Lewis the Graduate Student: An Entry in the AAAI Robot Challenge

William D. Smart Michael Dixon Nik Melchior
Joseph Tucek Ashwin Srinivas

Media and Machines Laboratory
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130
United States
{wds,msd2,nam1,jat2,avs1}@cse.wustl.edu

Abstract

In this paper, we describe Lewis the Graduate Student, Washington University's entry in the AAAI Mobile Robot Competition Challenge Event. Lewis successfully completed a modified subset of the Challenge tasks, with a minimum of human intervention. We describe the architecture of our system, and how each sub-task was achieved. We also offer some thoughts on the performance of the system, and highlight our plans for future work.

Introduction

This paper describes Washington University's entry in the Robot Challenge event of the AAAI mobile Robot Competition, held at IJCAI 2003, in Acapulco, Mexico. The entry was fielded by the Media and Machines Laboratory, in the Department of Computer Science and Engineering. The main programming team consisted of four undergraduate students and one faculty member.

Our lab robot, Lewis, was not brought to Mexico for the competition due to the cost of shipping. Instead, the hard drive, and various sub-systems were removed and transplanted into another robot, called Pinky, of the same type in Acapulco. Surprisingly, this did not substantially affect the performance of the software, once a few minor differences were accounted for. Leaving aside any philosophical questions of self, the robot will be referred to as Lewis in this paper.

Lewis is a standard iRobot B21r mobile robot platform, with a Directed Perception pan/tilt unit, two color cameras, and SICK laser range-finder.

We begin by outlining the Robot Challenge task. We then discuss our overall software architecture, and then go into greater depth for each of the component parts. Because of the strict modularity of design, each of the individual sub-tasks corresponds to one or more stand-alone programs.

The Task

The full Robot Challenge task can be summarized by a sequence of sub-tasks.

1. Start the robot at the door to the convention center.
2. Autonomously locate the registration desk, asking for help when appropriate.
3. Stand in line at the registration desk. Upon getting to the front of the line, interact with the official at the desk. Get registration materials (including a map of the conference center), and find out the talk location.
4. Plan a path to the talk location, and navigate there. Arrive in time to give the talk.
5. If there is time on the way, interact with conference-goers. Accept tasks from designated conference officials (such as delivering a message to a specified room).
6. Give a talk, and answer questions.

We chose to concentrate on a modified subset of these tasks because of our limited manpower. Specifically, we attempted the following:

1. Start the robot at the door of the conference center.
2. Autonomously location the registration desk using pre-positioned directional signs.
3. Identify the registration desk, and join the end of the line. Move towards the head of the line. Once at the desk, ask for registration materials, and allow the conference official to designate a talk location on a touch-panel displaying a pre-learned map of the conference center.
4. Plan a path to the talk location, and navigate to it directly.
5. Give a pre-scripted talk, and answer questions.

It should be noted that our robot-human interaction was extremely limited, and consisted of touch-screen interactions. Also, it could be argued that the amount of "real" artificial intelligence in the system was quite low.

This was our first attempt at the challenge event, and we made the conscious decision to engineer a system that would address the specific technical challenges imposed by the rules. While this is not really in the spirit of the event (which was designed to showcase AI research), we believe that it gives us an excellent platform on which to build in the future. Now that we can handle the basic requirements of the event, we are in a good position to add in subsystems based on the ongoing research projects in our lab.

The Software Framework

We used a simple finite state machine (FSM) based control structure. Each state in the machine roughly corresponds to a subtask, and is performed by a single stand-alone program. The amount of explicit interaction between these programs is extremely limited. For the challenge event, only two programs directly interact by passing two floating point numbers.

The programs have well-define pre- and post-conditions, which allows each part of the overall solution to be developed independently. These conditions are explicitly stated, but not specified algorithmically. For example, the `line-wait` state, used for standing in line at the registration desk, has the precondition of being at the end of the line, pointed at the desk. The post-condition is that the robot is at the head of the line, pointed at the desk.

By specifying the pre- and post-conditions in this manner, we were able to minimize the amount of integration work that had to be done after arriving at the conference. Although we had to alter internal elements of almost all of the behaviors on-site, this did not affect the overall system much, because of the limited interactions between programs.

Sequencing the Programs

The structure of the FSM was specified using a simple description language, illustrated in figure 1. The description specifies the starting state, the optional recovery state (see below), and whether or not a graphical control interface should be used. Each state has a symbolic name, an associated program (possibly with arguments), and a list of transitions. Transitions are made based on the value returned by the program when it terminates.

The individual programs are sequenced by an executive that monitors for abnormal termination, and allows a human supervisor to interrupt the normal flow of control. If a recovery state is specified, the FSM transitions to this state on a human interrupt or abnormal program termination. If no such state is specified, the sequencer terminates and closes all active control programs.

Graceful Recovery

One of our primary goals was to have a graceful recovery strategy, in case one of the control programs suffered a segmentation fault, or if the robot was about to crash into something. The sequencing code keeps watch over the control programs looking for the former problem, while a human-activated on-screen "panic button" takes care of the latter.

```
start follow-sign
recovery follow-hat
use gui

state follow-sign gotosign
  action 0 trans follow-sign
  action 1 trans door-wait
  action 2 trans line-wait
  action 3 trans seek-sign

state seek-sign seeksign
  action 0 trans follow-sign
  action 1 trans seek-sign

state door-wait doorwait
  action 0 trans follow-sign

state line-wait QueueStander
  action 0 trans register

state register get_goal smrt/maps/mexico
  action 0 trans localize

state localize smrt_start_loc
  action 0 trans goto-talk

state follow-hat followhat
  action 0 trans stop
```

Figure 1: A portion of the task sequence specification for the challenge event in Acapulco.

If something goes wrong, the system enters a special recovery state. the default recovery state gracefully stops the robot in its tracks. However, for the challenge event another, more intelligent strategy was implemented.

On entering the recovery state the robot looks for, and tries to follow, a (specific) red baseball cap. If no such cap is seen, the robot does not move and waits for ten seconds. If no cap appears within ten seconds, the recovery state terminates.

If the robot does see a red baseball cap (using the color-finding techniques outlined below), it attempts to follow the wearer of the cap. By fusing the information from the camera and the laser range-finder, the robot is able to robustly follow the wearer, keeping a safe distance. We use this technique to guide the robot to a safe place, often the start position for some other state in the FSM. Once the robot is in the correct position, the hat-wearer moves quickly out of the robot's field of view. This is not difficult, since the hat-tracking software is designed only to follow slow-moving hats. After ten seconds of not seeing a hat, the recovery state terminates.

On termination of the recovery state, the sequencer launches a graphical interface that allows the human supervisor to pick the next state of the robot. Once the state has been picked and verified, the robot enters that state, and continues operating according to the FSM.

Following Signs

To reach the registration desk without prior knowledge of its location, we chose to follow signs. To make this task more feasible, we provided our own signs designed to be easily detected and positioned such that one sign points roughly in the direction of the next, leading from the starting location to the registration desk. This turns the task of finding the registration desk into the much easier task of detecting arrowed signs, measuring their arrow directions, and following the vectors defined by this.

All movement is subject to low-level obstacle avoidance, implemented by a greedy navigation algorithm. For a given final destination point, we select the point nearest our destination that can be reached with a straight line path. To calculate this point, we first compute the horizon of ”reachable” points. This is done by performing a closing operation on the 180 degrees of laser readings. We take the robot's radius plus a safety tolerance and calculate how far along each of the 180 headings the robot would be able to travel without intersecting any of the laser endpoints. The closest reachable point will be either on this horizon or at the intersection point of a laser heading and the line perpendicular to the laser heading which intersects the goal point. We compute these points and select the one nearest the goal as our intermediate destination. We continually recompute and approach this intermediate point until either the intermediate point or the final destination is reached. If the intermediate point is reached, the obstacle avoider signals its failure to find a path connecting the start point and the destination, otherwise success is signaled.

Finding Signs

We use a color-based approach for our sign detection. All signs are to be constructed of a known outer color (in this case magenta) with an arrow of a known inner color (black). To detect arrowed signs we detect blobs of the outer and inner colors. Since the specific colors are known in advance, this is a very straightforward process, that classifies pixels according to their values in the YUV color space. Once we have detected all the color blobs, we find the set of all outer-colored blobs which contain exactly one inner-colored blob, with appropriate outer-inner size ratios. Using two nested blobs eliminates many of the problems with false positives typical with color blob detection, but it is limited based on the size of the center blob. From a distance, the center blob may not be visible, so in the case where no nested blob signs are detected, the largest outer-colored blob is used.

Following Signs

To follow signs we simply approach the most prominent sign, and upon reaching the sign orient the robot in the direction of the sign's arrow. To do this, we must be able to measure the relative position of a sign, the orientation, and the arrow direction. Given the measurements of the sign and the intrinsic parameters of the camera, one can calculate the position of the sign in relation to the camera. We make the assumption that the sign will be hung at roughly camera level, and is more-or-less perpendicular to the floor plane, so finding the plane in which the sign lies is straightforward.

To measure the arrow direction, we look for the presence of diagonal edges in the inner blob. The only strong diagonal edges within this region will be the head of the arrow, so by determining which half of the box has more diagonal edges, we can determine which way the arrow points. Although not as robust to errors as other techniques, such as template matching, this method for arrow-direction measurement was simple to implement and was independent of the specific arrow used on the sign. This allowed us to test multiple sign configurations without modifying the code.

Once the robot finds a sign, it orients itself parallel with the sign, facing the direction in which the arrow is pointing. The robot the begins searching for the next sign. If no sign is found, the robot simply drives forwards, and begins searching for the next sign by panning the camera back and forth.

Dealing with Doors

We chose to use an upward pointing arrow to indicate a closed door. When an upward arrow is encountered, the robot approaches and enters the door-waiting process. This process begins by playing a please-open-door message. It then waits for laser readings to indicate a space wide enough to move through to open in front of the robot. As this occurs, the robot will pass through the door. Upon crossing the threshold, the robot will play a thank-you message and re-enter the sign following process.

Finding the Registration Desk

We used a downward pointing arrow to indicate the registration desk. Upon detecting a downward pointing sign, the sign following process ends, signaling for the line following process to begin.

Registering for the Conference

Once the robot found the registration desk, the next task was to stand in line and receive its registration materials. This breaks down into two sub-tasks, standing in line and actually registering. These sub-tasks are dealt with in this section.

Standing in Line

The precondition for this task is that the robot is facing the registration sign, is roughly pointing in a direction perpendicular to the registration desk, and is beyond the end of the line. Previously, we had developed code that would actively find the line, and place the robot at the end. However, the particular arrangement of the environment in Acapulco made this code redundant and it was removed. The sign-following sub-system reliably placed the robot in the correct position for the start of this sub-task.

Standing in line is fairly simple. The robot moves forwards until it is within 80cm of the closest object in front (assumed to be a person). When this object moves forwards, the robot follows. Some hysteresis was built into the system, to ensure that small movements by the human in front of the robot did not trigger short, jerky movements by the robot. This procedure is repeated until the robot is at the registration desk.

To recognize when it is at the registration desk, the robot constantly performs a Hough transform on the laser contact points over the front 60 degrees. If more that 80% of the points line close to the line predicted by the Hough transform, this means that there are no objects between the robot and the desk. The value of 60 degrees and threshold of 80% were empirically found to give good performance. When the robot is closer than 80cm from the desk, and there are no objects between it and the desk, the line-standing module terminates, and control passes to the registration interaction module.

Registration Interaction

The interaction at the registration desk consists of a GTK+ graphical user interface (GUI) and some associated auditory prompts. The GUI displays a map of the convention hall (constructed previously) and a selection box for listing maps of other areas of interest to the robot. Instructions for selecting the talk location are printed beside the map, and an abbreviated form is repeated out loud (using the Festival speech synthesis system) to the judge. The GUI is meant to be displayed on a small touch-screen, which would allow the judge to simply tap the location of the talk. However, due to problems connecting the touch-screen to the robot's power supply in Acapulco, the GUI was displayed on the laptop sitting on top of the robot.

Our interface allows the judge to change his or her selection any number of times, prompting them to press "OK" when satisfied with his choice. At this point, the robot thanks the judge, and the registration interface terminates. The output from this module is too complicated to return to the FSM with a simple exit code. Instead, the (x, y) coordinates selected by the judge are written to a temporary file which is read by the next module, the navigator. This is the only explicit inter-module communication used by the system for this event.

Getting to the Talk Location

Up to this point, the robot had no sense of its location within the convention hall. It had navigated from sign to sign, following the direction of each successive arrow, and conducting a visual search when it failed to detect a sign. After interacting with the judge at the registration desk, the FSM started a localization process which continued to run for the remainder of the challenge. Since we knew the location of the registration desk, the localizer was given a hint as to the robot's starting location. In addition, the entire conference hall was contained within one map. Since the challenge had been simplified such that the robot was never required to leave this room, the localizer was never required to change maps, or reinitialize its position estimate.

As the robot moves, the estimate of its location on the map is updated by Monte Carlo localization, based heavily on the Carmen software from Carnegie Mellon University. Roughly 6000 particles were used to form the estimate, and they were contained within a Gaussian centered at the last-estimated position. Although we considered allowing a small portion of the particles to be uniformly distributed throughout the map (to handle the kidnaped robot problem), this feature was turned off during the challenge. We expected the robot to be surrounded by a dense crowd of spectators during the entire run, and this expectation was proven correct. The Monte Carlo localization algorithm weights particles based on the correspondence between actual laser range-finder data and the data that would be generated by a robot located at each particle. Since the crowd would make the laser range-finder all but useless, we were afraid that random particles would be weighted unreasonably high (*e.g.* particles in corners of the room), and the estimated location could jump about the room. Instead, the other component of Monte Carlo localization was relied upon more heavily: as the robot moves, odometry data is used to update the location of each particle. Odometry can be trusted for short distances, and particle weighting would still have a significant effect if a unique environment feature (such as the small columns the filled the venue) was detected. Unfortunately, no record of the localizer's performance was kept during the challenge. However, given our navigation strategy (described below in section), the direct path taken by the robot is indicative of satisfactory performance.

Planning a Path

Although the robot is aware of its location and the location of its goal, navigating between these points is not trivial. While the obstacle avoider described above in section is useful for simple movement commands, it cannot be trusted for navigating any significant distance. It can easily become trapped in "box canyon" obstacles, and it will give up if faced with a large (or persistent) obstacle. Instead, we constructed a roadmap off-line using the map of the conference hall, and the obstacle avoider was used as a local planner between nodes (or way-points) of the roadmap.

The nodes of the roadmap were chosen from the map using a number of heuristics. Since we knew that our local planner works well in open areas, very few nodes were placed in empty areas of the map. Approaching obstacles, though, the nodes become much more dense. Finally, no nodes were placed within a robot's radius of any obstacle. To form the roadmap, each node was connected to a small number of its nearest neighbors to create an undirected graph.

Navigating to the Talk

We use Dijkstra's shortest path algorithm to choose a series of nodes between the given starting point (the location estimated by the localizer) and the goal point (the location of the talk specified by the judge). The chosen nodes, optimized for traveling distance, become the way-points for our navigator. Although optimizations are apparent, the navigator simply attempts to reach each way-point in turn using the obstacle avoider as a local planner. At each way-point, the navigator queries the localizer for a current estimate. It uses this estimate to give the obstacle avoider the relative location of the next way-point. If the obstacle avoider were to fail to reach any way-point, the navigator will retry a number of times before simply planning a new route from the current location to the remembered goal.

The Presentation

The final task of the challenge is for the robot to give a talk about itself. We did not consider this a core part of our system (this year), and our solution is somewhat straightforward. The talk-giving module relies on a human helper who advances the slideshow, and who signals the completion of a question from the audience with a key-press.

Giving the Talk

The robot is equipped with a soundcard and uses the Festival text-to-speech system to generate speech. A simple piece of sequencing code reads the text for each slide from a file, then asks the human helper to advance to the next slide. Appropriate pauses are inserted at the start and end of each slide. However, all of this is done without reference to the outside world. The robot does not check to make sure that the slide has advanced, or that there is even a slideshow running.

Answering Questions

Once the robot has finished reading from the slideshow script, it asks if there are any questions from the audience. The robot has no speech input, and only one answer to questions, triggered by a key-press from the human helper:

> "That's a good question. The answer involves very complicated mathematics, but I'd be happy to talk about it off-line, perhaps over coffee."

Conclusions

Performance in Acapulco

The system performed extremely well in Acapulco, far exceeding our expectations. Our main failure mode was the misidentification of two signs. The orientation of one directional sign was wrongly calculated (but not by much). The registration sign was confused with a similar-colored sign on the wall two meters behind it. Both of these incidents required some minimal intervention. Other than those two problems, however, the system operated flawlessly.

Lessons Learned

The simple modular framework was a great asset, since it allowed us to minimize integration problems. Each of the modules worked well individually, and only a few minor problems had to be ironed out when they were sequenced together.

The main feature lacking from the robot in Acapulco was some form of monitoring and reporting system. Being able to log sensor data and decisions made by the processes would have greatly accelerated the debugging process, and enabled us to carry out a better post-mortem performance analysis. Having systems that would also display (some subset of) this information on demand during the run would also have been helpful.

The main lessons learned were ones we knew already: integration is hard, and environment changes can ruin your day. We were able to limit the former, but some of the latter were beyond our control. Highly variable lighting conditions in the conference center proved to be less of a problem than we had first feared. However, we are indebted to the local lighting crew who crawled into the ceiling to string a spotlight that illuminated our "this is a door" sign. Without them, we would probably have had to pay for a new glass door.

The Future

Although the current control framework is adequate for the task in Acapulco, we have plans to improve on it. In particular, programs should have a way of passing information back upon termination that is better than a single return code. We are investigating a

We are also keenly aware that our human interaction is seriously lacking. We are investigating a simple gestural interface to allow the robot to take directions from a human. We are also carrying out human-robot interaction studies to determine the effects of subtle cues (such as gaze-direction, and the notion of "personal space) on humans' responses to the robot.

Our main goal, however, is to build on the framework that we have developed this year, and insert more "science" into the system. The modular nature of the design makes it a perfect vehicle for testing the results from the various robot- and vision-related projects currently underway in our lab. This will bring us closer to the spirit of the rules when we enter Lewis in the 2004 Challenge event in San Jose.

Acknowledgments

The work described in this paper was partly funded by NSF under award #0196213 and REU award #0139576, and partly the Washington University Sesquicentennial fund. We would like to thank Mark Bober, Rob Glaubius, Ben LeGrand, Andy Martignoni, and Jim Tucek for their help with testing, tweaking, and fixing various parts of the system. We would also like to thank iRobot Corporation for generously allowing us to borrow one of their robots in Acapulco.