Scavenging with a Laptop Robot

Alan Davidson, Julian Mason, Susanna Ricco, Ben Tribelhorn, and Zachary Dodds
Harvey Mudd College Computer Science Department
1250 N. Dartmouth Avenue
Claremont, CA 91711
adavidso, jmm, sricco, btribelh, dodds@cs.hmc.edu

Abstract

This paper overviews the hardware and software components of a robot whose computational engine uses only commodity laptop computers. Web cameras, sonar, and an inexpensive laser-pointer-based ranger provide sensing. By shifting emphasis from engineering precision to computational heft, the robot succeeded at several tasks in the 2005 AAAI scavenger hunt. As a result, this "robot-as-computer-peripheral" design confirms that capable, low-overhead robots are within the price range of a large group of educators and enthusiasts.

Introduction

We designed and implemented a robot to participate in the AAAI 2005 Robotic Scavenger Hunt Competition. The Scavenger Hunt is a showcase for robotics researchers to exhibit the abilities of their robots to find and identify objects in a dynamic environment.

Our goal was to demonstrate that a low-cost, sensor-limited robot could be competitive against well-equipped conventional robots. Our focus on computer vision and software processing led us to write all of our own software, including drivers for the motors.

The following two sections highlight the platform's physical resources, including both stepper and servo motors, along with two cameras and six home-brewed rangers. We then outline the software structure that coordinates these subsystems, with a focus on the navigation and visual processing. We conclude with several of the challenges and successes of working with our robot.

Hardware

Our entry was based on an Evolution Robotics ER1 (Evo 2002). The standard ER1 provides a differential-drive platform and a web camera. The robot's computational power is provided by a laptop which rests on the back of the chassis. We extended our ER1 to use five Devantech SRF04 sonars and a second web camera, and have removed the IR sensors. Since the web cameras' combined bandwidth requirements exceeded a single USB bus, we mounted a second laptop as shown in Figure 1. We also designed and built a laser rangefinder from two Maxx MX-400 servo motors, a laser pointer, and a camera. The laptop communicates with the motors and sonar rangers through a Pontech SV203 servo controller board and our own custom interface circuit.

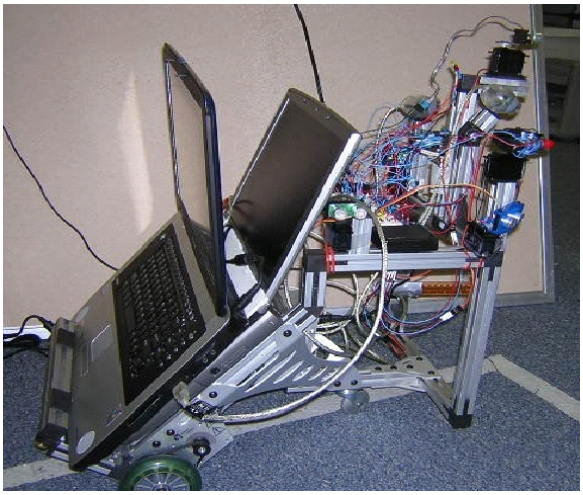

Figure 1: The laptop-controlled scavenging robot built atop an Evolution ER1. The sensor suite points toward the typical direction of travel, and onlookers can follow the robot's reasoning by watching the screens from behind. Both custom and commercial interface circuits mediate between the two sides.

We fashioned a makeshift gripper using a front-mounted strip of duct tape, which could reliably grab and carry a beach ball simply by colliding with it. Two rulers mounted on additional servo motors could detach the ball when the robot arrived at its destination.

This design led to a remarkably inexpensive platform: the entire machine cost approximately $600. The laptops, of course, are considerably more expensive, but it's important to note that those two computers are not dedicated to running the ER1. Indeed, they are now in use for much more typical tasks. For example, the team gave its presentation during the AAAI robotics workshop on one of them. Their role is analogous to that of a laptop mounted atop a Pioneer or communicating via wireless with an AIBO. Such support

computers are not typically quoted in the platforms' costs.

Range Sensing

The five sonar units mounted around the front half of the robot act as primary proximity sensors – basically virtual collision detectors. When a sonar detects an obstacle within about 35 cm, it triggers a "bump."

The other range sensor is a custom laser rangefinder made from a laser pointer, two servo motors, and a camera. The servos enable the laser to pan and tilt. The camera's exposure is adjusted in software to a level so low that only very bright features such as the laser dot are visible. Because we perform a short horizontal sweep of the laser, only a small, known subset of the observing camera image can detect the dot. Thus, other bright features such as lights and floor reflections are easy to ignore.

We calculate the distance to a surface by intersecting the vector from the camera center to the surface (extracted from the camera pixel of the laser dot) and the vector made by the laser pointer (via the kinematics of the pan/tilt mechanism). Typically, noise and/or modeling error prevents these two rays from intersecting. We take the least-squares solution to find the point that minimizes the distance to both rays simultaneously. At ranges up to a meter our system is accurate to roughly two centimeters, and at ranges up to four meters it is accurate within twenty centimeters. Beyond this, the laser dot becomes too faint to be identified by our camera.

Architecture

A finite state machine coordinates the scavenger's subsystems and subtasks as shown in Figure 2. We implement the state machine using an object-oriented approach, following standard design patterns (Adamczyk 2003). This design facilitates extending our control system to support new behaviors. Additionally, our control system keeps the behavior of the robot independent from the specific sensors being used, which allows us to substitute different sensors quickly and easily.

The robot's default state is called *Wander*. *Wander* has two different modes. One uses virtual waypoints on the map to explore the world, and the other attempts to travel as straight as possible in order to find the next arrow in a sequence. When the robot detects an object of a known color, it enters the *Found Object* state. In this state the robot servos towards the object in order to identify it. After visual identification, it is marked on the map, and the robot returns to *Wander*. In the case of an arrow, the robot enters *Follow Arrow*, calculates the angle of the arrow, travels to it, and turns before returning to *Wander*. When the robot encounters a beach ball, it "grabs" the ball with the duct tape and enters the *Found Beach Ball* state. In this state, the robot uses its localization routines to assist travel towards a prespecified destination. It chooses this path via predefined waypoints. Once there, the robot releases the beach ball and returns to *Wander*.

When the sonar detects an obstacle nearby, the robot pauses and enters *Panic*. The *Panic* state uses data from the laser rangefinder to perform a localization routine. The robot then turns away from the object and resumes its previous behavior.

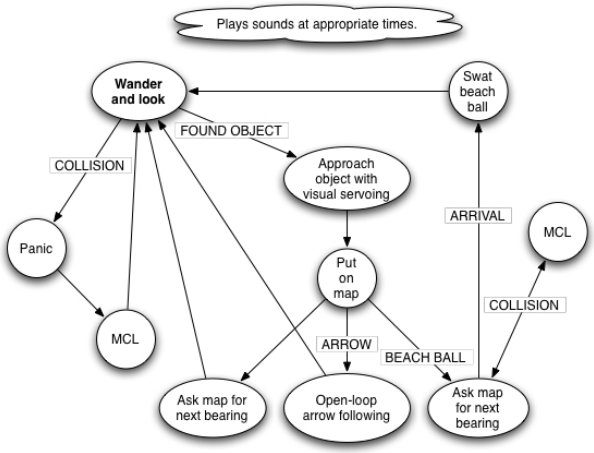

Figure 2: A graphical summary of the state-machine controller and the sensor conditions that trigger transitions among states. Music and voice files chime in as appropriate.

This state machine, implemented in Python, communicates with all of the other software components using standard socket protocols. The computationally expensive navigation and vision subsystems are written in C++ for speed, while the motor drivers and range-sensor interface remained in Python for simplicity.

Navigation and Localization

To effectively compete in the scavenger hunt, our robot had to report the locations of objects it encountered and to return a specific object to a designated location. To support these tasks, we hand-crafted a map of the competition area and used it within our own software framework for spatial reasoning. Our robot localizes itself using Monte Carlo Localization (MCL) (Thrun *et al.* 2000). Our implementation uses data from the laser rangefinder with empirically derived triangular probability distributions of sensor and motion error. MCL, however, does not provide a single position for the robot, but rather a collection of hypotheses of where the robot might be. We require the robot to have a single, specific location on the map, so we assume the robot is actually at the position of the most likely hypothesis (the probability mode).

We have found that this approach is effective, even though it can cause the robot's estimated position to change drastically. In runs at AAAI, the robot was able to correct a six-meter error using MCL and find its actual position within a third of a meter. During the competition run, a similar adjustment of about thirty centimeters occurred as the robot traveled to its destination.

In addition to localization, our software provides two kinds of path planning: "wandering" planning and "destination" planning. Both are based on a series of virtual, pre-

specified waypoints placed throughout the map, as shown in Figure 3. For simplicity, we create enough waypoints to have a line-of-sight from every point on the map to a waypoint. Each waypoint stores the heading to two neighboring waypoints, one for each type of planning.

To wander, the robot asks the navigation software for directions to the nearest visible waypoint. The robot then follows this bearing until it is within a predefined distance of the next waypoint. Once there, it gets the heading towards the next waypoint, and continues this pattern. If the robot encounters an obstacle along this path, it updates its MCL particle filter. As this might move the robot's estimated position by a large amount, the robot requests new instructions to the (possibly changed) nearest waypoint. When the robot encounters an object, it goes into the *Found Object* state. This results in object-dependent behavior. For instance, when the object is an arrow, the robot follows the arrow and returns to *Wander*, ignoring waypoints.

The robot also relies on waypoints to deliver the beach ball. Instead of following the "wandering" headings, the robot follows the "destination" headings, traveling as before. In this mode, the robot does not stop to identify other objects it encounters, though it still localizes itself when it finds an obstacle.

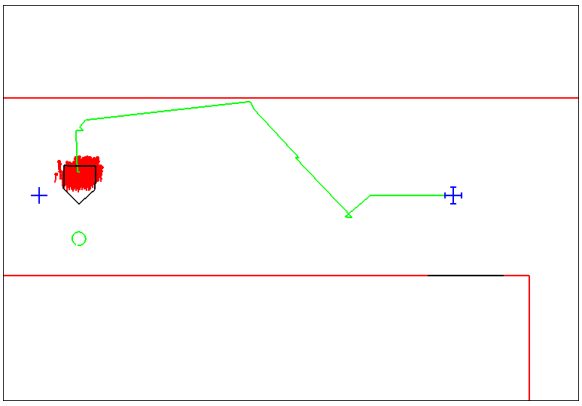

Figure 3: In this map snapshot, the pentagonal robot has wandered from the destination waypoint (the Jerusalem cross at right) towards a second waypoint (the cross at left). When obstacles are detected, a localization update occurs. The red dots represent the particle filter's pose hypotheses with the robot placed at the most likely. Observed objects, e.g., the empty circle representing the beach ball, are mapped relative to the current maximum-likelihood pose.

Visual Object Recognition

We use an exclusively vision-based system for identifying scavenger hunt objects. One of the robot's cameras is angled downwards to better view nearby objects on the floor. To recognize a particular item, our software applies the sequence of image-processing routines detailed in Figure 5.

Each color of interest is defined by thresholds in both RGB and HSV space. For convenience, our software allows the user to change these thresholds in real time by selecting

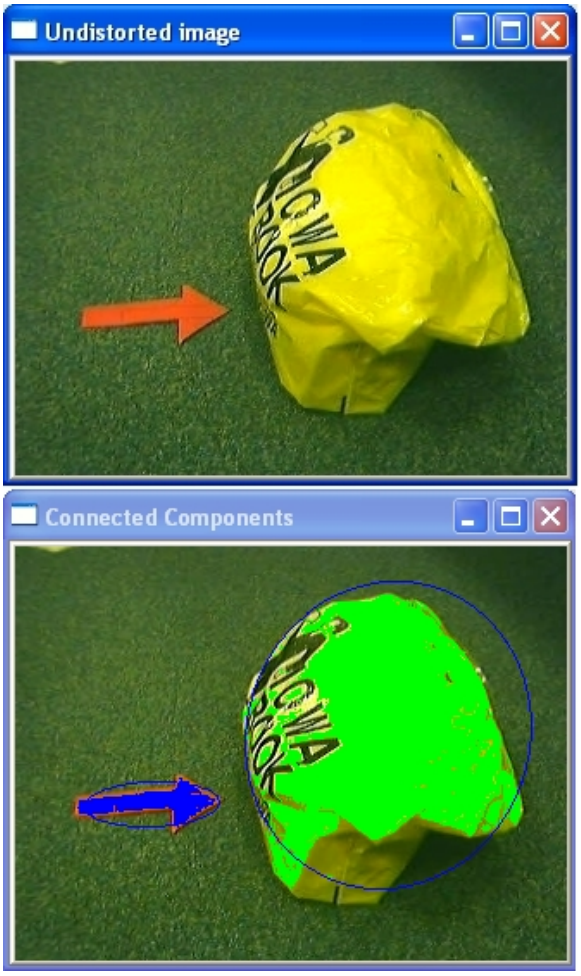

Figure 4: A raw image and the results of segmentation, connected components, and shape processing: best-fit ellipses surround both the arrow and the yellow bag.

a region in the video window. These color definitions can be read from and saved to a file. This interface made quick work of the frequent and important task of calibrating for different lighting conditions.

After the image is segmented with these color definitions, a connected-components routine extracts the largest contiguous region of the most abundant color and passes it to several shape filters.

Shape analysis consists first of PCA-based (i.e., coordinate-free) ellipse-fitting (Jolliffe 1986), which estimates the major and minor diameters and orientation of the region as shown in Figure 4. Using these statistics and the area (raw pixel count) of the component, the program calculates the object's roundness (Russ 1998) as the ratio of its area to that of a circle of the same diameter:

$$\text{Roundness} = \frac{4 \cdot \text{Area}}{\pi \cdot (\text{Max Diameter})^2}$$

For simplicity, we assume that the maximum diameter of an object is the diameter of the major axis of the associated

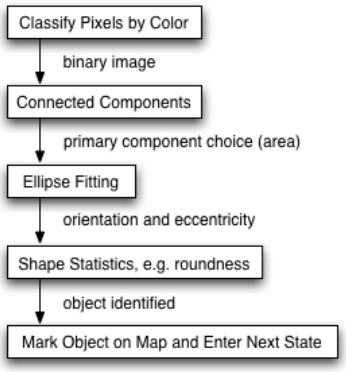

Figure 5: Diagram of the visual processing pipeline used in object identification.

ellipse.

Empirically derived thresholds for a gamut of characteristics, as shown in Table 1, enable the system to consistently distinguish among objects as similar as the orange dinosaur, orange bowl, and orange cone. These items can be seen in Figure 6.

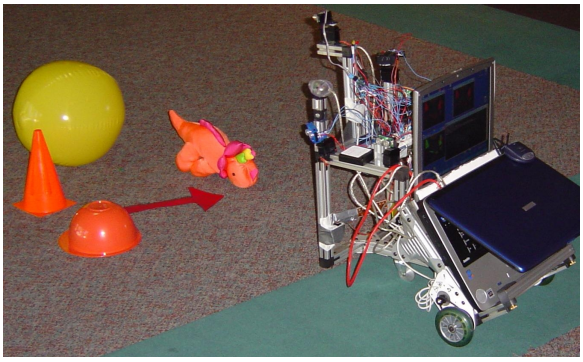

Figure 6: Our robot with several of the items it can recognize. Since the color definitions are easily changed, we could use red arrows in this building and off-white arrows at the competition.

Our robot can also follow a sequence of arrows placed on the floor. When it sees an arrow, the robot calculates the distance to the arrow and the direction in which the arrow is pointing. The distance to an object in front of us on the floor is proportional to the vertical position of the object in the camera image. This relationship is governed by an empirically derived exponential curve. We also found the number of pixels per horizontal unit of length as a linear function of the pixel height in the image. From this, we can calculate the position of both the head and the tail of the arrow relative to our robot. With this information, the robot can position itself on top of the arrow and turn to face the indicated direction. It then returns to the *Wander* state and continues to search for objects. The robot has successfully followed arrows in a variety of situations including around corners and down hallways.

Challenges

A major goal of the AAAI scavenger hunt event was to showcase a robot's ability within a real-world environment. The transition from the lab to the contest floor introduced a variety of challenges. In particular, people tended to cluster at the front of the robot – interactions which interfered with its map-based localization routine. While our avoidance behavior could adapt to this situation, our localization software depended on being able to reliably determine the distance to the static obstacles in the hallway. With people blocking clear paths to the obstacles we had included in our map, our otherwise accurate localization failed. Additionally, we found that the behavior of the spectators was often adversarial rather than cooperative, e.g., attempting to crowd the robot. To mitigate this, we added a routine that asked obstacles to step aside – people usually complied while other obstacles remained behind to assist localization.

People were not the only obstacles in the contest area which caused problems for our robot. With only five sonars, the robot had poor proximity resolution. Consequently, it easily missed narrow obstacles such as the legs of tables and easels. All of our sonars were mounted high enough off the ground that they also missed smaller robots, such as the Pioneer by iRobot. Both of these problems could be solved by adding extra sensors.

By far the most challenging difference between the lab environment and the competition setting was the lighting. Consistent lighting is essential for our vision-dependent robot. Our lab has white floors and relatively constant overhead lighting. The carpet at the hotel had an elaborate pattern containing many different colors, several of which were similar to the colors of the scavenger hunt's objects. The hotel's mood lighting created extremely bright patches interspersed with large shadowed areas. We had difficulty distinguishing colors in the shadowed areas, so we tuned our color definitions to accept only in the well-lit areas. Naturally, this limited the places where we could detect objects.

Finally, Evolution Robotics has discontinued the ER1 platform, limiting future development of this robot. What's more, some early ER1s have a defective control module. Figure 7 shows one of the two burned power regulators we replaced in the week before the competition.

Results and Perspective

During the competition, we demonstrated the ability to identify a beach ball and deliver it to a specified location. We differentiated between an orange cone, an orange stuffed dinosaur, and an orange bowl. We also successfully followed a series of arrows. These successes earned First Place in the scavenger hunt and a Technical Innovation Award for "Overall Excellence in a Fully Autonomous System."

We have shown that a laptop robot can be a cost-effective yet powerful system. The substantial onboard processing enables investigation of algorithms that exhaust other small models. By utilizing commodity hardware such as web cameras and laptops, our approach allows a broader group of

Object	Red	Green	Blue	Hue	Saturation	Value	Additional Constraints
Arrow	91 – 127	32 – 57	0 – 22	0.25 – 0.52	0.79 – 1.00	0.36 – 0.50	$0.1 \leq$ Roundness $(\mathcal{R}) \leq 0.5$
Beach Ball	148 – 253	1 – 94	0 – 72	0.01 – 0.23	0.69 – 1.00	0.56 – 0.99	Pixel Count ≥ 2500
Bowl	98 – 183	97 – 182	0 – 4	-0.66 – 1.00	0.98 – 1.00	0.38 – 0.72	$\lvert$Angle$\rvert < 72^\circ$ and $\mathcal{R} \geq 0.35$
Cone	98 – 183	97 – 182	0 – 4	-0.66 – 1.00	0.98 – 1.00	0.38 – 0.72	$\lvert$Angle$\rvert \geq 72^\circ$
Dinosaur	98 – 183	97 – 182	0 – 4	-0.66 – 1.00	0.98 – 1.00	0.38 – 0.72	$\lvert$Angle$\rvert < 72^\circ$ and $\mathcal{R} < 0.35$

Table 1: Several of the visual characteristics and their admissible ranges used by the vision system to identify scavenger hunt objects. The color values listed are those from the conference, though they change significantly based on lighting conditions. These values are for off-white arrows, yellow beach balls, and orange bowls, cones, and dinosaurs. Note that the bowl, cone, and dinosaur all use the same color definition, and are distinguished solely by shape.

Figure 7: One of the power regulators within Evolution Robotics' early Robot Control Modules looking a bit worse for wear...

educators and enthusiasts to explore the field of robotics.

Future Work

Future work at Harvey Mudd College will focus on improving robots' spatial-reasoning algorithms by incorporating vision. In particular, there is ongoing research in three dimensional mapping (Wnuk, Dang, & Dodds 2004), using visual data to extend the capabilities of MCL, and improving odometry using an optical flow approach (Campbell, Sukthankar, & Nourbakhsh 2004).

For additional resources including the scavenger hunt robot's source code, please visit `http://www.cs.hmc.edu/~dodds/AAAIsh`.

References

Adamczyk, P. 2003. The anthology of the finite state machine design patterns. In *Proceedings of the Pattern Languages of Programs Conference (PLoP)*.

Campbell, J.; Sukthankar, R.; and Nourbakhsh, I. 2004. Visual odometry using commodity optical flow. *Proceedings of the American Association of Artificial Intelligence (AAAI)*.

Evolution Robotics. 2002. *http://www.evolution.com*.

Horn, B. K. P., and Schunck, B. G. 1992. Determining optical flow. *Shape Recovery* 389–407.

Intel Corporation. 2000. *Open Source Computer Vision Library: Reference Manual*.

Jolliffe, I. T. 1986. *Principal Component Analysis*. Springer.

Lucas, B. D., and Kanade, T. 1981. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, 674–679.

Performance Motion Devices, Inc. 2002. *Pilot Motion Processor: Programmer's Reference*.

Russ, J. C. 1998. *The Image Processing Handbook*. CRC Press, Boca Raton, FL.

Shi, J., and Tomasi, C. 1994. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*.

Thrun, S.; Fox, D.; Burgard, W.; and Dellaert, F. 2000. Robust monte carlo localization for mobile robots. *Artificial Intelligence* 128(1-2):99–141.

Wnuk, K.; Dang, F.; and Dodds, Z. 2004. Dense 3d mapping with monocular vision. In *Proceedings of the 2nd International Conference on Autonomous Robots and Agents (ICARA '04)*.

Yau Lam Yiu. 2004. *http://www.cs.ust.hk/~yiu*.