The Georgia Tech Yellow Jackets: A Marsupial Team for Urban Search and Rescue

Frank Dellaert, Tucker Balch, Michael Kaess, Ram Ravichandran, Fernando Alegre, Marc Berhault, Robert McGuire, Ernest Merrill, Lilia Moshkina, and Daniel Walker
College of Computing, Georgia Institute of Technology

Abstract

We describe our entry in the AAAI 2002 Urban Search and Rescue (USAR) competition, a marsupial team consisting of a larger wheeled robot and several small legged robots, carried around by the larger robot. This setup exploits complimentary strengths of each robot type in a challenging domain. We describe both the hardware and software architecture, and the on-board real-time mapping which forms the basis of accurate victim-localization crucial to the USAR domain. We also evaluate what challenges remain to be resolved in order to deploy search and rescue robots in realistic scenarios.

1 Introduction

Figure 1: Our robot team, consisting of 1 ATRV-Mini and 4 Sony Aibo legged robots.

A team from the Georgia Tech's College of Computing participated in the AAAI 2002 Robotic Search and Rescue competition, held annually to advance the state in the art of robotics in the Urban Search and Rescue (USAR) domain. In the competition, a standard test arena provided by the National Institute of Standards and Technology (NIST) provides a setting to evaluate different robot hardware and software architectures in a more realistic setting than in the lab. The goal of the competition is to accurately locate victims in the test-arena, which is off limits to human operators. The arena has three sub-areas with varying degrees of debris, but most of the competing teams never enter the more difficult orange or read areas, which are not navigable by typical research robots.

Our entry (see Figures 1 and 2) was motivated by the observation that a heterogeneous team of a large robot and several smaller robots compliment each other's strengths. A larger robot can carry more capable sensors, provide substantial computation, is faster and has more autonomy than smaller robots. However, because of its size it is not able to travel everywhere in the environment. This is certainly the case for the challenging environments one expects to find in the aftermath of a disaster, with fallen debris, collapsed walls, and cluttered floors. Smaller robots, while having less autonomy and sensing, compensate by their ability to go in small and inaccessible places, small crevices etc.

Figure 2: The robot team while operating in the NIST-USAR arena. One of the Aibos has un-docked and is looking for victims, not seen in the image.

The marsupial team we entered consisted of an iRobot ATRV-Mini as the main robot, providing many sensors and computational resources, and four Sony Aibos legged robots, able to explore small areas the main robot cannot

reach. Figure 2 shows the ATRV and three of the Aibos, while in the NIST-USAR arena. The hardware used is discussed below in more detail in Section 2.

One perhaps unusual feature of our entry was that all the software (except interaction with the Aibos) was written in a functional programming language, ML. This is including low-level communication to the sensors and cameras on the robot. This ensured that the probability of run-time errors and crashes due to segmentation faults etc. was minimized. Also, ML lent itself surprisingly well to the specification of the robot architecture, which was based on an ML version of Clay. Clay is a robot architecture proposed by one of us (Balch) as part of Teambots, a software architecture for distributed robot teams (www.teambots.org). Finally, the strong typing imposed by ML combined with its mechanism of type inference allowed for a fast development cycle and made it easy to share code between the different programmers on our team. ML-Clay and the architecture we used is discussed briefly in Section 3.

Figure 3: The "away team" in Edmonton, with the robots. From left to right: Tucker, Ram, Frank and Michael.

The main sensors on the ATRV are a SICK laser range scanner and a custom-built omni-directional camera rig. One of the main capabilities the larger robot brings is the ability to construct an accurate map of the environment on the fly, based on laser range scans provided by the SICK scanner. This map can then be relayed to the operator for tele-operation, and is also the basis for providing information as to victim location. Mapping is explained in more detail in Section 4. Where the scanner provides a map, the camera rig functions as the "eyes" of the operator, who can choose between each of the 8 cameras that together provide a 360^o field of view. Using these cameras, victims can be identified and placed in the map by the operator.

The Sony Aibos were dispatched whenever the ATRV was unable to enter a specific area, or when it was deemed unwise to do so. Operator training and experience played a large role during the competition itself, where one operator controlled all the robots, albeit in a serial manner. The localization of the Aibos was done manually by the operator by monitoring them from the ATRV camera rig. When an Aibo (or the ATRV) then found a victim, it was marked manually on the map by the operator. In Section 5 we discuss the operator interface for this and general tele-operation.

Our successful entry into the competition was made possible with the help of many students, graduate students as well as undergraduates, and three dedicated faculty/research scientists. The "away-team" in Edmonton consisted of two faculty (Dellaert & Balch), one graduate student (Michael Kaess), and one undergraduate (Ram Ravichandran), shown in Figure 3. As with any experience, we have learned valuable lessons about our specific approach, as well as about the realities associated with the USAR domain. They will be discussed in Section 6.

2 Hardware Platform

2.1 ATRV-Mini

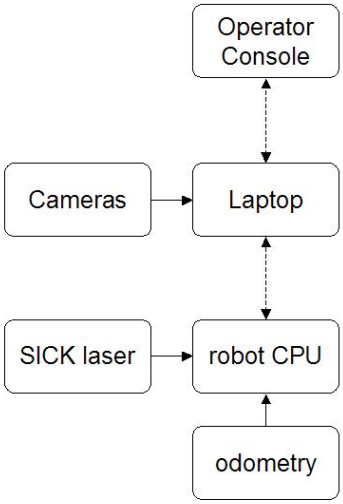

Figure 4: System architecture for the larger robot: the odometry and SICK laser scans are handled by the on-board computer, and relayed via Ethernet to a laptop on top of the robot. This laptop interfaces to the cameras, and also provides wireless communication to the operator console.

The ATRV-Mini is a commercial robot platform from iRobot. A SICK laser scanner was added in the front, as well as an aluminum mount on the top to fit the omni-directional camera rig together with a laptop for computer vision tasks. As illustrated in Figure 4, the laptop connects to the on-board PC over an Ethernet connection, and to the operator console using a PCMCIA 802.11 card, providing wireless communication with the robot. On the back of the robot an aluminum platform was mounted which can hold up to four Sony Aibos.

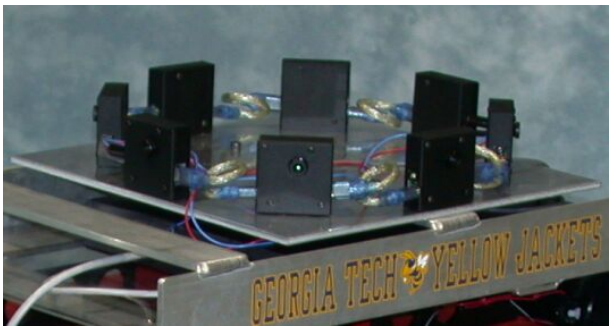

Figure 5: Omni-directional Camera Rig consisting of 8 FireWire DCAMs.

2.2 Omni-directional Camera Rig

The omni-directional camera rig was custom built in our lab, and consists of eight FireWire DCAMs from Videre Design. They are mounted every 45 degrees around a circle, their principal axis parallel to the ground. They are connected to the laptop over two FireWire busses, since each bus can only handle synchronous data transfer from four cameras at a time. Power is provided from an extra battery residing between the cameras.

2.3 Sony Aibos

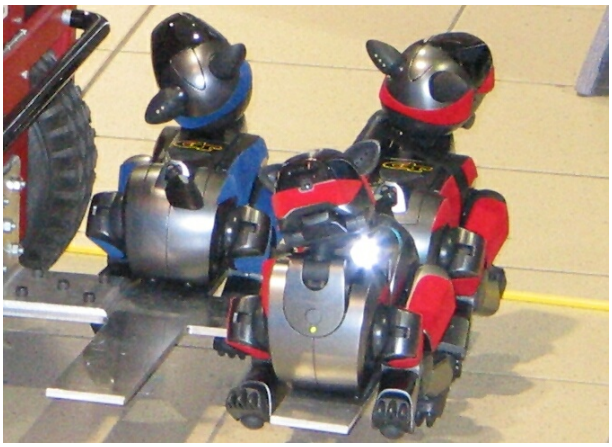

Figure 6: Sony Aibo with LED flashlight.

Four Aibos were used, which can be carried on the aluminum platform on the back of the ATRV. The Aibos are commercial robot platforms from Sony with 17 degrees of freedom, a color video camera built into their nose, and a PCMCIA 802.11 card for wireless connection. They provide on-board processing and were programmed using the Sony SDK. To also enable vision in dark rooms, a LED flashlight was added. Figure 6 shows three of the Aibos mounted on the platform, which was designed so the robots can easily dock and undock, respectively by standing up or taking on a "clamp" position.

3 Software Architecture

The software architecture we used to control both types of robots consists of a distributed set of applications, which are:

1. Aibo-Commander: a C application running on the Aibo operator console, wirelessly controlling the Aibo legged robots. Its interface is discussed below.
2. Control: a C application on board of the robot that provides low-level control and acquires SICK and odometry measurements. It also acts as a server in communicating over the Ethernet wire with the laptop.
3. Commander: the user interface on the operator console, written entirely in ML, which acts as both client and server in a wireless client-server communication scheme with the laptop. It serves commands and consumes images, laser scans, and incremental map updates to relay feedback to the operator.
4. Rescue: the main mapping and high-level robot control software running on the laptop, also written in ML, acting as a client to both Commander and Control, and also serving images, laser, and map updates to Commander. Images are JPEG-compressed before transmission to conserve bandwidth.

3.1 Use of a Functional Language

Both Rescue and Commander were written in a functional language, ML, which provides strong typing, type inference, and higher order functions. Strong typing proved to be invaluable in having a large team of programmers put together a complex application in a relatively short amount of time. Type inference is a mechanism by which types are automatically inferred from context, at compile time, and hence obviate the need to annotate the code with types. This gives a 'scripting' feel to the language, albeit with strong types. Higher order functions are functions that take other functions as input, and allows one to specify, for example, a generic particle filter as in Figure 7.

The compiler we used was Objective Caml, which produces very fast code, achieving performance on par with C/C++ code. We used Caml to do client-server communication, image processing, particle-filtering, map building etc, all in real-time and distributed over two computers. Functional languages are useful in the real world, can handle the real-time constraints of robotics and vision, and provide a vastly superior programming language than C or C++.

3.2 Clay Architecture

The Rescue application is configured as a Clay architecture shown in Figure 8. Clay (Balch, 1998) was originally written in Java as a group of classes that can be easily combined to create robot control systems. We have written a version of Clay in ML, taking advantage of polymorphic types to facilitate combining, blending and abstracting behaviors and processes.

```
type 'x likelihood = 'x -> float
type 'x samples = 'x list
type 'x motionModel = 'x -> 'x
val step: 'x likelihood -> 'x motionModel -> 'x samples -> 'x samples
```

Figure 7: Type specification of a generic particle filter in ML, using higher order functions. The function step takes two functions as arguments, one that computes the likelihood for each sample, and another that moves a sample according to a motion model. These functions generally are partially applied versions of other functions that take additional arguments concerning the likelihood calculation and motion model used.

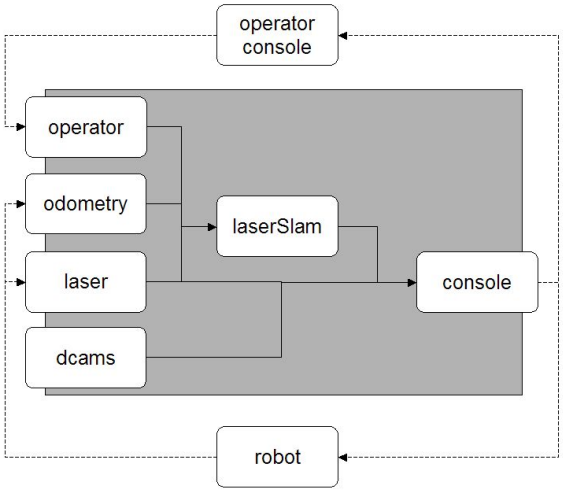

Figure 8: Clay node configuration.

The basic building block in Clay is a *node*. There are two important phases in a node's life: initialization and run time. Most nodes have only two methods, corresponding to these phases: the constructor, used for initialization; and `value`, called repeatedly at run time. Nodes often have other nodes embedded within them (e.g. an `avoid_obstacle` node typically has a `detect_obstacle` node embedded within it). The embedding is specified at initialization time using the node's constructor. Here is an example of how we would embed one node in another:

```
let avoid_obstacles  =
let detect_obstacles =
new obstaclesNode abstract_robot in
new avoidNode 2.0 1.0 detect_obstacles;
```

In this example, a `detect_obstacles` node is created using the `obstaclesNode` class (the `obstacleNode` class knows how to query the robot hardware for information about obstacles). Next an `avoid_obstacles` node is generated by embedding the `detect_obstacles` node in a `avoidNode object`.

Once a Clay node configuration has been specified, it is repeatedly called at run time for it's present value, based on the current situation of the robot. All the `value` methods take a time stamp as a parameter, so it will only compute its value once per time step.

4 Mapping

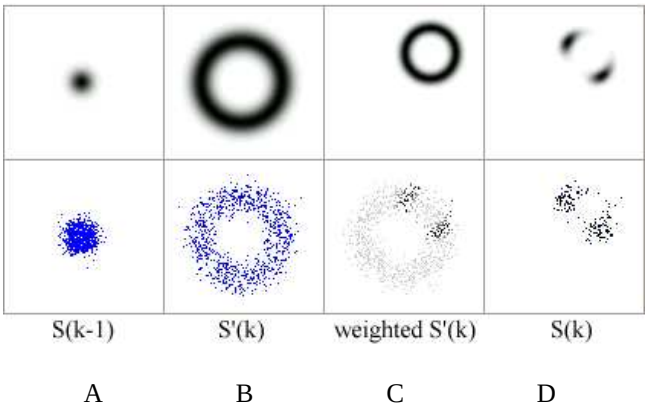

Figure 9: The probability densities and particle sets for one iteration of the MCL algorithm. See text below.

The real-time mapping algorithm we used is based on Monte Carlo Localization (MCL) (Dellaert et al., 1999; Fox et al., 1999), using the SICK laser range scanner and the odometry as measurement inputs. MCL represents the probability density over robot pose by maintaining a set of *samples* that are randomly drawn from it. By using a sampling-based representation one obtains a localization method that can represent arbitrary distributions, in contrast to Kalman-filter based approaches which are limited to using normal densities over the state space.

One iteration of the algorithm is illustrated in Figure 9. In the figure each panel in the top row shows the exact density, whereas the panel below shows the particle-based representation of that density. In panel A, we start out with a cloud of particles $S(k-1)$ representing our uncertainty about the robot position. In the example, the robot is fairly localized, but its orientation is unknown. Panel B shows what happens to our belief state when we are told the robot has moved exactly one meter since the last time-step: we now know the robot to be somewhere on a circle of 1 meter radius around the previous location. Panel C shows what happens when we observe a landmark, half a meter away, somewhere in the top-right corner: the top panel shows the likelihood $P(Z_k|X_k)$, and the bottom panel illustrates how the sample set $S'(k)$ is weighted according to this likelihood. Finally, panel D shows the effect of resampling from this weighted set, and this forms the starting point for the next iteration,

$S(k)$.

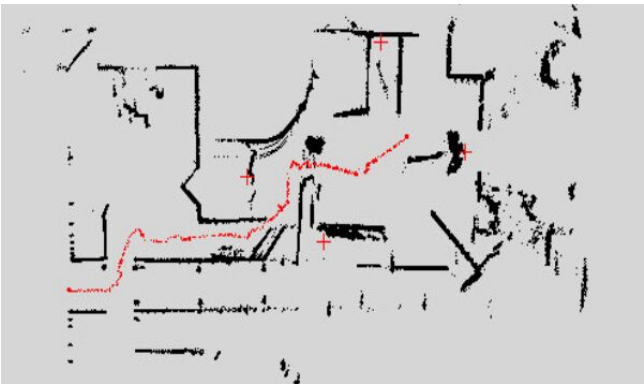

Figure 10: Laser map built in real-time, with marked victims (plus signs).

The measurement model used for the SICK laser calculates the overlap between the endpoints of the current laser scan and an occupancy map that is created on the fly. The overlap is used as an energy measure which is transformed into a probability using the Gibbs distribution, and then used as the likelihood. We typically used 250 samples, and use the weighted mean pose of the weighted samples as the point estimate of robot pose. This estimate is then used at each time step to update the occupancy map with the current laser-points.

A laser-map built in this way, during one of the competition runs, is shown in Figure 10. Victims found during that run are shown as crosses. Note that the map is noisy in places because some assumptions we made were violated. In particular, several walls of the arena were made of transparent Plexiglas that behaved as a mirror under certain grazing angles. Also, at one point the robot pushed open a doorway which in effect changed the map, and this is not currently handled by the algorithm.

5 Interface & Experience

A large part of the programming effort concentrated on the remote control user interface and the wireless communication protocols to make real-time control a possibility. The interface is split in two parts running on two different laptops to allow the operator to see everything at once. The ATRV interface controls the main robot and its cameras, the Aibo interface provides control over one of the Aibos at a time.

The ATRV interface (Figure 11) consists of four parts:

1. A control window to switch between cameras and modes.
2. A camera window showing live images from one of the eight cameras.
3. A laser window showing the most recent scan consisting of 361 points with an outline of the ATRV for easy navigation.

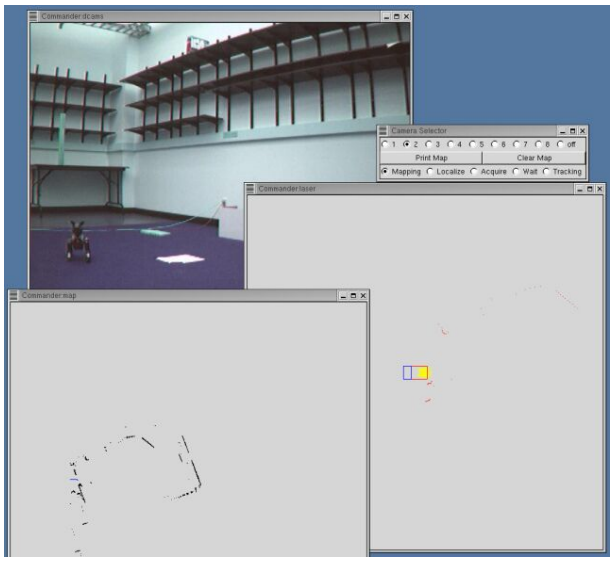

Figure 11: ATRV Operator remote-control interface.

4. A map window providing an integration of all laser scans into an accurate map.

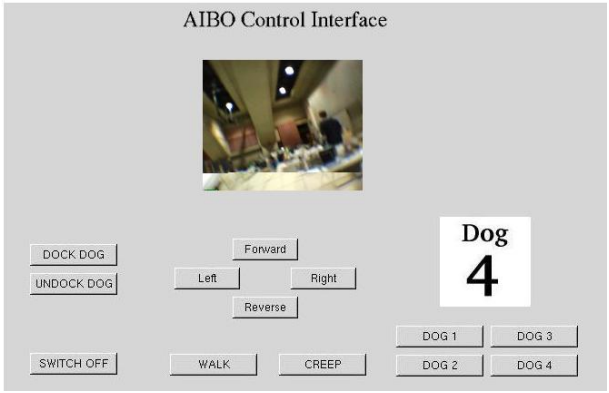

Figure 12: Aibo remote-control interface.

The Aibo interface, shown in Figure 12, allows the operator to select one of the four Aibos for control. It shows live images from the selected Aibo's built-in camera, allows to undock and dock the robot on the aluminum platform on the back of the ATRV, walk forward and backward, turn left and right, and select from several different gaits.

6 Discussion

Our team came in third in a field of 9, and won a technical award for its real-time mapping capabilities.

We also participated in the HCI study by the University of Massachusetts at Lowell, led by Holly Yanko. As part of that study, a local fire-chief evaluated our system by taking the operator role during an evaluation run. The resulting

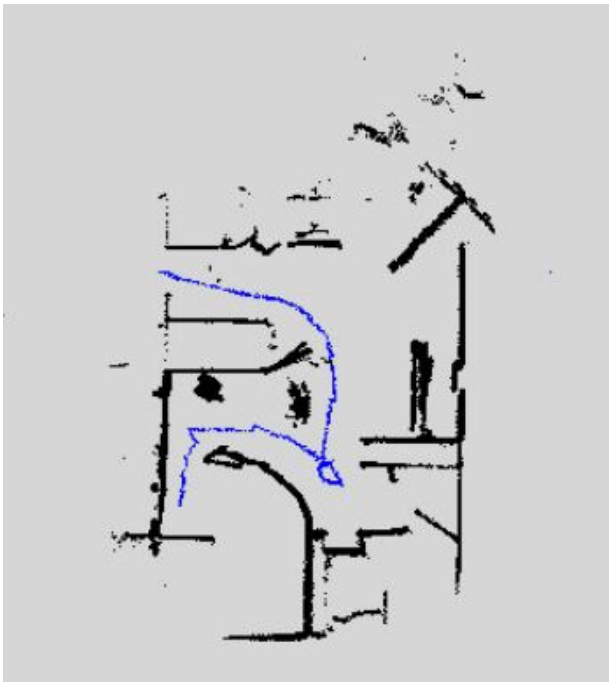

Figure 13: Map created during the fire-chief run.

map is shown in Figure 13. The chief communicated to us that especially the accurate and real-time mapping provided a valuable resource that could be used in realistic rescue scenarios. A negative was that our larger robot proved to be difficult to navigate in the confined space of the test arena.

The difficulty in navigating with the robot in the context of a real-time, remote link suggests an immediate improvement, namely semi-autonomy. The operator should be able to provide a goal location and leave the robot to do the necessary configuration planning and path execution to achieve the goal with a much tighter control loop than possible over the remote link. This is well within reach of the current state of the art and would save a lot on time now lost by an overly conservative control strategy on the part of the operator.

The absence of reliable wireless communication was the biggest handicap that faced our team in the competition, and hence robust communication has to be considered as much part of the problem as the more academic research questions of semi-autonomy and real-time mapping. In the world-trade center disaster, where USAR robots where deployed in a realistic setting for the first time, all robots were tethered with long lines to avoid this type of communication breakdown.

Ultimately, one of the most crucial factors in making robots useful in the USAR domain is purely mechanical: can the robots robustly navigate in a post-disaster environment cluttered with debris and non-trivial obstacles ? Not surprisingly our robot platforms, one a wheeled research robot and the other designed for the toy-market (albeit very capable), were not able to enter either the orange or red areas of the test arena, which resembled more closely the type of

Figure 14: During one of the competition runs, the ATRV was retro-fitted with an improvised contraption to avoid getting tangled in debris hanging from the ceiling, which would have otherwise torn off some of the cameras or cables connecting them together.

environment USAR robots will eventually encounter. Even the "easy" area sometimes provided challenges that needed some improvising at the time of the competition, as shown in Figure 14. The design of rugged robots that can perform the task is really up to industry, although researchers could play an important role in devising new paradigms for robust locomotion, for example inspired by the fundamental principles of effective animal locomotion (Clark et al., 2001).

Acknowledgments

We gratefully acknowledge Sven Koenig, who funded the ATRV platform. Jonathan Shaw, Amy Hurst, Karthik Subramanyam, and Brian Feinstein provided additional support. We are especially grateful to NIST and the organizers of the competition for organizing a successful event.

References

Balch, T. (1998). *Behavioral Diversity in Learning Robot Teams*. PhD thesis, College of Computing, Georgia Institute of Technology.

Clark, J., Cham, J., Bailey, S., Froehlich, E., Nahata, P., Full, R., and Cutkosky, M. (2001). Biomimetic design and fabrication of a hexapedal running robot. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo Localization for mobile robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte Carlo Localization – Efficient position estimation for mobile robots. In *AAAI Nat. Conf. on Artificial Intelligence*.