Multi-Robot Disaster Recon: Building Software Foundations with Real and Simulated Robots to Work with a Common Interface

Roger Wu, Curtis M. Humphrey, Sanford T. Freeman, Julie A. Adams
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, Tennessee USA 37235

Abstract

As progress towards automation and decreased human teleoperation of multi-robot systems continues, the need arises for a software system encapsulating the robotic device to train new users and test new autonomous algorithms. This system carries several advantages: reducing wear and tear on the physical robots during training and testing, reducing redesign at the user interface level, and reducing time spent with users in training. The system provides support to the Flash User Interface (UI) (Humprey et. al, 2007), developed to improve situational awareness while operating multi-robot systems, via the Gamebots protocol (Wang et. al, 2005) parsing and adapting these commands to the Player (Gerkey et. al, 2001) provided interface on-board the robot or in conjunction with the Gazebo simulator (Koenig and Howard, 2004) via TCP. Design and functionality of the current system is reviewed, and milestones for future versions are outlined.

Introduction

Multi-robot systems are complex, and require extensive testing, in terms of both automated control and human-robot interaction (Adams, 2006). Economic pressure stemming from hardware cost creates the need for a cheaper alternative for early testing, which is where software simulation comes into play.

Though not completely precise, simulations are still valuable for detecting early errors in algorithms, or for familiarizing users to new control interfaces. While pure simulation training is not viable, it is crucial for mitigating cost. The software system was designed to take advantage of incorporating both software simulation and actual robotic hardware. Our system reduces costs, design time, implementation time, and the learning curve of the system.

The system supports use of both the Urban Search and Rescue simulation (USARSim, 2007; Wang et. al, 2005; and Wang et. al, 2007) and physical robotic hardware. This integration is achieved through support for the Adobe Flash User Interface (UI) (Humprey et. al, 2007) in conjunction with either a combination of USARSim and Unreal Tournament 2004 for simulation testing, or through Player (Gerkey et. al, 2001; Player 2007) for utilizing physical robotic equipment.

Our objective for the Player based system is to create a bridge that adapts the hardware to the Flash UI commands via the Gamebots Protocol, created at the University of Southern California, as used and documented in the USARSim manual (Wang and Balakirsky, 2007). This system will allow users to become familiar with the Flash UI, instead of having to learn to use separate controls for simulation and physical hardware.

This paper presents the details of the design and development of the system regarding the software bridge in conjunction with Player, as well as how the bridge performed during initial testing.

The Software System

Overview

The Flash UI is designed to interact with both USARSIM and the UT2004 engine (Humprey et. al, 2007) when in simulation mode. For physical robot control, the software system communicates with the network messaging thread (see Messaging System section) as shown in Figure 1.

The network messaging thread passes commands to the robot control thread (see the Robot Control System), where the commands are parsed and then are sent to the robot class that controls the robot. The robot control functions within the robot class work with the Player software, which we chose due to the freedom it provides users in designing control programs (Gerkey et. al, 2001). Player manages the physical devices on board the robot or the simulated devices within a Gazebo test simulation, which attempts to realistically simulate the behavior of an ActivMedia Pioneer 3 (Koenig and Howard, 2004), controlling hardware and relaying data to the bridge.

The user can command the robot to send camera frames to the image transfer thread (see Image Transfer System) as shown in Figure 1. In

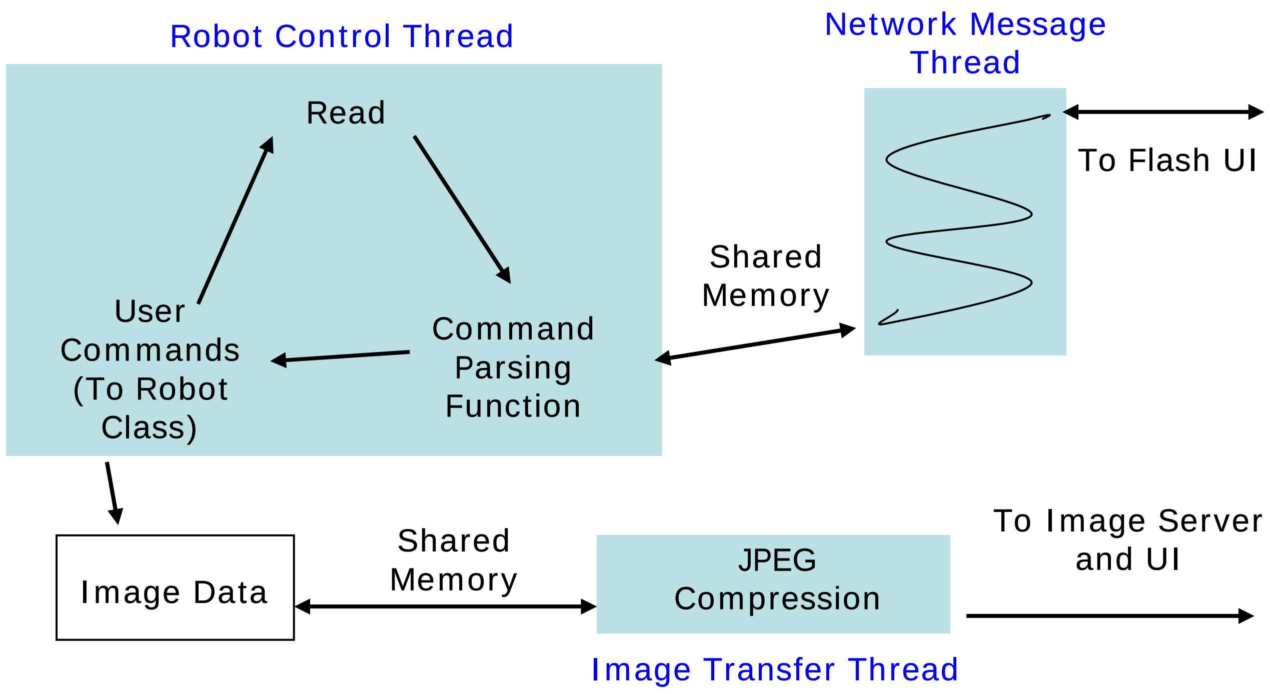

Figure 1: This shows the layout of the three threads of the software bridge.

the image transfer thread, camera frames are compressed into a JPEG file and are then sent to the Flash UI. A C++ image server program facilitates this process, accepting JPEG images through TCP/IP and writing them to the local hard drive, where the Flash UI reads the JPEG image files.

Messaging System

Commands passed from the Flash UI to the message thread conform to the Gamebots protocol (Wang and Balakirsky, 2007). These commands are copied into shared memory buffers and passed into the command parsing function that runs in the robot control thread. The received commands are parsed and the appropriate functions are activated and pass the message to Player using the provided proxies over TCP/IP (Gerkey et. al, 2001).

The message thread also receives an acknowledgement message after the command parsing function parses a command, placing that message within a separate shared memory buffer for return to the message transfer thread. The message transfer thread forwards the resulting message to the UI.

Robot Control System

The robot control thread contains a user terminated loop that cycles through the functions of a robot class instance, shown in Figure 2. The Player default update rate is set to every 10 milliseconds (Gerkey et. al 2001) in order to reduce the network load. This default update rate provides the frequency ceiling for the loop in Figure 3. The robot class contains all the Player device proxies that the bridge uses to control the physical hardware remotely through a TCP/IP connection, as well as functions necessary to load and stop the functions for initialization and termination of the robot class. Using the Player model, each device on the robot has its own Player device proxy (Gerkey et. al, 2001).

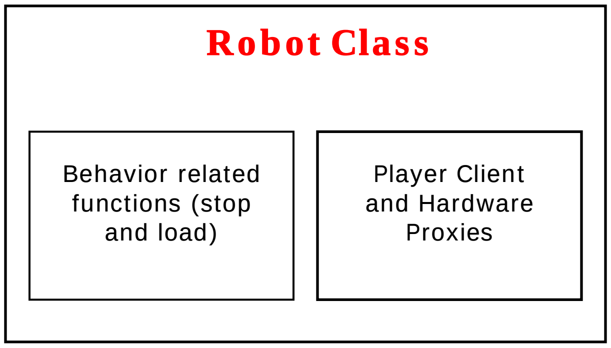
Robot Class
Behavior related functions (stop and load)
Player Client and Hardware Proxies

Figure 2: The robot class contains device proxies for robot control, as well as initialization and termination functions.

Figure 3 shows the behavior loop, which begins with a read command that receives data from Player regarding the state of the hardware. Once data is read from Player, the command parsing function executes by default, presuming the user desires control. The command parsing function checks for new commands in the shared memory buffer, and if there are new commands they are parsed. The parsing of these commands sets the parameters within respective behavior functions and places a response in the shared memory.

The remainder of the loop involves the other user defined command functions, with each command contained in an array of function instances. The loop checks for any actively flagged behaviors, the first being a user signaled quit. If no signal for quit has been received, then the loop proceeds with the other behavior functions.

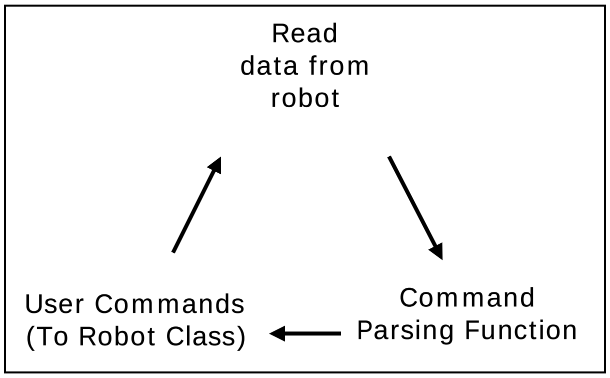

Figure 3: This loop is the center of the robot control system.

The Commands

The currently implemented behavior functions are simple prototype functions that permit experimentation with the physical robot hardware: *quit*, command parse, *drive*, *PTZ* (pan, tilt, zoom), and image transfer. The *quit* behavior terminates all commands and then signals the system threads to terminate. *Drive* and *PTZ* both set the Player simulated robot's position, speed, and camera position variables respectively with Player. They copy their own local position, speed, and camera position values set by the command parsing function to the Player stored values. Finally, the image transfer behavior checks repeatedly if 100 milliseconds have passed since the last image was transferred. The system updates the image data from the robot every 100 milliseconds and this minimizes the issuing of image grabbing instructions for images that have already been transferred by the system. If the image data is new, the raw camera data is placed into memory that is shared with the image compression thread. Once these steps are completed the waiting compression thread is notified.

Image Compression System

Interacting together with the image transfer behavior function, the image compression thread runs as a state machine. State information is maintained with two boolean variables protected with a mutex. A reader/writer's lock is maintained in cyclic fashion using this state machine (see Figure 4).

The initial state resides with the image transfer function. When a tenth of a second passes, the function accepts a new frame from the camera, which is placed in a mutex protected shared memory. A signal to a condition variable is sent to the image compression thread, which, when initialized, waits for a signal from the condition variable. After this write section, we proceed to the next state, reading the image.

State two involves reading the image from shared memory by the image compression thread. This step is also locked, to ensure that the entire image is read. After this step completes, we move to state three.

In state three, the lock is released on the shared memory, allowing the image transfer function to write a new frame. Meanwhile, the image compression thread compresses the JPEG, and then sends the image to the image server via the TCP/IP connection. From here, we return to state 2 once the image is transferred and the compression thread returns to wait on the condition test. If the image transfer thread finishes, we remain in this state, so that the frames are the most recent.

The state machine is required because signals sent to the image transfer thread are lost while the thread is not blocking execution. This method also allows for pipelining the image transfer, keeping the shared memory from causing a bottleneck.

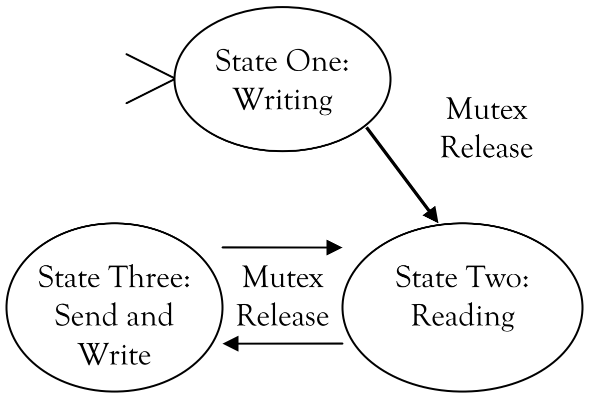

Figure 4: Image Transfer State Machine

Image Server

The stand alone image server functions to receive compressed JPEG files from the software system and writes them to the hard drive, deriving its functionality from a similar program that works with USARSIM (Wang et. al, 2005). This behavior supports the Flash UI users by writing frames to the hard drive and allowing then to rewind and review all images recorded by the robot, thus permitting analysis. This feature will also be useful during design and testing, especially when testing for image transfer fidelity.

Results

Two main aspects of the software system were tested, the functionality and the performance of the image system. Both sets of tests conclude that further investigation is necessary, but the initial results are promising.

Testing the functionality involved using the Gazebo 3D simulator (Koenig and Howard, 2004) as well as an ActivMedia Pioneer 3 robot. Tests involved sending instructions corresponding to our behavior functions (see Behavior Functions section) via the Flash UI. In simulation, our behavior functions performed as expected, providing motor and camera control, as well as providing images to the Flash UI. However, when using the Pioneer 3, we ran into trouble with the camera system, most likely due to camera driver issues with the Canon VCC4 camera (see Future Work). The software bridge used was able to provide PTZ control but was unable to grab actual camera images. The software bridge was also able to control the robot's motors for driving.

Image transfer tests were performed in three steps to analyze the transfer subsystem. As shown in Figure 5, we tested the system during the three image transfer points. Test 1 measured the time to compress 1,000 separate images. Test 2 involved both the image compression and transmitting the image over the TCP/IP connection. Test 3 includes compression, image transfer, and writing the images to the hard drive on the receiving machine. The tests involved twenty trials, with timing code measuring the values with precision to the second. The mean and standard deviation of the resulting data is provided in Table 1. Comparing the averages, I derived the percentage decreases in the system progressing from Test 1 to Test 3.

Our test hypothesized that the network transfer and hard drive writing would not be statistically significant in decreasing the frame rate. Using the data taken from the three tests, p and t values, shown in Table 2, were calculated to determine statistical significance. Using standard significance values, the data shows that we can accept the null hypothesis.

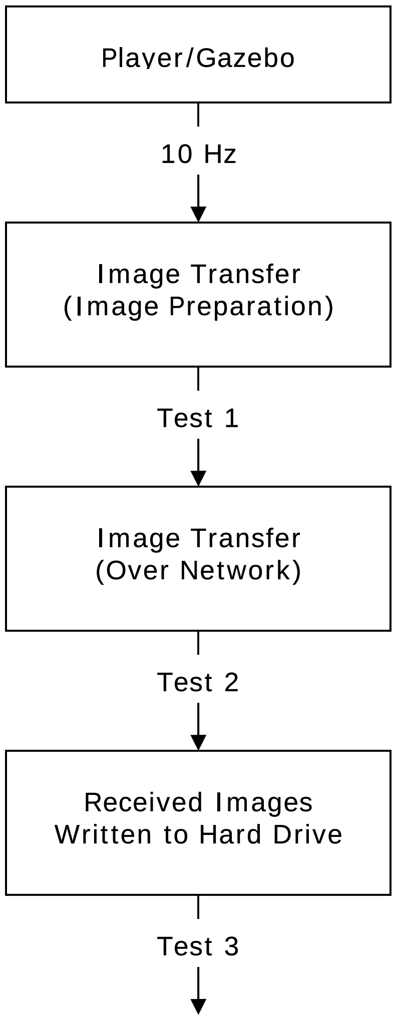
Player/Gazebo
10 Hz
Image Transfer (Image Preparation)
Test 1
Image Transfer (Over Network)
Test 2
Received Images Written to Hard Drive
Test 3

Figure 5: This figure provides tests run analysis points on the image transfer system. Test one involved only retrieval and compression of camera data for 1,000 images. Test two involved retrieval, compression, and transfer of 1,000 images. Test three involved retrieval, compression, transfer, and storage of 1,000 images to the hard drive of the computer running the Flash UI. All tests were performed on the Vanderbilt University wired campus network.

Table 1: Results from 1,000 image frame transfer over 20 trials.

	Test 1	Test 2	Test 3
Average	4.402	4.399	4.393
Standard Deviation	0.037	0.046	0.054
Decrease From Test 1	---------	0.06%	0.15%

Table 2: Test results for analysis of the null hypothesis.

Tests compared	1 vs. 2	2 vs. 3	1 vs. 3
TTEST t-value	0.1895	0.4092	0.6166
TTEST p-value	0.8517	0.6870	0.5448

Discussion

Multi-robot systems involve a great deal of human robot interaction difficulties (Adams, 2006). Specifically, operators have difficulties when sensory data is limited. This system ties into the solution for this particular problem, a better user interface.

Using this bridge system model, new interface designs can easily be tested as long as they comply with the Gamebots protocol (Wang and Balakirsky, 2007) and can connect using TCP/IP communication. This bridge system modularity and smooth interchangeability between USARSim and actual robots has been achieved, the primary goal of the work. The system fits into the larger design of a multi-robot system, aimed at running on each individual robot to provide control for operators.

The system performance on received image data is very efficient, as seen in the data in Figure 1. Evidence shows the software system loses less than 1% of the frame rate through network transfer, though this loss occurred over the campus wired local area network.

These tests also demonstrate the system reliability via each test during which all 1000 images were successfully transferred and were uncorrupt. Further considerations would be to test reliability over wireless networks, test for long term fidelity, and test for network interruption recovery.

Future Work

The software system currently requires additional network coding with the TCP/IP communication to handle connection interruption. Desirable behavior would be for the robot to cease all movement and wait for the user to reconnect as a client and reestablish new commands. The current state of the bridge software is not far from this objective, except that the robot control system terminates upon any disconnection after issuing a stop command, to prevent any damage to the hardware.

The image system requires improvement that will provide a higher frame rate to aid the human users. Proper integration of the VCC4 camera is required before further conclusion. Further tests with a functional camera on the physical hardware may reveal if the slow image feed was due to the Gazebo/Player interaction or the system code itself.

Other updates should include function behaviors to control additional future peripherals future such as grippers, ultrasonic sonar, laser range finders, and speakers. Thanks to Player's modular design (Gerkey et. al, 2001) these capabilities can be added easily with additional device proxies in the robot class.

Lastly, more advanced function behaviors and automation algorithms should be investigated. Ultimately, we would like to see development of team based automation, with image feeds being transmitted from remote robots directly to a human operator responsible for directing relief, rescue, or recon efforts during an emergency response.

Conclusion

This software bridge is still in its early stages, despite successful functioning in Gazebo simulation. There is room for improvement, including final work with physical hardware to ensure the software works as desired.

Designed to interact with the Flash UI in a similar manner to USARSim, the multithreaded bridge utilizes Player as the middleware system the UI and the physical robot. With the back end hidden from the user, we have the possibility of designing and testing control systems that can be easily tested and quickly utilized with real multi-robot systems.

This communication and control bridge system will continue to prove useful, even after we create autonomous control algorithms. As we look into automation, few changes will be required to test new self-controlled unmanned vehicles.

Acknowledgement

This project has been completed with support from the 2007 Vanderbilt Undergraduate Summer Research Program.

References

Adams, Julie A. (2006) "Supporting Supervision of Multiple Robots." The Journal of the Robotics Society of Japan. 24(5): 17-19.

Gerkey, B.P., Vaughan, R.T., Stoy, K., Howard, A., Sukhatme, G.S., and Matarić, M.J. (2001) “Most valuable player: A robot device server for distributed control.” IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1226-1231.

Humphrey, Curtis M., Henk, Christopher, Sewell, George, Williams, Brian W., and Adams, Julie A.

(2007) "Assessing the Scalability of a Multiple Robot Interface." Proceedings of the 2nd ACM/IEEE International Conference on Human-Robotic Interaction.

Koenig, Nathan and Howard, Andrew. (2004) "Design and use paradigms for Gazebo, an open-source multi-robot simulator." IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2149 -2154.

Player v2.0.4. Retrieved June, 2007. http://playerstage.sourceforge.net

USARSim v3.11. Retrieved July, 2007. http://sourceforge.net/projects/usarsim

Wang, Jijun and Balakirsky, Stephen. Retrieved July, 2007. "USARSim Manual v3.11." http://ftp.roedu.net/mirrors/sf.net/u/us/usarsim /USARsim-manual_3.1.1.pdf

Wang, Jijun, Lewis, Michael, Hughes, Steven, Koes, Mary, and Carpin, Stefano. (2005). "Validating USARSim for use in HRI Research." Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting, pp. 457 – 461.