# The Autonomous Observer: A Tool for Remote Experimentation in Robotics

Héctor González-Baños[a], José-Luis Gordillo[b], David Lin[a],
Jean-Claude Latombe[a], Alejandro Sarmiento[b], Carlo Tomasi[a]
[a]Robotics Laboratory, Computer Science Dept., Stanford University, CA 94305
[b]Centro de Intelligencia Artificial, ITESM, Monterrey, México, 64849

## ABSTRACT

This paper describes a robotics technology – the Autonomous Observer (AO) – developed to facilitate experimentation over the Internet. The AO is a mobile robot equipped with visual sensors. It applies visual tracking and motion planning techniques to track a designated moving object (the target) in an environment cluttered by obstacles and repeatedly measure the target's pose. This pose is sent over the Internet to remote users who can observe 3-D real-time graphic renderings of the target's motion in its environment under individually selected viewpoints. The AO was used to set up an experiment in which a can-collecting robot (playing the role of the target) equipped with a range sensor and a simple arm automatically detects coke cans and collects them in a bag. Observation of the can-collector's behavior through the AO allowed remote experimenters to correct software bugs causing failures on this robot (e.g., colliding with obstacles, missing cans). It is well-known in experimental robotics that direct observation of the performance of a robot is crucial for debugging and tuning the software controlling this robot. The AO has proven to be an adequate (though perfectible) telepresence tool for remote experimenters. Other information, e.g., ambient sound and live videos, can also be transmitted to these users to complement the graphic rendering made possible by the AO. Multiple AO's could be used in the future to observe more complex environments with multiple moving targets. Applications of AO technology are not limited to collaborative experiments. The same basic technologies could benefit other domains as well, such as teleconferencing, surveillance, and interactive TV.

**Keywords:** Remote experimentation, telepresence, distributed teams, autonomous observer, target tracking, mobile robotics, asynchronous control, distributed systems.

## 1. INTRODUCTION

This paper describes a joint project between the Computer Science Robotics Laboratory at Stanford University and the Center for Artificial Intelligence at the Instituto Tecnologico y de Estudios Superiores de Monterrey (ITESM) in Mexico. The goal of this project was to develop robotics technology to assist geographically dispersed groups who jointly perform robot experiments. A result of this project is the *Autonomous Observer* (AO), a mobile robot equipped with visual sensors, which applies visual tracking, motion planning, and landmark-based navigation techniques to track a designated moving object, called the *target*, in an environment cluttered by obstacles. The AO repeatedly measures the target's pose (position and orientation) and sends this information over the Internet to remote users who observe in real-time a graphic rendering of the target's motion in its environment from individually selected viewpoints. The 3-D model of the environment is sent only once, prior to the experiment, so that only small amounts of data are later exchanged between the AO and the users. Figure 1 illustrates the concept of an AO. Compared to sending live videos over the network, the two key ideas behind the AO concept are (1) to reduce data exchanges between the experimental site and the remote users and (2) to allow the remote users to locally select the most pertinent viewpoints.

To demonstrate this concept, we have built and used an AO in the following experiment. The ITESM team developed navigation and vision software to control a can collector, an in-door mobile robot that navigates according to user-input high-level commands, such as "follow the wall on the right," and analyzes data provided by a laser range-finder

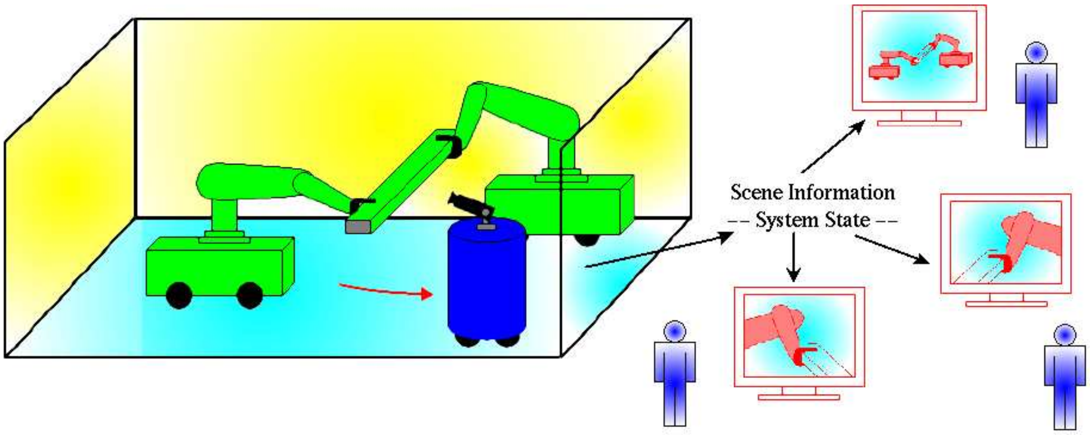


Figure 1: The concept of an Autonomous Observer

sensor to automatically detect coke cans. The can collector is also equipped with a simple arm (a rod mounted on a turret) to collect the detected cans in a trash bag. The ITESM team transfered the software that they had previously developed for their own robot in Monterrey, to a similar one at Stanford. The AO was used by the ITESM experimenters to remotely observe the behavior of the can collector (then playing the role of the target) and to fix software bugs causing failures on this robot at the Stanford's environment (e.g., going to incorrect locations, colliding with obstacles, missing cans).

It is well-known in experimental robotics that direct observation of the performance of a robot is crucial for debugging and tuning the software controlling this robot. Our experiments show that the AO is an adequate (though perfectible) telepresence tool that provides direct observation to remote experimenters. Other information, e.g., ambient sound, data available on the robot, as well as live videos, can also be transmitted to these users in order to complement the real-time graphic rendering made possible by the AO. Moreover, the application of the AO is not limited to collaborative experiments in robotics or other domains involving complex equipment. We believe that the same basic technologies could benefit other tasks, such as telepresence, teleconferencing, surveillance, and interactive TV.

Our current AO is implemented using a Nomad 200 robot from Nomadic Technologies, Inc., equipped with two cameras. One camera points horizontally and is used to track targets; the other points upward and is used to detect ceiling landmarks for precise navigation. The AO is equipped with two onboard P166 computers and it is connected by radio ethernet to the local area network. In our experiments, the target is another Nomad 200 robot.

This paper presents the software techniques implemented in the AO. Perhaps the most critical of these techniques is a target-tracking planner that periodically decides where the AO must move to both keep the target in its field of view, despite potential obstruction by obstacles, and remain at appropriate distance from the target. It must be noted that the software controlling the target (the software of the can collector in the experiment reported below) is designed independently from the AO. For that reason, there is no communication between the target and the AO.

The paper is organized as follows. Section 2 relates our work to previous research. Section 3 presents the components of the AO software and their current implementation. Section 4 describes our experimental work on the can collector and the use of the AO in this experiment. Finally, Section 5 draws conclusions from our results and discusses possible extensions, including the use of multiple cooperating AOs.[1]

## 2. PREVIOUS WORK

Most robot tasks consist of moving objects between locations, e.g., manipulator arms assembling products and mobile robots transporting parts. However, collecting pertinent information about environments through sensing is another important task for robots. For instance, robots equipped with sensors can be used to build models of objects[1,6]

[1] For more information about the AO and ITESM's can collector visit `http://underdog.stanford.edu/` and `http://renoir.mty.itesm.mx/~gordillo/OI/` , respectively.

and/or environments,[15,18] find objects (static or moving),[5,11] detect defaults (e.g., cracks in a structure), track moving targets,[2,13,17,23] and perform surveillance operations. Such tasks often require computing motion strategies to adequately place the sensors.[4] For example, building a 3-D map of a large environment may require a strategy that will reduce the number of sensing operations;[1,15,18,21] finding a moving object requires sweeping the environment in such a way that the object cannot sneak into an already cleared subset of the environment.[5,17]

This paper considers an application where a mobile robot (the AO) uses a camera to track a target and estimate the target's pose several times per second. Visual tracking is a well-studied problem in computer vision.[13,14,23] However, traditional visual tracking techniques ignore obstacles and track a target until it escapes the field of view of the camera. In some systems, the camera is mounted on a fixed-base pan-tilt platform that is servoed to keep the target close to the center of the image. The AO camera has much greater motion capabilities; but, to make the most out of them, the AO must continuously position itself with respect to the occluding obstacles in order to keep the target in sight. This led us to equip the AO with a target-tracking motion planner that deals both with visibility constraints and collision constraints (to avoid the AO to bump into obstacles, including the target itself). A model of the environment (2-D layout of the obstacles) is given in advance, but it should be possible in the future to combine our target-tracking planning techniques with on-line model-building capabilities. An alternative to a moving AO would be to rig the environment with enough cameras, so that one of them, at least, would always have the target in its line of sight; an art-gallery algorithm[22] could be used to compute the camera locations for this setup. The AO offers several advantage over multiple fixed cameras: reduced engineering of the environment and ability (in the future) to deal with dynamically changing environments.

Our target-tracking planner derives from our previous work,[2,10,17] in which we considered two main cases: (1) the target's trajectory is known in advance and (2) it is unknown or only partially predictable. Here, we want the target and the AO to be independently designed systems. Hence, the planner must operate on-line; this means that it must iteratively compute the next "best" position of the AO in response to the most recent estimation of the target's pose. At each iteration, the planner computes the AO position that minimizes a measure of the probability for the target to escape the AO's field of view over a certain time span (the planner's scope). In theory, the longer the scope, the better the AO can anticipate target motions out of its field of view. But, simultaneously, the more expensive the computation becomes[17] and less pertinent the outcome (since the target keeps moving during the computation). In our application, we do not expect the target to act antagonistically by trying to escape the AO's field of view. Therefore, we have opted for a fast planner that anticipate possible target motions over a short time span, roughly equal to a few cycles of the AO operations. Planning for target tracking in cluttered environment has applications other than telepresence. For instance, in computer graphics, it can be used to automatically control the motion of a virtual camera tracking a moving digital actor.[20]

The domain of application considered in this paper is telepresence. Though telerobotics can be seen as possible component of a telepresence system,[7] our work is very different. The AO is a robot, but it is not teleoperated; in the experiment reported below, the target is also a robot, but it is computer controlled. The AO moves autonomously to collect information about the target (currently, its pose), and transmits this information to user's workstations for 3-D graphic rendering. The need to observe remote experiments is not unique to robotics and has been noted in other domains, including physics[24] and medicine.[9] Light-field rendering is another technique that allows the observation of a distant environment from selected viewpoints without 3-D modeling;[19] but it requires using a large number of cameras, each precisely positioned in the environment.

## 3. AO SOFTWARE

### 3.1. Overview

As mentioned above, the AO is basically a mobile robot equipped with a camera to acquire images of a moving target. Its software analyzes these images to estimate the pose of the target in the environment several times per second. It sends this information to the user's workstations over the Internet. It also computes the sequence of positions where the AO should go in order to maximize its chances of keeping the target in view. In a cluttered environment, the target may move around the corner made by an obstacle and be suddenly occluded by this obstacle. As there is no explicit communication between the AO and the target, the software must anticipate such a possibility and, for example, command the AO to swing around the corner before the target gets too close to it. Of course, the AO must simultaneously avoid colliding with obstacles and remain at an appropriate distance (a user-specified parameter)

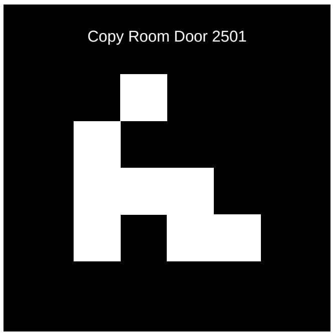


Figure 2: Example of an artificial landmark used for localizing the AO

from the target.

The AO performs the same set of operations several times per second. These consist of: localizing the target, sending the target's pose to the user's workstations, computing the AO motion to best keep the target in view, and performing this motion. These operations do not have to be performed sequentially, or with the same frequecy, and may be distributed over several processors as is the case in our implementation. To perform these operations the AO software was designed in modules, each one presented int the following subsections. One of these modules, the coordinator, supervises the rest of the software. Prior to its operations, the AO is given a polygonal map (2-D layout) of the environment.

### 3.2. Landmark Detector

As it moves, the AO must keep track of its current position and orientation in the environment. This information is needed not only for reliable navigation among the obstacles, but also to accurately estimate the target's position. Our approach to AO self-localization is by means of artificial landmarks and odometric sensing. This combination is not new and our techniques derive directly from the work presented in reference 3.

The AO uses a camera pointing upward to detect and recognize artificial landmarks placed at the ceiling of the environment. Each landmark is a $4 \times 4$ array of black or white squares, such as the one shown in Figure 2. The $4 \times 4$ pattern is unique to the landmark and is called its ID. For each landmark, there is a subset of AO positions from which it is visible to the AO.

The positions of the landmarks and their ID's are included in the 2-D map given to the AO. Whenever the AO detects a landmark, it recognizes the $4 \times 4$ pattern, computes its own position and orientation relative to the landmark, and derives its position in the environment. When no landmark is visible, the AO updates its position based on odometric sensing.

The landmark detector receives a $320 \times 240$ grayscale image as input. It first computes edge points and then looks for edge chains that are consistent with the boundary of a square. If such a chain is found, lines are fitted to the pixels which make up each edge. These lines yield the orientation of the landmark. Next, the positions of the inner squares are computed and their intensities are read in order to identify the landmark. Four of these values are used to disambiguate the landmark's orientation, while the others encode the landmark ID. At a ceiling height of 9 feet, the translational and rotational errors of the self-localizer with respect to the landmark were unbiased with standard deviations of 0.75 inches and 0.5 degrees, respectively.

### 3.3. Visual Tracker

A camera pointing horizontally allows the AO to track the target. The visual tracker finds the target in the current image provided by this camera and estimates its pose in the environment.

In the case where the target is a non-rigid object capable of performing complex motions, one would have to use real-time versions of general-purpose tracking algorithms.[14,23] The visual tracker implemented in our current AO is simpler, but limited to rigid targets moving in translation and rotation on a horizontal floor (three degrees of freedom). The target is equipped with unobtrusive rectangular black-and-white patterns placed on its surface, as

$(a)$ $(b)$

Figure 3: Barcode patterns placed on the target's surface

shown in Figure 3. Each pattern consists of a set of three vertical lines (used for its detection) and a horizontal barcode identifier.

The visual tracker receives a $320 \times 240$ grayscale image as input. It first searches the image for occurrences of a three-line pattern by scanning a few rows of the image and looking for the appropriate sequence of intensity changes. Since the pattern is rather large and located at an approximately fixed height, it is not necessary to parse more than 2 or 3 image rows. Once a sequence is found, a second procedure locates the pattern boundaries and fits straight lines in order to compute the corners (with subpixel precision) through line intersection.

After the corners are detected, the software module reads the binary barcode at the bottom of the pattern (a number between 0 and 31). It exploits the fact that the four detected corners are perspective projections of a rectangle in space to compute the position and orientation of the rectangular pattern in space.[16]

Finally, with the poses and barcodes of the detected patterns, the visual tracker infers the location and bearing of the target. Typically, the target is reliably detected at distances ranging between 2 and 8 feet from the camera. The relative errors on its position and bearing are less than 5% within the operational range.

### 3.4. Motion Planner

Our solution to the on-line target-tracking motion planning problem is deliberately simple and fast. Whenever it is invoked, the planner picks a small set $S$ of candidate positions toward which it can move within a short amount of time $\delta$ (the planner's scope), given its maximal velocity. For each position $p \in S$, it uses the input polygonal map of the environment to compute the region $V(p)$ that the AO would see if it where at $p$. It then selects the position $p$ in $S$ that maximizes the chances that the target be in $V(p)$ given its most recent estimated position.

There are several ways to pick the set $S$ of candidate positions. In our implementation, we place a grid over the map of the environment and consider the grid points that are reachable by the AO along a collision-free straight-line path, in a time $\delta$ set to 2 or 3 times the average planning time (as we will discuss later, our planner takes approximately constant time). We limit $S$ to $K$ candidate positions, at most, where $K$ is a user-input parameter. If the grid's resolution is too fine, we pick the $K$ positions at random among the reachable grid points.

Computing the visibility region $V(p)$ at every candidate position is rather fast, but still too expensive to be done on-line. So, we precompute $V(p)$ at every grid point before starting the normal AO operations. At every grid point $p$, the computation is done using a classical line-sweep technique (here, the "line" is a ray emitted from $p$ that we rotate from 0 to 360 dg.). This computation takes $O(n \log n)$ time, where $n$ is the number of vertices in the input polygonal map. The total precomputation is linear in the number of points in the grid. Our computation of $V(p)$ takes into account that the range of the AO's camera is both lower and upper bounded. Figure 4 shows the visibility region at a position $p$ – the obstacles are depicted in dark color and $V(p)$ in light, and no upper limit is shown.

The visibility region $V(p)$ could be restricted to the cone of vision of the camera tracking the target. However, as the turret of the Nomad 200 rotates relatively quickly and its orientation can be controlled independently of the heading of the robot, a separate servo-loop keeps the target-tracking camera oriented toward the target. For planning

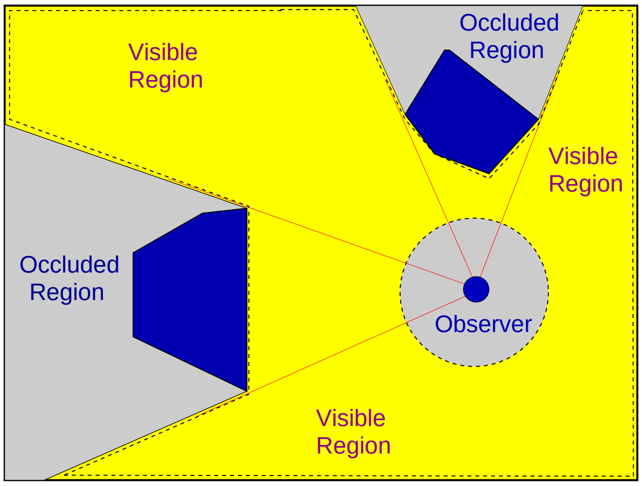


Figure 4: Visibility region of a point $p$

purposes, the observer has omnidirectional vision.

Thanks to the precomputation step, the on-line planner has quasi-instantaneous access to the visibility region of each of the candidate positions in $S$. The visibility region of each candidate is intersected with $R$, the region reachable by the target during the planner scope. The position with highest intersection is selected as the best candidate. This computation corresponds to selecting the point that has the highest probability of observing the target within the scope $\delta$. We call this planning strategy *maximizing the probability of future observation*. Ties among the candidates are broken by selecting the position that keeps the distance between the AO and the target closest to the value specified by the user.

The above technique works well if the AO can move as fast as the target (or faster), and the time needed by the target to travel outside the region visible to the observer remains significantly greater than the planner's scope. If this distance gets smaller, the planner should reduce its scope accordingly.

In the can-collector experiment, the planner used a a $129 \times 142$ grid, which corresponds to a spatial resolution of about 6 in. per increment. The parameter $K$ is set to 15. The precomputation (carried out on a SGI INDIGO$^2$ RS10000) takes about 23 sec. Planning takes about $10 - 15$ msec. Although the computation of the visibility regions depends on the number of vertices in the map, this operation is done off-line. Therefore, the on-line planning time depends mostly on $K$ and is approximately constant for fixed $K$.

More sophisticated planning techniques could be used to select the next AO position. An interesting extension, if the landmarks are sparsely distributed, is to maintain an explicit representation of the uncertainty on the AO position and to aim at positions where a landmark is visible when localization uncertainty becomes too large. Such techniques have been separately developed and tested successfully.[8]

A completely different strategy, suitable for antagonistic targets, will be to solve a local mini-max problem. For each candidate in $S$, evaluate the distance between the most recent estimation of the target's position and the boundary of $V(p)$. This distance, which we call the *distance-to-escape*, is roughly proportional to the minimal time needed by the target to escape $V(p)$; hence, it is a worst-case measure of the likelihood that the target escapes the AO's field of view. Said otherwise, the larger the distance-to-escape, the more likely the target will remain in the field of view of the AO. The best candidate is the one with the largest distance-to-escape. This strategy, which we call *maximizing the minimum escape-time*, has been succesfully implemented in simulation and we are currently developing a robot implementation based on this planner.

### 3.5. Motion Controller

The controller issues commands for each of the three actuators on the Nomad 200 robot – translation, steering, and turret rotation. Translation and steering are actuated in order to head toward the positions (set-points) successively selected by the motion planner. Note that the planner may generate a new intermediate position while the robot is still moving toward the previous one. The controller then switches immediately to aim for this new set-point.

As mentioned above, the turret is controlled by a separate loop that keeps the horizontal camera pointing toward the target. The quick response of the turret loop allows us to assume that the robot is equipped with an omnidirectional target-tracking camera.

High-level motion control cannot be achieved by directly applying classic design methods since the information flow between the AO system and the robot motors is an asynchronous transmission. The Nomad 200 motors are driven by a GALIL DMC-630 board through a system daemon running as part of the Linux kernel in the robot's computer. The DMC-630 is in effect a low-level position servo. For regular operations we can safely represent the response of each axis using a simple kinematic model. However, since motion requests within the AO ultimately originate from an active feedback process running at the top-level, and given that the OS on-board the robot is not based on a real-time kernel, the axes motion responses become non-linear.

Given the sample rates required by the AO (10-15 Hz), we felt the need for a real-time kernel (which is an expensive solution) could be avoided with a more sophisticated controller design. Our approach consists on a triple-layered strategy: a linear compensator, an adaptive scheme that keeps the compensator tuned, and a time pacer that regulates the control cycle to the specified rate.

The linear compensator is designed using a pole-placement scheme. The controller parameters are selected in accordance to the desired control rate (15 Hz), the communication delays between the AO modules, and the time the controller spends executing the necessary operations (duty cycle).

The adaptive loop keeps the compensator poles in place by recomputing the controller parameters as the control rate and the duty cycle drift from their initial estimated values. The adaption maintains the closed-loop response close to design specifications (e.g., settling-time of 1 sec. and overshoot of 1 %).

Finally, the time pacer verifies that the control cycle stays close to the specified rate. The pacer estimates on-line the controller duty cycle, and pauses execution the amount necessary to keep the total cycle time within specification – i.e., it keeps the sampling time approximately constant. The pausing function is particularly important, as it gives the kernel a good time slack to attend other processes. Without this slack, the control cycle will be interrupted at arbitrary instants during operation.

### 3.6. User Interface

Each user can monitor the status of the remote experiment using the Graphical Robot Interface Program (GRIP). GRIP also allows each user to teleoperate the target robot kept under observation (to emulate the robot joysticking function that is usually available at the experimental site). The information about the experiment is retrieved from the AO and displayed in graphical windows. A top-view window shows the 2-D map of the environment and the current location of the target in this map, while other windows show 3-D renderings of the experimental site under viewpoints selected by the user (see Figure 5). In our current system, we also allow the user to access video images taken by the tracking camera.

The core AO system is separated from the user interfaces by a gateway module. The gateway module contains up-to-date information of the changes in the environment observed by the AO. Any user interface module communicates with the gateway module to obtain this information. For better fault tolerance on the Internet, the communication protocol is a traditional acknowledgment protocol based on UDP. Thus, a target position sent to a user interface module, but not received by this module, is simply forgotten. This prevents sporadic transmission problems from queuing at the user's end and significantly delaying the graphic visualization (a frequent problem when the standard TCP/IP protocol is used). As new positions are computed 5 to 10 times per second, sporadic loss of information is usually not perceptible by the users. The protocol gives reliable performance under low Internet traffic, and degrades gracefully when traffic increases. The video images are displayed at a slower rate (on the order of one every 5 sec.).

### 3.7. Coordinator


$(a)$ $(b)$

Figure 5: Rendering of the experimental site: (a) top view showing the target and the observer, as well as the ceiling landmarks; (b) view from a selected viewpoint

Each of the above components runs as a separate Unix process. The communication between processes uses standard TCP/IP protocols (with the exception of the UDP gateway), making it possible to run them on different machines and test each module independently. In our implementation, the landmark detector and the visual tracker run on one of the AO's two on-board P166 processors, the motion controller and coordinator run on the other P166 processor, and the motion planner and the UDP gateway run on a SGI INDIGO$^2$ RS10000 station. The graphical interfaces run at the users' remote ends.

The coordinator supervises the rest of the AO's software (see Figure 6). It integrates the information from each module into a single representation of the state of the AO and the target. The state is essentially a vector pair $(p, q)$, where $p$ and $q$ are the most recent estimates of the position of the AO and the target, respectively. It connects as a client to the landmark detector, the visual tracker, and the motion planner, and acts as a server to any process requiring online information about the system state. The coordinator and the motion controller are integrated into the same Unix process.

The coordinator estimates the current state from readings coming from the visual tracker and the robot's onboard odometry. The estimated AO position is corrected for odometric drifts by using the information received from the landmark detector. The visual tracker typically runs 20 to 25 times per second, which allows the coordinator to update the target state with this same frequency. The landmark detector is invoked less often, as odometric sensing is precise enough over short periods of time. The planner is invoked on the order of 5 times per second. Whenever it is invoked, it computes the next set-point based the most recent state estimate.

## 4. CAN-COLLECTOR EXPERIMENT

The AO was used in several experiments between Stanford and other research centers (GRASP Laboratory at the University of Pennsylvania, University of Illinois at Urbana-Champaign, and Iowa State University). The most complex experiment is the can-collecting robot experiment done with the ITESM in Monterrey, Mex.

The main goal of this experiment was to evaluate the AO as a facilitating tool for collaboration in experimental robotics among dispersed groups. The can-collector system was designed and developed at ITESM. It consists of a semi-autonomous robot that follows routes specified using high-level commands, searches for soft-drink cans, and collects them in a bag.

The can collector is a Nomad 200 robot running three main software modules performing the following tasks: robot navigation according to the input script; 3-D sensing and can recognition and localization; and can collection.

In the experiment an operator located at ITESM designs a script of high-level navigation commands (e.g., "follow the wall on the right at some distance, until a corner is reached"), which is downloaded to the can collector. The can

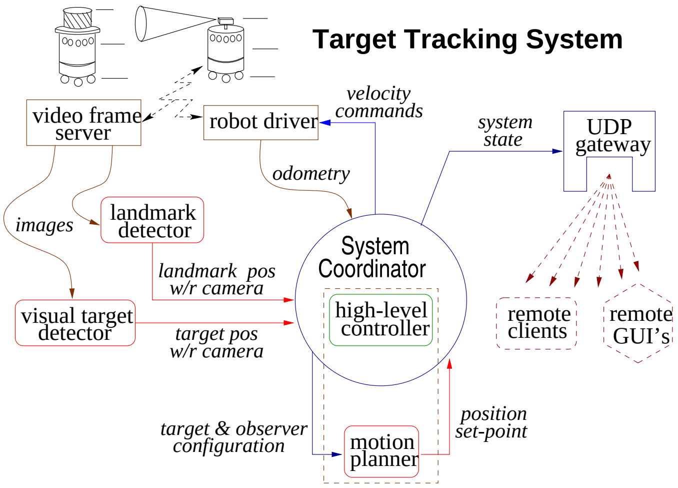


Figure 6: Interaction among the AO software modules

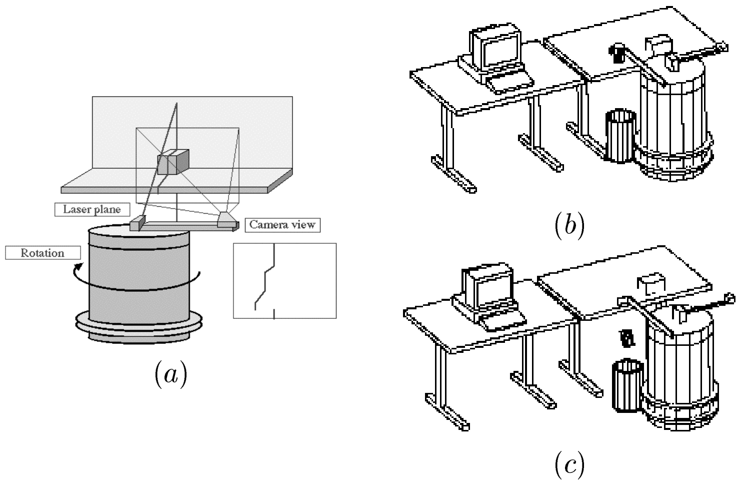


Figure 7: The can collector system

collector then executes these commands autonomously by running sophisticated on-board primitives that incorporate sensing to manage uncertainty in the can-collector's position. The can collector estimates its position using its own odometric and sonar sensors. It ignores the existence of the AO – it does not use any of the information computed by the AO.

The script also instructs the can collector when to perform a 3-D sensing operation to detect cans. The 3-D object recognition module receives data from a laser range finder mounted on top of the robot (see Figure 7(a)). A 3-D image is obtained at a fixed location of the can collector by rotating the robot's turret and thus sweeping a vertical plane of light. The input data is segmented into 3-D surface patches using a generalization of the one-pass segmentation algorithm for binary images. The surface patches are then characterized using central second-order moments and curvature descriptors in order to detect cylindrical surfaces.[12]

The collection of each detected can is performed by using a hooked rod mounted on the robot's turret. The motion of the robot and its turret is controlled by a specific software module in order to pull and drag a can over the table, into a basket attached to the robot (see Figure 7(b)). This module first checks that the can is reachable by the rod (which is mounted at a fixed height and has a fixed length).

While the can collector is in operation, the AO observes the can collector at some distance (see Figure 8), constantly

Figure 8: The AO and the can collector

sending information about its location and bearing to the operator at ITESM. There, the information is displayed as shown in Figure 5.

Using the AO feedback, the operator checks whether the downloaded script is achieving its purpose or needs to be corrected. Although the software was debugged on a similar robot at ITESM, a number of problems occured at Stanford. Because sonars are imperfect sensors and odometric sensing is quite sensitive to the physical properties of the ground surface, the can collector collided several times with obstacles. Moreover, the control module failed on ocassions to collect cans into the trash bag. The ITESM team was able to fix all those errors by looking at the graphic rendering and low-frequency video provided by the AO.

## 5. CONCLUSION

As robotics hardware become more sophisticated, it is increasingly important for researchers to be able to experiment with remote systems. Remote experimentation may drastically reduce hardware costs (including maintenance) by allowing more researchers to share the same equipment. By allowing researchers to experiment with their software on various hardware systems at different locations, it is also likely to yield more robust and portable software.

However, remote experimentation in robotics is complicated by the fact that the experimenters absolutely need to observe the robots in operation. Indeed, it is virtually impossible to correct most software bugs without observing the behavior of the robots controlled by this software. The Autonomous Observer is one tool developed to help remote experimenters. It presents two major advantages over the mere transmission of live video: low volume of data are exchanged between the experimental site and the experimenter's sites, and each experimenter can select the most pertinent viewpoints. However, nothing prevents the graphic rendering provided by the AO to be augmented by video images and sound.

Experiments with our AO have given very satisfactory results. However, these should be considered preliminary. The can-collector experiment was developed in ITESM before being downloaded at Stanford. Therefore, a large portion of the debugging had already been done in advance. Developing new software modules from scratch at a remote site would be much harder. Also, the can collector is a relatively simple robot with only 3 degrees of freedom. The AO technology must be extended to handle articulated robots (e.g., manipulator arms) and multiple robots. We currently work on multiple cooperating AO's tracking several targets. The AO could also be used to track objects other than the robots. For example, the current AO does not track cans, making video images necessary in order to detect when a can is missed by the collector and when it falls outside the bag. Finally, we have not used the AO yet when experimenters are located at multiple remote sites.

Another interesting extension of the current AO would be to equip it with on-line techniques to build environment models. Such an ability will eliminate the need to provide a map of the environment in advance. It would also enable the AO to operate in dynamic environments containing moving objects other than the targets.

## ACKNOWLEDGMENTS

This work was done under a project "The Intelligent Observer: A Tool for Collaborative Experimental Research Among Geographically Dispersed Groups" jointly supported by NSF (award IRI-9506064) and by México's CONACyT (grant 500100-5-C007-A). Alejandro Sarmiento was the recipient of a CONACyT fellowship. Stanford research on motion planning under visibility constraints is also supported by ARO MURI grant DAAH04-96-1-007 and NSF grant IIS-9711380.

## REFERENCES

1. J.E. Banta, Y. Zhien, X.Z. Wang, G. Zhang, M.T. Smith, and M. Abidi, "A Best-Next-View Algorithm for Three-Dimensional Scene Reconstruction Using Range Images," *Proc. SPIE*, Vol. 2588, pp. 418-429, 1995.

2. C. Becker, H. González-Baños, J.C. Latombe, and C. Tomasi, "An Intelligent Observer," *Proc. 4th Int. Symp. on Experimental Robotics*, pp. 153-160, 1995.

3. C. Becker, J. Salas, K. Tokusei, and J.C. Latombe, "Reliable Navigation Using Landmarks," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 401-406, 1995.

4. A.J. Briggs and B.R. Donald, "Robust Geometric Algorithms for Sensor Planning," *Algorithms for Robotic Motion and Manipulation (WAFR'96)*, J.P. Laumond and M. Overmars (eds.), A K Peters, Natick, MA, pp. 197-212, 1997.

5. Crass D. Crass, I. Suzuki, and M. Yamashita, "Searching for a Mobile Intruder in a Corridor - the Open Edge Variant of the Polygon Search Problem," *Int. J. of Computational Geometry and Applications*, **5(4)**, pp. 397-412, 1995.

6. B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *SIGGRAPH'96.* 1996.

7. M. Doherty, M. Greene, D. Keaton, C. Och, M. Seidl, W. Waite, and B. Zorn, "Programmable Ubiquitous Telerobotic Devices," *Proc. SPIE on Telemanipulator and Telepresence Technologies IV*, vol. 3206, pp. 150-158, 1997.

8. P. Fabiani and J.C. Latombe, "Dealing with Geometric Constraints in Game-Theoretic Planning," *Proc. IJCAI'99*, 1999.

9. H. Fuchs and U. Neumann, "A Vision of Telepresence for Medical Consultation and Other Applications," *Robotics Research – The Sixth Int. Symp.*, T. Kanade and R. Paul (eds.), pp. 565-571, 1993.

10. H. González-Baños, L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, R. Motwani, and C. Tomasi, "Motion Planning with Visibility Constraints," *Robotics Research - The 8th Int. Symp.*, Y. Shirai and S. Hirose (eds.), Springer, pp. 95-101, 1998.

11. L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, and R. Motwani, "Visibility-Based Pursuit-Evasion in a Polygonal Environment," *Proc. 5th Workshop on Algorithms and Data Structures (WADS'97)*, Springer-Verlag, pp. 17-30, 1997.

12. G. Hermosillo, A. Sarmiento, and J.L. Gordillo, "Differential Geometry Surface Descriptors Applied to Real-Time Object Recognition in Range Images," *Memorias de Visión-Robótica del Primer Encuentro de Computación (ENC'97)*, UA de Querétaro, Querétaro, pp. 124-133, 1997.

13. S. Hutchinson, G.D. Hager, and P. Corke, "A Tutorial on Visual Servo Control," *IEEE Tr. on Robotics and Automation*, **12(5)**, pp. 313-326, 1995.

14. D.P. Huttenlocher, J.J. Noh, and W.J. Rucklidge, "Tracking Non-Rigid Objects in Complex Scenes," *Proc. 4th Int. Conf. on Computer Vision*, pp. 93-101, 1993.

15. K. Kakusho, T. Kitahashi, K. Kondo, and J.C. Latombe, "Continuous Purposive Sensing and Motion for 2D Map Building," *Proc. IEEE Int. Conf. on Syst., Man and Cyb.*, pp. 1472-1477, 1995.

16. K. Kanatani, *Geometric Computation for Machine Vision*, Oxford Science Publications, 1993.

17. S.M. LaValle, H. González-Baños, C. Becker, and J.C. Latombe, "Motion Strategies for Maintaining Visibility of a Moving Target," *Proc. IEEE Int. Conf. on Robotics and Automation*, 1997.

18. P. Leven, S. Hutchinson, D. Burschka, G. Färber, "Perception-Based Motion Planning for Indoor Exploration," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 695-710, 1999.

19. M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. SIGGRAPH'96*, 1996.

20. T.-Y. Li and T.-H. Yu, "Planning Tracking Motions for an Intelligent Virtual Camera," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1353-1358, 1999.

21. J. Maver and R. Bajcsy, "Occlusions as a Guide for Planning the Next View," *IEEE Tr. on Pattern Analysis and Machine Intelligence*, **15(5)**, pp. 417-433, 1993.

22. J. O'Rourke, "Visibility," *Handbook of Discrete and Computational Geometry*, J.E. Goodman and J. O'Rourke (eds.), CRC Press, Boca Raton, FL, pp. 467-479, 1997.

23. N.P. Papanikolopoulus, P.K. Khosla, and T. Kanade, "Visual Tracking of a Moving Target by a Camera Mounted on a Robot: A Combination of Control and Vision," *IEEE Tr. on Robotics and Automation*, **9(1)**, pp. 14-35, 1993.

24. S.R. Sachs, D. Agarwal, and M. Blaufuss. "A Remote Camera System for the ALS Collaboratory," `http://www-itg.lbl.gov/~ssachs/telepresence/telepresence.html`, Lawrence Berkeley Lab., U.C. Berkeley.