KSU Willie in Scavenger Hunt at AAAI '06

Aaron Chavez, Michael Marlen, Chris Meyer,
Andrew King, Joseph Lutz, and Dr David Gustafson
Computing and Information Sciences
Kansas State University
Manhattan, KS 66506

Introduction

The Kansas State University entry into the AAAI 2006 robot scavenger hunt consists of a Pioneer P3AT robot (see figure 1) running Windows 2000, scalable client/server software architecture, blob-based object recognition, and a path-planning package coupled with an off-the-shelf Monte-Carlo package that we have augmented.

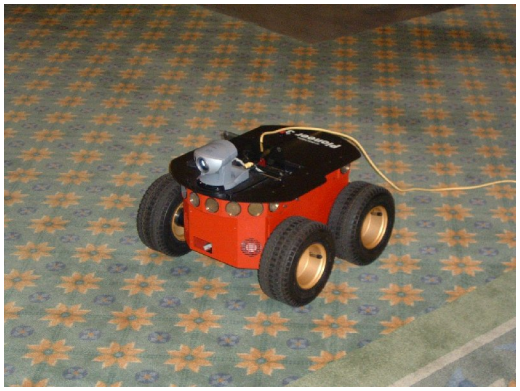

Figure 1: KSU's P3AT robot

Client / Server Architecture

For flexibility in development, we built a client / server model (see figure 2) to abstract the ARIA (ActivMedia) API and allow for distributed computation. The server process(es) run on the robot(s). This allows for clients to be written that can control multiple robots. The server keeps track of robot state; such as the velocity of the drive motors, the data being returned from the sonar sensors, the pan/tilt angle of the camera, and the charge of the internal battery. As the state of the robot changes (either from default behavior coded into the server, or by other clients requesting a state change) the server generates a series of messages which are broadcast to all connected clients to notify them of the state change. All state change requests from clients are served in order of the time they were received.

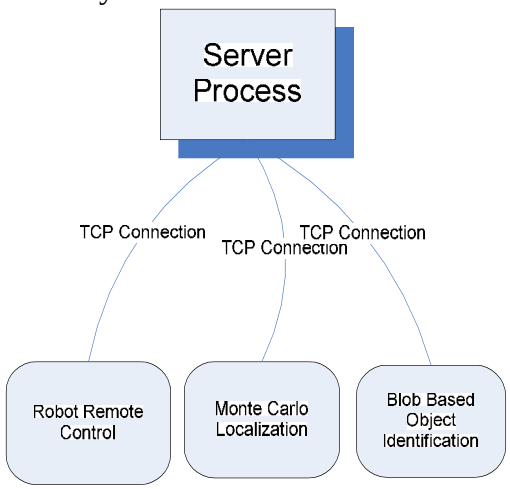

Figure 2: Server process with remote clients performing various robot related computation

A client is simply any process that connects to the server. For the scavenger hunt we developed two clients: one to provide robot remote control and visual object identification, the other to perform path planning and Monte-Carlo localization. Clients can

send requests to the server for information that the broadcast messages

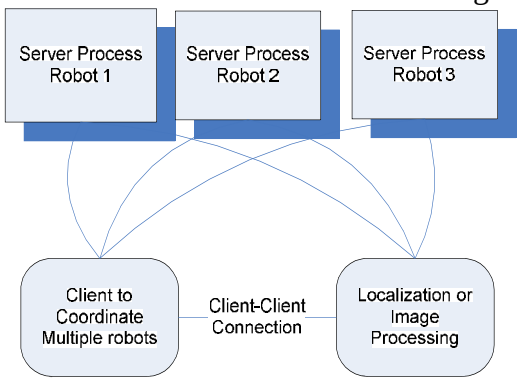

Figure 3: An example of a multi robot architecture using our framework

don't provide or to alter the running state of the robot. Clients can ask for the current image from the camera, request the motors to change velocity, request the camera to tilt and request the camera to pan. Clients have to be written to play nice with other clients, because the server has no facility to resolve any contention between requests (i.e. two clients keep requesting different drive motor velocities.) See figure 3.

Blob-Based Image Recognition

Because the colors of the objects in the scavenger hunt (lots of fluorescent colors) were not likely to be found in the competition environment, we developed a simple blob-based object recognition system (see figure 4). Although we started using the ACTS software, we eventually switched and integrated the CORAL Group's Color Machine Vision Project blob software (CMVision) into the server process at the AAAI '05 Conference. This year we developed our own blobbing software tuned to meet the requirements of our own image recognition algorithms (such as storing silhouette maps). This blobbing software works similarly to CMVision in that it works in the YUV color space, but the algorithm that is used to build the blobs is different and caches explicit blob silhouettes (Boolean maps of blob regions). Our blobbing facility is implemented in C# and managed C++.

The image recognition client has three purposes: 1) Allows the human operator to calibrate the blobbing system with a YUV color space interface to isolate the color regions that should be blobbed 2) Training: the human operator can present the robot with an example of an object and the software keeps track of the blob and silhouette statistics associated with that image 3) Real time object recognition: visually shows where in the image an object has been detected in real time.

Calibration allows the operator to choose what part of the YUV color space is to be associated with a channel. For instance, an operator may wish to assign channel 1 to blob colors that are orange with low luminosity (such as the orange cone) then the operator could take another channel to associate with orange of a different luminosity (the orange of the stuffed dinosaur seems to have a higher luminosity component than the cone.) The operator, of course, can calibrate other channels to blob other colors that are present in the objects being searched for.

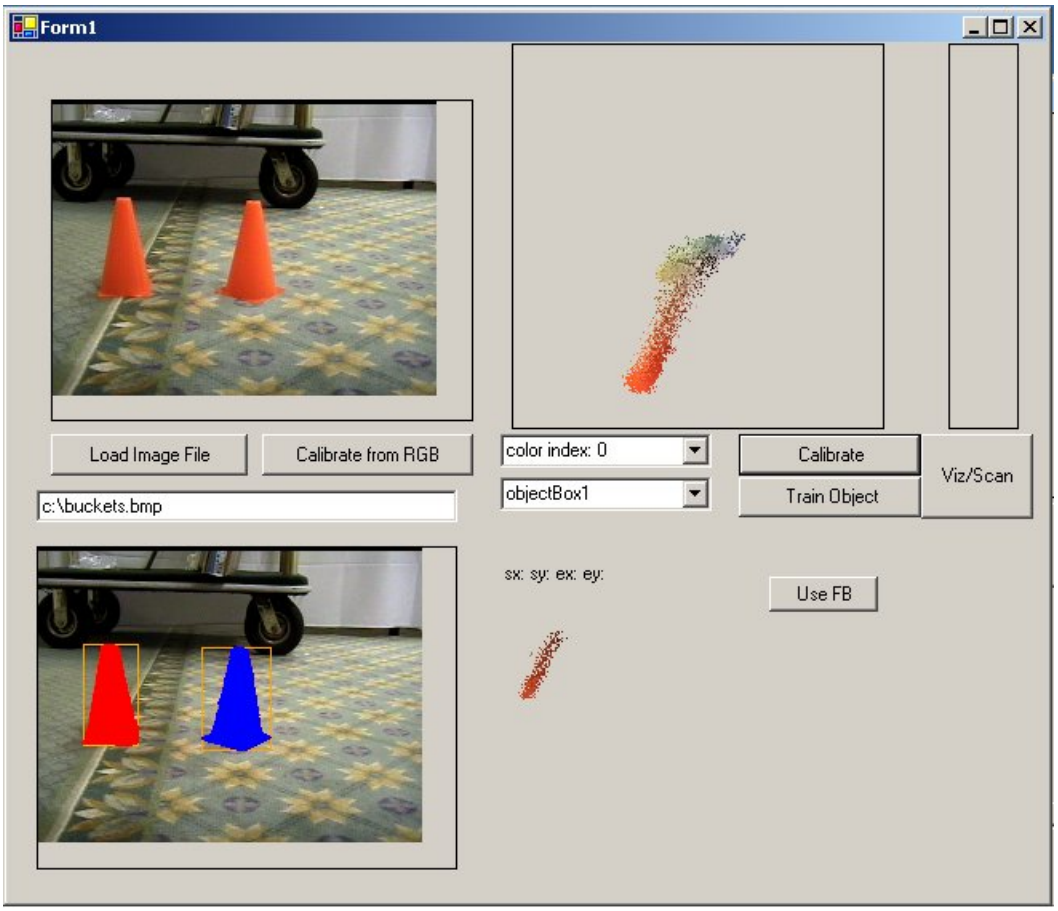

Figure 4: This is the object recognition interface processing a video feed in real time. It has been trained to recognize cones. The blob silhouettes and object regions are visualized in the lower right section of the interface. The UV projection of the image is in the upper right.

Object training is a process that takes place after the blobbing system's channels have been calibrated to the appropriate colors. The training system allows the operator to present the robot with a good example of the object. The blob and silhouette statistics from the example image are recorded and associated with the object label that the operator provides. In essence, the operator would place a cone in front of the robot, type in the name "cone" and click the "train" button. That training data and its associated label is then stored in a list of known objects.

Object recognition itself is very simple: Our blobbing software is set up so that it has multiple channels that it is performing blobbing on. Each channel is calibrated to a different region of the YUV color space that could be relevant to the scavenger hunt objects. Each time new blob information is sent to the client, the client compares the blob statistics and relative locations of the blobs in the image against learned statistics for the object type.

If the statistics are similar then the object is considered to be found. The centroid of the object is computed by averaging the centroids of all the blobs that compose it, and the region of

the object in the image is computed by bounding an area with the maximum X and Y edges of all the blobs in the RGB version of the image. In addition, if there is silhouette statistics associated with the object it thinks it has found, the silhouette data is used as a sanity check before declaring that an object is "found." This allows the robot to distinguish between two different object types that are of the same or similar color.

Localization and Path Planning

For basic localization and local path planning, we use MobileRobotic's SONARL package. Due to the unreliable nature of the sonar sensors used in conjunction with basic Monte-Carlo localization, we added visual landmark recognition to the robot's localization tool box. If we have an environment that is problematic for sonar but has visual landmarks (such as unique signs or objects that are stationary) we can train the robot to see those landmarks and associate a certain robot pose with visually seeing those landmarks. In addition, if the robot identifies an object that is not already a landmark it will generate a landmark on the fly and place it on the correct location in the human generated map of its environment (see figure 5).

SONARL is used to do local path planning (such as navigating around dynamic objects detected by the sonar.) We also built long term path planning software on top of SONARL. Our software uses a "Zamboni" sweep pattern to navigate across a search grid. If grid locations are unreachable, due to either SONARL being unable to plan a path to the grid location or because there is an obstruction detected visually, the "Zamboni" algorithm looks for another grid location to search. If the robot had to skip a grid location because it was previously obstructed from it will try to search it again after it has traveled to other locations. If the location is still obstructed it will end its search, noting which grid locations where obstructed.

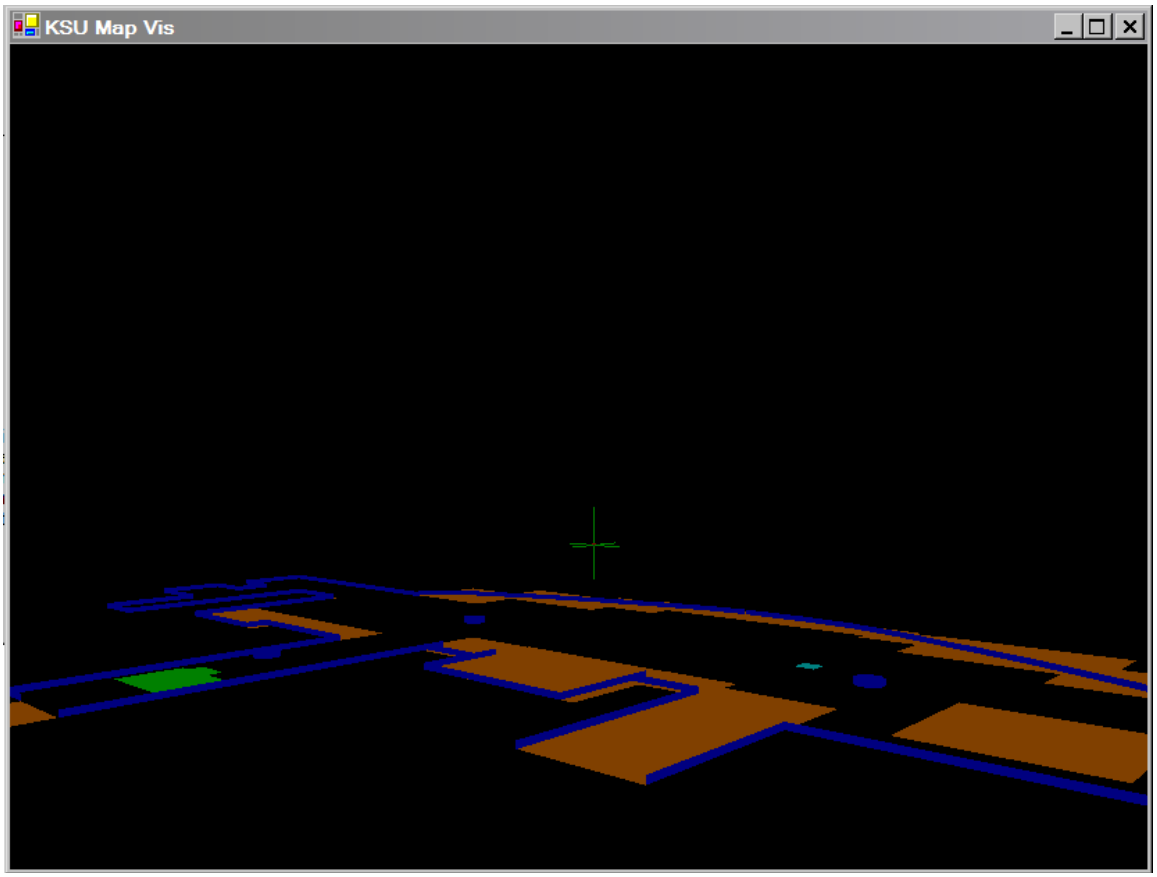

Figure 5: A birds eye view of the competition area from the AAAI '06 scavenger hunt. As the robot encounters hunt items it renders an icon at the object's perceived location, as well as the robot's current location in the map.

Summary/Conclusion

At AAAI '06 our robot performed as we (see figure 6) wanted it to. The visual recognition was able to identify and discern the objects it was supposed to and the localization and path planning worked fairly well given the number of dynamic objects (people) in the environment.

We did have some issues with our path planning building a path around visually detected obstructions too closely that caused our robot to clip some objects and drag those objects under its wheels.

Figure 6: the KSU robotics team

The biggest issue we encountered during the event was the amount of computational load we were putting on the robot and the client computers. Even

with a 1.86 Ghz dual core laptop doing the vision processing we were having trouble maintaining an acceptable frame rate (>10FPS) with all of our processing features active. We feel that a significant part of this problem has to do with allocating new heap regions for each pass, and then deleting those heap regions after each pass. We also could have multi-threaded the processing framework in a way to take better advantage of the dual core setup.