PAPER

Development of a Flexible Command and Control Software Architecture for Marine Robotic Applications

AUTHORS

Brian S. Bingham
Department of Mechanical Engineering, University of Hawaii at Manoa

Jeffrey M. Walls
Department of Mechanical Engineering, University of Michigan

Ryan M. Eustice
Department of Naval Architecture and Marine Engineering, University of Michigan

ABSTRACT

This paper reports the implementation of a supervisory control framework and modular software architecture built around the lightweight communication and marshalling (LCM) publish/subscribe message passing system. In particular, we examine two diverse marine robotics applications using this modular system: (i) the development of an unmanned port security vehicle, a robotic surface platform to support first responders reacting to transportation security incidents in harbor environments, and (ii) the adaptation of a commercial off-the-shelf autonomous underwater vehicle (the Ocean-Server Iver2) for visual feature-based navigation. In both cases, the modular vehicle software infrastructures are based around the open-source LCM software library for low-latency, real-time message passing. To elucidate the real-world application of LCM in marine robotic systems, we present the software architecture of these two successful marine robotic applications and illustrate the capabilities and flexibilities of this approach to real-time marine robotics. We present benchmarking test results comparing the throughput of LCM with the Mission-Oriented Operating Suite, another robot software system popular in marine robotics. Experimental results demonstrate the capacity of the LCM framework to make large amounts of actionable information available to the operator and to allow for distributed supervisory control. We also provide a discussion of the qualitative tradeoffs involved in selecting software infrastructure for supervisory control.
Keywords: autonomous vehicles, maritime domain awareness, supervisory control, interprocess control

Introduction

The role of unmanned systems in the maritime domain continues to grow as robotic tools continue to mature and evolve, serving an increasingly broad set of users. The Department of Defense, the energy industry, and the scientific community have been early adopters of this technology, guiding the development of new capabilities to satisfy the needs of applications such as ocean mine countermeasures (Alt, 2003; Schnoor, 2003; Eickstedt & Schmidt, 2003; Rish et al., 2001), seafloor survey for oil and gas exploration (Chance et al., 2000; Vestgard et al., 1999), and studying deep-sea hydrothermal vents (Yoerger et al., 2007a, 2007b; Sohn, et al., 2008; Kunz, et al., 2009). More recently, autonomous vehicles have been used for disaster response such as responding to oil spills similar to the recent Deepwater Horizon incident (Camilli et al., 2010; Bhattacharya et al., 2011) and natural disasters such as hurricanes (Murphy et al., 2008, 2009).

The focus of this article is on the development and implementation of a flexible software architecture based around the lightweight communications and marshalling (LCM) message passing protocol. We report on the development of the command and control software developed for two marine robot applications: supervisory control of a maritime domain awareness (MDA) mission and the modification of an Ocean-Server Iver2 autonomous underwater vehicle (AUV) for real-time visual feature-based navigation. This software is based on the open-source LCM project initiated to support the Defense Advanced Research Projects Agency (DARPA) Urban Grand Challenge. The architecture we propose has distinct advantages over some of the available software tools for robotics, but it is not a unique solution. We describe a generic framework for this software along with benchmarking tests to compare the performance of this system with other robotics operating tools.

Background and Closely Related Work

Real-Time Command and Control

A fundamental aspect of robotic software is the use of modularity to cope with the complexity of real-time operation. For example, each hardware sensor typically has a software driver that manages the sensor communication and passes the data from each sensor to estimation, control, and planning software modules. Generically, the process of communicating between modules is called interprocess communication (IPC). The choice of IPC method is the key to the performance of the real-time robotics software.

The LCM system is a relatively new alternative for implementing IPC for real-time robotics in the marine environment (Huang et al., 2009, 2010). LCM was developed specifically for low-latency, high-throughput communication to support the MIT entrant in the DARPA Urban Grand Challenge (Leonard et al., 2008). The authors of LCM have compared their design with similar systems currently in use by the robotics community including the Player project, the CARMEN robotics package, the Joint Architecture for Unmanned Systems, the Robotics Operating System (ROS), Microsoft's Robotics Development Studio, and the Mission Oriented Operating Suite (MOOS) (Huang et al., 2009). LCM provides a set of software tools for message[1] passing between modules (processes or threads). These tools provide the following functions for the development of a real-time system:

- message type specification for language-independent data structures;
- marshalling (encoding and decoding) of LCM messages via software libraries for C/C++, Java, Python, MATLAB, and C#;
- scalable communication via user datagram protocol (UDP) multicast.

The design of LCM emphasizes the high-throughput, low-latency needs of distributed robotics applications and accomplishes this by providing the infrastructure that holds together the modules of a software solution rather than prescribing the structure of any of the processes or threads. The result is an extremely general software architecture that is amenable to the needs of a supervisory control scheme described below.

The LCM messages are transmitted across the network via multicast UDP packets. In this method, a process publishes the data on a particular channel, defined by a unique name. Any other process on the network can subscribe to the same channel and receive the UDP message. The marshalling capabilities encode and decode the structured data transmitted over the network. The fingerprinting functions of LCM guarantee that the messages are type-safe, i.e., that the communicating processes/threads agree exactly about how to interpret the messages. This design allows for transparent distribution of the system across a standard Ethernet or wireless network.

The MOOS software system is being applied to a growing number of marine robotic applications. In contrast to the LCM libraries, which provide a minimal message passing functionality, MOOS is a more complete set of communicating applications including both the libraries and executables for a real-time platform (Newman, 2003). Communication is done in a publish/subscribe model, referred to as a star topology where each client subscribes to a central database where all messages are stored. Message passing is done using human-readable text strings containing one or more name/value pairs via transmission control protocol (TCP). Typically processes and threads that interact with the database are derived from a common application. The central database can then control the message passing and execution of software developed for the particular platform. MOOS has been used for a variety of ocean robotics applications including AUVs and unmanned surface craft (Curcio et al., 2008). The MOOS software has more recently been extended to support cooperative control of multiple underwater platforms (Benjamin et al., 2010; Tye et al., 2009). One of the goals of this research is to qualitatively and quantitatively compare the MOOS system with an LCM implementation of real-time robotic command and control (Table 1).

Unmanned Vehicles for Port and Harbor Security

Unmanned marine vehicles (both surface vehicles and underwater vehicles) continue to evolve as important components of defense, industry, and scientific ocean operations. As these platforms have matured, they have found application in an increasingly wide variety of operational scenarios. The research presented here is a part of a burgeoning impetus to apply robotic technology to MDA.

One aspect of MDA that has seen considerable attention is the need for unmanned systems to assist in disaster

[1] A "message" is any digital data that can be encapsulated into a C-like data structure. For example, messages could be simple text, images, files, or binary data.

TABLE 1

Comparison of LCM and MOOS communication implementations.

	LCM	MOOS
Topology	Broadcast: Each process/thread sends message throughout the network.	Star: Centralized database. Each client allowed to publish and subscribe.
Transport	UDP multicast	TCP
Message types	User defined, type-safe binary	Text strings

response, including search and rescue. A number of commercial platforms (e.g., the iRobot Packbot and Foster-Miller Talon) are available for urban search and rescue (USAR). This technology has aided a number of important USAR incidents, the World Trade Center disaster being a particularly dramatic example (Casper & Murphy, 2003). Unmanned aerial vehicles (UAVs) have also been tasked with disaster response such as surveying structural failures like the Berkman Plaza II collapse in 2007 (Pratt et al., 2008). Similarly, UAVs have been applied to wilderness search and rescue (Goodrich, et al., 2008). These examples demonstrate the capability of unmanned systems for security applications using terrestrial and aerial applications, but this potential is yet to be realized by the maritime security community.

In the marine environment, researchers have leveraged the robotic technologies developed for defense for maritime event response. The Center for Robot-Assisted Search and Rescue has deployed both Unmanned Surface Vehicles (USVs) and UAVs for post-disaster port and littoral inspection after hurricanes Ike and Wilma (Steimle et al., 2009; Murphy et al., 2008, 2009).

Supervisory Control

Supervisory control explicitly includes the human operator in the feedback loop of an automated system. Thomas Sheridan coalesced much of the early thought on this paradigm, "supervisory control means that one or more human operators are intermittently programming and continually receiving information from a computer that itself closes an autonomous control loop through artificial effectors to the controlled process or task environment" (Sheridan, 1992).

The framework we have developed is inspired by Sheridan's emphasis on careful attention to the capabilities of the human and robotic components of such a system. Figure 1 illustrates the supervisory architecture where the human supervisor can interact with the system at a variety of levels. By building this flexibility into the software, we can provide users with an interface that allows them to simultaneously automate tasks and take direct control of the platform depending on the situation.

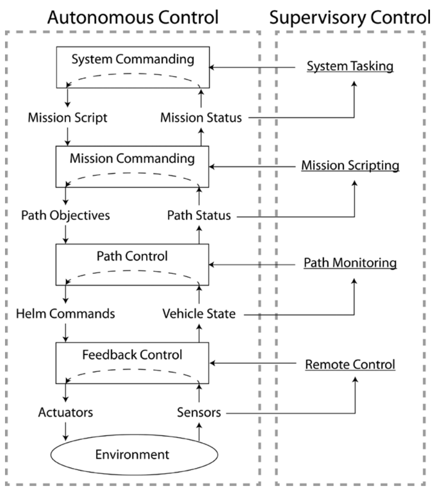
Case Study I: University of Hawaii Field Robotics Laboratory

We present the design of an unmanned system for port and harbor security as part of the Department of Homeland Security's (DHS) initiative to enhance MDA. At the Field Robotics Laboratory (FRL) at the University of Hawaii, we have designed, built, and tested the unmanned port security vessel (UPSV[2]) to investigate the unique challenges of the homeland security mission.

The UPSV concept is designed around the need for port resilience after a transportation security incident (TSI). A TSI can be the result of a natural disaster (e.g., hurricane, earthquake, or tsunami), environmental damage (e.g., chemical spill), or terrorist attack. As an example of the potential impact, U.S. West Coast ports support 8 million U.S. jobs as 14 million TEUs (20-foot equivalent units) are moved through these ports (Annual Report, 2009). Because of the economic and security risks associated with the impact of such an event, there

FIGURE 1

Conceptual flowchart of the automatic and supervisory control tasks for the UPSV platform. The tasks on the left are automated by the use of computational elements for real-time feedback. The tasks on the right represent the various ways the human supervisor may intervene and redirect the system.



[2]The authors' use of the acronym "UPSV" is not meant to add a new acronym to the long list of acronyms in the literature, but this acronym is used for brevity in this article.

is considerable interest in a variety of techniques to minimize the down time of port facilities affected by such an event. The UPSV is designed to provide information to decision makers that can help speed the process of bringing the port back to operation. In particular, the UPSV is capable of bathymetric mapping to identify obstructions and to perform acoustic and optical surveys of coastal infrastructure such as piers and pilings.

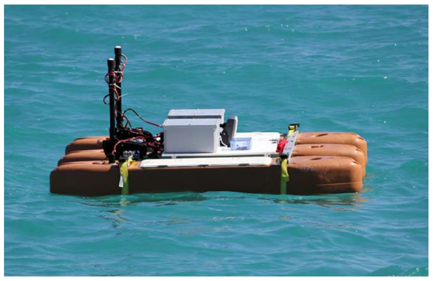
The FRL Vehicle Software (FVS) is a general purpose real-time supervisory command and control system for robotic development. This software is a derivative of the LCM-based software infrastructure developed by the Perceptual Robotics Laboratory (PeRL) at the University of Michigan (discussed in Case Study II: University of Michigan PeRL). The FVS system is specifically designed to support the supervisory control necessary to include the operator as an explicit component in the vehicle command and control.

Unmanned Port Security Vessel

Our initial prototype platform is shown in Figure 2. In working with the DHS first responders, we have discovered the following concepts that have led us toward the supervisory control framework for the UPSV.

FIGURE 2

Photograph of the first prototype UPSV vehicle designed for software and instrument evaluation.



- Port and harbor environments are exceedingly complex environments. Fully autonomous control is not suitable for such an environment because of the changing nature of the operation and the need of first responders to provide real-time commands.
- The decision making structure for homeland security during an incident in a port or harbor is distributed. Actionable information must be relayed to the decision makers in near real-time and the task of first responders is constantly changing, resulting in dynamic goals for a robotic platform.
- To successfully support homeland security's MDA, robotic tools must augment the multiagency teams. This requires close collaboration between first responders and the unmanned system.
- First responders need real-time access to all the data being collected as well as an interface to seamlessly retask the unmanned system without requiring the full attention of the operator.

One particularly important facet of these concepts is the need for unmanned surface systems to obey the "rules of the road" and avoid collisions. Current thinking suggests that unmanned marine vessels must comply with the International Regulations for Preventing Collisions at Sea (COLREGS) while on the surface (Showalter & Manley, 2009). A number of research projects have addressed this requirement by seeking to advance the technology to autonomously comply with such a mandate (Benjamin et al., 2006; Larson et al., 2006). The missions of the UPSV (explained below) are intended to take place immediately after an incident that would close a port, so the vehicle would be operating in scenarios where the United States Coast Guard (USCG) has complete control of surface traffic. However, there is significant interest from these users to maintain authority over any vessel, even if it had the ability to autonomously follow the rules of the road.

FRL Vehicle Software

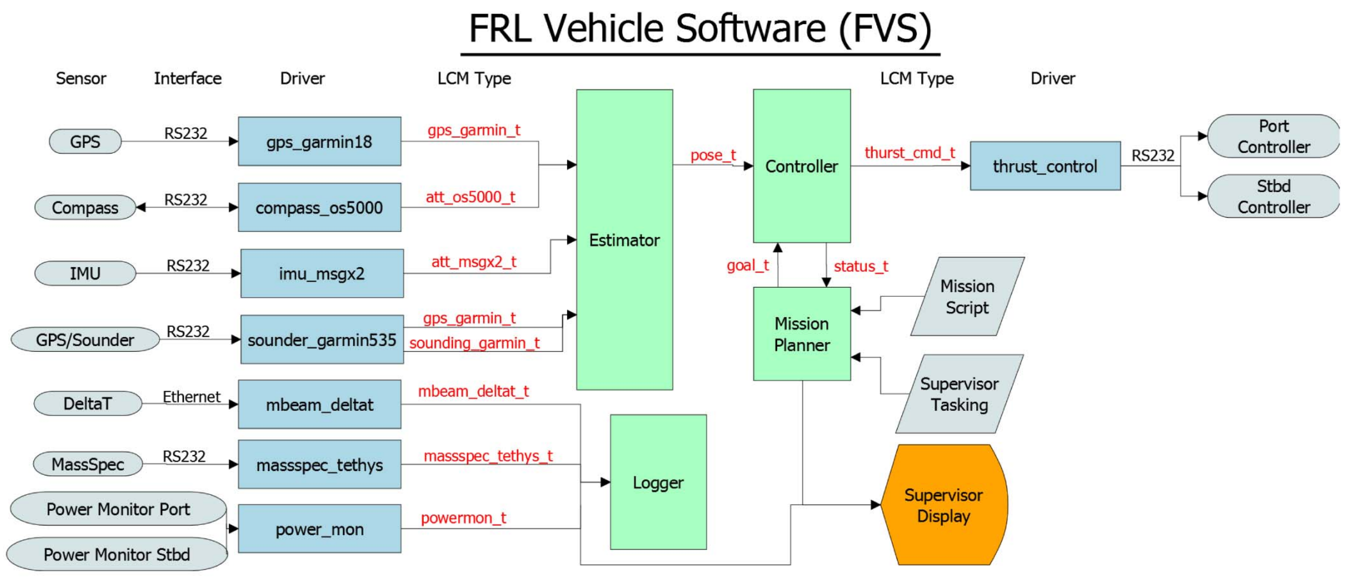
Figure 3 illustrates the FVS software design. The sensors are generically categorized as either navigation or payload sensors. The difference is that the navigation sensors are used in real-time by the onboard autonomous control to guide the vehicle state. The payload sensors provide the environmental measurements and are archived onboard and transmitted to the supervisor to provide feedback for intervention and tasking. All onboard software is running on a singleboard PC running the Ubuntu distribution of Linux.

Each sensor has a corresponding software driver (written in standard C), which maintains the communication interface to the vehicle. Each driver is a standalone process that delivers the sensor data to the software estimator and LCM logger via LCM messages. The estimator and controller (written in Python) are two threads of the same process. Using threads simplifies the asynchronous coordination so that the estimator can provide a fresh state estimate to the controller at a predetermined rate (typically 10 Hz). The controller provides the low-level feedback control that responds to the state estimate and generates the thrust commands for the port and starboard thrusters. These commands are then sent to another driver that manages the interface with the motor controllers.

To illustrate the function of the LCM messaging, two examples of

FIGURE 3

Flowchart of the FVS LCM-based software infrastructure for supervisory control.



LCM data type specification are shown in Figure 4. The LCM toolbox provides executables for automatically generating the necessary source code so that all the processes and threads (in C, C#, C++, Python, and Java) can make use of these custom data structures. The gps_rmc_t data type is for transmitting GPS position information from the driver to the estimator. The message is composed of a timestamp (in microseconds), latitude, longitude, and the speed-over-ground. Each of these numerical components is specified by an operating system, language, and processor-independent type specification (such as 64-bit integer or double). The encoding and decoding of the message is accomplished by the LCM generated software and is transparent to the FVS processes that actually manipulate the information. The pose_t data type is a simple container for a timestamp and a 12-element array of doubles that contain the estimated position, attitude, velocities, and attitude rates. This simple message is used extensively by the robot command and control software as an output of any of the estimation algorithms and input to any of the control algorithms. This example illustrates how simple it is to asynchronously pass arrays of data between various processes and threads written in different languages. Processes and threads can seamlessly be distributed on any machine throughout the network. This is of critical importance for supporting supervisory control since the human supervisor needs to interact directly with both the navigation and payload.

FIGURE 4

Two examples of LCM type definitions. The definition on the left is associated with the recommended minimum data (RMC) sentence from a GPS receiver. The definition on the right is associated with the estimated state broadcast from the estimator thread.

```
structgps_rmc_t
{
int64_t utime;   // Timestamp (us)

doublelat; // latitude (dec. deg.)
doublelon; // longitude (dec. deg.)
double sog; // speed-over-gnd (kts)
}
```

```
structpose_t
{
int64_t utime;    // Timestamp (us)
double state[12]; // Vehicle state
}
```

Case Study II: University of Michigan PeRL

The PeRL Software Library (PSL) is a general purpose robotics toolkit initially developed by the Perceptual Robotics Laboratory at the University of Michigan for use with Ocean-Server Technology, Inc., Iver2 AUVs. PSL is based around the LCM messaging paradigm (Huang et al., 2010) allowing for modular development and operation. Multiple processes can communicate by both publishing messages and assigning callback functions that execute upon receiving messages over a given LCM channel. A suite of useful tools are also included with LCM such as logging, playback, and real-time data visualization. These utilities ease the process of debugging existing software as well as writing new functions. This system lends itself to a clean organizational structure that can easily partition jobs such as hardware interfacing, state estimation and control tasks.

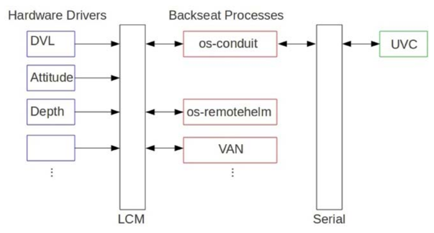
Major thrusts of current development within PeRL focus on perception-driven control and real-time visually augmented navigation (VAN) (Brown, Kim, & Eustice, 2009). Perception-driven control refers to a vehicle autonomously adapting its trajectory, either returning to a known landmark or continuing exploration, in order to optimally cover a survey area with minimal pose uncertainty. The VAN framework uses perceptual data from onboard cameras to correct drift inherent in dead-reckoned navigation. The following section provides additional detail through an example of a real-world implementation of the PSL architecture on an Iver2 AUV.

Iver2 Implementation

The basic system design is illustrated in Figure 5. PSL software handles all sensor processing and communication with a separate Ocean-Server vehicle controller called Underwater Vehicle Console (UVC). The modularity provided by LCM allows the user to launch any one of a variety of sensors or use different filtering schemes to process raw data, such as vehicle attitude or velocity, sent to the vehicle controller. Furthermore, PSL interfaces with the vehicle controller through a library of "backseat-driver" commands to execute perception-driven control (Figure 6).

FIGURE 5

Iver2 System Architecture illustrating information exchange onboard an Iver2 AUV: sensor-level information is passed via LCM toward backseat processes that in turn communicate serially with the low-level vehicle controller.



The Iver2 AUV is a commercial off-the-shelf underwater platform providing great flexibility in terms of hardware and software that can be added by the end user. Two Iver2 AUVs currently serve as real-world testbeds for simultaneous localization and mapping (SLAM) research within PeRL. The vehicles were chosen for their small size (easing transportation and launch/recovery constraints) while still providing capable performance. Each vehicle was integrated with additional navigation and perceptual sensors to perform SLAM as well as multi-AUV communications and operation. A 32-bit embedded PC104 CPU payload was also added for data logging and real-time control through the vehicle controller.

In the Iver2 software configuration, PSL accomplishes two main tasks: publishing sensor data and interfacing with the lower level vehicle controller.

FIGURE 6

PeRL Iver2 AUV.

Sensor Drivers

Independently running sensor drivers publish all sensor information over the network through LCM. Table 2 lists available sensors on the PeRL Iver2 AUVs. Other processes that have subscribed to the sensor data are then able to execute callback functions based upon sensor publish events.

Backseat Driver

All low-level vehicle control such as waypoint and depth tracking is carried out by the proprietary Ocean-Server software, UVC, on an embedded CPU. Additionally, UVC can process a large set of commands from a payload CPU. This architecture is designated a frontseat/backseat scheme where intelligent control can be relayed from the backseat to a frontseat processor handling dynamic low level control. Commands are formatted ASCII NMEA strings exchanged over a serial connection between the two CPUs. Table 3 lists some of the available backseat driver messages and their corresponding LCM structures. The backseat driver has the ability to send servo and primitive commands as well as higher level waypoint information. Additionally, the backseat driver can update the vehicle state as perceived by UVC. UVC also transmits acknowledgement of received commands with an error flag when appropriate. Finally, the backseat driver can query UVC for basic state information including mission status with a data request message.

All commands sent to UVC from PSL are sent serially through the os-conduit process running on the backseat CPU. In addition to UVC

TABLE 2

PSL available sensors.

Sensor	State Property	Update Rate (Hz)	LCM Structure	LCM Channel
Ocean-Server OS5000 Compass	Attitude	0.01-20	os_compass_t	OS_COMPASS
USGlobalSat EM-406a GPS	XYZ position	1.0	gpsd_t	GPSD
Prosilica GC1380H(C) Camera	gray/color image	1-5	prosilica_t	PROSILICA_M PROSILICA_C
Teledyne RDI 600kHz Explorer DVL	body velocity	7	pdi_pd5_t	RDI
KVH DSP-3000 single-axis FOG	Yaw rate	100	kvh_dsp3000_t	KVH
Desert Star SSP-1 300PSIG Digital Pressure Transducer	Depth	0.0625-4	dstar_ssp1_t	DSTAR
Microstrain 3D-GX1 AHRS	Attitude, body rotation rate	1.0-100	ms_gx1_t	MICROSTRAIN

commands, the backseat CPU can forward vehicle state information. The os-conduit process listens to the full LCM stream and processes sensor information and backseat driver commands by correctly packaging messages and sending them to the serial bus.

A higher level process, os-remotehelm, tracks mission status and publishes LCM backseat driver commands that os-conduit later relays to UVC. This process performs tasks such as loading, starting and aborting missions. Furthermore, os-remotehelm is able to control other devices via LCM, such as cameras, which it only powers between designated mission waypoints. Through this process, we have begun to implement perception driven control. Future work will adaptively alter waypoint parameters in order to aid optimal exploration.

Visually Augmented Navigation (VAN)

The real-time VAN library is currently an active area of development within the PSL. VAN is a real-time feature-based navigation and mapping framework, which augments onboard dead-reckoned navigation with constraints derived from visual data. Pairwise navigation constraints between overlapping images are fused within a pose-graph SLAM framework. This VAN methodology improves upon traditional navigation techniques by bounding pose uncertainty and has the ability to scale to large environments capturing thousands of image keyframes. For a detailed description of the VAN algorithm, the reader is referred to Eustice (2008) and Kim and Eustice (2009).

Figure 7 depicts algorithmic and communication flow in the VAN framework. VAN is a multithreaded process exploiting LCM as well as shared memory for communication when necessary for reduced bandwidth or increased speed. For example, static information, such as the camera calibration matrix, is required across all threads and therefore uses shared memory. Threads also communicate via LCM to share dynamic information such as an image feature vector. The use of LCM allows the refined navigation estimate to be easily passed to other processes, such as the backseat driver detailed in the previous section.

Within VAN, the feature thread first extracts feature keypoints from each image. Currently, fast feature extraction (up to 10 Hz) is achieved by

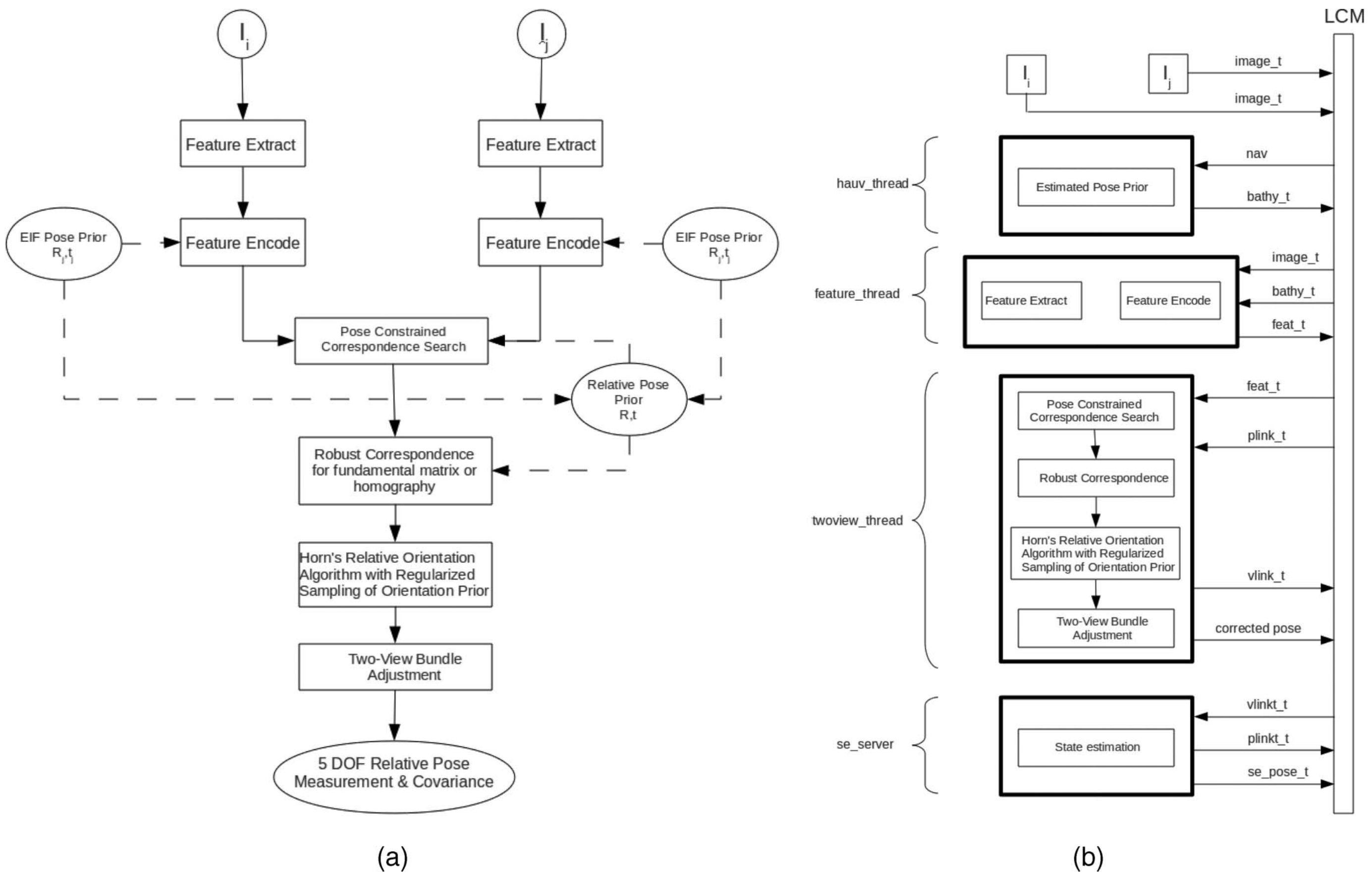
TABLE 3

Backseat driver commands.

UVC ASCII Command	Action	LCM Structure
$ACK	command acknowledgment	ack_t
$OMP	primitive control mode	omp_t
$OMS	servo control mode	oms_t
$OMP	primitive control mode	omp_t
$OMW	mission control mode	omw t
$OMSTART/STOP/LOAD	mission start/stop/load	omstart/stop/load_t
$OSD	data request	osd_t
$OSI	state information	osi_t
$OJW	jump to waypoint	ojw_t

FIGURE 7

(a) Van systems-level approach to image registration. (b) VAN software architecture. Overview of VAN framework illustrating algorithmic and communication flow, solving for the relative pose between the camera pair corresponding to image I_i and I_j. Bold boxes denote each thread while inner boxes designate sub processes within each thread. Arrows depict the direction of communication with the structure name to the LCM stream. SIFT features detectors are used for extraction. Prior knowledge of relative-pose is derived from the state estimate, which is then used in the two-view registration framework. The final output from this framework is an optimal 5-DOF camera constraint.



I_i
I_j
Feature Extract
Feature Extract
EIF Pose Prior R_i,t_i
Feature Encode
Feature Encode
EIF Pose Prior R_j,t_j
Pose Constrained Correspondence Search
Relative Pose Prior R,t
Robust Correspondence for fundamental matrix or homography
Horn's Relative Orientation Algorithm with Regularized Sampling of Orientation Prior
Two-View Bundle Adjustment
5 DOF Relative Pose Measurement & Covariance
(a)
LCM
image_t
image_t
hauv_thread
Estimated Pose Prior
nav
bathy_t
feature_thread
Feature Extract
Feature Encode
image_t
bathy_t
feat_t
twoview_thread
Pose Constrained Correspondence Search
Robust Correspondence
Horn's Relative Orientation Algorithm with Regularized Sampling of Orientation Prior
Two-View Bundle Adjustment
feat_t
plink_t
vlink_t
corrected pose
se_server
State estimation
vlinkt_t
plinkt_t
se_pose_t
(b)

using a feature detector running on a Graphics Processing Unit (GPU) (Wu, 2007). The hovering AUV (HAUV) thread is then responsible for assigning a camera pose prior to the time of each image event. This thread also prepares bathymetric information from Doppler Velocity Logger (DVL) range observations that is later used for estimating scene depth in the camera frame. Our state estimation engine is then able to hypothesize potential matching pairs. The final task of the image registration engine is to recover an optimal estimate of relative poses between corresponding camera pairs. The two-view thread first performs a pose-constrained correspondence search (Eustice, 2008), which rejects geometrically inconsistent feature matches based upon the uncertainty in the relative pose prior. A robust correspondence method rejects outliers from the putative set of feature correspondences and refines the initial relative pose estimate for bundle adjustment. The two-view thread then minimizes a two-view bundle adjustment cost function composed of the sum squared reprojection error in order to accomplish this.

VAN constraints are fused under a pose-graph SLAM framework, referred to as the state estimation server. This server communicates via LCM with the state estimation engine. Currently, we use an open-source implementation of iSAM (Kaess et al., 2008; Kaess & Dellaert, 2009), although another estimation framework could be used. This state estimation server also communicates with the image registration engine by providing hypothesized candidate overlapping image pairs based upon an estimated overlap between pairwise camera nodes in the pose-graph.

Real-time VAN performance is currently being field-tested for ship hull inspection applications using the Bluefin HULS3 HAUV. Preliminary real-world experiments have shown the ability of this software to handle high bandwidth data and correct poor dead-reckoned navigation by matching visual features in the environment in real-time running on standard laptop hardware. LCM has helped to provide a basic set of functions and tools leading to a successful real-time VAN implementation.

Results: Software Benchmarking

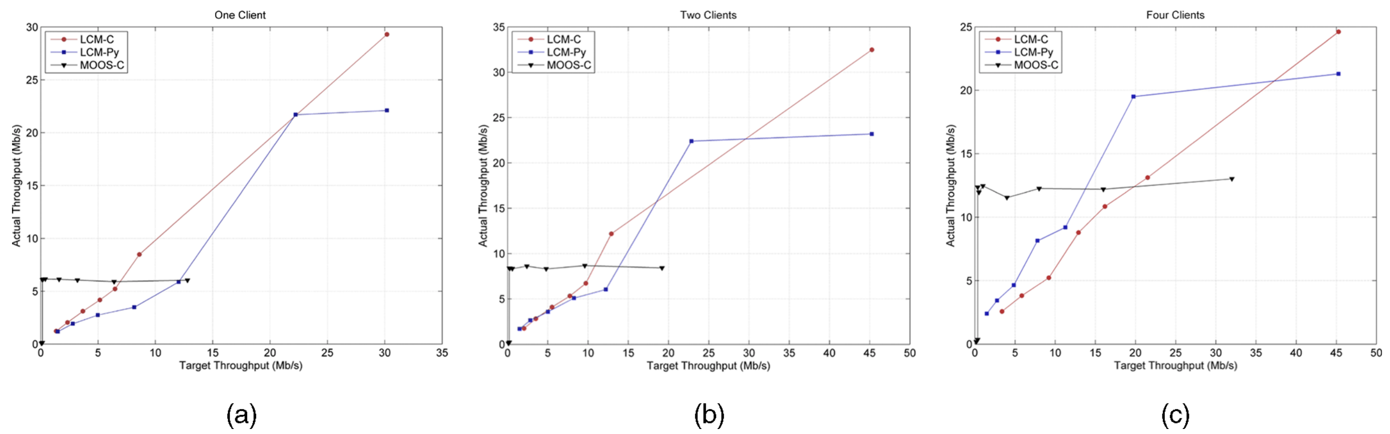
For supervisory control, the software system must be scalable to allow large amounts of data to be transmitted through the system, which includes the human supervisor. For example, in addition to updates on vehicle state, the UPSV operator needs to review chemical data (from the mass spectrometer) and bathymetry data (from the multibeam sonar) to provide high-level intervention and tasking of the vessel while under operation. The authors of LCM have characterized the bandwidth performance of the LCM software by comparing the performance of an LCM system (in C and in Java) with other robotics software packages such as IPC, Player, and ROS (Huang et al., 2009, 2010). We have extended this characterization to include a Python implementation of LCM (because we make extensive use of Python in the FVS software) and to compare the performance with the MOOS system, which is currently in use in a variety of marine robotic applications.

The experimental procedure is similar to that discussed in the original LCM comparisons by the authors of LCM where fixed size messages were transmitted from a source node to one, two, or four clients simultaneously. We followed the same protocol as reported previously (Huang et al., 2009) so that our analysis would be consistent with their tests and provide new comparisons. The results are shown in Figure 8. For each data point in the figures 100 MB of data is split into 800 byte messages at a fixed rate. The 800 byte message is of similar size to a single multibeam scan from the UPSV Imagenex sonar. All the tests were run on an Intel Core 2 Duo CPU running Ubuntu 10.04 with 2 Gb of RAM. Three different implementations were run for comparison: (1) the LCM sending and receiving nodes written in standard C, (2) the LCM sending and receiving nodes written in Python, and (3) the MOOS clients written in C++.

The results in Figure 8 illustrate some important conclusions about the performance of these three systems. First, performance of the LCM implementations in C and Python were comparable. The maximum throughput for these implementations showed that the Python implementation achieved only slightly less throughput than the C implementation: 29.3 MB/s for C and 22.1 MB/s for Python. The MOOS implementation is qualitatively and quantitatively different. Because MOOS is a full-operating suite of software tools the update rate of the source node was specified indirectly through MOOS at a fixed rate. Setting this update rate above 90 Hz (which equated to 0.144 MB/s for one client) resulted

FIGURE 8

(a) Results for one client. (b) Results for two clients. (c) Results for four clients. Message passing throughput test results with one, two, and four clients. In each comparison, three implementations are illustrated as indicated in the legend. LCM-C is a C implementation of LCM. LCM-Py is a Python implementation of LCM. MOOS-C is a C implementation of MOOS.



in MOOS running at its maximum update rate. This resulted in a fixed, maximum throughput for any target throughput above this threshold. This is illustrated in the horizontal portions of the throughput curves in Figure 8. For the single source, single client case the throughput achievable with MOOS was found to be roughly 6 MB/s, or about 27% of the throughput using the Python implementation of LCM under similar circumstances.

Discussion

The quantitative results presented above demonstrate the ability of the LCM-based software architecture to move large amounts of data throughout a distributed supervisory control system. However, there are qualitative reasons for our choice of using LCM compared with other tools available from the robotics community. One important consideration is that LCM is based on a common set of libraries that encapsulate the important task of message passing. This is in contrast to many of the other robotics toolkits that provide a full infrastructure by including libraries, pre-defined data types and prescribed event loops. The developers of LCM have taken a minimalistic approach, they have termed an "a la carte" system, that serves the needs of the UPSV project because it is scalable, affording the ability to simultaneously develop low-level feedback control and to provide high-level supervisory interfaces with the same set of tools.

Another important design aspect is the choice of binary message passing rather than using plain text or a text-based protocol such as XML. The obvious tradeoff is between the bandwidth efficiency of binary messages and the readability of text-based communication. LCM includes both an API for developers and a set of debugging, monitoring and playback tools that successfully manage the additional complexity of binary messages. Our experience with LCM binary messages has been seamless and the added functionality of passing rich, user-defined data structures has eased our development tremendously.

Lastly, a key aspect of the LCM design is the choice of a UDP multicast for communication that allows all clients to see all messages rather than including a centralized hub to mediate communication between clients. This design choice emphasizes the need for low-latency communication, which is critical for real-time implementation at the expense of dropped messages. (In the two applications discussed above, the number of dropped messages was near zero during field applications.) This also eliminates the need for the added complexity of a centralized hub for communication. Our experience with this feature is very positive as demonstrated by the data presented above. The data presented above suggests that the minimalistic approach taken by the LCM provides for a leaner software implementation with a greater capability for moving messages through the network.

Conclusions

This paper has described the development of a distributed, LCM-based software architecture for robotic systems. This architecture leverages several advantages of LCM for real-time low-latency robotic command and control. In order to emphasize the advantages of a modular architecture, the successful implementation of this software design for both an UPSV and within the PSL were reported.

We first presented the software infrastructure for supervisory control of an UPSV to augment MDA for homeland security operation. Homeland security operations present new challenges to unmanned marine systems because of the complexity of both the operating environment (ports and harbors) and the mission, which can be multifaceted because of the distributed nature of decision making in a multiagency incident response. We presented a supervisory control framework that provides first responders with a tool to increase their situational awareness during the response to a TSI. We argue that supervisory control balances the need for extending the abilities of first responders with their need for high level intervention and tasking of assets during a response.

The PSL used for real-time visual feature-based navigation was also presented. An overview of the PSL has demonstrated the ability of the LCM framework to promote a clean, modular architecture. Furthermore, a review of the backseat driver and VAN implementation has shown that LCM-based software can easily interface with both existing software and handle large bandwidth data in real time.

The quantitative results of the study illustrate the performance of the LCM framework for implementing such a command and control system. We have extended the comparison of LCM with other robotic control software by comparing it to the MOOS system, which is common in marine robotics applications. We have shown that the LCM framework can provide data throughput on the order of 25 MB/s, which is more than sufficient to support real-time feedback while providing the human supervisor with actionable remote data and the interface to interact with the

UPSV for intervention and reactive tasking.

Acknowledgment

The work on the UPSV was supported in part by a grant from the Department of Homeland Security Science and Technology Directorate (2009-ST-061-MD001); the work on PSL was supported in part by grants from the National Science Foundation (IIS-0746455) and through a grant from the Office of Naval Research (N00014-07-1-0791).

Lead Author:

Brian S. Bingham
Department of Mechanical Engineering, University of Hawaii at Manoa,
2540 Dole Street,
Holmes Hall 302, Honolulu, HI
Email: bsb@hawaii.edu

References

Benjamin, M., Curcio, J., & Newman, P. 2006. Navigation of unmanned marine vehicles in accordance with the rules of the road. In: Proceedings 2006 IEEE International Conference on Robotics and Automation. 3581-7. doi: 10.1109/ROBOT.2006.1642249.

Benjamin, M.R., Schmidt, H., Newman, P.M., & Leonard, J.J. 2010. Nested autonomy for unmanned marine vehicles with MOOS-IvP. J Field Robot. 26(6):834-74. doi: 10.1002/rob.20370.

Bhattacharya, S., Heidarsson, H.K., Sukhatme, G.S., & Kumar, V. 2011. Cooperative control of autonomous surface vehicles for oil skimming and cleanup. In: Proceedings of the IEEE International Conference on Robotics and Automation.

Brown, H.C., Kim, A., & Eustice, R.M. 2009. An overview of autonomous underwater vehicle research and testbed at PeRL. Mar Technol Soc J. 43(2):33-47. doi: 10.4031/MTSJ.43.2.4.

Camilli, R., Reddy, C.M., Yoerger, D.R., Van Mooy, B.A.S., Jakuba, M.V., Kinsey, J.C., … Maloney, J.V. 2010. Tracking hydrocarbon plume transport and biodegradation at Deepwater Horizon. Science. 330(6001):201-4.

Casper, J., & Murphy, R. 2003. Human-robot interactions during the robot-assisted urban search and rescue response at the World Trade Center. IEEE Trans Syst Man Cybern B Cybern. 33(3):367-85.

Chance, T., Kleiner, A., & Northcutt, J. 2000. The autonomous underwater vehicle (AUV): A cost-effective alternative to deep-towed technology. Integr Coastal Zone Manage. 2(7):65-9.

Curcio, J., Schneider, T., Benjamin, M., & Patrikalakis, A. 2008. Autonomous surface craft provide flexibility to remote adaptive oceanographic sampling and modeling. In: Proc MTS/IEEE OCEANS Conf. 1-7. IEEE. doi: 10.1109/OCEANS.2008.5152078.

Eickstedt, D., & Schmidt, H. 2003. A low-frequency sonar for sensor-adaptive, multistatic, detection and classification of underwater targets with AUVs. In: Proc MTS/IEEE OCEANS Conf. 3:1440-7.

Eustice, R.M. 2008. Toward real-time visually augmented navigation for autonomous search and inspection of ship hulls and port facilities. Intl. Symposium on Technology and the Mine Problem. Mine Warfare Association (MINWARA).

Eustice, R.M., Brown, H.C., & Kim, A. 2008. An overview of AUV algorithms research and testbed at the University of Michigan. In: IEEE/OES Autonomous Underwater Vehicles Conference. 1-9. doi: 10.1109/AUV.2008.5290531.

Goodrich, M.A., Morse, B.S., Gerhardt, D., Cooper, J.L., Quigley, M., Adams, J.A., & Humphrey, C. 2008. Supporting wilderness search and rescue using a camera-equipped mini UAV. J Field Robot. 25:89-110. doi: 10.1002/rob.20226.

Huang, A., Olson, E., & Moore, D. 2009. Lightweight Communications and Marshalling for Low-Latency Interprocess Communication. Computer Science and Artificial Intelligence Laboratory Technical Report. http://dspace.mit.edu/bitstream/handle/1721.1/46708/MIT-CSAIL-TR-2009-041.pdf.

Huang, A.S., Olson, E., & Moore, D.C. 2010. LCM: Lightweight communications and marshalling. In: Proc. Int. Conf. on Intelligent Robots and Systems (ROS). IEEE.

Kaess, M., & Dellaert, F. 2009. Covariance recovery from a square root information matrix for data association. J Robot Autonom Syst. 57:1198-210.

Kaess, M., Ranganathan, A., & Dellaert, F. 2008. iSAM: Incremental smoothing and mapping. IEEE Trans. Robot. 24:1365-78.

Kim, A., & Eustice, R. M. 2009. Pose-graph visual SLAM with geometric model selection for autonomous underwater ship hull inspection. In: Proc IEEE/RSJ Int Conf Intell Robots Syst. 1559-65.

Kunz, C., Murphy, C., Singh, H., Willis, C., Sohn, R., Singh, S., … Bailey, J. 2009. Toward extraplanetary under-ice exploration: robotic steps in the Arctic. J Field Robot, 26(4):411-29. doi: 10.1002/rob.20288.

Larson, J., Bruch, M., & Ebken, J. 2006. Autonomous navigation and obstacle avoidance for unmanned surface vehicles. In: Proceedings of the SPIE: Unmanned Systems Technology VIII. 6230:17-20. doi: 10.1117/12.663798.

Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., … Williams, J. 2008. A perception driven autonomous urban vehicle. J Field Robot. 25(10):727-74.

Murphy, R.R., Steimle, E., Griffin, C., Cullins, C., Hall, M., & Pratt, K. 2008. Cooperative use of unmanned sea surface and micro aerial vehicles at Hurricane Wilma. J Field Robot. 25:164-80. doi: 10.1002/rob.20235.

Murphy, R.R., Steimle, E., Hall, M., Lindemuth, M., Trejo, D., Hurlebaus, S., …

Slocum, D. 2009. Robot-assisted bridge inspection after Hurricane Ike. 2009 IEEE International Workshop on Safety, Security Rescue Robotics (SSRR). 1-5. doi: 10.1109/SSRR.2009.5424144.

Newman, P. 2003. A mission oriented operating suite. Technical Report OE2007-07. MIT Department of Ocean Engineering.

Pacific Maritime Association. 2009. Annual Report. Pacific Maritime Association. http://www.pmanet.org/pubs/AnnualReports/2009/PMA%202009%20Annual%20Report.pdf.

Pratt, K., Murphy, R., Burke, J., Craighead, J., Griffin, C., & Stover, S. 2008. Use of tethered small unmanned aerial system at Berkman Plaza II collapse. IEEE International Workshop on Safety, Security and Rescue Robotics. 134-9. doi: 10.1109/SSRR.2008.4745890.

Rish, J., Willcox, S., Grieve, R., Montieth, I., & Vaganay, J. 2001. Operational testing of the Battlespace Preparation AUV in the shallow water regime. In: Proc MTS/IEEE OCEANS Conf. 1:123-9.

Schnoor, R.T. 2003. Modularized unmanned vehicle packages for the littoral combat ship mine countermeasures missions. In: Proc MTS/IEEE OCEANS Conf. 3:1437-9. doi: 10.1109/OCEANS.2003.1282587.

Sheridan, T.B. 1992. Telerobotics, Automation, and Human Supervisory Control. Cambridge, MA: MIT Press. 1.

Showalter, S., & Manley, J. 2009. Legal and engineering challenges to widespread adoption of unmanned maritime vehicles. In: Proc MTS/IEEE OCEANS Conf. 1-5.

Sohn, R., Willis, C., Humphris, S., Shank, T., Singh, H., Edmonds, H.N., … Soule, A. 2008. Explosive volcanism on the ultraslow-spreading Gakkel Ridge, Arctic Ocean. Nature. 453(7199):1236-8. doi: 10.1038/nature07075.

Steimle, E., Murphy, R., Lindemuth, M., & Hall, M. 2009. Unmanned marine vehicle use at Hurricanes Wilma and Ike. In: Proc MTS/IEEE OCEANS Conf. 1-6.

Tye, C., Kinney, M., Frenzel, J., O'Rourke, M., & Edwards, D. 2009. A MOOS module for autonomous underwater vehicle fleet control. In: Proc MTS/IEEE OCEANS Conf. 1-6.

Vestgard, K., Storkersen, N., & Sortland, J. 1999. Seabed surveying with Hugin AUV. In: Proceedings of the 11th International Symposium on Unmanned Untethered Submersible Technology. Durham, NH: Autonomous Undersea Systems Institute.

von Alt, C. 2003. REMUS 100 transportable mine countermeasure package. In: Proc MTS/IEEE OCEANS Conf. 4:1925-30.

Wu, C. 2007. SiftGPU: A GPU implementation of Scale Invariant Feature Transform (SIFT). Retrieved from http://cs.unc.edu/~ccwu/siftgpu (accessed 8 April 2011).

Yoerger, D.R., Bradley, A., Jakuba, M., German, C., Shank, T., & Tivey, M. 2007a. Autonomous and remotely operated vehicle technology for hydrothermal vent discovery, exploration and sampling. Oceanography. 20(1):152-61.

Yoerger, D.R., Jakuba, M., Bradley, A.M., & Bingham, B. 2007b. Techniques for deep sea near bottom survey using an autonomous underwater vehicle. Int J Robot Res. 26(1): 41-54. doi: 10.1177/0278364907073773.