press, Cambridge, MA

Thrun S, Burgard W, Fox D (2005) Probabilistic Robotics. MIT Press, Cambridge, MA

Trucco E, Verri A (1998) Introductory Techniques for 3-D Computer Vision. Prentice Hall

Ulrich I, Borenstein J (1998) VFH+: Reliable obstacle avoidance for fast mobile robots. In: IEEE International Conference on Robotics and Automation (ICRA), IEEE Xplore, vol 2, pp 1572 – 1577

Vardy A (2012) Accelerated patch sorting by a robotic swarm. In: Canadian Conference on Computer and Robot Vision, IEEE Xplore, pp 314–321, URL http://youtu.be/BVDy2qOG9OM

Vardy A, Möller R (2005) Biologically plausible visual homing methods based on optical flow techniques. Connection Science 17(1/2):47–90

Verret S, Zhang H, Meng MQH (2004) Collective sorting with local communication. In: IEEE/RSJ International Conference on Robots and Systems (IROS), IEEE Xplore, vol 3,

pp 2687–2692
Wang T, Zhang H (2003) Multi-robot collective sorting with local sensing. In: IEEE Intelligent Automation Conference (IAC)