A Scalable Architecture for Multi Agent Vision Based Robot Scavenging

Kamil Wnuk, Brian Fulkerson, Jeremi Sudol
Computer Science Department
University of California, Los Angeles, CA, 90095

Abstract

This paper outlines the technical details and presents early results from our multi agent robot scavenging system, as demonstrated at the 2006 AAAI Robot Exhibition and Scavenger Hunt Competition. Our system is designed to be easily scalable in the number of agents, which are currently limited to Evolution ER1 laptop robots capable of monocular vision. Each individual robot possesses the ability to localize itself, recognize a set of objects, and communicate with peer robots to coordinate exploration.

Overview

The robot scavenger problem, as posed in the 2006 AAAI Robotics Competition, is to search the conference environment for a list of objects and report their position according to some map of the environment. For convenience, teams were provided with floor plans for the competition spaces. However, since the conference environment is highly dynamic, with people constantly moving about, as well as exhibits, posters, and other typically characteristic environmental items often shifting, agents acting in the space require robust reasoning algorithms in addition to simply utilizing floor plans. In addition, conference spaces are large and typically require long periods to explore for slow moving robots.

In an attempt to minimize exploration time, we have chosen a scalable multi agent approach to the scavenging problem, that focuses strongly on the spatial reasoning, planning, and object recognition elements of the task. We are also particularly interested in exploring low cost robotics and the potential of vision as a primary sensing modality, which has led to our choice of platform. We believe that vision could prove to be a flexible sensor, able to be more robust in a highly dynamic environment.

Platform

Our chosen platform is the ER1 robot from Evolution Robotics (Fig. 1). It serves essentially as a laptop peripheral, connecting all sensors and motors through USB. The robot is equipped with differential drive and a single camera, and on its own sells for under $200, placing it well in the low cost robotics bracket. We have augmented each platform with two inexpensive ($11 each) short-range infrared sensors, which we use to avoid hitting unanticipated obstacles. For low level robot control we adapted the ER1 Python driver developed at Harvey Mudd College, by adding multithreaded serial communication, drivers for our infrared sensors, and a simple simulator.

Figure 1: The Evolution ER1 platform.

The Highly Organized Bunch Of Scavengers (HOBOS) Architecture

The philosophy driving our design is that each robot should be able to reason and act completely on its own, and an arbitrary number of robots should be able to cooperate by simply running a duplicate of the same software.

Each member of the HOBOS system has five key capabilities, which will be described in detail below. These include localization, path planning, optimal deployment, object recognition, and mobile ad hoc peer-to-peer networking. Each capability was designed and tested as a separate component, and final integration is still in progress at the time of this writing.

Localization

A key component of any agent desiring to do spatial reasoning is having some representation of its surrounding environment and being able to locate itself in that environment based on sensory input.

We have chosen to leverage the provided floor plan data to implement a vision based Markov Localization (Fox,

Burgard, & Thrun 1999) method that incorporates structure from motion techniques to accurately estimate position (Kosecka, Li, & Yang 2005). This approach utilizes Scale Invariant Feature Transform (SIFT) keypoints (Lowe 2004) to represent views in the environment and match locations. A Hidden Markov Model (HMM) is then incorporated to improve localization accuracy by adding constraints enforcing motion only between adjacent locations. Once a location is matched, structure from motion methods are used to estimate the offset from the matched view. The algorithm requires an initialization stage in which an environmental map is gathered as a set of captured views before the robot can localize.

SIFT Keypoints A necessary background topic for vision based localization is the SIFT method for detecting keypoints, developed by David Lowe.

The SIFT keypoint detector can be briefly summarized in four steps. The first is detection of extrema in scale space, which is performed by finding maximum and minimum points in difference of gaussian images. In the second stage of detection, the location and scale of each keypoint is determined and low contrast or badly localized keypoints are removed. In the third stage of SIFT, each keypoint is assigned an orientation based on the surrounding image gradients. Finally, a 128 bit descriptor is created for each keypoint based on the local image orientation histograms. This incorporation of scale and orientation information into the keypoint descriptor is what enables good matching of keypoints across scale and rotation.

Given two images, corresponding keypoints are matched according to the Euclidean distance between two descriptors. Two points are considered a match only if the ratio between the second best match and the first best match is greater than 0.6.

Environmental Representation The environment is represented as a set of *locations*. Each location consists of a number of *views* (Fig. 2), and each view is stored as a set of SIFT keypoints extracted from an image captured at a recorded physical location.

The sequence of images captured on an initial mapping run of the robot is partitioned automatically into locations. A new location aggregates views until there is an insufficient number of matching keypoints between sequentially captured images. Two sequential views are considered to be part of the same location if at least 4 percent of keypoints are matched and if there are at least 6 matched keypoints (Fig. 3). As expected, this constructs a representation where long hallways, for example, are categorized as a single location due to the continuity of stable keypoints throughout the image sequence. This feature is nicely leveraged to efficiently incorporate a Hidden Markov Model into the algorithm, as described next.

Hidden Markov Model Once the environmental representation is constructed the robot can localize. The initial

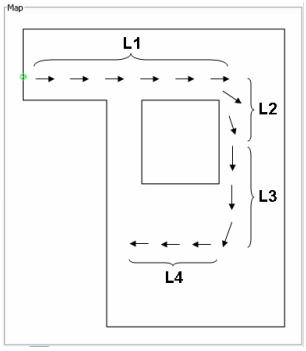

Figure 2: The representation of a partial model of the UCLA Vision Lab. The arrows indicate the location and orientation of captured views, and the labels indicate their grouping into locations.

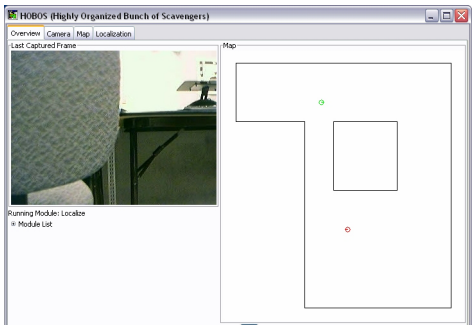

Figure 4: This figure demonstrates the fundamental problem of the naive view matching approach. Here an unanticipated chair occludes the camera, blocking characteristic keypoints and causing the robot, whose actual position is shown by the red circle at the bottom of the map, to localize itself completely incorrectly (as shown by the green circle at top).

naive approach to localization is to simply find the stored view in the representation that best matches what the robot is currently seeing, according to the number of SIFT keypoints matched. Thus the view with the most matching keypoints is the view most likely to be close to the current location of the robot. This algorithm performs relatively well even with partial occlusions from humans walking in front of the robot at a reasonable distance.

The fundamental problem with this approach is that when distant views look similar or a majority of characteristic keypoints are not visible, as during a major occlusion, the best matching view can be located very far from the robot's actual position (Fig. 4). Thus the natural thing is to attempt to exploit temporal information, such as the last estimated position of the robot, in determining the most likely current position.

In order to prevent huge errors in location recognition, Kosecka et. al. introduce a HMM to represent neighborhood constraints in the map, and essentially adapt the naive keypoint matching technique into a Markov Localization algorithm. The environmental representation explained earlier

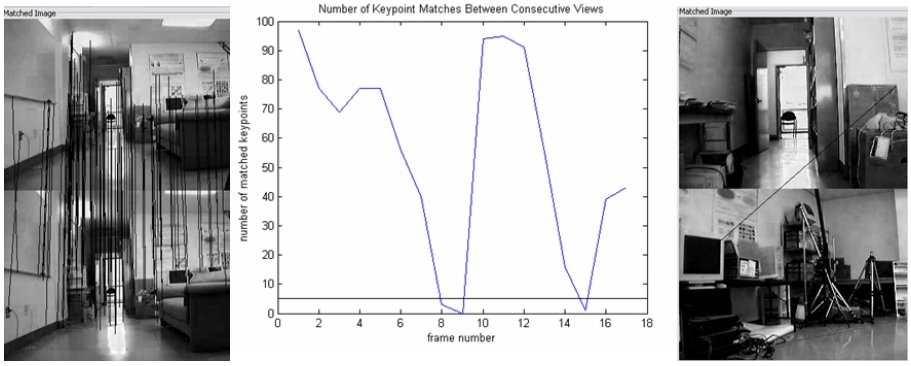

Figure 3: The graph in the center shows the results for SIFT keypoint matching in sequential frames of an 18 frame dataset gathered in the UCLA Vision Lab. The black line indicates the threshold below which a view is categorized as a new location. The left and right images show matching between sequential frames, demonstrating good and bad matches respectively.

is particularly conducive to a Markov Localization scheme because it partitions the space into large areas rather than a fine scale uniform grid. This greatly reduces the computation time of the localization step, since there are fewer locations to consider. For this localization method, it is sufficient to match to a view within the correct location, as that will likely provide a good enough correspondence to perform a structure from motion step to determine the camera offset of the robot's current view from the best match in the representation.

With the Markov Model the most likely robot location is decided by maximizing location likelihood:

$$P(L_t = l_i|o_{1\ldots t}) \propto p(o_t|L_t = l_i)P(L_t = l_i|o_{1:t-1}). \quad (1)$$

In the above equation, L_t is the location at time t; l_i is the ith location in the set of all locations; and o_t is the data observed by the robot at time t. The posterior represents the most likely location of the robot conditioned on all of the observations made so far. The first term on the right hand side of equation (1) is referred to as the observation likelihood and defines the probability that an observation came from a particular location. Again borrowing Kosecka et. al.'s notation, the observation likelihood is computed as the cardinality of the correspondence set between the input image and the ith location $C(i)$, normalized by the sum of all correspondence sets.

$$p(o_t|L_t = l_i) = \frac{C(i)}{\sum_j C(j)}. \quad (2)$$

Where $C(i)$ is defined as the maximum correspondence with a constituent view of the ith location. Finally, the last term of equation (3) is expanded to represent the neighborhood constraints,

$$P(L_t = l_i|o_{1:t-1}) = \sum_j^N A(i,j)P(L_{t-1} = l_j|o_{1:t-1}), \quad (3)$$

where

$$A(i,j) = P(L_t = l_i|L_t = l_j), \quad (4)$$

and $P(L_t = l_i|L_t = l_j)$ is the transition probability, set to 1 if two locations are neighbors of each other. After construction, the matrix A is normalized by rows.

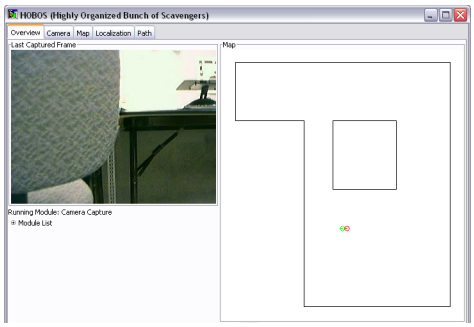

Figure 5: Once the HMM is incorporated in the localization, the correct location is matched for the scenario in Fig. 4, even though the best matching view is in a different location. This is because the preceding correct match has changed the probability density to focus in the lower region of the map, making a fair match in the lower region more likely than a great match in a non-neighboring location.

This localization method has proved remarkably robust even to temporary full occlusions. In dynamic environments it recovers quickly from errors since occlusions don't typically last long, and recognizable features reappear after a bad frame is captured. The knowledge from the robot's previous state keeps the probability distribution from quickly drifting too far during such short interferences (Fig. 5).

Structure From Motion The final step in the localization system is leveraging structure from motion methods to estimate the robot's offset from the best matched view in the saved environmental representation. According to the epipolar geometry between two views (Ma *et al.* 2003), if the camera is calibrated corresponding points x_1 and x_2 in two images are related by:

$$x_2^T \widehat{T} R x_1 = x_2^T E x_1 = 0 \quad (5)$$

where R is a rotation and $\widehat{T}$ is the 3×3 skew symmetric version of the translation vector $T = [t_x, t_y, t_z]^T$. E is referred to as the essential matrix, and in the special case of planar motion ($t_y = 0$) the essential matrix looks as follows:

$$E = \begin{bmatrix} 0 & -t_z & 0 \\ t_z \cos\theta & 0 & t_z \sin\theta - t_x \cos\theta \\ 0 & t_x & 0 \end{bmatrix} \quad (6)$$

In this special case only four corresponding points are necessary to obtain a least squares solution to the essential matrix. The rotation angle θ and translation can then be inferred directly by imposing the positive depth constraint (Ma *et al.* 2003) on the four possible solutions. This gives an estimate of the current position of the robot relative to the best matched view in the stored environmental representation.

In their paper, Kosecka et. al. faced the additional difficulty of having to deal with unknown focal length, however since our cameras had a manually adjusted static focal length, we computed it in the calibration step and avoided the extra computation otherwise necessary. For camera calibration we utilized Intel's OpenCV calibration procedures.

Path Planning and Navigation

The agents will work in a partially-known dynamic environment. Thus, adaptive and cost effective path planning is a crucial design component. We use the D* algorithm (Stentz 1995) for its robustness to dynamic environmental changes while preserving optimality and completeness.

The D* Algorithm D* is an adapted version of the A* informed search that is more suitable in dynamically changing environments. Typically when an uncharted obstacle is discovered while using A*, one would have to recompute the entire state-space to maintain guarantees of optimality. However, by keeping track of changes of the nodes' hypothetical costs, D* allows for a significantly reduced update of only the affected nodes on the way to the path. It still preserves optimality and completeness (they are optimal based on known information).

Like A* the algorithm maintains an Open list to keep track of nodes to be explored, and a Closed list for nodes that have already been explored. The Open list is used to propagate discovered changes in the arc and path costs, where arcs are single step jumps and the path cost is the overall cost to the goal. States X on the Open list are sorted in ascending order by the function $key(X)$, which is an estimate of the minimal path cost. When states are expanded they are classified as either Raise or Lower states. The Raise states are those such that $key(X) < h(X)$, where $h(X)$ is the estimated sum cost to the goal. They propagate information about arc cost increases. The Lower states, $key(X) = h(X)$ carry information about cost reductions – discovered cheaper arc costs, or new paths found.

The algorithm has two main components, $ProcessState$ and $ModifyCost$. These two functions are responsible for computing the optimal path and introducing discovered changes respectively.

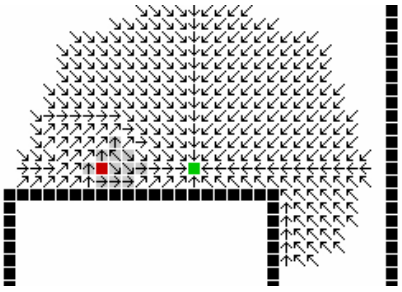

Figure 6: This figure shows the D* algorithm at work on a zoomed in map segment. The arrows indicate the path planned for the robot (red dot on left) to the destination (green dot in the center), and the black squares represent walls. Gray squares indicate a dynamic obstacle that was detected during travel, and the dynamically replanned path is shown by the changed direction of neighboring arrows.

Additionally, there are variations in the results of the algorithm that depend on the quality of initial knowledge and the exploration approach. Exploration style is an interpretation of the cost of traversing unknown regions. If such a cost is very low, then the approach is considered optimistic, and if it's more expensive than using already explored paths then it's pessimistic. If there is prior information, an average approach may be taken where costs to traverse unknown areas of the map are approximated by an average of known costs and priors surrounding that area (a lower resolution is used for this computation). Comparisons of these different techniques (Stentz 1996) suggest that given prior information about the path costs in the environment, the average method finds the best solutions overall. Also the optimistic approach is slightly better than the pessimistic. In our case the dynamic aspects are highly volatile (like humans wandering through the space), thus it is difficult to use the average approach, and we employ the optimistic one.

Optimal Deployment

Ultimately, our goal is to find scavenger hunt items as quickly as possible. In order to facilitate this, we have constructed an architecture which allows multiple robots to cooperate on the search task by dividing the space between them and allowing each robot to cover one section. In our framework, adding robots will at best linearly decrease the search time.

There are two problems to be addressed here. The first is how to divide the search space between robots such that each robot covers approximately equal portions of space. This is an optimal deployment problem. Once the area has been divided, the second task is to globally plan a path which covers this area. Here, we will focus on the deployment problem. We refer those interested in gaining a recent overview of path coverage to (Choset 2001).

Deployment Problem We pose the deployment problem as follows: Given n agents and a map of the environment to search, how should the agents divide the area such that the

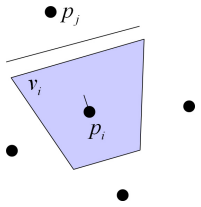

Figure 7: v_i attempts to move its boundary, taking a polygon of area from v_j

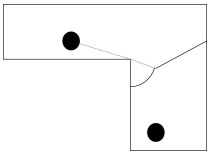

Figure 8: In this example, a non-convex portion of the region v_i from v_j. Therefore, the distance between them is not a line, but rather an arc which is formed by taking a line from v_i to the convex

entire space is covered in minimal time? Since all agents can move at the same rate and have the same visibility cone, the time to cover the space will be smallest when each agent covers an equal amount of area.

When the area to be covered is convex, an elegant distributed algorithm exists which uses voronoi regions to cover the area. This algorithm is described in (Cortes *et al.* 2002). When the region to be covered has uniform probability of containing the object (in other words, no prior information about the space is available), this algorithm corresponds to iteratively perturbing the voronoi boundaries in order to minimize:

$$C = \sum_{i=1}^{n} Area^2(v_i(p_1, p_2, \cdots, p_n)) \tag{7}$$

Where v_i is the voronoi region centered at point i. An illustration of the idea of perturbing the boundary is shown in Fig. 7. This algorithm can be extended to non-convex regions, but it becomes a bit messy. This is because the voronoi regions, and hence the area added by perturbing the border, are not lines when the region is non-convex. Consider Fig. 8. When a non-convex portion of the space separates two agents, the voronoi partition between them becomes a line augmented with an arc.

Instead, we consider the task as an area equalization problem in a non-convex domain. First, we convert the space into a grid. We initially divide the grid among n agents by initializing their positions by sampling uniformly among the space. One the agents have been placed, they claim responsibility for exploring grid squares near them iteratively until all grid squares have been claimed. We next iteratively update the bordering regions between agents until the areas have been equalized. This step is described in more detail below.

Area Equalization After the agents have been initialized and the region has been divided among them, we use Potts model to iteratively update the areas until they are equal. Potts model is a generalization of the Ising model to an arbitrary number of labels. For labels $x_i \in [1 \cdots L]$, the probability of label being a particular value is controlled by it's neighbors:

$$p(x_i) = \frac{1}{Z} e^{\beta \sum_{x_j \in N(x_i)} 1_{x_i = x_j}} \tag{8}$$

Where $N(x_i)$ are the neighbors of square x_i. Intuitively, this suggests that when all of the neighbors of x_i have a different label, x_i will be flipped to that label. In order to realize this model, we use a sequential scan Gibbs sampler with three constraints. The constraints are:

1. Only consider changing the value of grid squares along the border between regions.
2. Only consider changing the value of grid square if the region it currently belongs to is larger than the optimal area.
3. Consider updating grid squares which are further away from the center of mass of the region more often then grid squares which are closer to the center of mass.

Together, these constraints equalize the areas each agent has to cover and try to make the covered regions contiguous. The first constraint is common sense, there is no reason to flip grid squares which are internal to the area. The second constraint assures that we do not take area away from a region which is already too small. We calculate the optimal area by counting the number of grid squares in the search region and dividing by the number of agents. The last constraint makes grid squares which have been separated from the main body of the area very likely to be flipped, and also prefers that the border of the region should be as close to the center of mass of the region as possible.

Pseudocode for the implementation of our Gibbs sampler follows:

```
for each grid square i:
    build a histogram of the occurrences
      of each label in N(i)
    apply potts model to each non-zero
      entry in this histogram
    normalize this to a probability
      density
    sample from the CDF of this density
      function by:
          generate a random number r
          find the label where the
            CDF is first greater than r
          switch the grid square to
            that label
```

We start by iterating many times with high temperature (small β). This equalizes the areas. Once the areas have been nearly equalized, we smooth the boundaries by iterating with a low temperature (high β). Fig. 9 shows our results with a U-shaped non-convex space filled with a varying number of agents. Fig. 10 shows the results of our algorithm in a space which contains holes. Note that when in

places where the map is locally convex, the partitions resemble voronoi partitions.

In principle, this algorithm could potentially be performed in a distributed manner since the decision to flip involves only knowledge of neighboring squares in the grid which belong to a neighboring agent. However, our implementation was centralized.

Object Recognition

In order to recognize the objects in the scavenger hunt, we utilize a simplified version of the method described in (Lowe 2004). First, a database of object views is constructed by imaging the objects of interest from multiple viewpoints and computing the SIFT keypoints for these images. During the scavenger hunt, SIFT keys are constructed for each image and matched with the object SIFT keys in the database. If more than 10% of the keys from any view of the object in the database match, we consider the object to be present at our current location.

Provided that enough viewpoints are captured in the database building stage, this method is invariant to the pose of the object. Additionally, due to the nature of the SIFT descriptor, the method is also somewhat insensitive to the scene lighting. It also handles partial occlusions of the target object, as long as enough of the object is unoccluded to gather the required number of matching keys.

Communication Over Sparse Mobile Ad Hoc Peer-to-Peer Networks

A critical component of any multiagent system is communication. In order to reason effectively as a team, robots must be able to automatically discover each other and exchange information. Keeping our design philosophy in mind, the leading goals for communication were easy scalability, no centralized services, and robustness to mobility. As these conditions were typical characteristics of peer-to-peer systems, and a number of projects in the field of Mobile Ad Hoc Networking (MANET) have recently addressed communication issues similar to those that we predicted to encounter, we selected an application layer peer-to-peer approach, adapting methodologies from MANET research. Due to our platform, our inherent network layer utilizes 802.11 wireless cards installed in the laptops used to control the robots.

How Does Scavenger Hunt Affect Communication Based on results from our optimal deployment module, one can see that it will often be the case that robots will be assigned to explore regions of a map that may be divided by significant obstacles or have points on their extremes that may be very distant from each other. Since the conference center is large and will have multiple different halls and rooms, it is highly likely that during exploration the robots will never all be simultaneously connected to each other for long. This means that it is critical that our communication system be tolerant of sizeable periods of net partitioning. It also implies that agents must be able to discover autonomously when they are within communication range and with whom they can communicate.

Related Work A number of related work has been done addressing the above problems under the category of delay tolerant networking. The standard method of addressing sparse networks with long periods of network partitioning is referred to as the store-carry-forward (Zhao, Ammar, & Zegura 2004) paradigm. Variations of this method are classed into two categories: proactive and reactive. In reactive approaches agents typically disseminate packets to multiple peers, who then either propagate or store those messages in hopes that the receiving node soon becomes available. In proactive approaches, agents have partial knowledge about the movement patterns of others and utilize predictions of where an unreachable destination node might be to alter their course, so that they can get close enough to send a message. The proactive approach however, is not appropriate for our problem. Since our primary focus is full coverage and efficiency in exploration, our robots cannot afford to deviate from their planned paths to improve communication.

Related work has also been done on message ferrying (Zhao, Ammar, & Zegura 2004), in which additional mobile message ferry agents deliver messages to peers. This approach is related but does not apply to our problem specifically, as our only agents are the robots.

HOBOSnet The HOBOSnet protocol is a peer-to-peer overlay that is currently configured to send and respond to two types of messages: discovery and robot state update. A two-hop multicopy (Groenevelt, Nain, & Koole 2005) approach is used to deliver the important state update messages to the designated recipients, while discovery messages are only sent to reachable nodes and not propagated. Each node in the overlay stores two tables. One table consists of all nodes that it has ever contacted, and the other consists of all connected nodes (those that have replied to a recent discovery message). Entries in the connected nodes table expire with a time to live (TTL) of 10 seconds. This TTL is reset each time a message is exchanged between the two nodes, whether it is a discovery or state update message. In addition each node stores a queue of messages it is to forward to specific peers if it happens to encounter them. Each message also has its own TTL value.

Node Discovery The node discovery messages are implemented as a UDP broadcast that includes the IP address and robot identification (stored internally on each robot) of the sender. All nodes receiving the discovery message reply back with an acknowledgement that includes their identifying information. Based on this brief communication both the sender and receiver update their global knowledge and currently connected tables.

Node discovery is performed as a combination of an on-demand and proactive approach. The algorithm can best be summarized in pseudocode as follows:

```
while (1):
    // proactive
```

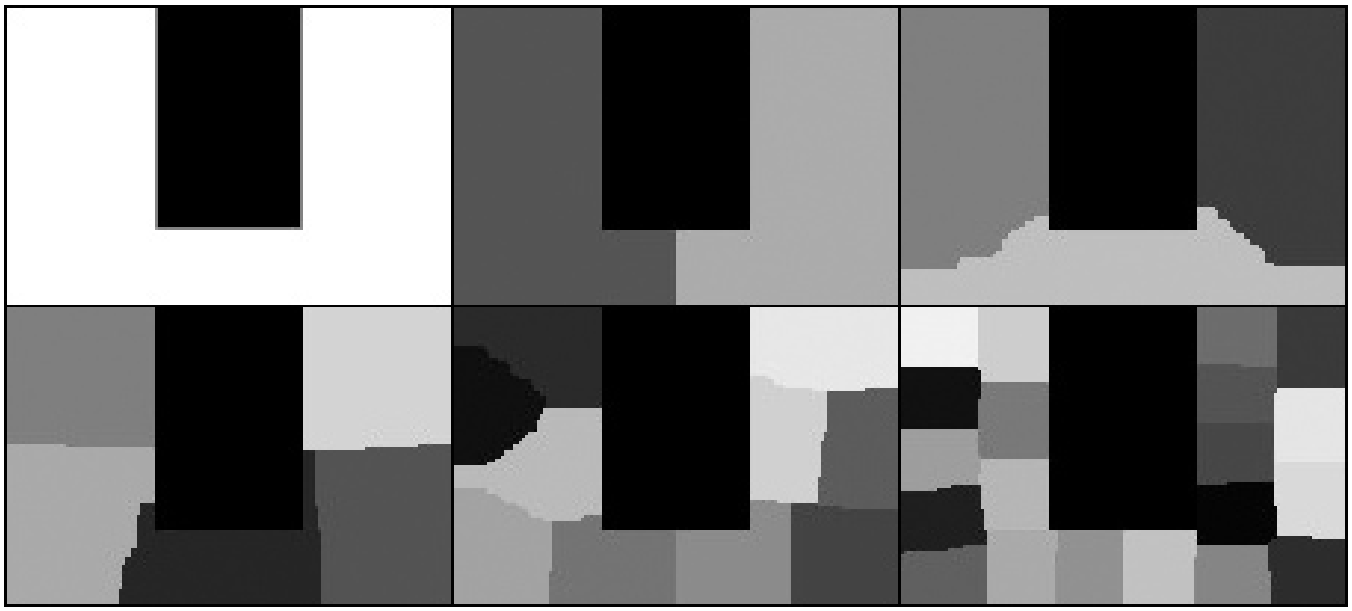

Figure 9: The partitions of a U shaped region for a varying number of agents. Top row from left: Original map, 2 agents, 3 agents. Bottom row from left: 5 agents, 10 agents, 20 agents.

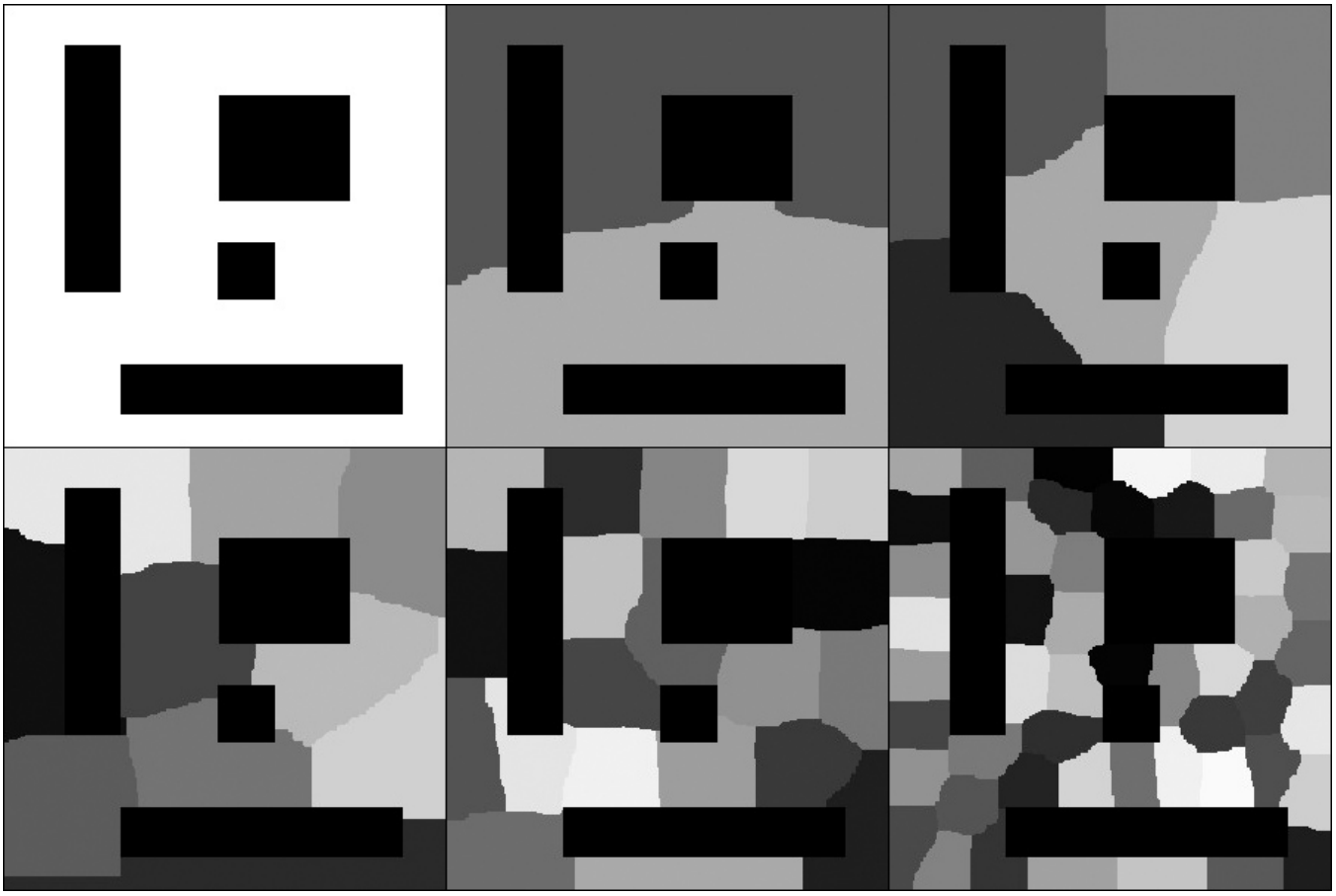

Figure 10: The partitions of a non-convex region which contains holes for a varying number of agents. Top row from left: Original map, 2 agents, 5 agents. Bottom row from left: 10 agents, 20 agents, 50 agents.

```
if ( isempty( global node list ) ||
     exist( stored messages ) )
    Send discovery message

// on-demand
else if ( send message request )
    if ( unknown( destination ) ||
         destination.ttl < time )
        Send discovery message

sleep( interval )
```

State Update The state update messages are implemented via directed TCP messages. To minimize the amount of data transmitted at once, there are two possible types of state update messages. The first type is a *general info* message, which includes the source and destination addresses, the source robot position, the total number of objects that the source robot knows have been found globally, and a timestamp. If the receiver of the message determines that the sender knows of more objects that have been found it follows up with a query, to which the sender sends out an *objects found* message, including a list of object IDs and the locations in which they were found. Python's pickle module is used to serialize data for transfer.

Unlike discovery messages, state updates use the two-hop multicopy algorithm to communicate with mobile nodes to whom they may not be immediately connected. This algorithm copies a message from the sender to all immediately connected nodes, who are then only allowed to forward a message specifically to the destination node. In an extremely sparse system like ours, there would be no benefit from a multicopy approach of higher degree, which would only clutter the network with many duplicate messages. Each message is kept in node queues for a 30 second limit, and if an original sender does not receive a reply within 40 seconds it resends the original message, likely with updated state information.

Discussion

The majority of our system capabilities were demonstrated in simulation at the 2006 AAAI Robotics Exhibition. Unfortunately full integration of all components was not yet completed by the time of the scavenger hunt competition, thus only our autonomous obstacle avoidance and object recognition systems were demonstrated. Integration work is currently continuing and the system is expected to be fully functional soon.

We plan for the HOBOS framework to serve as a general research platform for exploration of both single and multi agent approaches to various problems in robotics. Once full integration is completed, various object recognition routines, communication protocols, task division methods, and localization and mapping approaches may be tried on the system.

Significant extensions planned for the future of the system include the development of a real time visual simultaneous localization and mapping (SLAM) method to replace the current localization component, elimination of the planar motion assumption, and the addition of visual odometry. These changes would enable the framework to accept any agents capable of carrying a camera and a processing unit, which would have obvious ramifications for large scale search and rescue tasks, enabling autonomous agents to cooperate with humans and rescue animals such as dogs.

Improvements in the path planning algorithm are also in design. One of the aspects of D* is re-tuning it to account for the fact that the robot itself has significant size, without sacrificing the resolution of the path planner and the IR sensor data. We also plan to implement an improved version of D* where an additional soft constraint is minimized, which in our case would be the number of overall turns (Stentz 2002).

References

Choset, H. 2001. Coverage for robotics - a survey of recent results. *Annals of Mathematics and Artificial Intelligence* 31:113–126.

Cortes, J.; Martinez, S.; Karatas, T.; and Bullo, F. 2002. Coverage control for mobile sensing networks. In *Proceedings of the International Conference on Robotics & Automation (ICRA)*, volume 20(2), 1327–1332. IEEE.

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

Groenevelt, R.; Nain, P.; and Koole, G. 2005. The message delay in mobile ad hoc networks. In *Proceedings of SIGMETRICS*. Association for Computing Machinery.

Kosecka, J.; Li, F.; and Yang, X. 2005. Global localization and relative positioning based on scale-invariant keypoints. *Robotics and Autonomous Systems* 52(1):27–38.

Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *Int. Journal of Computer Vision* 60(2):91–110.

Ma, Y.; Soatto, S.; Kosecka, J.; and Sastry., S. 2003. *An Invitation to 3-D Vision: From Images to Geometric Models*. Springer Verlag.

Stentz, A. 1995. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation* 10.

Stentz, A. 1996. Map-based strategies for robot navigation in unknown environments. In *Proceedings of AAAI*. AAAI.

Stentz, A. 2002. Cd*: a real-time resolution optimal replanner for globally constrained problems. In *Proceedings of AAAI*. AAAI.

Zhao, W.; Ammar, M.; and Zegura, E. 2004. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of MobiHoc*. Association for Computing Machinery.