Preprints of the Fourth International
Symposium on Experimental Robotics, ISER'95
Stanford, California, June 30–July 2, 1995

An Intelligent Observer*

Craig Becker H.H. González-Baños Jean-Claude Latombe Carlo Tomasi

Robotics Laboratory
Department of Computer Science
Stanford University, Stanford, CA 94305
{cdb, hhg, latombe, tomasi}@flamingo.stanford.edu

Abstract

This paper describes an integrated mobile robotic system dubbed the *intelligent observer* (IO). The IO is a mobile robot which moves through an environment (such as an office building or a factory) while autonomously observing moving targets selected by a human operator. The robot carries one or more cameras which allow it to track objects while at the same time sensing its own location. It interacts with a human user who issues task-level commands, such as indicating a target to track by clicking in a camera image. The user could be located far away from the observer itself, communicating with the robot over a network. As the IO performs its tasks, the system provides real-time visual feedback to the user. We have implemented a prototype of the IO which integrates basic versions of five major components: landmark detection, target tracking, motion planning, motion control, and user interface. We have performed initial experiments using this prototype, which demonstrate the successful integration of these components and the utility of the overall system.

1. Introduction

This paper describes the concept, design, and initial implementation of a system we call the *intelligent observer* (IO). Our goal for the IO is to develop a system that provides a human user with intuitive, high-level control over a mobile robot which autonomously plans and executes motions to visually track a moving target (see Figure 1). The user sends commands, such as "follow the next moving object which enters the view", and receives real-time feedback, such as a graphical display of the positions of the observer and target overlaid on a map of the environment.

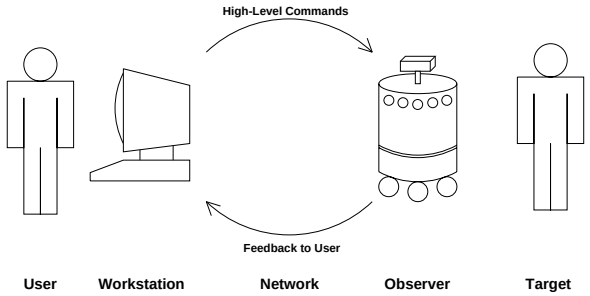

Figure 1. Interaction between a user and the intelligent observer.

Although the IO can be seen as an extension to a traditional teleoperation system, there are several major differences. First, it responds to high-level commands which are specified at the task level. There is no need for a "virtual joystick" or any other such control. Removing such low-level responsibilities from the human user provides many benefits, such as reducing the likelihood of human error and allowing the user to focus attention on more important, higher-level issues.

Second, it uses its internal representation to provide a more flexible feedback mechanism than would be possible by simply displaying the image seen by the observer's cameras. The IO can fuse information from various sensors and, using geometric information about the environment, reconstruct a view of the observed scene. This view could be a simple two-dimensional, "top-down" view or a more realistic three-dimensional view rendered from an arbitrary viewpoint.

The IO project brings together concepts and algorithms from computer vision, motion planning, and computer graphics in order to create a robust, useful, and integrated system. One important aspect of the system is that vision, often viewed merely as a mech-

*This research was funded by ARPA grant N00014-94-1-0721-P01 (ONR) and by an NSF/ARPA contract for ANVIL through the University of Pennsylvania. C. Becker is supported in part by an NSF Graduate Fellowship.

anism for aiding robot navigation, itself becomes the central objective of the system.

There are a number of possible applications for the IO system. Consider an engineer who must remotely monitor operations on a factory floor in order to analyze the performance of an assembly line. The IO would provide a convenient and automatic telepresence which could, for example, automatically move to track a single part as it travels through the assembly sequence. The engineer need not focus on the mundane task of moving the observer, but can instead concentrate on the assembly line task itself.

As another example, consider a security surveillance system for a large building. Typically a human operator is presented with a large number of views from various cameras positioned throughout the building. To be sure of catching any unusual activity, the user must cycle regularly through many views, possibly watching several views simultaneously. Here, the IO would alert the operator of motion in any camera view, and also plan new views as required to keep any moving objects in view. In this case the mobile robot is replaced by a large number of cameras, each of which could be either fixed or able to pan and tilt. The basic problem, however, is the same: the IO must still choose the best view of the target. This is accomplished by either moving the current camera or switching to a different camera.

The rest of this paper is organized as follows. In Section 2 we describe the overall design of the IO and of the roles of the various components that make up the system. In Section 3 we describe an initial implementation of each component of the IO. In Section 4 we describe the results of our experimentation with the system. Finally, in Section 5 we draw conclusions from our work and discuss possible directions for further research.

2. Overall Design

The complete IO consists of five major modules: landmark detection, target tracking, motion planning, user interface, and motion control. The relationships between these modules and the actual robot are shown in Figure 2. Each module is described more fully below.

Landmark Detection As the observer moves around in its environment it must always keep track of its current position. Our approach to solving this problem involves placing artificial landmarks throughout the environment. Many researchers have studied the use of landmarks to aid robot navigation; for examples see [1, 3, 5, 6, 7].

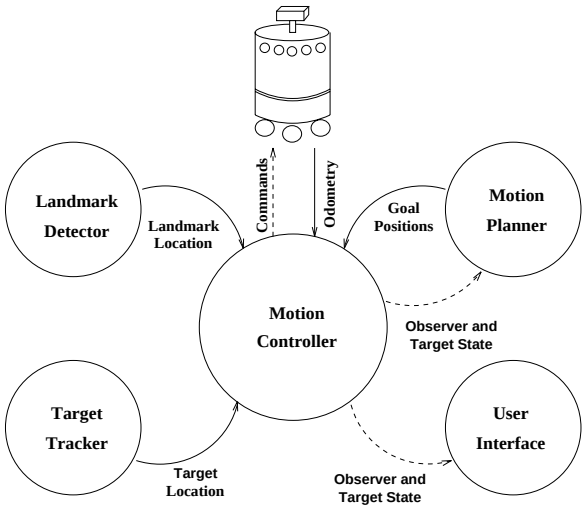

Figure 2. The components of the IO system.

In our system, the positions of the landmarks are provided as part of a map which is provided to the IO. Each landmark induces a *landmark region* such that the landmark is visible to the robot whenever it moves within that region. The robot localizes itself by visually detecting landmarks and determining its position relative to them. Since the success of the robot depends on this self-localization, the vision algorithms used to detect the landmarks must be fast, accurate, and robust.

Target Tracking The central task of the IO is to observe moving objects, or targets. There are two main requirements: first, that the IO recognizes when a new target enters its field of view; and second, that the IO is capable of tracking the desired target as it moves. Since the robot must respond to the movement of objects, all tracking must happen in real time. In general, target motions may be almost totally unconstrained and the targets themselves may by nonrigid and of unknown shape; humans are a good example. In order to handle such targets, we can apply real-time versions of tracking algorithms such as those described in [4]. Currently, however, we use a simplified approach as detailed in Section 3.

Motion Planning Traditional motion planning problems involve finding a collision-free path from an initial region I to a goal region G in the presence of obstacles whose geometry is known in advance. The assumption of a static world allows the entire path to be precomputed in an off-line fashion. A robot can then follow the path without fear of collision.

In the context of the IO, the planning problem is quite different. The goal is no longer a fixed location; in-

stead, our goal is to remain in view of a moving target at each point in time. Since the target motion is not known in advance, we must employ an on-line algorithm. In order to maintain a view of the target we must avoid occlusions due to obstacles and, as in the traditional case, we must also avoid collisions with obstacles. Any given obstacle may obstruct the robot's view, or its motion, or both. For example, a glass wall obstucts motion but not visibility, while a table may obstruct both, depending on its height.

We define an *on-line view planning problem* which must be solved by the IO as follows. The planner is given as inputs the position R_t (at time t) of the observer, the position T_t of the target, the region $\mathcal{R}_{t+1}$ of reachable locations of the observer at time $t+1$, and a function $\mathcal{T}_{t+1}$ which gives the probability of the target moving to any position T_{t+1}. As output, the planner produces a new observer position $R_{t+1} \in \mathcal{R}_{t+1}$ which maximizes the probability of maintaining an unobstructed view of the target for all possible T_{t+1}.

The problem outlined above is "local" in the sense that it only considers the probability one step in advance. A less local version would attempt to maximize the probabilty over some window of time, instead of only at time $t+1$.

User Interface One simple way of providing feedback to the user of the IO is to display live video from the robot's point of view. There are, however, several drawbacks to this approach. First, it makes no use of the higher-level representation maintained by the IO. Second, it limits the viewpoint to that of the robot's camera. Third, we would like to avoid "information overload" in cases where there are multiple cameras (or multiple observers). Finally, full-motion video requires high transmission bandwidth; this problem is especially important when the user is far away from the observer.

Instead, we have chosen to use a different approach. The user is presented with a synthetic reconstruction of the environment in which the IO is operating. This reconstruction can be either a two-dimensional overhead view or a three-dimensional view from an arbitrary vantage point. This module relies on an appropriate geometric model of the environment as well as information about the positions of the observer and the target.

Assuming that the environment is largely static, only a small amount of new information is required to update the scene, alleviating the bandwidth problem. This approach also allows us to fuse input from multiple cameras into a single, unified view.

Motion Control The motion controller takes the role of the top-level supervisor in the IO system (see Figure 2). It coordinates communication with the other components and produces appropriate low-level commands to control the physical robot. In addition, it periodically updates the current estimate of the robot's position based on feedback from the landmark detector and the odometric information from the robot. When appropriate, it also requests a new goal point from the motion planner. Finally, is sends updated information to the user interface.

The IO system is made up of a number of different processes, each of which has its own characteristic cycle time. Because of this, the motion controller must communicate asynchronously with the other processes, using new information as it becomes available. The exception is communication with the path planner, which follows a transaction-based model: the controller requests new a goal once it has achieved the previous one.

3. Implementation

This section describes our initial implementation of each of the components of the IO. Each of the five components has been implemented as a separate Unix process. They communicate with one another using standard TCP/IP protocols, making it possible to run them on different machines to increase performance. In fact, during our experimentation we ran the landmark detector, motion controller, and user interface on a Sun SPARCstation 20, the target tracker on an SGI Indigo^2, and the motion planner on a DEC Alpha/AXP. The implementation of each process is detailed below.

Landmark Detection As discussed earlier, we rely on artifical landmarks to localize the robot. Our landmarks, shown in Figure 3, are placed on the ceiling at known positions throughout the robot's workspace. Each landmark consists of a black square with a 4×4 pattern of smaller squares inside of it. The detection algorithm first identifies edge pixels in the image (using a variant of the algorithm presented in [2]), and then looks for edge chains that are consistent with the boundary of a square. When such a chain is found, the corners are detected and then lines are fit to the pixels which make up each edge. The slopes of these lines yield the orientation of the landmark; their intersection points locate the landmark's corners. Once the landmark is localized, the positions of the inner squares are computed and their intensities are read from the image using bilinear interpolation. These intensities are grouped into "black" and "white" sub-

Figure 3. A sample ceiling landmark.

groups to determine the 16 binary values they represent. Four of these values are used to disambiguate the landmark's orientation, and the others encode the landmark's unique ID.

The landmark detector uses 320×240 grayscale images as input. Each frame requires approximately 0.5 seconds to process; this includes detecting, localizing, and identifying the landmark. The algorithm is very accurate: the translational error has zero mean and a standard deviation of 0.75 inches, while the rotational error is also zero mean and has a standard deviation of 0.5 degrees.

Target Tracking This component is used by the observer to detect and visually track a moving target. Currently the target consists of a number of black, vertical bars on a white cylindrical "hat" which sits on top of the target robot (see Figure 5). The tracking algorithm detects these bars in each image and, given the camera parameters and the physical size of the bars, computes the target's location relative to the camera.

The tracking system operates at approximately 5 frames per second using 160×120 grayscale images as input. For each frame, it determines the angle θ and distance r to the center of the target. Typically the target can be detected at distances ranging from 2 feet to 10 feet from the camera. Experimentation shows that θ is accurate to within ± 1 degree, and r is accurate to within ± 4 inches.

Motion Planning Our implemented solution to the on-line view planning problem is deliberately simple so that planning time is reasonable. The planner is given a polygonal map of the workspace, as well as bounds v_R and v_T on the maximum velocities of the observer and target, respectively. These velocities are given in terms of distance per planning cycle.

Suppose a new goal position is requested at time t. The planner is given the positions R_t and T_t of the observer and target. We first find $\mathcal{R}_{t+1}$ by constructing a disk of radius v_R centered at R_t, and intersecting that with the free space in the environment. Similarly, we take $\mathcal{T}_{t+1}$ to be the intersection of the free space with a disk of radius v_T centered at T_t. We assume that all positions within this region are equally probable.

To compute a new goal position R_{t+1}, we use the following approach. First we uniformly sample $\mathcal{T}_{t+1}$, producing m possible positions of the target. For each position we then compute the resulting visibility polygon V_i $(i = 1, \ldots, m)$. We then sample $\mathcal{R}_{t+1}$, yielding n possible goals for the observer. From these samples we select the one which falls within the maximum number of visibility regions V_i. Ties can be broken in any number of ways; currently we choose the goal which results in the smallest motion of the observer. Assuming that we sample densely enough, this approach approximately maximizes the probabilty of maintaining a view of the target at time $t + 1$.

In the experiments we have performed, a single application of this planning technique requires between 0.2 and 0.8 seconds depending on the configuration of the observer and target relative to the obstacles in the environment.

User Interface In our initial system we have adopted a simple, two-dimensional user interface as shown in Figure 4. Physical obstacles are shown in black and configuration space obstacles are dark gray. The positions of the observer (R) and target (T) are updated in real time. In addition, the visibility region of the observer is shown in light gray.

Motion Control As mentioned before, this module not only controls the motion of the robot, but it also coordinates communication between all of the other modules. In a sense it is the top-level supervisory process for the whole system.

The most basic task of the motion controller is to issue velocity commands to the robot. This is accomplished using a tight feedback loop which runs at approximately 10 cycles per second. At each cycle, the loop reads odometry information from the robot and computes a velocity command to move the robot toward the current goal position. The control system measures and compensates for the communication delays present in the system. When new information is available from the landmark detector, the controller uses this to update the current position of the robot; this is done to compensate for errors which would build up over time if only odometric information were used.

The landmark detector returns the position of the robot relative to a landmark at the time s that a par-

ticular image is captured by the camera. The information is not received by the motion controller until some later time t. Due to latency in our digitizing hardware and to the image processing time, the difference between s and t can be as long as two seconds. To compensate for this delay, the controller keeps a history of the robot's position over the past several seconds. An updated position estimate at time t is computed by adding the difference between the stored positions at times t and s to the position sensed at time s using landmarks.

The error signal for steering and translation is based on the (x, y) goal points received from the motion planner. An error in both cartesian directions is computed by using the current position estimate of the observer. The steering error and the translation error are then calculated in such way that the required steering is minimized. Both of these errors are then provided to the control law.

A totally separate control loop is used to position the camera in response to feedback from the target tracking system. The camera is mounted on a turret which can rotate independently of the robot's drive wheels. At each cycle, the turret control loop obtains the (r, θ) coordinates of the target, where r is the distance from the observer to the target and θ is the angle between the camera's optical axis and the line between the observer and the target. To account for delays, the actual target position is estimated using the last reading from the target tracker, an estimate of the target velocity, and the measured delay between actually acquiring an image of the target and completing the processing of that image. The estimated target position is the error signal used by the turret control law.

The control laws for the turret, steering, and translation are all similar in structure. These are simple P-controllers with compensation for the time-delay found inherent in communication with the robot. This time delay is present both when new odometric information is requested and when new velocity commands are issued. The control laws attempt to compensate for this lag based on estimates of previous delays. This compensation ideally performs a linearization by feedback, and the whole system is reduced to three second-order systems once the loops are closed. The simplification is valid as long as the feedback gain is not excessive.

The three P-controllers have exactly the same gain. In our system we attempted to minimize both time-response and oscillations. In theory, the ideal gain is the one that makes the two closed-loop poles at each loop identical. In practice the ideal value is 60% of the theoretical one, which shows that the linearization by feedback approach works satisfactorily, and that the system was effectively reduced to three first-order systems under the imposed operating conditions.

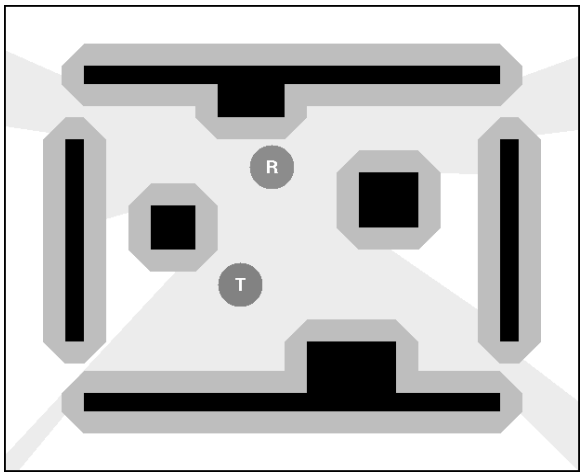

Figure 4. The display presented to the user.

4. Experimental Setup

This section describes our initial experiments with the IO system. Our experimentation has two goals: first, to validate the utility and robustness of the chosen algorithms for each of the various system components; and second, to demonstrate the feasibility of integrating all of the components into a unified system.

Our experiments took place in our laboratory, a fairly typical office environment. As obstacles we used desks, chairs, and large cardboard boxes. A map, as required by the motion planner, was constructed by approximating the obstacles by polygons; this map is shown in Figure 4.

The observer itself is a NOMAD-200 mobile robot. It is equipped with an upward-pointing camera for landmark detection and a forward-pointing camera for target tracking. Both cameras are mounted rigidly to the robot's turret, which can rotate independently of its drive wheels. This allows the turret, and therefore the tracking camera, to rotate based on the motions of the target without affecting the motion of the robot. The robot's onboard computation is provided by an internal Pentium-based computer.

As a target, we used a second NOMAD-200 equipped with a special "hat" required by our simplified tracking algorithm, as described above. The target was moved under joystick control by a human operator. Figure 5 shows a view of both the target and the observer.

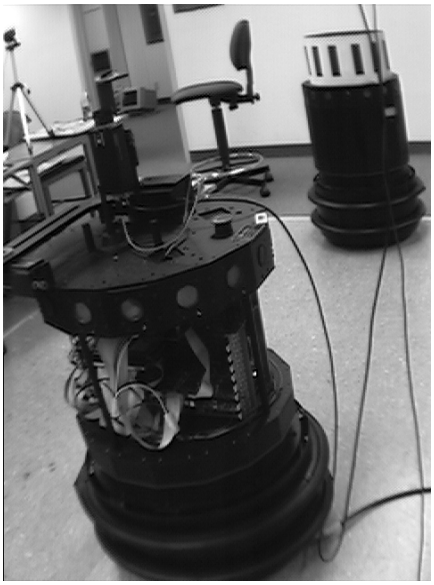

Figure 5. A view of the IO in action, following a second robot which serves as the target.

As stated above, the purpose of our experimentation was to verify each of the individual components of the system, as well as to show that they could be successfully integrated. Although we have been able to quantify the performance of several of the components, it is too early to quantitatively describe the performance of the system as a whole. Qualitatively, however, the system was successful: the observer was consistently able to visually track the target and keep it in view in the presence of obstacles.

The most important conclusion of our experimentation is that, even though each of the components which makes up the system may periodically fail, the overall system is robust in the face of those failures. For example, the target tracker or landmark detector may periodically given an inaccurate result, but these errors are tolerable since they are random variables with zero mean and a relatively small variance. In addition, the sampling rate is high enough relative to the reaction time of the control system that sporadic sensing failures do not significantly alter the long-term performance of the system.

5. Conclusion

In this paper we have described the concept and high-level design of the intelligent observer. We have also described our initial implementation of the components of the system as well as early experiments with the system as a whole. Up to this point, our goal has been to design the overall system and to develop simple versions of all of the system components. Our continuing work focuses on two types of extensions: first, those which increase the generality and robustness of current components; and second, those which add new functionality to the concept of the overall IO system.

In terms of improving upon the current components, we have the following goals. First, we are working on ways to remove the discretization in our planning implementation by developing new visibility algorithms more directly related to the view planning problem. Second, we are working on ways to allow the planner to look farther ahead in time when it considers possible target moves. Third, we are working to implement a more general target tracking mechanism which removes the need for a special visual cue on the target.

In terms of adding functionality to the IO system, we are working on several extensions. First, we plan to add a component which automatically builds a 3D model of the observer's environment. The model will be generated using a laser rangefinder (for geometry) and a camera (for color). This model would then be used to reconstruct a view for the user. Second, we plan to extend the view planner to deal with multiple observers which can cooperate to track a single target. Finally, we will also consider the case of multiple targets and the problem of dynamically assigning observers to these targets.

References

[1] C. Becker, J. Salas, K. Tokusei, and J.C. Latombe. Reliable navigation using landmarks. In *Proc. IEEE Int'l Conference on Robotics and Automation*, 1995.

[2] J.F. Canny. A computational approach to edge detection. *IEEE Transactions on PAMI*, 8(6):679–698, 1986.

[3] S. Hutchinson. Exploiting visual constraints in robot motion planning. In *Proc. IEEE Int'l Conference on Robotics and Automation*, pages 1722–1727, 1991.

[4] D.P. Huttenlocher, J.J. Noh, and W.J. Rucklidge. Tracking non-rigid objects in complex scenes. Technical report, Cornell University Department of Computer Science.

[5] D.J. Kriegman, E.Triendl, and T.O. Binford. Stereo vision and navigation in buildings for mobile robots. *IEEE Transactions on Robotics and Automation*, 5(6):792–803, 1989.

[6] A. Lazanas and J.C. Latombe. Landmark-based robot navigation. *Algorithmica*, 13:472–501, 1995.

[7] T.S. Levitt, D.T. Lawton, D.M. Chelberg, and P.C. Nelson. Qualitative navigation. In *Proc. DARPA Image Understanding Workshop*, pages 447–465, 1987.