From: AAAI Technical Report WS-02-18. Compilation copyright © 2002, AAAI (www.aaai.org). All rights reserved.

The Personal Rover

Emily Falcone *etf@andrew.cmu.edu*
Rachel Gockley *rgockley@andrew.cmu.edu*
Eric Porter *eporter@andrew.cmu.edu*
Illah Nourbakhsh *illah@ri.cmu.edu*

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper, we summarize a new approach for the dissemination of robotics technologies. In a manner analogous to the personal computer movement of the early 1980's, we propose that a productive niche for robotic technologies is as a creative outlet for human expression and discovery. This paper describes our ongoing efforts to design, prototype and test a low-cost, highly competent personal rover for the domestic environment.

Introduction

As with most leading technological fields, robotics research is frequently focused upon the creation of technology, not on creating compelling applications. Although the search for new technologies is a valid scientific process, one important aspect of robotics is not appropriately explored in this manner: human-robot interaction.

Robotics occupies a special place in the arena of interactive technologies because it combines sophisticated computation with rich sensory input in a physical embodiment that extends well beyond the desktop. Moreover robots can exhibit tangible and expressive behavior in the physical world.

In this regard, a central question that occupies our research group pertains to the social niche of robotic artifacts in the company of the robotically uninitiated public-at-large: *What is an appropriate first role for intelligent robot-human interaction in the daily human environment?* The time is ripe to address this question. Robotic technologies are now sufficiently mature to enable long-term, competent robot artifacts, at least in prototype form, to exist (Nourbakhsh et al. 1999, Thrun et al. 2000).

We propose that an appropriate first application for robotics within the human social domain is as a creative and expressive tool rather than a productive tool optimized for consumer use. Consider the history of the personal computer. In the early 1980's, advances in low-cost computer manufacturing enabled individuals to purchase and use computers at home without specialized knowledge of electrical or computer engineering. These early computers were tools that forged a new creative outlet for programmers of all ages. Before long, video games as well as more business-savvy applications were born from the tinkering of these initial computer hobbyists. In effect, the early adopters of the personal computer technology constituted a massively parallel effort to explore the space of possible computer programs and thus invent new human-computer interaction paradigms.

The goal of the Personal Rover project is analogous: to design and deploy a capable robot that can be deployed into the domestic environment and that will help forge a community of creative robot enthusiasts. Such a *personal rover* is highly configurable by the end user, who is creatively governing the behavior of the rover itself: a physical artifact with the same degree of programmability as the early personal computer combined with far richer and more palpable sensory and effectory capabilities.

Our goal is to produce a Personal Rover suitable for children and adults who are not specialists in mechanical engineering or electrical engineering. We hypothesize that the right robot will catalyze such a community of early adopters and will harness their inventive potential.

As in the toy industry, the first step toward designing a Personal Rover for the domestic niche is to conduct a User Experience Design study. The challenge in the case of the Personal Rover is to ensure that there will exist viable user experience trajectories in which the robot becomes a member of the household rather than a forgotten toy relegated to the closet. Contracting with Emergent Design, Inc., we produced an internal experience design document that describes the interaction of a fictional child, Jenna, with her Rover over the course of several months.

The user experience design results fed several key constraints into the Rover design process: the robot must have visual perceptual competence both so that navigation is simple and so that it can act as a videographer in the home; the rover must have the locomotory means to travel not only throughout the inside of a home but also to traverse steps to go outside so that it may explore the back yard, for example; finally, the interaction software must enable the non-roboticist to shape and schedule the activities of the rover over minutes, hours, days and weeks.

In Sections 2-4, this paper describes the ways in which we are working to satisfy these constraints. In addition, Section 5 describes the Personal Rover's performance at the recent 2002 AAAI Conference.

Rover Mechanics and Control

Rover Hardware

The rover's physical dimensions measure about 18"x12"x24" (length, width, height). A CMUcam vision system is mounted on top so that it can pan and tilt (Fig. 1). Its ability to track colorful objects permits the rover to easily navigate a room using vision. On either side of the rover, there are forward-facing infrared range finders. The most unique feature of the rover is a movable center of mass mechanism that allows it to actively shift its weight forwards and backwards (Fig. 2). Because of elevated Omni-Wheels located behind the rear wheels, the rover can tip backwards and climb steps greatly exceeding its wheel diameter. The two sides of the rover are connected with a differential, which lets one wheel move over an obstacle while keeping the remaining three wheels on the ground. Two servo motors provide Ackerman steering and the ability to rotate in place by turning the two front wheels inward.

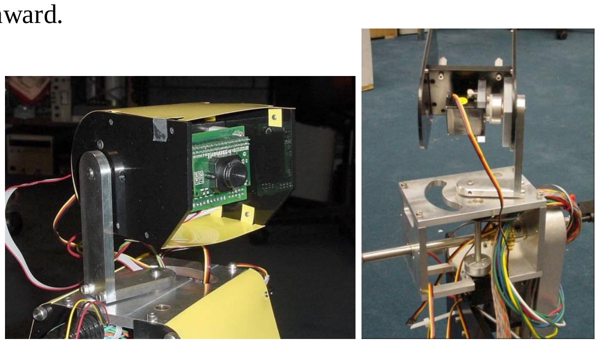

Figure 1: A CMUcam is mounted in the rover's head, where it can pan and tilt.

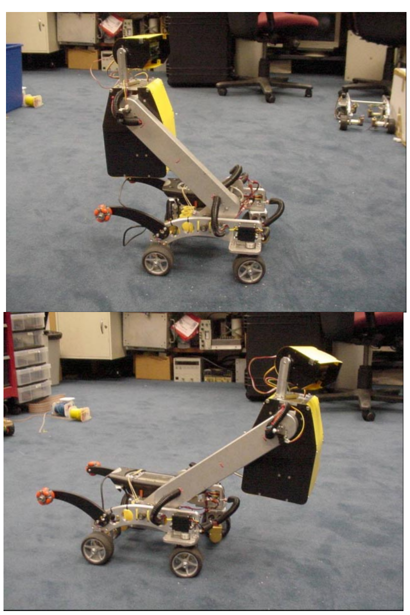

Figure 2: A moveable boom gives the rover a variable center of mass.

A Compaq iPAQ on the rover provides 802.11 networking, communicates with the CMUcam, and sends motion commands to the Cerebellum microcontroller. The role of the iPAQ is mainly as a wireless to serial bridge, but it has a tracking routine built in to control pan and tilt of the camera at 17 frames per second, which keeps the tracked object centered in the camera. The Cerebellum controls the servo motors, reads the range finders, and tells the four daughter boards (one for each wheel) a speed to maintain. Based on the directional encoders attached to the motors, the daughter boards use PID control to adjust the duty cycle. The daughter boards keep track of the encoder counts as a 16 bit unsigned value that wraps around when the value overflows or underflows. While 16 bits allows for values from 0 to 65535, the encoders overflow quite often because the encoders provide 30250 ticks per revolution. Because the wheel circumference is about 9.3 inches, this means that the encoder wraps around after a wheel has moved about 20 inches.

Rover Control

Command packets from the controlling computer to the rover can specify any combination of the following commands: a speed, a turn angle, a boom position, camera pan and tilt, plus commands supported by the camera. There are 36 possible turning angles: 17 to each side, straight ahead, and rotation in place. From the rover, the controlling computer receives a state array containing the velocity, encoder counts, and duty cycle from each of the wheels. In addition it is given the servo positions, the range from the infrareds, and the boom position.

Encoders. The controlling computer calculates the rover's position and angle by integrating the encoders from all four wheels. Because the turning radius is known, only one encoder is required for the calculations. With four encoders, the problem is over constrained. Having four wheels helps overcome errors and provides greater accuracy than just one encoder. When a user wants to update the encoders, they simply call a function that takes the state array as an argument. This function begins by finding the difference between the current encoder values and the previous ones and possibly correcting for wrap around. We check all four of the encoder differences to make sure that they fall within an acceptable range. If the encoder difference indicates that the wheel has spun in the wrong direction or spun in the correct direction but too quickly, that value is discarded.

In a vehicle with Ackerman steering, the axes of rotation for all four wheels cross at a single point. That point is always on the line that goes through the centers of both rear wheels (Fig. 3).

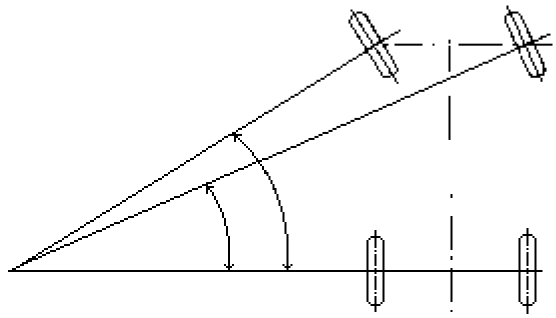

Figure 3: The axes of rotation for all four wheels cross at a single point.

For each pair of wheels, we calculate the ratio of the distance traveled by each wheel. We then use the ratio to calculate the radius of the turn. We average the results to identify the point about which the rover is turning. We can use the resulting turn angle and turning radius values to correct for real-world events such as wheel resistance or unexpected wheel angle changes over the course of the turn.

After we have the turning radius, we need to find out how far the rover has moved along the circle in order to determine its new location. We use the wheel that is farthest away from the center of rotation, which is also the wheel that has moved the greatest distance. Dividing the distance that wheel traveled by the turning radius gives us how many radians the rover has moved along its arc. Call this angle alpha. Calculating the final position of the rover requires three steps. First, we translate the coordinate frame to the point around which the rover is rotating. Second, we rotate the coordinate frame in place by alpha. Third, we do a translation in the opposite direction than that of the first step.

More simply, given r, the positive radius, and α, the angle in radians the rover has moved around the circle, we can calculate the new location of the rover with the following formulas:

$x_1 = r*[\cos(\theta_0 + \alpha - \pi/2) + \cos(\theta_0+\pi/2) + x_0]$

$y_1 = r*[\sin(\theta_0 + \alpha - \pi/2) + \sin(\theta_0+\pi/2) + y_0]$

$\theta_1 = \theta_0+ \alpha$.

GoTo and TurnTo. Two simple movement functions, GoTo and TurnTo use the encoders to move and turn accurately. While the rover is moving, a global x, y, and theta are continuously being updated. Since the movement commands run in a separate thread, they can be interrupted at any time. These commands are needed for more advanced motion control.

The GoTo(x, y) function moves to an arbitrary point by moving along the arc between the starting and ending points. The ending position is relative to where the rover is when the command is issued with the rover facing the positive x-axis and having the positive y-axis on its left. The rover moves along the arc of the circle having its center on the y-axis and containing points at (0, 0) and (x, y). Because there are only 36 possible turning angles, the rover starts off by turning the wheels to the position that will take it closest to the ending point. The function drives the rover towards the destination. After each command, the encoders are updated and the function recomputes the best turning angle. When the rover gets close to the goal, it slows down so as to minimize dead reckoning error while stopping. The TurnTo(theta) function rotates in place the specified number of degrees. It works just like GoTo, but does not need to recompute the turning angle between cycles.

Landmark-Relative motion. We can create advanced motion control functions by using the tracking routine on the iPAQ and the global coordinate frame. The function called “landmark lateral” moves the rover a specified distance towards a landmark, using the pan angle of the camera to keep the rover moving straight, and using the global coordinate frame to keep track of how far the rover has gone. The position of the landmark in the global coordinate frame is calculated by using the pan and tilt angles of the camera, along with the known height of the camera above the ground. Once we know the position of the landmark, we can use the GoTo function to provide landmark-relative motion. For example, the rover is able to stop one foot in front of the landmark or two feet to the left of it (Fig. 4).

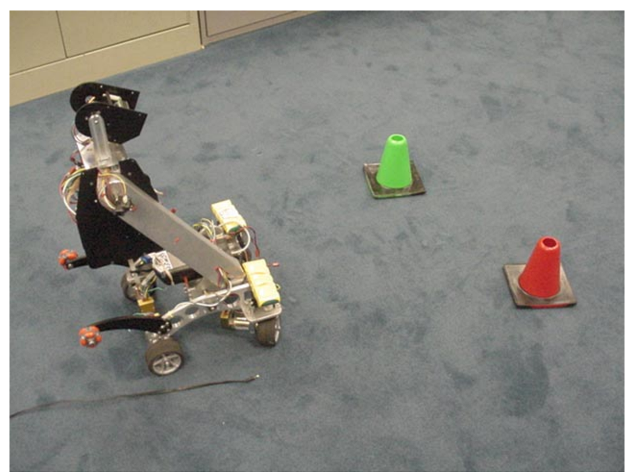

Figure 4: The rover uses landmarks to navigate.

Climbing. By moving its center of gravity, the rover can climb up stairs that a robot with a fixed center of gravity would not be able to. While the rover is climbing, the code controlling the rover detects the changes of state by reading the motor currents in the wheels (Fig. 6, 7). Because speed control is done on each wheel independently, we can get an idea for how much resistance each wheel is experiencing by reading how much power it takes to keep the wheel spinning at a constant velocity.

The omni-wheels can be moved to allow the rover to climb up stairs as high as 7 inches tall. In the proceeding paragraphs, I will be referring to the angle of the boom. For reference, 0 degrees is all the way back (near the omni-wheels) and 180 degrees is all the way forward. I will also be referring to the duty cycle, which in the units of the robot ranges from -1024 to 1023; the duty cycle is not represented as a percentage. While moving forward at a small speed, the duty cycle is about 650. The actual stair that the rover climbs in this example is about 6 inches high.

Climbing Up. The first step in stair climbing is moving the boom to 50 degrees and waiting until the rover hits the stair (Fig. 5a). We detect the rover hitting the stair by waiting for the duty cycle from one of the rear wheels to go above 950. Once both front wheels have hit the stair, the back wheels are moving forward with full power and the front wheels are actually applying force backwards to keep them from moving too fast. This is because when the back wheels move horizontally towards the stair one inch, the front wheels have to move vertically up the stair by 4 inches.

The second step is to shift the boom back, causing the rover to fall backwards onto the omni-wheels (Fig. 5b). When the duty cycles for both back wheels drop below 900, we know that the rover has fallen back on the omni-wheels and we start moving the boom forward to 145 degrees. With the rover moving at about half an inch per second, the boom has moved far enough forward to tip the rover forward when the rear wheels are about an inch from the step. The rover now has its center of gravity on top of the stair.

When the duty cycles from the front wheels goes above 800, we know that the back wheels have hit the stair because it takes more work to pull the back of the rover up (Fig. 5c). Once the back wheels hit the stair, things get tricky. If we leave the boom at 145 degrees, when the rover makes it up, its weight will be too far forward and it will fall on its face. If we immediately move the weight back, the rover won't be able to make it up because not enough weight will be on the front wheels. We solve this by waiting three seconds and then moving the boom to 125 degrees. This way the boom is moving backwards just as the rear wheels are coming over the top of the stair. We know that the rover has made it when the duty cycles from all wheels go back to about 650 (Fig. 5d). Figure 6 charts the duty cycles of the four wheels as the rover climbs a step.

Climbing Down. To climb down, the boom is moved to 120 degrees and the rover moves backwards until the back wheels drop off the ledge. When the rear wheels slide down the ledge, they start spinning faster than usual and the speed controller has to apply force in the opposite direction of motion. The boom is moved to 100 degrees and we wait until the front wheels drop off the ledge. Figure 7 charts the duty cycles of the four wheels as the rover descends a stair.

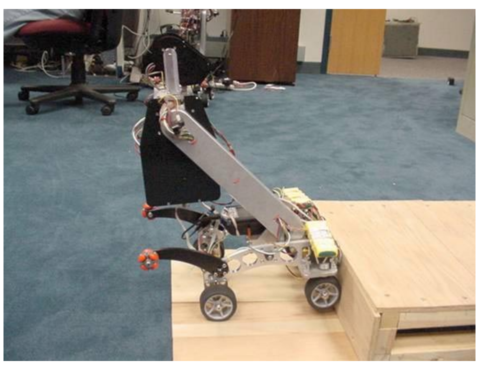

Figure 5a

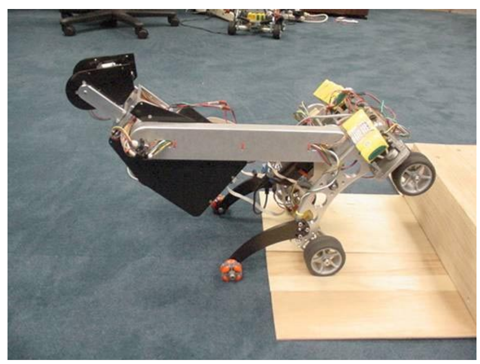

Figure 5b

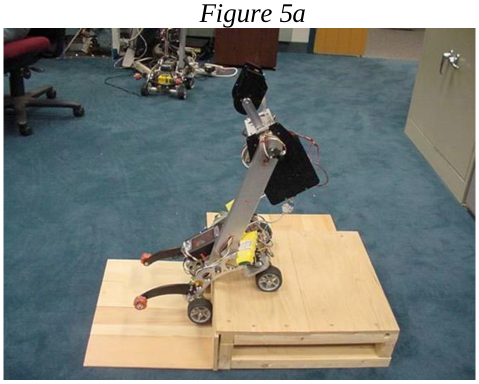

Figure 5c

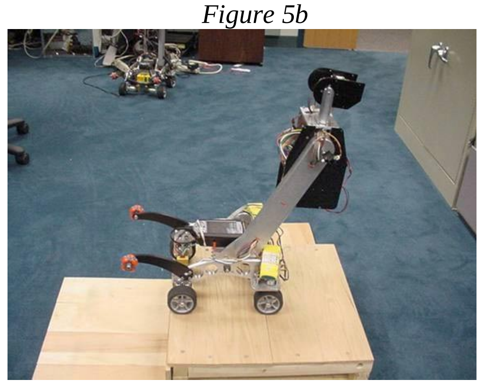

Figure 5d

Figure 5: Four different stages in climbing up a stair.

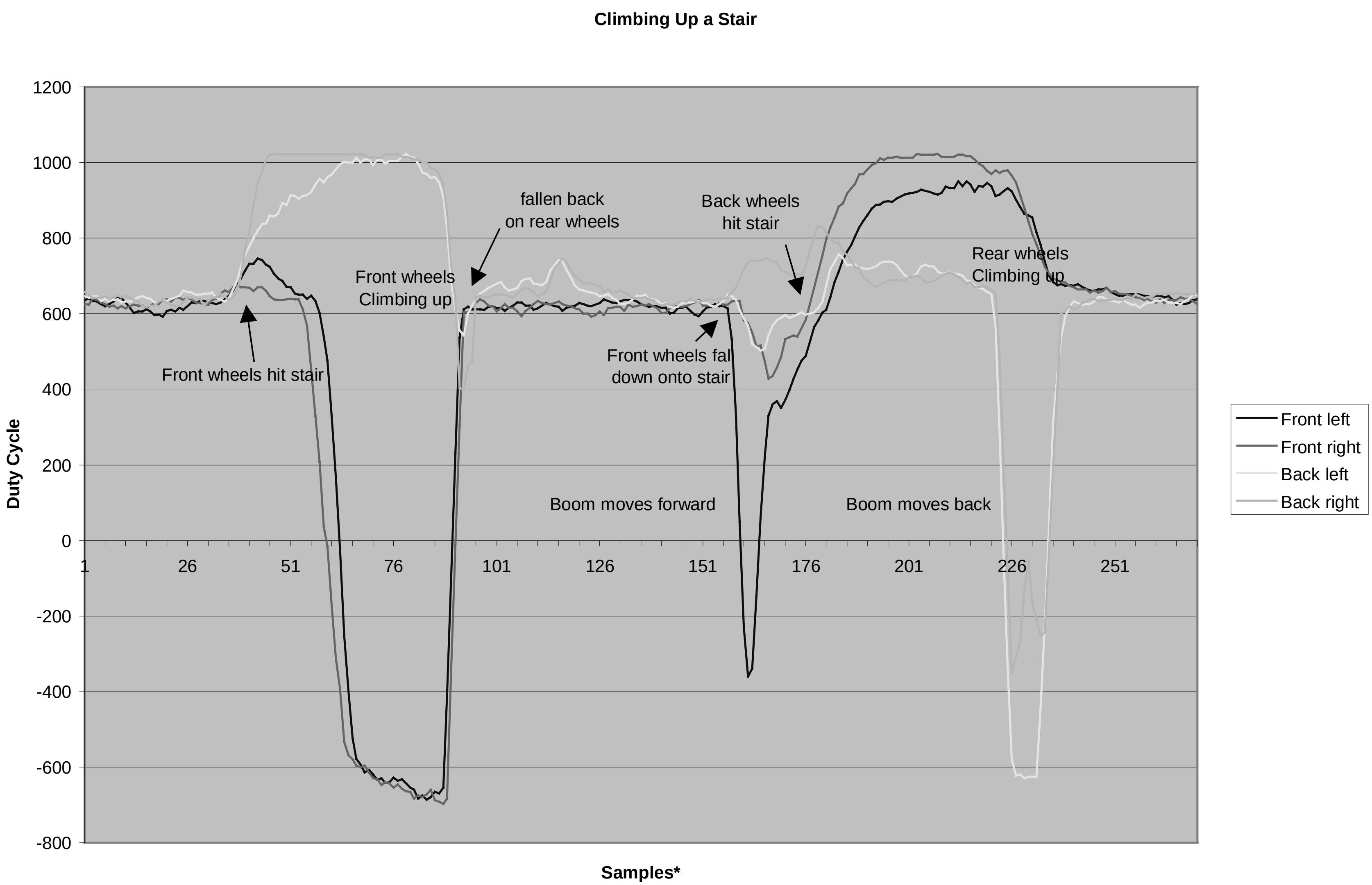

Figure 6: Back-EMF trajectories during stair climb

*There are approximately 200-300ms between samples.

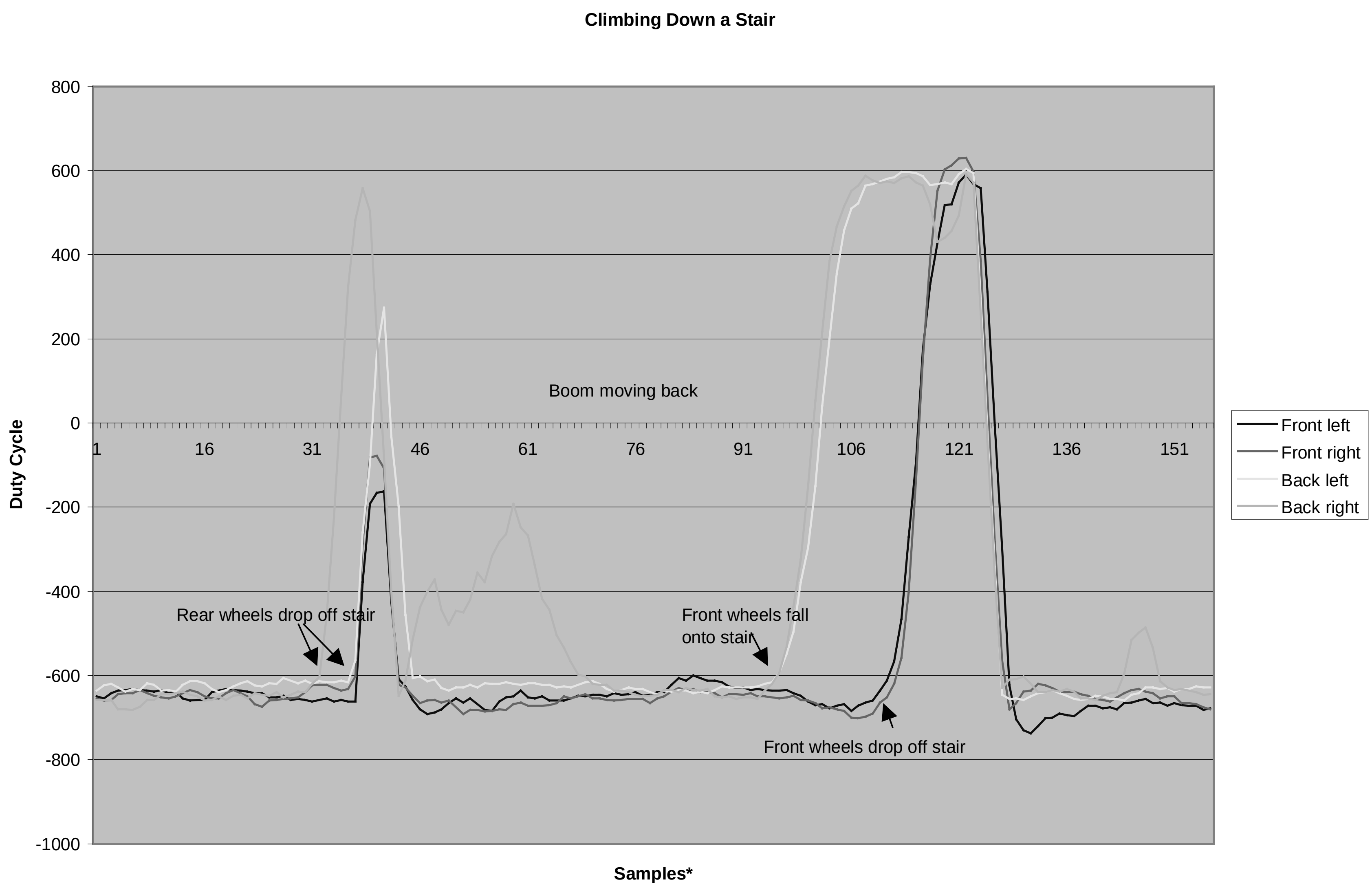

Figure 7: Back-EMF trajectories during stair descent

*There are approximately 200-300ms between samples.

Perception Based Teaching

A key aspect of this research addresses the question of how we can teach the rover to navigate an environment reliably, when the environment is as complicated and dynamic as a home. Our chosen approach involves the idea of landmarks, brightly colored objects that are purposely placed in static locations throughout the home. In this way, the rover can use its camera as a sensor, as well as its other perceptions (such as IR rangefinders), to successfully navigate its surroundings.

Goals

Our goals in developing a teaching environment for the rover include:

- The user environment must be highly intuitive.
- The language must be expressive enough to navigate a house.
- The navigational information must be stable to perturbations in the physical environment.

Implementation

Definitions. The basic data structures used to implement the teaching environment are Actions, LandmarkViews, Landmarks, Locations, and Paths.

Action: any basic task that the rover can perform. Actions include things such as pure dead-reckoning, driving to landmarks, turning in place, and checking for the presence of landmarks. Examples of Actions include:

- *ClimbAction*: climb up or down a stair
- *DriveToAction*: dead-reckon driving
- *DriveTowardMarkAction*: drive toward a landmark, stopping after a set distance
- *LookLandmarkAction*: check for the presence of a landmark
- *SendMessageAction*: send the user a message
- *StopAtMarkAction*: drive toward a landmark, stopping at a location relative to the landmark (e.g. two feet to the left, twelve inches in front, etc.)
- *TurnToAction*: turn a set number of degrees
- *TurnToMarkAction*: turn until facing a landmark

LandmarkView: what a landmark looks like; its "view." This can be thought of as a landmark "type," that is, it contains information about a landmark but not positional information. It keeps track of the camera track color parameters, a name for this type of landmark, and an image of the landmark.

Landmark: a landmark with positional information. A Landmark object contains a LandmarkView object as well as pan and tilt values for where the rover expects to see this landmark.

Location: a location is identified by a set of Landmarks and a unique name. A Location also stores the known paths leading away from that location. The rover neither independently determines where it is, nor compares stored images with what the camera currently sees. Rather, the user must initially tell the rover where it is, at which point it can verify whether it can see the landmarks associated with that location. If it cannot see these landmarks, then it can query the user for assistance.

Path: a series of Actions, used to get the rover from one Location to another. A Path executes linearly; one action is performed, and if it completes successfully, the next executes. Paths actually have a tree structure, so that they have the capability of having alternate Actions specified. Thus, for example, a Path from point A to point B might be "drive to the red landmark, but if for some reason you can't see the red landmark, drive to the green one and then turn ninety degrees."

User Interface. While the rover can dead-reckon with a high degree of accuracy, navigation robustness is achieved through the use of landmarks. Our teaching interface allows the user to specify a landmark by outlining a box around the desired landmark on the displayed camera frame, as is occurring in Figure 8. If the rover is able to track the landmark the user selected, it compares the new landmark to all the previously seen and named LandmarkViews. If no match is found, the rover asks the user whether she would like to save this new type of landmark. Saved landmarks can then be used offline in mission design.

To begin teaching the rover, the user must first specify the rover's current location. To do this, the user just needs to select one or more landmarks, so that the rover can identify the location in the future. The interface for teaching a location is shown in Figure 8. Note that on the right-hand side of the window is a small picture labeled "currently saved image." By default, this image is set to what the rover sees straight ahead from the location. However, the user can also choose to associate a different image with the location. This image merely serves as a visual aid for the user, when selecting locations during mission design.

To teach the rover paths between points in a home, the user is presented with a simple, wizard-like interface to define each step of the path. Each of these steps map directly to Actions, and may be something like "drive until you are directly in front of a landmark," "climb up a step," or "turn ninety degrees." Figure 9 depicts the start of path teaching. The user is presented with an image of what the rover can see, the wizard for instructing the rover, a box where the history of the actions performed will be displayed, and other information relevant to this path. By progressing through a series of panels, such as those shown in Figures 10 through 13, the user can instruct the rover exactly as necessary. The full wizard, along with the Actions that can be produced, is shown in Figure 14.

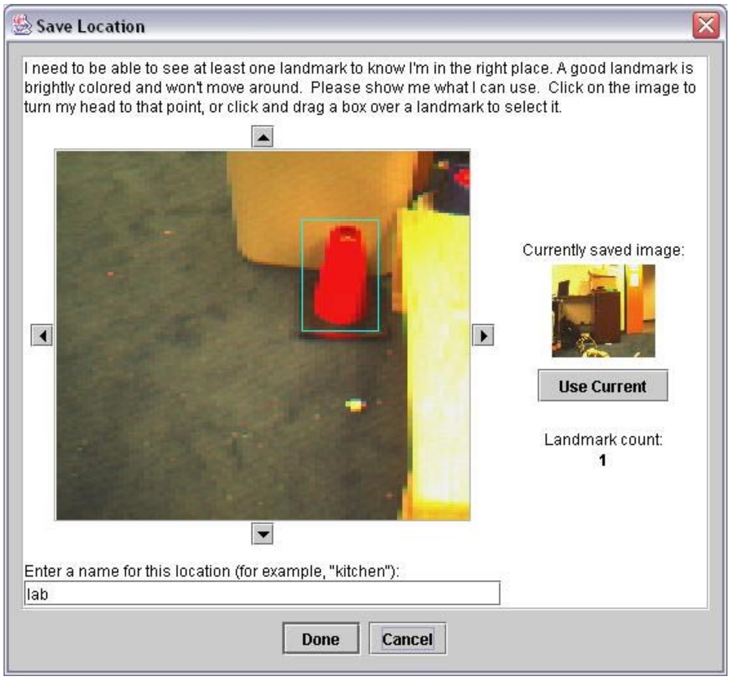

Figure 8: Selecting a landmark while saving a location

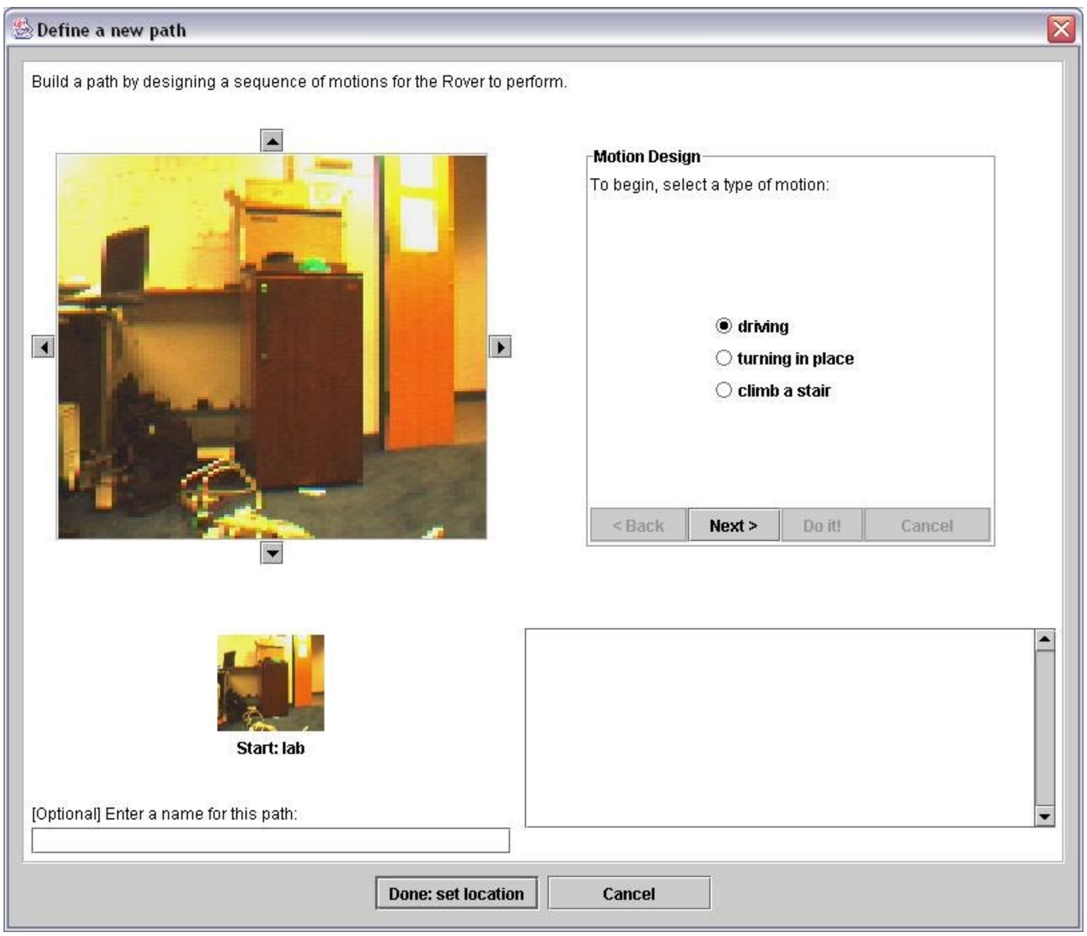

Figure 9: Start of path teaching

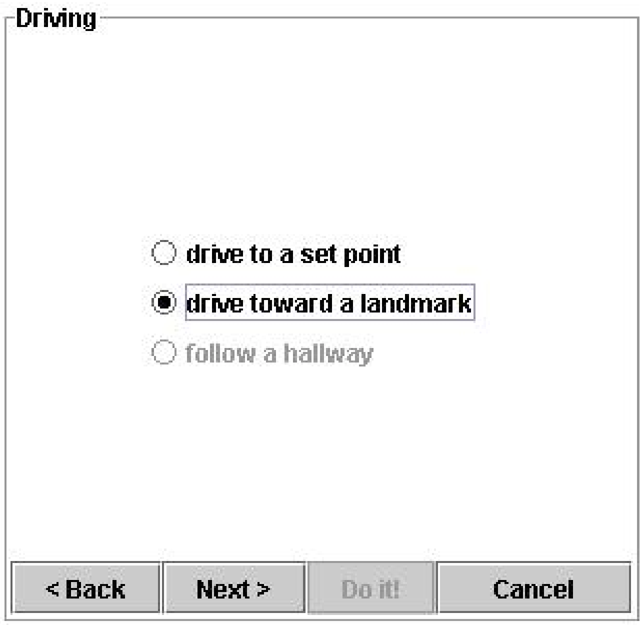

Figure 10: Driving options

Figure 11: Selection of a landmark

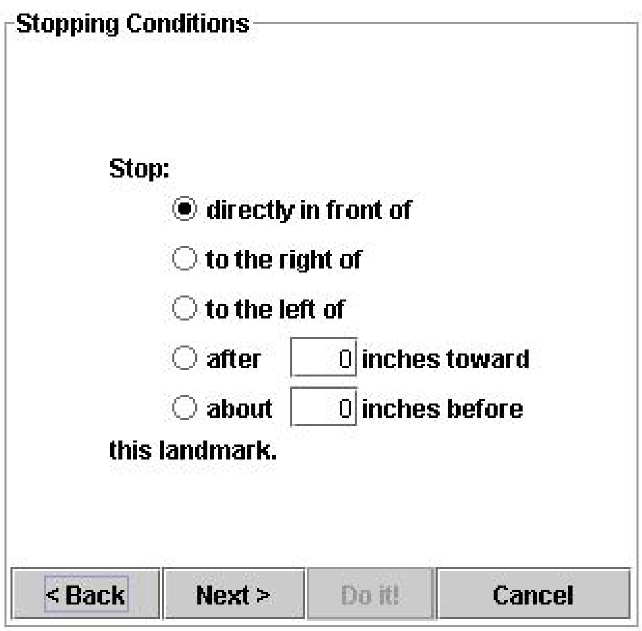

Figure 12: Stopping conditions

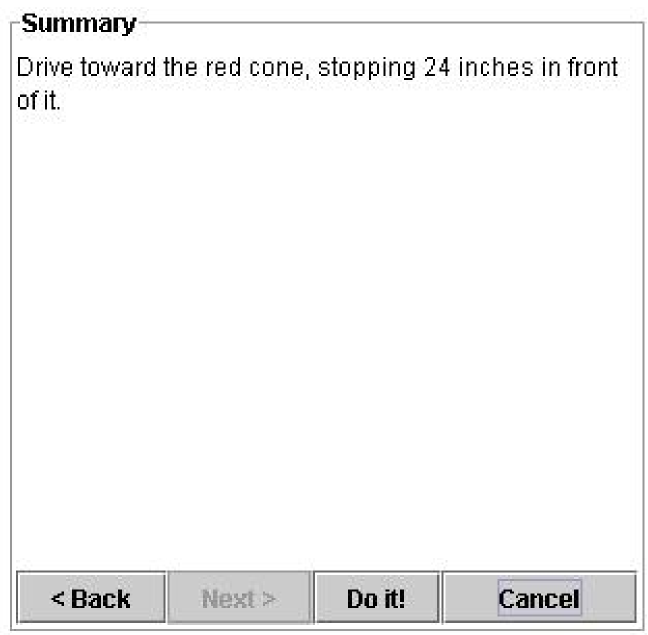

Figure 13: Summary

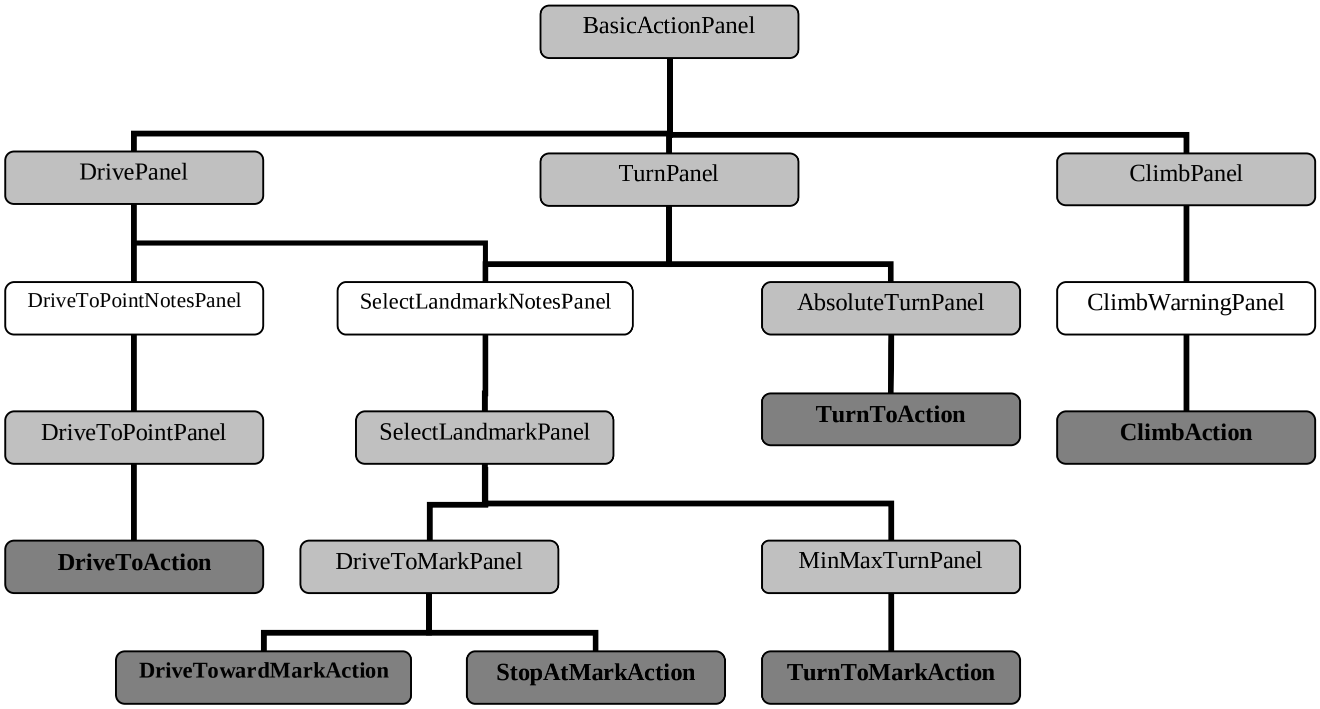

Figure 14: Flow of ActionPanels in action design wizard. Actions are shown in dark gray, panels which request user input are shown in light gray, and panels which merely provide information are shown in white.

Future Considerations

Repeatable Landmark Recognition. More work needs to be done to improve landmark recognition. While the current algorithm can detect landmarks repeatedly much of the time, recognition is still heavily dependent on lighting conditions and various other factors. The next version of CMUcam, currently in development, promises to offer histogram capabilities and other features that will be useful in this regard.

User Interaction and Error Handling. Currently, while path teaching is highly interactive, path execution is not as much so. In the future, we plan to provide a great deal of feedback and interaction while the rover is following a path. For example, if the rover loses sight of a landmark, it will contact the user, perhaps by email or via the internet, asking for assistance. The user would then be able, for example, to teleoperate the rover past an obstacle, or to specify an alternate route that the rover could use.

Mission Design, Scheduling, and Execution

The rover's daily activities are controlled through the design and execution of autonomous missions. Each mission is a task or experiment that the user has constructed from a set of individual rover movements and actions. Missions may mimic the exploratory and scientific missions performed by NASA's Mars Rover or accomplish new goals thought up by the user. Missions are fairly autonomous, with varying degrees of user interaction in the case of errors or insurmountable obstacles. Mission scheduling allows the rover to carry out missions without requiring the user's presence.

Goals

Our goals in developing a user interface for mission design, scheduling, and execution include:

- The mission design interface should allow the user to design and program creative missions by combining individual actions. The interface should be intuitive enough so that the user can begin using it immediately, but flexible enough so as not to limit the user's creativity as they grow familiar with the rover.
- Mission scheduling should make the user think beyond the rover's immediate actions to the rover's long-term future over days and even months.
- Mission execution should offer adjustable degrees of human-machine interaction and control for mission reporting and error handling.
- The software should support communication of the rover's status through different means such as email, PDA, or cell phone.

Implementation

Mission Development. To build a mission, the user first clicks on the Mission Development tab. Here there is a set of "blocks" grouped by function, with each block representing a different action that the rover can perform. Some of the blocks are static, such as the block used to take a picture. Others can be defined and changed by the user through the teaching interface. For example, the block used to follow a path allows the user to choose any path that they have previously taught the rover.

The user can select a block by clicking on it. While a block is selected, clicking in the Mission Plan section will place the block and cause a gray shadow to appear after it. This shadow indicates where the next block in the mission should be placed. To build a mission, the user simply strings together a logical set of blocks. Figure 15 shows the building of a small mission.

As each block is placed, a popup window is displayed. Here the user can enter the necessary details for the action, for example, the starting and ending location of a path (Fig 16).

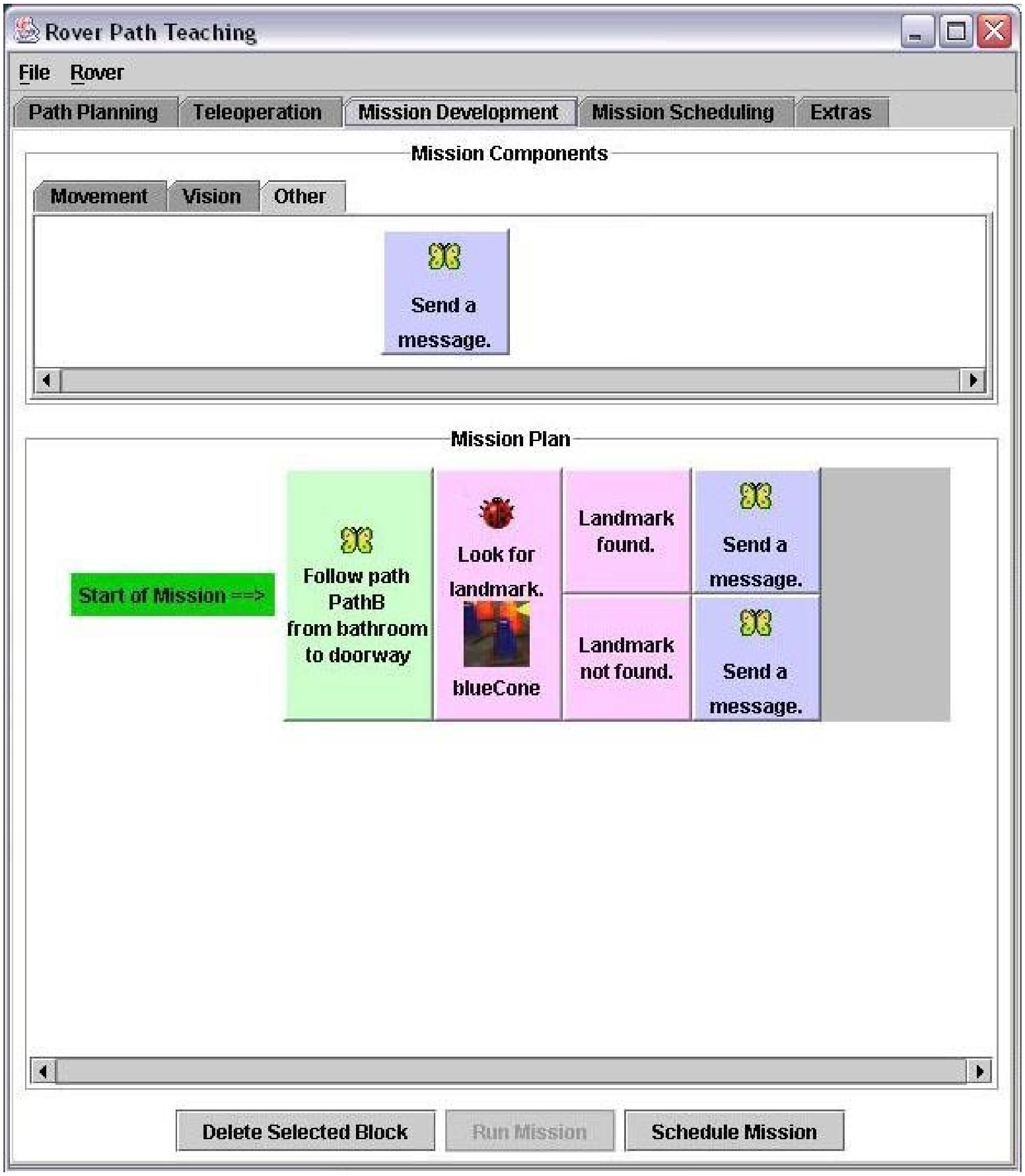

Fig 15: The user can build a mission by placing individual action blocks together.

Fig 16: A popup window prompts the user to select a starting location and then an appropriate ending location to create a path. Ending locations which would create an invalid or unknown path are disabled.

We have currently implemented two different types of blocks. The first simply represents a single action that can be followed directly by another action, for example sending a message (Fig 17). The second represents a conditional action, in which different actions can be taken based on the outcome. For example, when looking for a landmark, one action can be taken if a landmark is found and a different action can be taken if the landmark is not found (Fig 18). These blocks can have any number of conditions. As well as the true and false conditions shown in the landmark example, blocks can condition on equality and inequality. For example, one could implement a block for checking if the IR rangefinder value is less than x, equal to x, or greater than x.

It is possible to build a mission that cannot be run by the rover. For example, the simple mission “follow a path from A to B then follow a path from C to D” does not make sense. The step to get the rover from location B to location C is missing. When errors such as this occur, the mission may not be run or scheduled. A red X icon indicates the blocks where there are errors (Fig 19). The user can delete the bad blocks, or right click on a block to display the popup window and edit the details for that block. After the errors are corrected, the user is free to run or schedule the mission. Other than mismatched locations, currently supported errors are invalid paths and invalid landmark selections.

Fig 17: Sending a message is an unconditional action.

Fig 18: Looking for a landmark is a conditional action. There are two different possible outcomes.

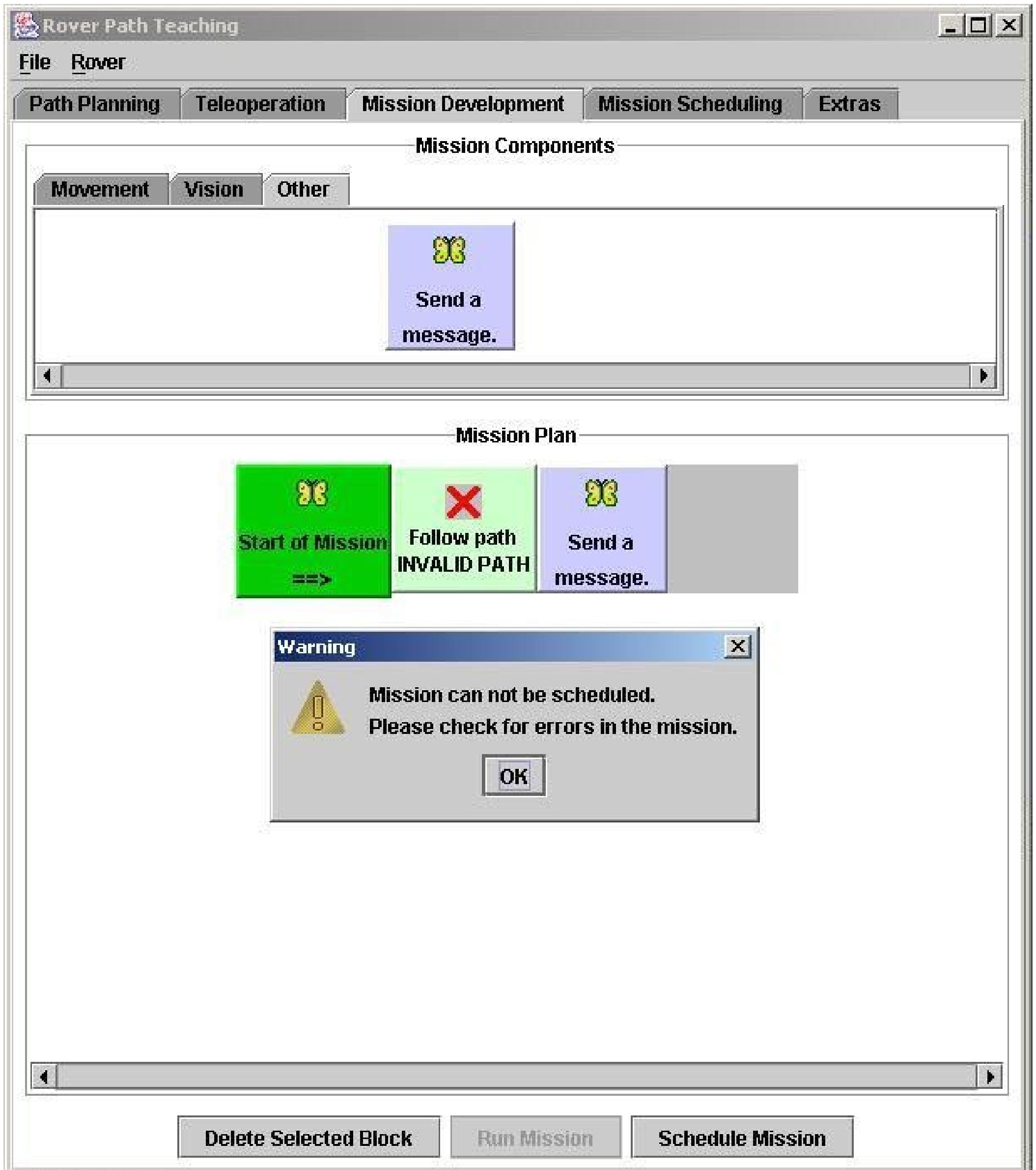

Fig 19: A red X icon indicates any blocks with errors. The mission may not be run or scheduled until the errors are corrected or removed.

One planned future improvement in the area of mission development is to implement two new block types. One type of block will allow sections of the mission to be repeated. The user will be able to choose a number of times to repeat the section, or to repeat until a certain condition is met. The other block type will allow the user to define her own subroutine blocks. These user-defined blocks can then be used as functions, allowing a set of actions to be added to the mission as a group. The user-defined blocks will also allow the same set of actions to be easily added to multiple missions. Other improvements include allowing the user to save missions, open saved missions, and copy sections of missions.

Mission Scheduling and Execution. After designing a mission, the user has the option to run the mission immediately or schedule the mission. Scheduling the mission allows the user to select a starting time and date as well as how often and how many times the mission should be repeated. The user also gives the mission a unique name. Figure 20 shows the scheduling wizard.

Before accepting a mission schedule, we check for conflicts with all of the previously scheduled missions. If any conflicts are found, we prompt the user to reschedule the mission, cancel the mission, reschedule the conflicts, or cancel the conflicts as shown in Figure 21. In the future, we plan to allow both the precise scheduling currently implemented and a less rigid scheduling method. For example, the user could schedule a mission to run around a certain time or whenever the rover has free time. For these more flexible missions, the rover will handle conflict avoidance without requiring additional user input.

All of the scheduled missions can be viewed by clicking on the Mission Scheduling tab (Fig 22). The user can select any of the scheduled missions to view the details of the schedule. The user can also cancel a mission or edit the schedule. In the future we plan to implement a graphical view of the rover's schedule. The Mission Scheduling panel will include a calendar showing all of the scheduled missions.

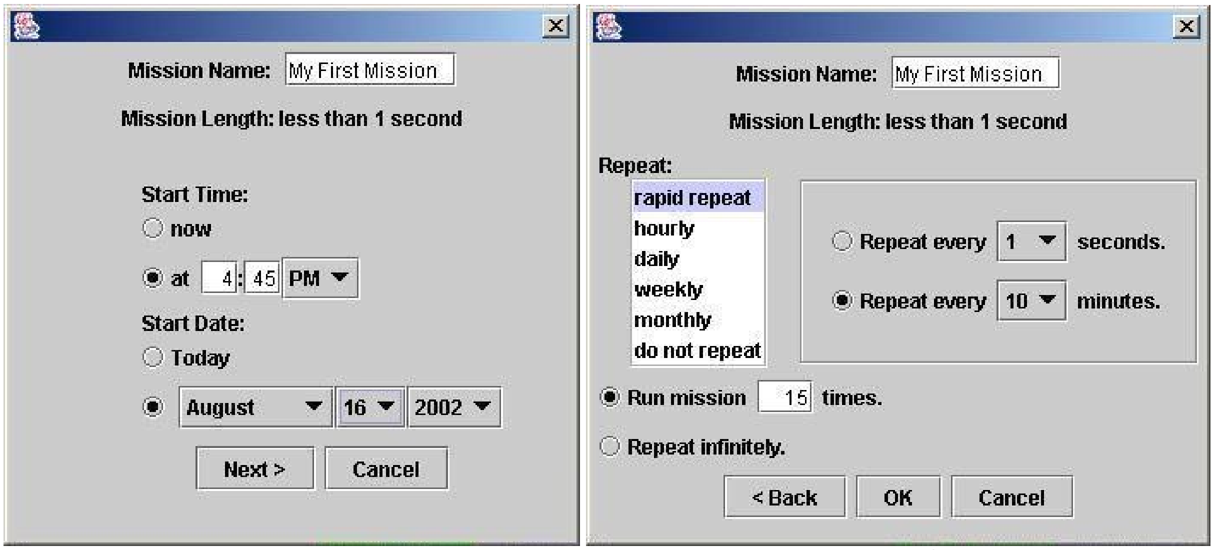

Fig 20: When scheduling a mission the user selects the start time and date as well as how often and how many times to repeat the mission.

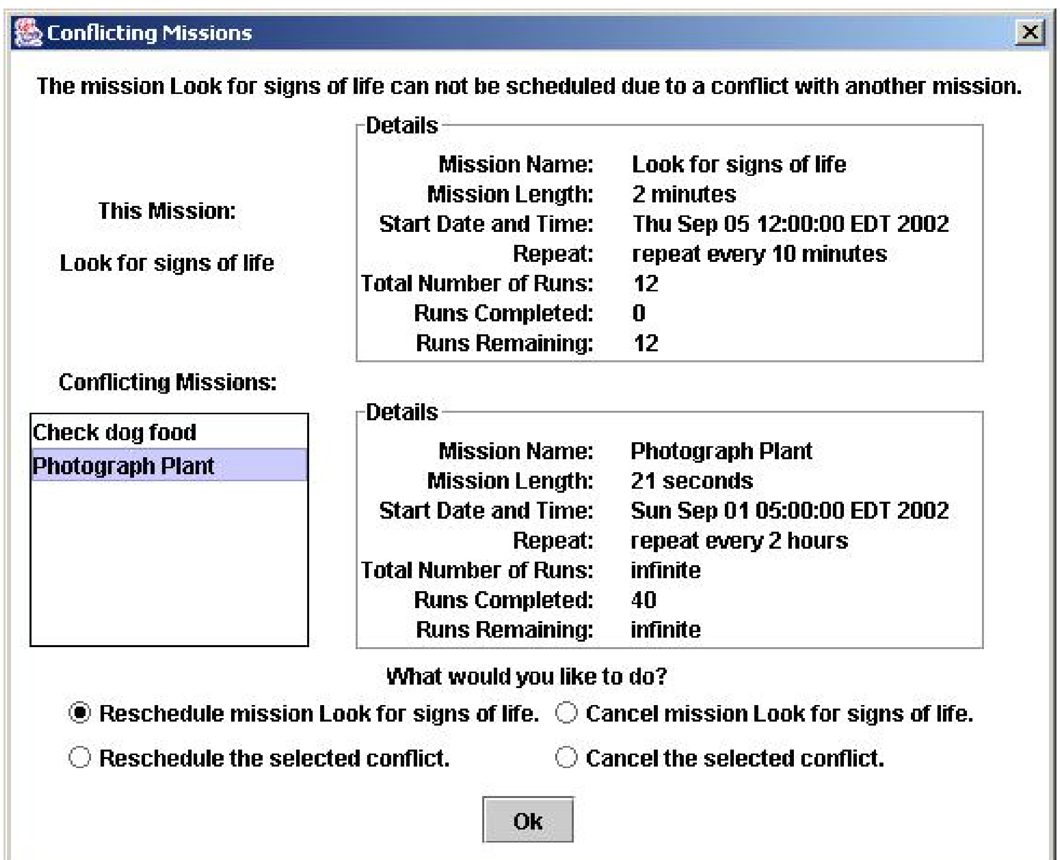

Figure 21: When there is a scheduling conflict, a dialog prompts the user to resolve the conflict.

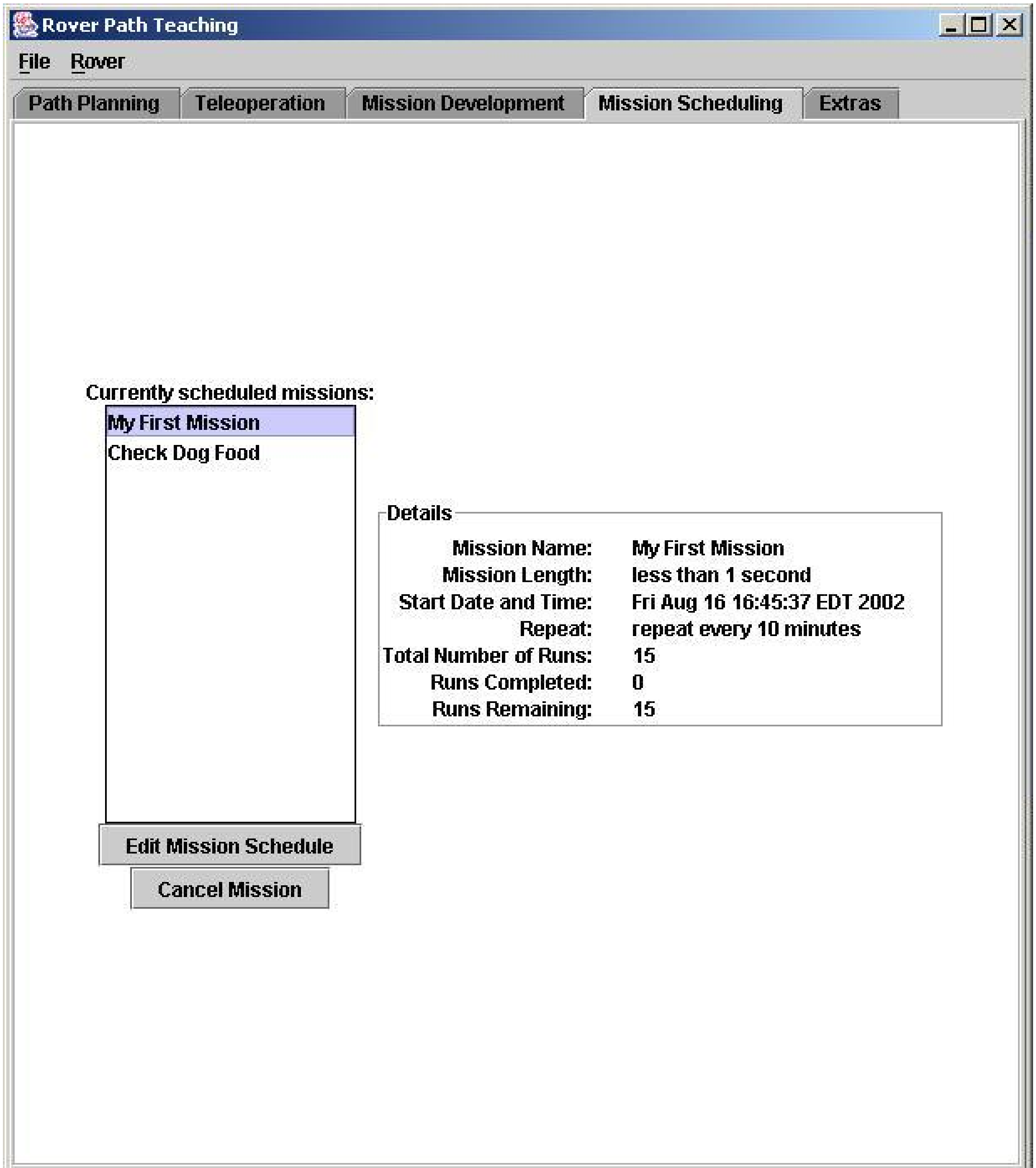

Fig 22: Under the Mission Scheduling tab the user can see all of the scheduled missions and view the details of their individual schedules. Here the user can also edit a mission schedule or cancel a mission.

Feedback. Currently under construction is a screen to display the rover's status. As a mission is executed, the current mission and the current step in that mission will be displayed. If the rover gets lost or stuck, this will be displayed as well. Eventually the goal is for the rover to ask for help in such situations. The user could then do such things as instruct the rover to try again later, or help the rover to drive around an obstacle. When the user helps the rover, the rover could add the new actions to the mission in case it gets stuck in the same way again. We also plan to add communication through email, web page, PDA, or cell phone.

Performance at the 2002 AAAI Conference

Although people enjoyed the rover at the 2002 AAAI Conference, several things did not go as smoothly as we would have liked:

- *Wireless*: We experienced a number of issues with wireless networking, as many groups at the conference attempted to support their own 802.11 networks. However, many of the issues were resolved by the second day of the exhibition.
- *Electrical system*: The rover that we presented is still a fairly early prototype, and we are still working out bugs in the electrical system. We found that, under the heavy use at the conference, the rover's microcontroller would frequently freeze for a short period. The next version of the rover will fix such problems.
- *Interface*: Attendees at the conference were much more interested in direct interaction with the rover, such as having it track someone's clothing, than in our teaching interface. While we were prepared for such an occurrence, we perhaps should have also provided a scaled-down teaching interface that could have been briefly utilized by anyone interested.

Acknowledgments. We would like to thank NASA-Ames Autonomy for their financial support, Peter Zhang and Kwanjee Ng for the design and maintenance of the rover electronics, and Tom Hsiu for the design of the rover hardware.

References

Nourbakhsh, I.; Bobenage, J.; Grange, S.; Lutz, R.; Meyer, R.; Soto, A. 1999. An Affective Mobile Robot Educator with a Full-Time Job. *Artificial Intelligence Journal* 114(1-2):95-124.

Thrun, S.; Beetz, M.; Bennewitz, M.; Burgard, W.; Cremers, A.B.; Dellaert, F.; Fox, D.; Haehnel, D.; Rosenberg, C.; Roy, N.; Schulte, J.; and Schulz, D. 2000. Probabilistic Algorithms and the Interactive Museum Tour-Guide Robot Minerva. *International Journal of Robotics Research* 19(11):972-999.