Analyzing Mobile Sensor Placement for Distributed Object Tracking

Alexander J. Trevor
School of Computer Science
Carnegie Mellon University

Advisor: Paul Rybski
May 15, 2006

Contents

Abstract

Mobile sensor networks have a variety of applications in security and surveillance. One of the most important considerations for such a mobile sensor network is the problem of how to distribute the sensors to maximize coverage of the space to be monitored and detection of targets. Here we compare several strategies for positioning of mobile sensors, including: stationary sensor placement, sensors that rotate in place, mobile sensors moving between predefined waypoints, and an active search. The active search is a strategy where the mobile sensors repeatedly move to the edge of the recently viewed space in order to view areas which have either not yet been observed, or which have not been observed in a certain amount of time. Data from previous target observations is used to learn a model of target motion, which can be used to predict target movements and areas that targets may be likely to appear. Tests are performed in simulation in order to evaluate the effectiveness of the different sensor positioning strategies and sensor models.

Chapter 1

Introduction

1.1 Problem Statement

Mobile sensor networks have a variety of applications in security and surveillance. They are applicable in situations where there are a number of sensors and an area to be covered, but the area to be observed is larger than the combined sensing cones of the sensors. A team of mobile robots can be used to move through the space and search for targets to track. The mobility provides two main benefits. First, more space can be viewed by moving the sensors through the space than if stationary sensors were used. Second, once a target has been detected, we can use information from previous detections to more intelligently track the target through the space and learn about its movement. However, this leads to the question: in what way should the sensors be moved to best make use of them? This is the problem we will examine in this paper.

1.2 Importance and Applications

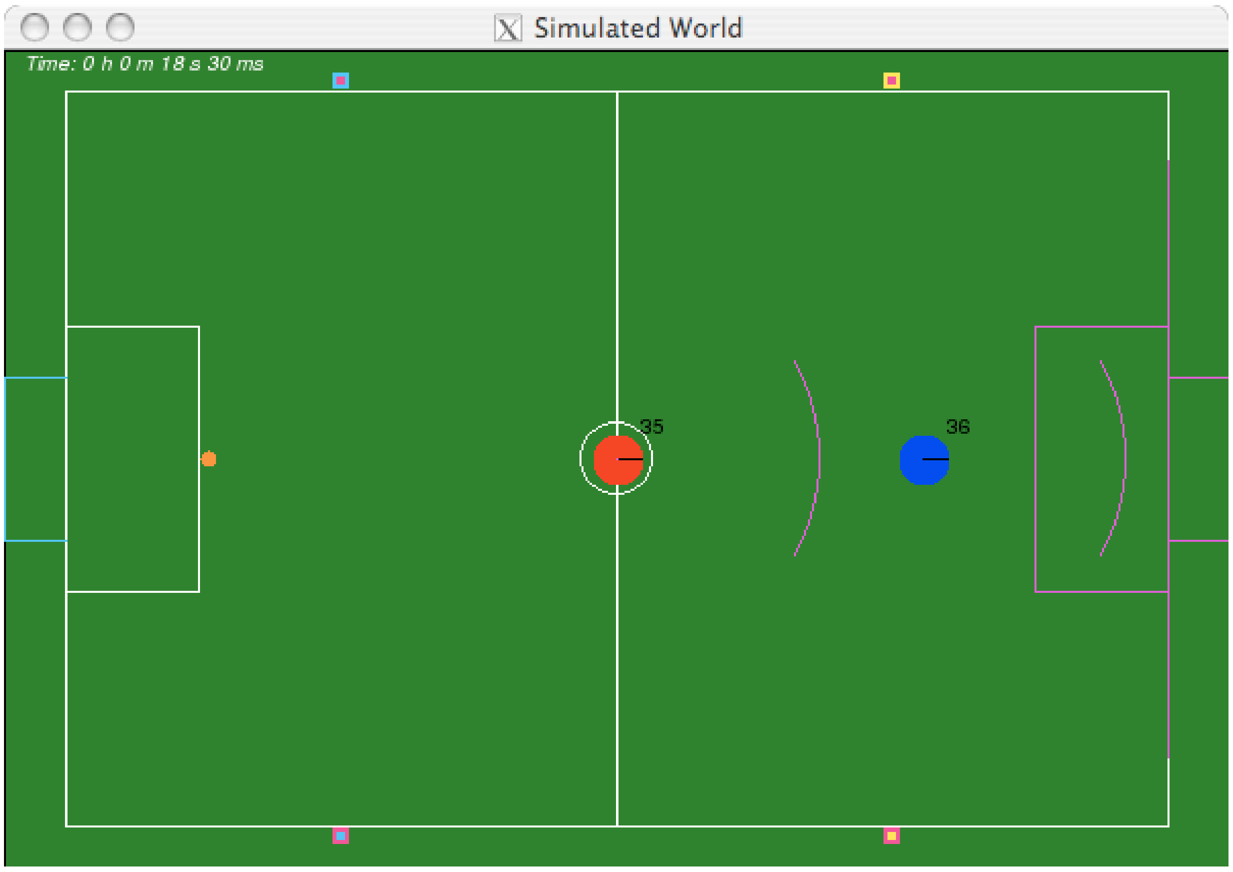
Security and surveillance are important examples of this type of problem. Such a solution is useful any time we have a large area to observe and not enough sensors to cover it. For example, we might want to monitor a large office or warehouse for intruders, but use a small number of mobile sensors rather than a large number of fixed ones.

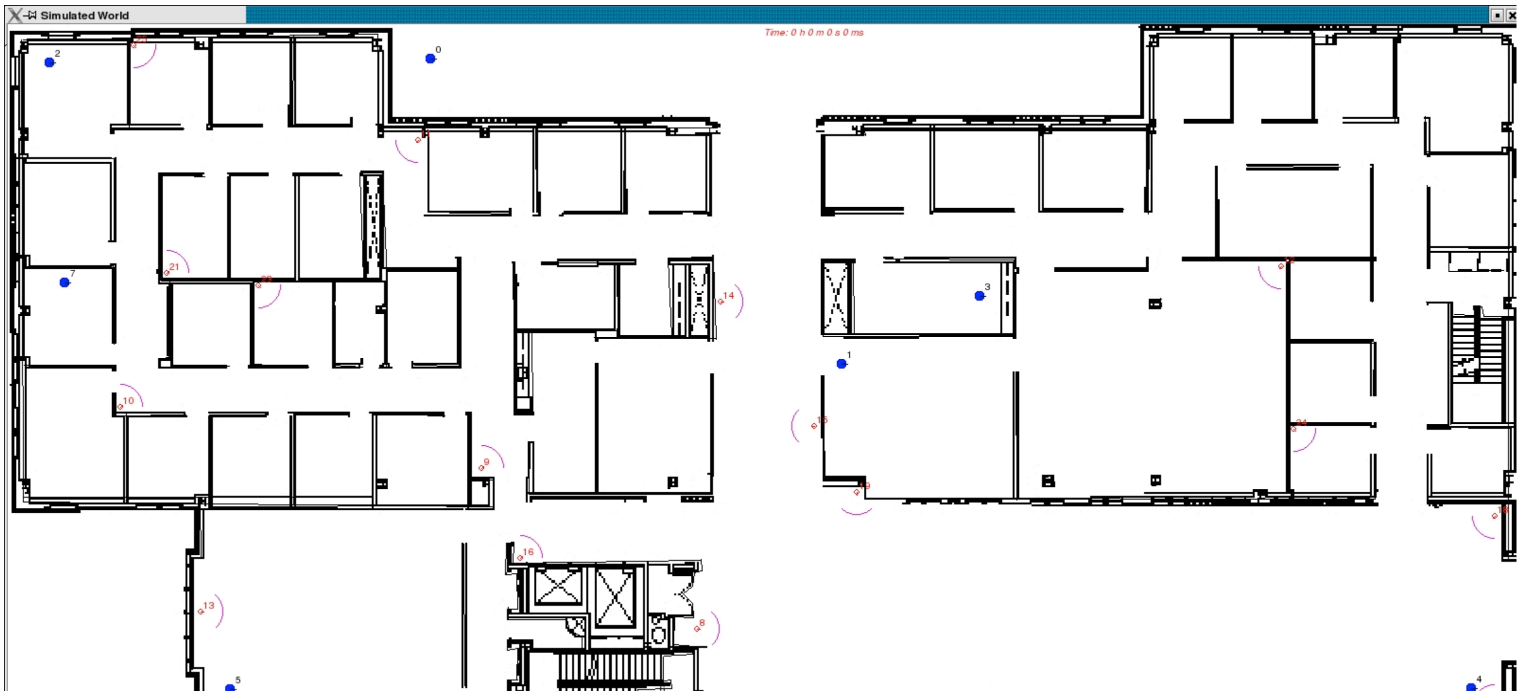
This type of solution could also be useful in situations where there is not time to install fixed sensors. If combined with a mapping algorithm, it would be possible to release mobile sensors into an unknown building or area first to map it, and then to monitor it for movements. This could be particularly useful in environments where there isn't time to install fixed sensors, or installing them would be too dangerous.

This also lets us study how we can make shared models between our distributed team of robots given uncertain sensor information, and how we can use such a model to improve the performance of our system as a whole for its detection and tracking task.

1.3 Approach

We empirically study several strategies for sensor placement and movement, and use the empirical data to compare and contrast them. The experiments are being performed using a multi-robot simulator developed by Paul Rybski. This simulator allows for basic physics simulation, communication between robots, simulation of sensing, etc. It can be used to simulate a variety of environments. Below are two example environments: a simulated robosoccer environment, as well as a simulation of one floor of Carnegie Mellon University's Newell-Simon Hall.





Experiments are performed in simulation rather than on physical robots to allow us to run experiments much more quickly and easily than would be possible with physical robots. This means that we can collect much more data than would be possible with physical robots. This also allows us to study more complex situations and strategies than would be possible with the physical robots, facilities, and time at our disposal. Finally, because the experiments are performed in simulation, we also know ground truth, so we can better analyze the results.

1.4 Related Work

This search problem is related to the well-known "art gallery" problem. The art gallery problem can be formulated in a variety of ways, but the basic idea is to determine how to place cameras in order to ensure that as much of a given space (the art gallery) is in view. Though this is related to the problem we are trying to solve, there are several differences. First, the art gallery problem typically refers to stationary cameras only, while we are interested in mobile sensors as well. Additionally, realistic sensor models are generally not used because sensor noise and limited range are usually not considered. We wish to examine this problem using a sensor model of a monocular camera, which includes noise, and limited range.

The work in [4] by L.E. Parker addresses a similar problem to what we are looking at here, though with a few key differences. This paper examines what they call "Cooperative Multi-Robot Observation of Multiple Moving Targets". One major difference is that they search for multiple targets, while we assume a single target.

The representation of sensors in this work is somewhat similar to our approach. These sensors are modeled as noisy, and have limited range. However, the sensors used in these experiments have an omni-directional view, while we use sensors with a limited field of view. This is a key difference, because with an omni-directional view, a robot's orientation does not matter; only the position affects the area it is able to view. Because we use sensors with limited field of view, we must also consider the robot's orientation in order to direct the camera in a useful direction. Additionally, our sensor model treats movement differently depending on it's direction relative to the camera, which is not taken into account here.

Also, the movement of these robots also differs from ours. They control robot movement by summing force vectors. The searcher robots are attracted to targets, but repelled by other searcher robots. The idea here is that this will make sure that targets that have been viewed are tracked, while maintaining space between the searcher robots. We use a variety of different movement strategies. Additionally, the work in this paper considers movement in a variety of complex patterns, while we only consider linear movement.

In the work by Gerkey et al ([5]), search problems in a polygonal environment are also discussed. This paper presents the ϕ-searcher, which also addresses the problem of search a given space for a moving target. The ϕ-searcher has a limited field of view, but has infinite range and perfect detection. This allows for the ϕ-searcher to move through the space in such a way that it can be certain that portions of the space it has previously viewed do not contain a target. We are interested in modeling sensor noise in order to more accurately simulate real sensors, particularly a monocular camera, so though the search strategies presented here are not necessarily applicable to our problem.

In [3], experimental results were given for a distributed surveillance task. We have recreated this task in simulation in order to empirically demonstrate our simulator. We model our sensors in a similar fashion to how the scout robot's cameras work in this experiment.

Chapter 2

Sensor Models and Sensing

2.1 Introduction

Because we are working in simulation, it is important to realistically model the type of sensor we are interested in studying. Here, we are particularly interested in detecting motion using a monocular camera. There are a variety of challenges faced when sensing using an inexpensive monocular camera. First, we want to make sure that our approach is robust enough to work under a variety of lighting circumstances. Additionally, we want to make sure we can run the detection in real time on a robot with limited processing capability, so we need to ensure that the detection method chosen is not too computationally intensive. It is common to use color segmentation based algorithms on mobile robots, but these are dependent on lighting conditions, and so are not suitable for our purposes. Instead, we chose a simple frame differencing algorithm, which is described in detail below.

Real sensors are inherently noisy, so we don't get perfect detections. This needs to be taken into account by modeling our sensors such that they only make detections with a certain probability when a target is in their field of view. Also, sensors might have other properties that we are interested in modeling. When detecting motion using a camera, it is difficult to detect movement directly toward or away from the camera, and easier to detect motion across the field of view. Also, nearby movement is easier to detect than distant movement. This means that what a sensor detects depends on where it is relative to the target motion. Finally, detecting motion in this way works only when a sensor is stationary. This is important to take into account when designing our sensor model.

2.2 Frame Differencing

Motion can be detected by using a simple camera such as the one on the Sony AIBO robots. The approach used here was to take an image from the camera, wait a short period of time, and then take another image. These images are then subtracted, which results in an image of the same resolution as the original images where each pixel's value is now the amount of change in that part of the image. The resulting image can be analyzed in several ways. First, we can look at the total amount of change in the image by summing the change of all the pixels. Using this approach, motion can be detected simply by setting a threshold for whether the image has changed "enough".

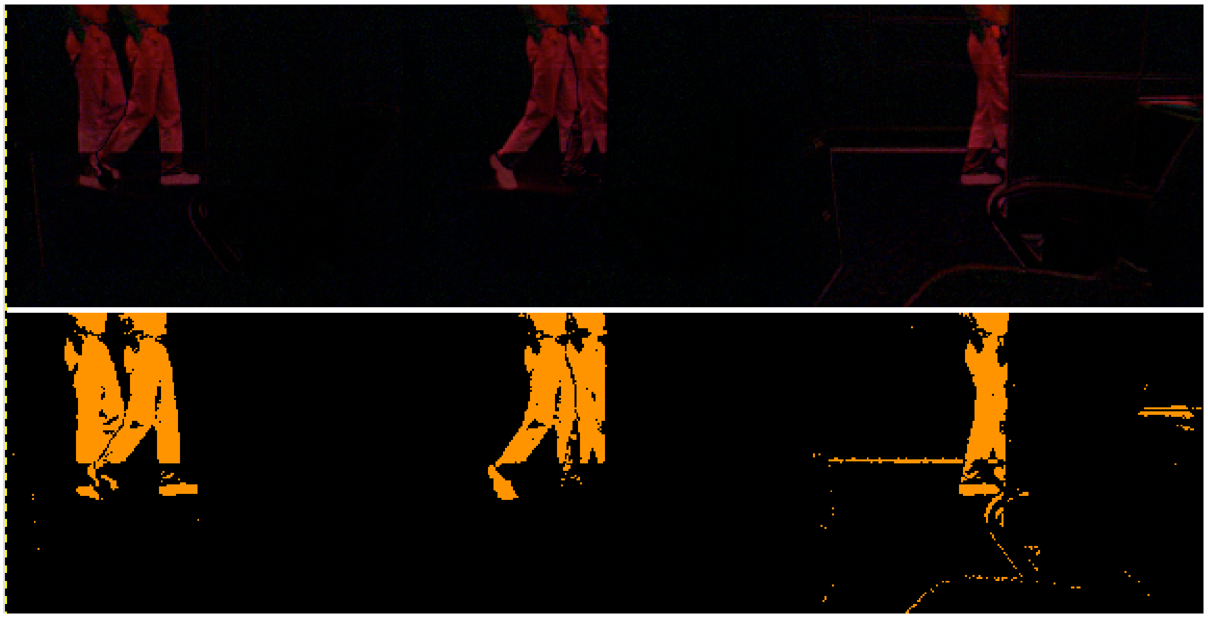
This algorithm works as follows:

1. total_diff = 0
2. Store image i_1

```
3. Wait for t_wait seconds
4. Store image $i_2$
5. For each pixel p:  total_diff += $i_1$.p.y – $i_2$.p.y + $i_1$.p.u – $i_2$.p.u + $i_1$.p.v – $i_2$.p.v
   (TODO: make this look nice)
6. if (total_diff > diff_thresh) return true  else return false
```

Another approach to motion detection on the resulting image is to use the CMVision package's color segmentation feature to segment the image into regions that have moved "enough" to be interesting. This allows us to look at different portions of the resulting image that have moved, which allows us to do somewhat more complex detection than the simple threshold described above. For example, we might be interested only in detecting large contiguous regions of movement as opposed to many smaller regions of movement. Also, this allows us to consider only areas that change more than a certain threshold, and ignore small changes to individual pixels.

The image shown here is an example of a frame differencing approach used to detect motion implemented on the Sony AIBO robots. Shown is an image of a person walking across the AIBO's field of view. The top is the raw differenced image, and the bottom shows contiguous regions of movement above a set threshold.



2.3 Sensor Model in Simulation

The above type of sensor and sensing technique is similar to what was used in the motion detection task with the scouts robots mentioned in [3], as well as the motion detection task described in the task allocation section of this document. Here we describe how this is represented in simulation.

Sensor Model Representation

Monocular cameras have a limited field of view, so the area such a sensor can view can be thought of as a cone. Our sensor model breaks the sensor cone up in to separate areas based on the distance from the camera, as well as relative direction of target motion. The sensor model used in our simulator is comprised of one or more sensor model areas. Such an area is represented as follows:

SensorModelArea d_start d_end direction threshold p_detect

d_start – the distance from the sensor this section of the sensor cone starts.

d_end – the distance from the sensor this section of the sensor cone ends.

direction – the component of the relative motion that this sensor area represents. This is 0.0 for motion across the field of view, and 90.0 for motion toward or away from the sensor.

threshold – the minimum magnitude of movement that we want to be able to detect

p_detect – the probability of detecting motion in this sensor area

This means that each *SensorModelArea* line means that motion with a component *direction* with a magnitude greater than *threshold* that is within the segment of the sensor's sensor cone starting at *d_start* and ending at *d_end.*

We determine whether or not something within the sensor's field of view is detected in the following way. First, we determine the components of the motion relative to the sensor. We examine the component that is parallel to the sensor's optical axis and the component that is orthogonal to the optical axis independently. For each component, we examine our list of *SensorModelAreas* to determine which (if any) *SensorModelArea* this motion falls under. If a *SensorModelArea* is found where the distance to the motion is greater than the *SensorModelArea*'s *d_start* and less than *d_end*, has the correct *direction*, and the magnitude is greater than *threshold*, we then return a detection with a probability of *p_detect.* A detection is made if at least one of the components of the motion is detected.

Example Sensor Model

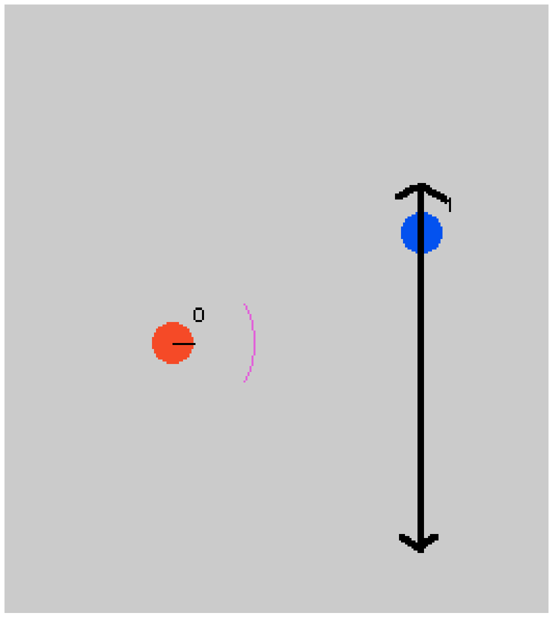
Here is one sensor model we have used:

```
#SensorModelArea start end direction thresh p_detect
SensorModelArea 0.0  1.21  0.0 0.001 0.9
SensorModelArea 0.0  1.21 90.0 0.001 0.2
SensorModelArea 1.21 2.43  0.0 0.001 0.7
SensorModelArea 1.21 2.43 90.0 0.001 0.16
SensorModelArea 2.43 3.65  0.0 0.001 0.24
SensorModelArea 2.43 3.65 90.0 0.001 0.05
SensorModelArea 3.65 4.87  0.0 0.001 0.02
SensorModelArea 3.65 4.87 90.0 0.001 0.005
```

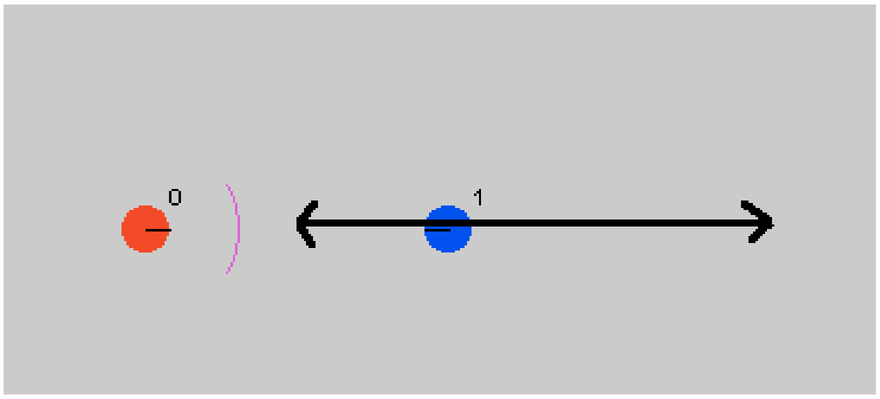
This represents a sensor with four regions, each represented by a *SensorModelArea* for both the component of the motion parallel to the optical axis, and the component orthogonal to the optical axis. The first line shows the region from directly in front of the sensor to 1.21 meters in front of the sensor, and represents the component of movement orthogonal to the optical axis. Movements within this region will be detected with a probability of 0.9 if their magnitude is greater than 0.001. The second line shows the same region, but is the component parallel to the optical axis, and gives detections with a probability of 0.2. The remaining lines are interpreted in the same way.

Demonstration of Sensor Model in Simulation

In order to demonstrate this sensor model, we can examine some example situations. First, let us look at motion orthogonal to the optical axis in the setup shown below. The red dot represents a stationary sensor, while the blue dot is a moving target which moves in the direction indicated by the arrow. The black line on the red dot shows the orientation of the sensor, and the red arc a short distance away from the sensor shows it's field of view. In one trial, of the time that the target is within the sensor's field of view, it is detected 65.4% of the time.



We can then examine the case where motion occurs parallel to the sensor's optical axis in the setup shown below. Again, the red dot is the sensor, and the blue dot is a moving target which moves along the path shown by the arrow. Of the time the target is in the sensor's field of view, it is detected only 16.2% of the time.


0
1

Chapter 3

Sensor Placement Strategies

3.1 Introduction

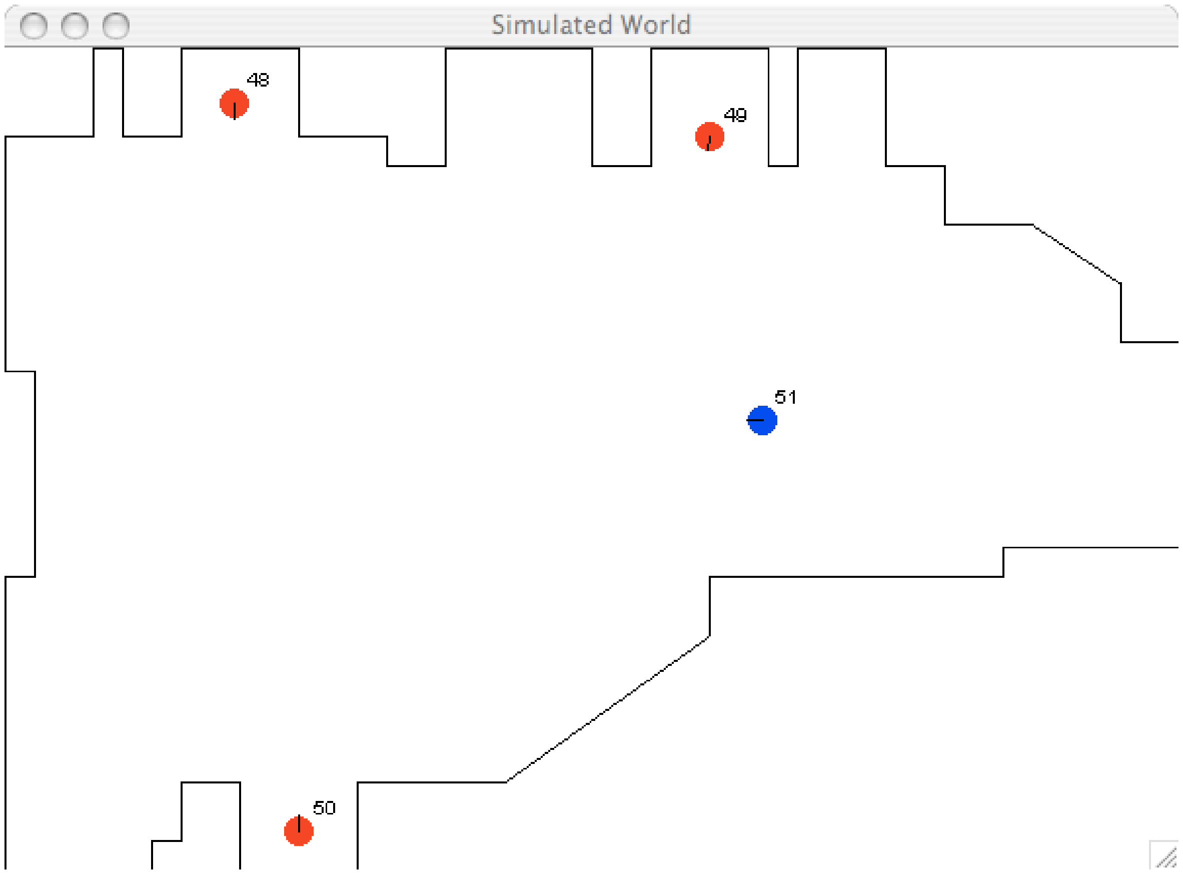
Now that we have developed the sensor model we are interested in, we examine several sensor placement strategies. We start with stationary sensors, which act as a baseline for comparison to mobile sensing strategies.

A physical experiment performed by Rybski et al in [3] was recreated in simulation for two reasons. First, it allows us to validate our simulator and sensor model because we can compare the simulated results to the actual results from physical robots. Also, this experiment used stationary sensors, so the results serve as our baseline for comparison to other search strategies.

The experiment involved using small mobile robots equipped with monocular cameras to perform a distributed surveillance task. These robots, which are described in detail in [3], were developed at University of Minnesota. In this experiment, one or more scout robots were placed in the center of the room, and would then place themselves such that they were positioned somewhere on the room's periphery, and were facing towards the open part of the room. The motion to be detected was a pioneer robot, which entered the room, moved across the room, then turned and exited the room at the same position that it entered.

The test environment for this experiment was described in detail in [3]. We have recreated this environment in simulation. First, we created a simulated test area with the same dimensions as the physical room. A diagram of the shape of the walls and other objects around the periphery of the physical room was also given, so we recreated this as well. In order to represent the physical scout robot's sensor as accurately as possible, a sensor model was constructed which used the same field of view as the physical Scout robots, and had detection probabilities similar to those described in [3]. Representative locations of scout robots were also specified in [3], so in our experiments we randomly selected from among these locations. Orientations were not specified, but since the robots in [3] were meant to face the center of the room, such orientations were chosen manually. Additionally, we created a simulated target which moved along the same path as the physical target used in the experiment in [3], and was in the room for the same amount of time as the physical robot was on average.

Here is an example of this test layout with three sensors and one target:



3.2 Evaluation Metrics

In order to compare these strategies, we will need to compare the results using quantitative metrics. Here we describe the metrics we are interested in. First, we can look at the total amount of time the target is viewed out of the time that the target is in the area to be viewed. This allows us to determine how well we are doing at detecting targets in the area to be observed.

Another interesting metric is how much of the time we detect a target out of the time it is within a sensor's field of view as compared with the highest probability of detection in the relevant sensor area. This tells us about how well we are using the sensors we have, given that detections depend on the direction of motion relative to the sensor.

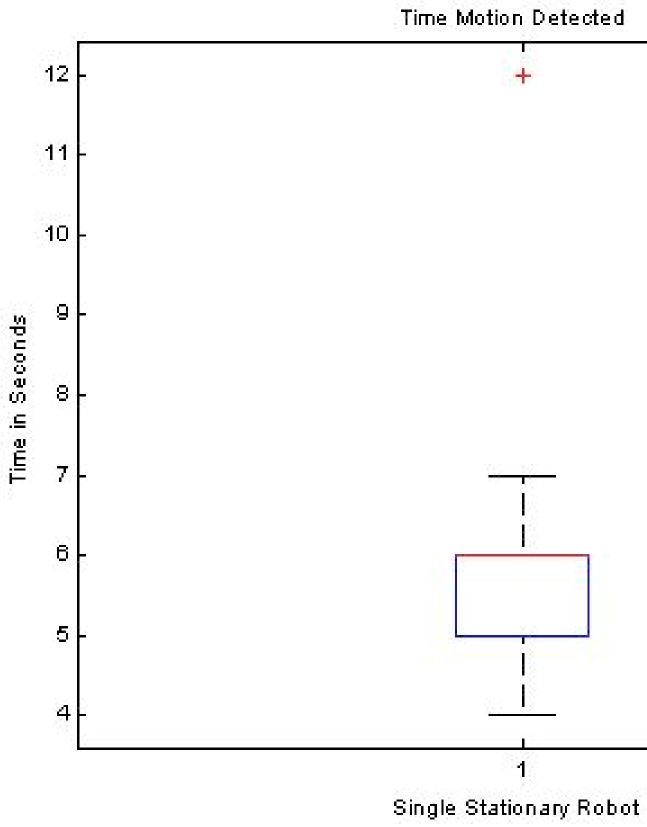
Also, we can examine how long it takes to get the first detection. This metric is interesting for search strategies that involve actively repositioning sensors based on previous detections, because we need to get an initial detection in order to start tracking a target. This metric tells us about how well we can search for a target as opposed to how well we can track it.

Because we are also interested in coverage of the space while searching for targets, we also examine the area enclosed within the sensors' field of view. For mobile sensors, we are interested in both the average area of the field of view, as well as the aggregate area covered by the sensors over time.

3.3 Stationary Sensors

The simplest sensor placement strategy we can use is that of stationary sensors. Here, we choose some placement and orientation for the sensors, and they never move. This strategy is used as a baseline for comparison with the other strategies. An example of this is the experiment mentioned above that we have recreated. This allows us to both compare the simulation to the actual results from the physical robots described in [3].

The test environment described in section 3.1 was used for these experiments. In all experiments, the target begins at the right side of the test area, moves to the right side, then turns around and returns to where it started. The target is in the test area for 22 seconds. First, a single sensor was placed in a randomly selected position representative of the scout robot placements. Twenty trials were run with this, and the amount of time target motion was detected was recorded. The results were as follows:

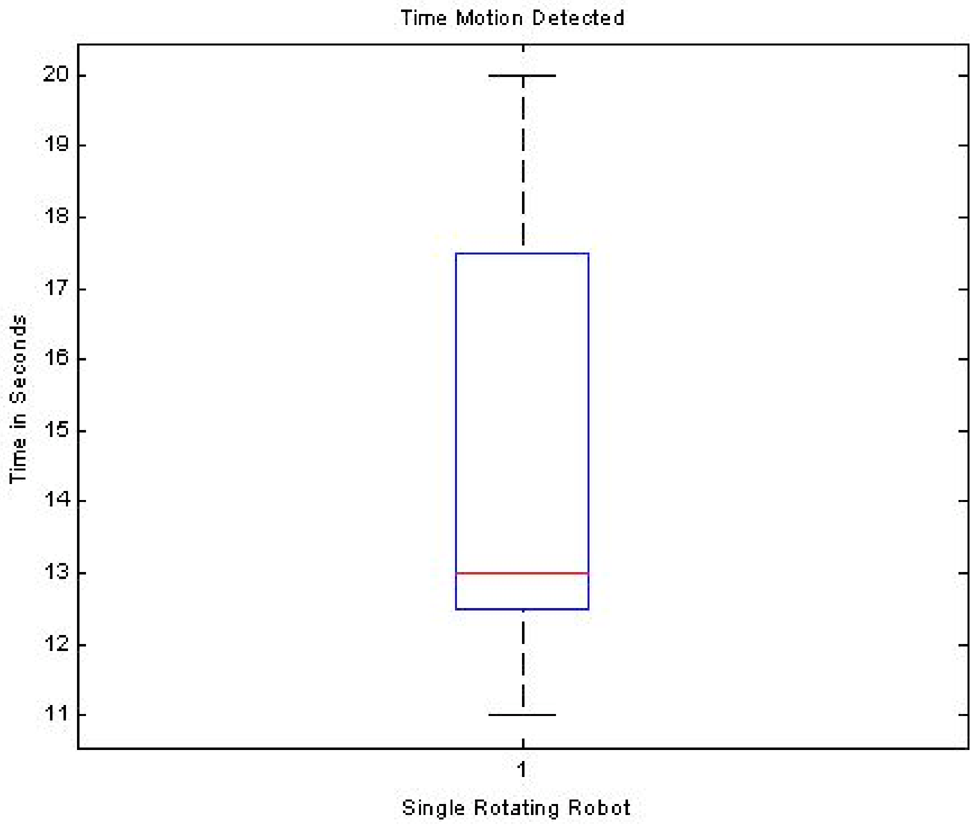


These results are similar to the results from the physical robots in the experiments performed in [3], though the results from the physical robots had a higher mean, as well as more variation.

3.4 Rotating Sensors

A simple improvement we can make to our sensing strategy is to use sensors that rotate. Here, we once a target is detected, we try to keep it within our field of view. Sensors remain stationary until the first detection, and then rotate to face the most recent detection once the target moves near the edge of the sensor's field of view.

Here we ran in the same test environment described above, but used the rotating sensor strategy instead. Again, twenty trials were performed with the same target motion, placing a single sensor at a randomly chosen position from the representative scout positions. The results were as follows:



We can see that this strategy greatly improves the amount of time we are able to see the target. In this simple test environment, this strategy generally allows us to track the target continuously after the initial detection.

3.5 Active Search

The active search strategy is still in development, so experimental results are forthcoming at a later date. However, the basic algorithm is described here.

Finally, we can perform a more active search where we decide where to move based on what areas we have not seen recently, as well as taking into account previous target detections we have made. This approach has two steps: first we search the space for targets until we make an initial detection, then we attempt to track a target once we have detected it.

The searching step for this approach is based on work done in [6]. The original work was done for SLAM (simultaneous localization and mapping) applications. The idea with this active search is that we can repeatedly go to the edge of the space that we haven't yet seen until we have fully explored the space. This is the "frontier". Then, once we have covered all of the space, the frontier becomes the areas we have not seen in the longest time. This lets us continuously search the space and do our best to observe the entire area, while trying to ensure that no portion of the space goes unobserved for too long.

Once an initial detection is made, we want to track the target we have detected. Because the type of sensor we are modeling works best when the motion being detected is moving across the sensor's field of view, we want to position the mobile sensors such that the detected motion is in this area. Once we have several target detections, we can use these to attempt to determine the direction of target motion. We can then attempt to position the sensors such that their optical axis is orthogonal to the target's direction of motion, which will allow us to get better detections.

Chapter 4
Task Allocation

4.1 System Description

In order to better coordinate between multiple robots, we have also developed a multi-robot task allocation system. This is useful in the context of multi-robot sensors networks in that it facilitates the coordination and cooperation between multiple robots. Here we describe the task allocation system we have developed.

Robots in the system begin in an idle state, where they await notification of available tasks. If one or more tasks become available, each robot will bid on the task it is best suited to perform. The robot will then be notified that it has been awarded the task that it bid on, or that the task was awarded to another robot in the system. If the robot was not awarded the task, it will attempt to bid on the task that it is most suitable for among the remaining tasks until it is either awarded a task, or there are no tasks remaining. If the robot was awarded the task, it will begin to execute it. This process will continue as long as there are tasks available in the system, at which point the robots will return to the idle state until more tasks become available.

4.2 Representation of Tasks

Each robot maintains a list of all tasks that the system is to perform, and tracks various state about each task. Each task in the list is comprised of:

- t_id - the task's unique id
- t_type - an id that indicates what behavior this task will run
- t_status - an id that indicates the current status of this task
- t_owner - the id of the robot responsible for this task
- t_reward - the value of this task, for comparison with other tasks in the system
- t_position - the starting position of the task, given in a global coordinate frame
- t_time_completed - the time at which a task was most recently completed
- t_duration - the maximum time the robot should spend attempting to perform the task
- t_repeat - which indicates whether a task needs to be performed only once, or is a repeatable task that should be continuously performed

Although this is a fairly simple task representation, we think that it is expressive enough to describe a wide range of tasks that a multi-robot system might need to perform, such as the distributed motion detection task described in this paper.

Tasks can be input to the system in two ways. First, tasks can be loaded from a configuration file on a robot's memory stick, and then transmitted over the network to any other robots in the system. Also, there is a utility that allows tasks to be transmitted to a group of robots over the network from a PC. The latter can be performed while the system is operating, allowing a human to inject tasks into the system while it is on-line.

4.3 Task Execution

When a robot begins to perform a task, it must first go to the starting location *t_position* of the task, which is specified in the task list. Once the robot has reached this location, it will begin performing the task specified by *t_type*. This means that it will begin running a behavior, such as detecting motion in the area, saving images from its camera, or some other task. The execution of these behaviors is beyond the scope of this paper. The task will be considered complete when the behavior finishes running, or when the behavior has been running for longer than *t_duration*.

In order to perform a task, a robot at position *my_position* will proceed as follows:

```
Procedure ExecuteTask(current_time, my_position, task t) {
        goto_point( t_position )
        time_started = current_time
        while (behavior t_type not done and current_time –
    time_started < t_duration)
                run behavior t_type
}
```

4.4 Communication with Peers

In order to allocate tasks between robots, the robots must be able to communicate with each other. To do this, they each maintain a list of other robots in the system, including a unique id for referring to each robot, the robot's IP address, the robot's current status, the last time a message was received from this robot, and the robot's current position. Robots send status packets occasionally (currently once per second), which inform the other robots of their current status and position. These packets also serve as a "heartbeat" packet, which allows robot to detect the presence or absence of other robots in the system, and act accordingly.

4.5 Task Allocation

Tasks are allocated between robots through a bidding process. The robot with the lowest id collects and processes all of the bids for tasks, whether they come from other robots or from itself. After receiving a bid for a task, the robot waits for a short time to allow other robots to bid on this task. It will then award the task to the robot with the highest bid, and send notification to all of the robots. The robot that was awarded the task will then begin to carry it out.

In order to determine which task to bid on, each robot in the system will calculate its bid for each available task currently in the system, and will then choose the highest of these. Several factors are considered when calculating bids. First, each task has a fixed reward *t_reward* associated with it. This allows for tasks given to the system to be valued relative to one another. The distance of a task's starting point *t_position* from the robot's current position is also considered. Next, for repeatable tasks, the time since the task has last been completed *t_time_completed* is also considered. Each of these factors can be scaled as necessary for the particular conditions or set of tasks that is being performed. The findBidValue procedure described below can be modified for the particular needs of a given task by adjusting the weights of these different aspects, or by including other factors that might be relevant. For a task list *task_list*, the algorithm works as follows:

```
Procedure BidOnTask(task_list) {
        Max_bid value = null
        Max_bid_id = null
        For each task t in task_list {
                If (findBidValue(t, current_time) > max_bid_value){
                        Max_bid_value = findBidValue(t, current_time)
                        Max_bid_id = t_id
                }
        }
        if(max_bid_id != null)
                sendBid(max_bid_id, max_bid_value)
}

Procedure findBidValue(Task t, current_time) {
        if (t_repeat) {
                return t_reward + (current_time – t_time_completed) –
                distance(my_position, t_position)
        }
        else{
                return t_reward – distance(my_position,
            t_position)
        }
}
```

4.6 Task Reallocation

If additional robots become available, it can be advantageous to reallocate tasks that have already been allocated. For example, an additional robot may be activated and added to the system near a task that another robot has been assigned, but is still very far away from. Or, a robot that had been busy performing another task might complete it, and become available near a currently assigned task. In situations such as these, reallocating the tasks can improve the overall performance of the system.

To handle this case, we extended our algorithm so that when a robot becomes available, it will examine all available tasks, as well as tasks that are currently assigned to other robots. The bidding process proceeds as described above if an unassigned task is the highest bid found. However, if an assigned task is the highest bid, the robot will compare it's distance to the task to the other robot's distance to the task. If the newly available robot is at least 10% closer to the task than the robot currently working on the task, the newly available robot will then carry out this task. The other robots in the system will then be notified that this task has been reallocated, and the robot that had been performing it will become available to perform another task.

4.7 Fault Tolerance

In order to ensure fault tolerance, all of the robots are aware of all tasks at any given time. If a robot stops sending status packets for more than five seconds, this robot will be considered disconnected. Any task it was performing will then be considered available to be performed by the remaining robots in the system. If the robot that is responsible for processing bids (the robot with the lowest id) becomes disconnected, the next robot with the next lowest id will take over this responsibility. This ensures that when a robot fails, the tasks the system needs to carry out will still be performed as long as there is at least one robot operating.

Because each robot is aware of all tasks in the system, if there is a communications failure for some reason, each robot will do its best to complete all of the tasks that it was aware of prior to the communications failure. This prevents any benefits that would have been gained from cooperating on tasks, but ensures that all tasks in the system will be carried out as long as operational robots remain.

4.8 Related Work

This system is similar to other "market based" multi-robot task allocation systems, such as Murdoch [1] and TraderBots [2]. Both of these systems use a form of auctions and bidding to allocate tasks. Our system's bidding and allocation works in a similar way. However, our task representation is somewhat simpler than what is used in these systems, and allows for all robots to be aware of all tasks the system has been asked to perform. This allows any robot in the system to attempt to complete the system's tasks in case of communications or robot failures. A simple task representation also makes the system easier for it's human operator to use.

In [3], a team of mobile robots was used to perform a distributed surveillance task. The robots were tasked with observing a room and monitoring it for motion using a frame-differencing algorithm. We have explored this domain with our system by using it to perform a similar task. The frame-differencing behavior used was also similar to the one described in [3].

4.9 Empirical Validation

To validate this system, we carried out a series of empirical tests using a distributed motion detection task. This task was similar to the distributed surveillance task domain in [3]. A team of robots was given a series of points to visit, where they were to run a behavior that uses a frame-differencing behavior to detect motion in the area. If motion was detected, an image from the camera was saved for later viewing. The behavior ran for a period of 5 seconds, after which it would finish, indicating that the task was complete and making the robot available to perform another task. The task was specified as a repeatable task, so after the task was completed, it would become available again. However, due to the cost functions being dependent on the time since that task was last visited, a task that has just been completed would be worth much less than a task that has not been completed recently, causing robots to move to points that had not been visited in some time. This caused the robots to "patrol" the points specified, detecting motion at each one. If a robot was deactivated while performing a task, the remaining robots in the system would continue to operate, and would complete this task when they became available. Also, if additional robots were added to the system, they would begin performing tasks as expected. Tests were also performed to compare the system's performance with and without the "task reallocation" feature described above. Preliminary tests indicate that the system performs better on many cases with this feature than without.

References

[1] B. Gerkey and M. Mataric’, “Sold!: Auction methods for multi-robot coordination” *IEEE Transactions on Robotics and Automation*, Special Issue on Multi-robot Systems, vol. 18(5), pp. 758-768, October 2002.

[2] M. B. Dias, “TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments”, doctoral dissertation, Robotics Institute, Carnegie Mellon University, January, 2004.

[3] P.E. Rybski, S. A. Stoeter, M. Gini, D. F. Hougen, N. Papanikolopoulos, “Performance of a Distributed Robotic System Using Shared Communications Channels,” *IEEE Transactions on Robotics and Automation*, Special Issue on Multi-Robot Systems, vol. 18(5), pp. 713-727, October, 2002.

[4] L. E. Parker, "Distributed algorithms for multi-robot observation of multiple moving targets," accepted for publication in Autonomous Robots , 1999.

[5] Brian P. Gerkey, Sebastian Thrun, and Geoff Gordon. "Visibility-based pursuit-evasion with limited field of view". In Proceedings of the National Conference on Artificial Intelligence (AAAI 2004), pages 20-27, San Jose, California, July 25 - 29, 2004.

[6] Brian Yamauchi. “Frontier-Based Exploration Using Multiple Robots”. Proceedings of the Second International Conference on Autonomous Agents (Agents '98), Minneapolis, MN, May 1998, pp. 47-53