Universiteit van Amsterdam

Project Report: SI4MS: Sensor Intelligence for Mobility Systems

IAS technical report IAS-UVA-12-01

A survey of the architecture of the communication library LCM for the monitoring and control of autonomous mobile robots

Arnoud Visser
Intelligent Systems Laboratory Amsterdam,
Universiteit van Amsterdam
The Netherlands

Abtract: This report gives a short overview of the potential of using the Lightweight Communication and Marshalling (LCM) framework for an autonomous mobile robot application.

Keywords: Robotic Architecture, Connecting Components.

IAS
intelligent autonomous systems

Contents

Intelligent Autonomous Systems
Informatics Institute, Faculty of Science
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
The Netherlands

Tel (fax): +31 20 525 7461 (7490)
`http://www.science.uva.nl/research/isla/`

Corresponding author:

A. Visser
tel: +31 20 525 7532
`A.Visser@uva.nl`
`www.science.uva.nl/~arnoud`

1 Introduction

The objective of the Dutch innovation project SI4MS (Sensor Intelligence for Mobility Systems) is to go beyond the state of the art and to combine research on architecture and deterministic behavior of heterogeneous, large scale and dynamic sensor systems for mobile autonomous systems [10]. The first delivery of the architecture project [9] contained an overview of several system models for data fusion, such as for instance the *JDL Model* [2, 21] and the *4D/RCS System Model* [3].

One of the information and communication architectures which is not mentioned in this overview is the Lightweight Communications and Marshalling (LCM) approach [6, 8], although communication architectures is an important aspect of the SI4MS project [17]. LCM allows to build a system as a number of distributed processes, which communicate with low-latency messages without the need for a central communications "hub". LCM is especially targeted at robotics and intelligent vehicles. The following architectures are among others mentioned as related work for this LCM approach: the Robotics Operating System (ROS) [18] and the Joint Architecture for Unmanned Systems (JAUS) [23].

2 Applications of LCM

The LCM system is applied on several occasions which are relevant for the SI4MS project. One is the 2007 DARPA Urban Challenge. In this challenge autonomous vehicles had to drive 90 kilometer through an urban environment in the presence of other vehicles. The LCM system worked as communication backbone for two teams [13, 16]. Both teams published several datasets in LCM-format. The team from Michigan published [16] their experiments at the Ford Campus and downtown Dearborn, MI.

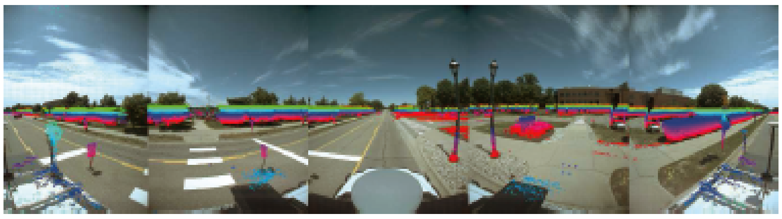


Figure 1: The perspective view of the Velodyne lidar range data, color-coded by height above the estimated ground, projected into the corresponding image from the Ladybug3 cameras. Courtesy [16].

An example of the sensor fusion that is required for the DARPA Urban Challenge is illustrated in Fig. 1. All of the perceptual and navigation sensors are fixed on a different location on the vehicle and are related to each other by static coordinate transformations. These rigid-body transformations, which allow the re-projection of any point from one coordinate frame to the other, were calculated for each sensor. The images of the omnidirectional LadyBug3 camera are spherical distorted, yet this distortion can be corrected with a calibration matrix. The 3D scans from the Velodyne laser scanner is corrected for the vehicle movement based on the measurements of the inertia sensor. The result of all those transformations and corrections is displayed in Fig. 1, where the obstacles to be avoided are clearly indicated.

The team from MIT published the dataset from the Urban Challenge Finals (all three missions) [7]. An example of the integration of all sensor streams is illustrated in the screenshot from their logviewer (Figure 2). A unique aspect of the MIT approach is the concurrent processing of laser range and radar measurements to track moving obstacles. Both systems were tuned to have a low false-positive rate, the output of the obstacle tracker was the union of the two subsystems. The system architecture of the MIT team [13], shows that the low level sensor data is processed in a number of perception modules, each dedicated to certain concepts (lanes, moving vehicles, obstacles, hazards). The information is collected in a drivability map, which is used by the motion planner. The complete set of distributed asynchronous software modules on top of the LCM library consists of approximately 140000 lines of source code. The LCM library was able during the final race to carry 8 MB/s on the primary network and 20 Mb/s on the secondary network. The LCM library was not only used for data, but also to make asynchronous rendering requests over the network, the LCGL library, to easily visualize the data interpretations.

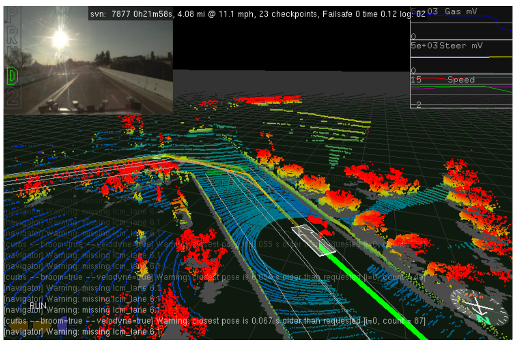

```
svn:  7877 0h21m58s, 4.08 mi @ 11.1 mph, 23 checkpoints, Failsafe 0 time 0.12 log: 02
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[curbs --broom=true --velodyne=true] Warning: closest pose is 0.055 s older than requested [i=0, count = [illegible]]
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[curbs --broom=true --velodyne=true] Warning: closest pose is 0.054 s older than requested [i=0, count = [illegible]]
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[navigator] Warning: missing lcm_lane 6.1
[curbs --broom=true --velodyne=true] Warning: closest pose is 0.067 s older than requested [i=0, count = 87]
[navigator] Warning: missing lcm_lane 6.1
```

Figure 2: A screenshot of the MIT logviewer, which displays the vehicle in God view, including the Velodyne lidar range data, several SICK laserscans, one of the camera images and planning information. Courtesy [7].

Another application of the LCM library was the team of the University of Michigan, which won the MAGIC 2010 competition [15]. The goal of this competition is to find as fast as possible a number of static and moving objects of interest (which could be an intruder or an explosive device). During this competition two operators controlled 14 robots connected via a long-distance radio network (900 MHz) capable of a bandwidth of 115.2 Kbps [19]. On short-distance the robots could start an 802.11g/n mesh network based on high-power wifi-routers (20 Mbps bandwidth). The different components on board of the robots and the operator control unit communicated via LCM messages [5]. The operator control unit (see Fig. 3) allows to monitor the fleet of robots and to give directions which adjust the automatic behavior of the robots.

The application of the LCM library in a multi-robot application shows how information can be shared over a mesh network of connected vehicles, which is one of the aspects studied in the SI4MS project. Yet, in this application the network consists of nodes with the same capabilities and sensor suite, while in a general traffic management case [11] an additional standardization is needed to negotiate communication protocols and the interpretation of information [17].

The survey will continue with a more detail description of the LCM framework and the analysis of the architecture of the applications build on the LCM framework against the SI4MS High Level Architecture. This is followed with a description of the experience to couple this framework to the USARSim environment [4].

3 The LCM framework

The Lightweight Communication and Marshalling (LCM) framework, as described in [8], consists of several parts. At the core LCM uses UDP multicast as transport layer. Most other approaches (except for JAUS [23]) use a "reliable" transport protocol like TCP, although this results in

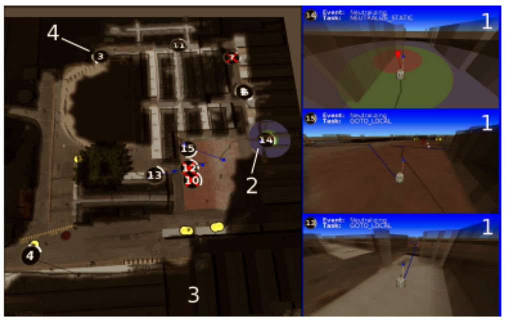


Figure 3: A screenshot of the SAGE interface of the University of Michigan team during the MAGIC competition. On the left a rasterized global map [22] in bird's eye view is shown, with the location of the robots spread over the environment (indicated with marker 4). On the right the local maps of three robots are displayed, which are automatically selected based on the occurrence of important events (indicated with marker 1). Courtesy [15].

additional latency. TCP includes mechanism for loss detection and packet retransmission. These mechanisms delay future messages, while for a robot platform the latest updates should have the highest priority. So a system working under real-time constraints, like a robot, may prefer to lose packages in favor of not delaying packages in the queue. Inside the LCM library there is also a possibility use a TCP multicast service, although this option is less well documented. In one occasion (a custom-built quadrotor helicopter [1]) experiments were performed to compare an UDP tunnel with additional forward error correcting (FEC) versus a TCP tunnel with the inherited correction mechanisms; this comparison was decided in favor of an UDP tunnel with FEC.

UDP Multicast is a protocol which is designed to distribute traffic over complicated networks. A parameter to control this is time-to-live (TTL) parameter, which controls the scope of the broadcasts. For instance, a TTL of 255 means that the messages are broadcasted over the whole internet. You can restrict the broadcast to your institute by using a value of 32. In the described applications, the broadcast is limited to the localhost (TTL=0) or the same subnet (TTL=1). Messages are broadcasted to a specific *group address*. The groups are open, no registration is needed to send information to the group address. Group addresses have the same format at as regular IP addresses, only a specific part of the IP address space is reserved for multicast applications: 224.0.0.255 to 239.255.255.255. When the broadcasts are limited to subnets conflicts between group addresses can easily be managed. The LCM library uses as default address 239.255.76.67, which is inside the local-scope address space[1].

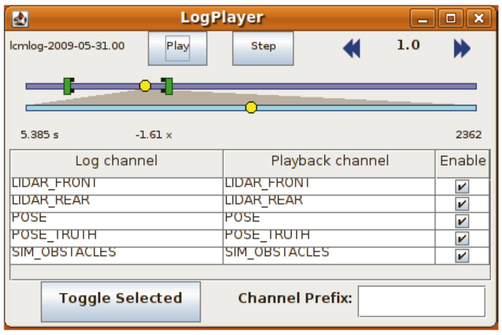
Since LCM uses a multicast mechanism, it does not require a centralized hub for either relaying messages or for 'match making'. There is also no need for a publisher to keep track of its subscribers, which can be error-prone if subscribers periodically start and stop (either by design or due to application errors). The April toolbox has mechanisms to restart modules via the LCM interface.

The next functionality of the LCM framework is marshalling. Marshalling refers to the encoding and decoding of structured data into an opaque binary stream that can be transmitted over a network. The LCM library contains a tool (`lcm-gen`) which automatically generates code with functions for marshalling and unmarshalling of each user-defined data type in each supported language (each specified by a different command-line option). Supported languages are C, C++, Java, Python and C#. The marshalling code generated by `lcm-gen` automatically ensures that the sender and receiver agree on the format of the message. The type checking

[1]`http://www.iana.org/assignments/multicast-addresses/`

of LCM types is accomplished by prepending each LCM message with a fingerprint derived from the type definition. The fingerprint is a hash of the member variable names and types. This mechanism ensures that the declarations were identical when the sender and receiver were compiled. When a system is in active development, the data types themselves can be in flux: this run-time check detects these sorts of issues.

Another aspect of the LCM framework is user-defined type specification. Processes that wish to communicate using LCM must agree in advance on the compound data types that will be used to exchange data. LCM defines a formal type specification language that describes the structure of these types. Modules can communicate about their state in the form of the exchanged data, but should not influence each other in another way. In this way the system becomes transparent, simplifying logging and playback. The LCM type specification allows several primitive data types and references to other LCM types. With these user-defined type specifications the interfaces inside the application architecture can be defined.



Log channel	Playback channel	Enable
LIDAR_FRONT	LIDAR_FRONT	✓
LIDAR_REAR	LIDAR_REAR	✓
POSE	POSE	✓
POSE_TRUTH	POSE_TRUTH	✓
SIM_OBSTACLES	SIM_OBSTACLES	✓

Figure 4: A screenshot of the LCM LogPlayer. With two green markers the user has added two "repeats" on the log. Courtesy [6].

The last functionality of the LCM framework is a number of general tools to inspect, filter, record and playback the information which is published to the udpm group address (see for instance Fig. 4). Those tools provide comparable functionality as the ROS framework. Also note the existence of LCM-Lite, a minimalist implementation especially meant for cross-compilation to resource-limited embedded platforms (as sensor networks).

The LCM framework provides a clear solution for the connecting components inside a system, yet gives no directions the functionality of the components or how they should be organized (for instance in a three-tiered architecture as described in [12] and [10]).

4 SI4MS High-Level Architecture

The high-level architecture defined in the SI4MS project [10] distinguishes several levels; those levels represent both a geometrical scale and a time scale after an analysis of intelligent vehicle applications indicated that the time and geometrical scale are highly correlated. The levels which are distinguished are:

0 **Vehicle:** the perception and control around a single vehicle. The typical time-scale is from 0.01 seconds (collision avoidance) to 2 seconds (headway time).

1 **Local:** the perception and control on a road segment between two intersections. The time scale is typical 1 to 10 seconds.

2 **Stream:** the perception and control on a main road, with traffic on multiple lanes and intersections with multiple minor roads. The time scale is typical 10 to 100 seconds.

3 **Network:** the perception and control around a network level. When needed, a separation can be made between a subnetwork (level 3) and a network level (level 4). The time scale is typically minutes to hours.

Those 4 levels are indicated in Fig. 5 with the Situation Assessment components labeled SA_0 to SA_3 and the Situation Management components labeled SM_0 to SM_3. All components compete for limited Communication and Computation resources. Note that in a distributed system multiple resources could be available (which could be shared by middleware in such a fair way that the components are not aware of the sharing). This is illustrated with the Computation and Communication components behind the components in the front. Also note that there exist multiple instances of SA and SM components. This can mean the equivalent component in another vehicle or a component which concentrates on the perception and control of another concept (for instance the Hazards and Obstacles concepts of the MIT Team [13]). Also note that each component has a 'shell' around its core. This represents that the components are adaptivein the form of self-optimization and self-organization of each component.

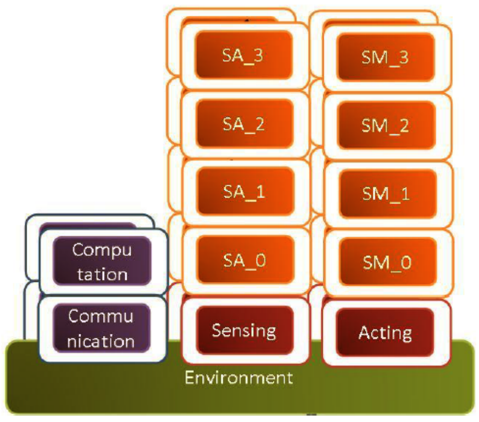


Figure 5: The high level architecture of SI4MS. Courtesy [10].

The DARPA Urban challenge evaluates capabilities, which are in [10] classified as on a vehicle or local level, which corresponds on time scales up to 10 seconds. Yet, no communication between the intelligent vehicles of the DARPA Urban challenge was allowed, so all decisions stayed on a vehicle level. Note at the left side of Fig. 6 that there are three levels of control: at the highest level the Navigator develops a plan by combining network information with the required mission (which represented a timescale between 12 minutes and 2 hours). In a total of 7 missions the vehicle traveled 55 miles in 6 hours.

So the Navigator can be identified as SM_2, the Motion Planner as SM_1 and the Controller as SM_0. At the Situation Assessment side the picture is less clear. The Drivability Map can be identified as SA_1, as it is heavily used by the Motion Planner SM_1. More difficult is the identification of the Lanes and Fast Vehicles percepts. Their information is used at a higher level (which would mean that these are two instantiations of SA_2). Yet, the Lanes percept information is also used in the Drivability map (indicating a level of at most SA_1). The Fast Vehicle percept interprets Radar data equivalently as the Hazard and Obstacle percepts interpret the SICK data, which would mean that those percepts would be from the same level. The Controller is intentionally decoupled from the sensor suite and concentrates in minimizing the difference between the current speed and the intended speed (and equivalently the steering angle). Because the lowest Situation Assessment level is not used, the Road paint detection, the Fast Vehicles, Hazards and Obstacles can be identified as SA_0. The modules Lanes and Drivability Map are two instantiations of SA_1, because this not only fits hierarchically, but

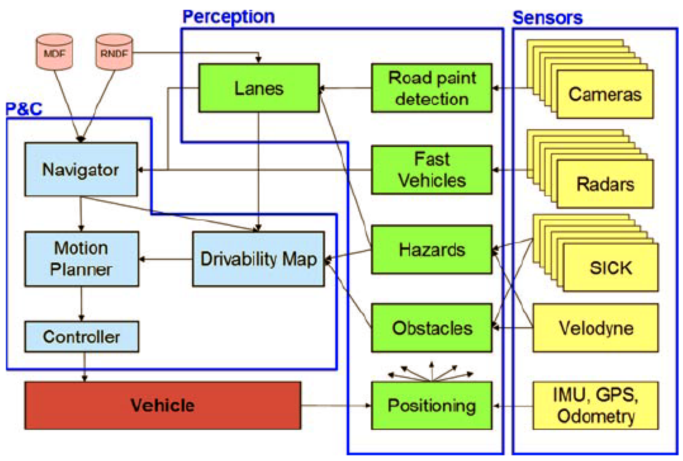


Figure 6: The system architecture of Team MIT. Courtesy [13].

also is consistent with the geometrical scale. So, in the system architecture of MIT [13] single instances of SM and multiple instances of SA are identified. Note that for various modules a multiple fail-safe logic is implemented, which can be identified with the adaptive shell in the high level architecture (Fig. 5).

The software architecture used in the MAGIC Challenge (see Fig. 7) consists of two main modules: the Ground Control Station (GCS) at the left and multiple instantiations of the robot controllers on the right. The Ground Control Stations was operated by two people: one concentrating on the tasks of the robots (Situation Management) while the other operator was concentrating on the confirming the observations of the robots (Situation Assessment). The number of operators compared to the number of robots (14) is quite low: the system could run without supervision, the input of the operators could be interpreted as advice (with more weight than an automatic decision).

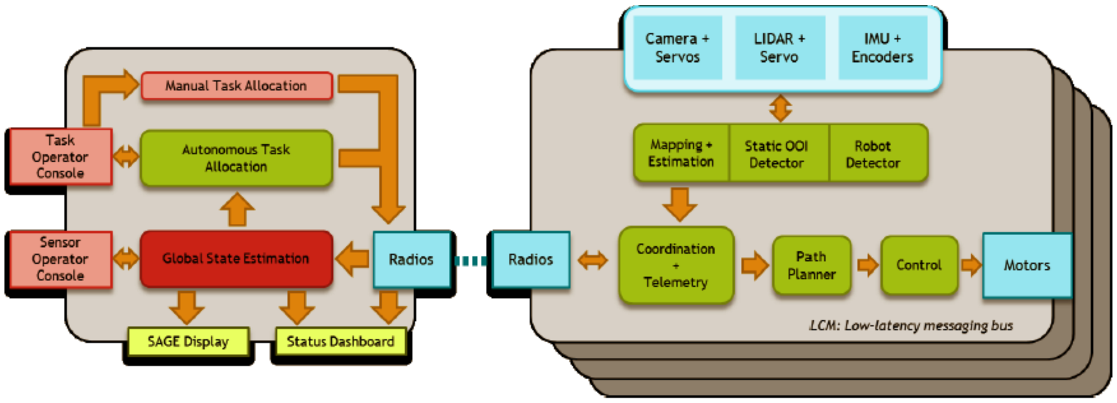


Figure 7: The software architecture of Team Michigan. Courtesy [15].

At the right sight of Fig. 7 a three-tier chain of control can be identified, equivalent with the MIT architecture. The Coordination could be identified as SM_2, the Path Planner as SM_1 and the Control as SM_0. At the perception side the situation is again less clear. Team Michigan uses for their state estimation a Decoupled Centralized Architecture [14]. The essence of this approach is that the robots use only a local map (SA_1), which they only share with the GCS, where a global map can be made (SA_2). The robots also had a number of advanced perception modules on board: such as terrain classification, object of interest detectors (based on shape and color) and robot detectors (based on April tags). Those perception modules can be classified as SA_0.

5 Coupling to LCM

The logfiles published by the MIT team[2] only consist of the messages from the sensors to the SA_0 modules. The camera images are published in another format. The provided sensor messages are `pose_t`, `laser_t`, `velodyne_t` and `gps_to_local_t`. The logfiles of the DARPA team of Michigan[3] consist of the same types, augmented with `image_t`.

The Magic Team of Michigan did not publish logfiles of their runs. Yet, in their April toolkit some control and higher level perception datatypes are available (`gamepad_t`, `procman_process_t`, `tag_detection_t`).Missing is the definition of a message format for sonar sensors (e.g. `sonar_t`).

6 Coupling to USARSim

USARSim (Unified System for Automation and Robot Simulation) is a high-fidelity simulation of robots and environments [4] based on the Unreal Tournament game engine. It is intended as a research tool and is the basis for the RoboCup rescue virtual robot competition (the UT2004 version was also used for the first DARPA Grand Challenge). Because at the Universiteit van Amsterdam none of the custom-build intelligent vehicles was available, the LCM interface was tested by building a proxy[4] to a simulated robot.

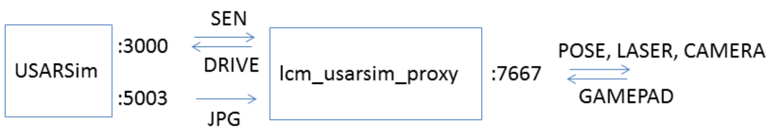


Figure 8: An overview of the communication channels opened by the proxy.

As illustrated in Fig. 8 this proxy publishes information on three LCM channels (POSE, LASER and CAMERA), with information of type `pose_t`, `laser_t` and `image_t`. The proxy listens to the channel GAMEPAD for information of type `gamepad_t`[20]. This information is converted to the format and communicated over the TCP ports which define the interface of USARSim.

The published information was verified with the LaserPlugin and VideoPlugin of the LcmSpy for four different robots: an humanoid Nao robot, a Pioneer Advanced Terrain robot (P3AT), a Kenaf robot (equipped with two cameras) and an AirRobot (see e.g. Fig. 9).

The LaserPlugin of the Kenaf shows a non-accurate map, because the Kenaf is equipped with a Hokuyo URG04LX laserscanner with a shorter maximum range than the SICK LMS 291-SO5. Yet, this parameter is not available in the `laser_t` format, so this flaw cannot easily be mended. Equivalently, inside the `pose_t` a field is missing for the angular acceleration. The formats could be adjusted, yet this would mean that the fingerprint based type checking of existing April modules would fail.

The appendix of [20] contains a number of Java-based tools inside the April toolbox which could be used to further process the information published by the proxy, such as for instance

[2]`http://grandchallenge.mit.edu/wiki/index.php?title=PublicData`
[3]`http://robots.engin.umich.edu/SoftwareData/Ford`
[4]http://usarsim.git.sourceforge.net/git/gitweb.cgi?p=usarsim/usarsim;a=tree;f=USARTools/LcmProxy

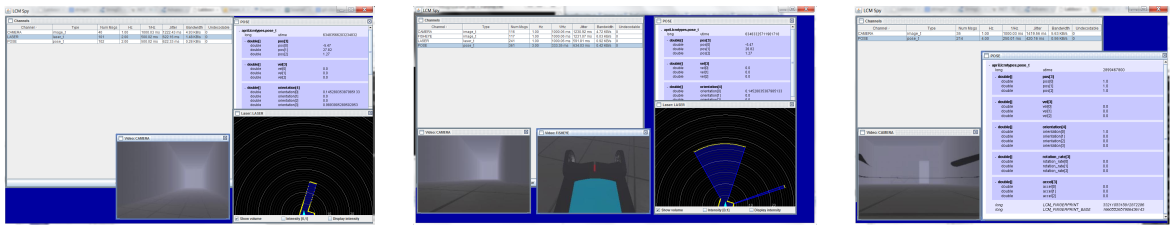


Figure 9: Screenshots of LcmSpy for a P3AT, Kenaf and AirRobot.

advanced scan matching and simultaneous localization and mapping algorithms [22].

7 Conclusion

The LCM library provides a very robust and fast communication channel, which allows to decouple modules very easily. It allows to monitor, record and replay the information streams between modules, which is great for research and development of complex perception algorithms.

In some cases the LCM library could be preferred above using the communication mechanisms of the Robotics Operating System (ROS). Assets are not only the speed and robustness, but also to use it for cross-compilation to resource-limited embedded platforms. ROS is known for its many dependencies, which all have to be cross-compiled.

The LCM framework doesn't provide directions how modules should be combined into a high level architecture. Two applications on top of the LCM framework have been analyzed and compared to the SI4MS High Level Architecture. The analysis showed that the LCM messages were mainly used at the perception side, although there is enough potential at the control side. The two studied applications showed a clear three-tier architecture at the control side and a more complex pattern at the perception side.

The LCM library has been tested in simulation by building proxy-software, which is published on sourceforge. The LCM framework with the sensor messages used in three applications was mature enough to control three different robots in simulation.

References

[1] M. Achtelik, A. Bachrach, R. He, S. Prentice and N. Roy, "Stereo Vision and Laser Odometry for Autonomous Helicopters in GPS-denied Indoor Environments", in "Proceedings of the SPIE Unmanned Systems Technology XI", volume 7332, Orlando, F, 2009.

[2] R. Ackoff, "From data to wisdom", *Journal of Applied Systems Analysis*, volume 16, 1989.

[3] J. Albus *et al.*, "4D/RCS Version 2.0 : A Reference Model Architecture for Unmanned Vehicle Systems", Technical Report 6912, NIST, Gaithersburg, MD, 2002.

[4] S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, "USARSim: a robot simulator for research and education", in "Proceedings of the International Conference on Robotics and Automation (ICRA'07)", pp. 1400–1405, 2007.

[5] J. Crossman, R. Marinier and E. Olson, "A Hands-Off, Multi-Robot Display for Communicating Situation Awareness to Operators", in "Proceedings of the International Conference on Collaboration Technologies and Systems", pp. 109–116, May 2012.

[6] A. Huang, E. Olson and D. Moore, “Lightweight Communications and Marshalling for Low-Latency Interprocess Communication”, Technical Report MIT-CSAIL-TR-2009-041, Massuchusetts Institute of Technology, September 2009.

[7] A. S. Huang, M. Antone, E. Olson, D. Moore, L. Fletcher, S. Teller and J. Leonard, “A high-rate, heterogeneous data set from the DARPA Urban Challenge”, *International Journal of Robotics Research (IJRR)*, volume 29(13), November 2010.

[8] A. S. Huang, E. Olson and D. C. Moore, “LCM: Lightweight Communications and Marshalling”, in “Proc. Int. Conf. on Intelligent Robots and Systems (IROS)”, October 2010.

[9] L. J. Kester *et al.*, “Sensor Intelligence for Mobility Systems - Design principles and methods for dynamic information systems”, Project Report, ICT Innovation Platform Sensor Networks, November 2011.

[10] L. J. Kester *et al.*, “Sensor Intelligence for Mobility Systems - High Level Architecture”, Project Report, ICT Innovation Platform Sensor Networks, April 2012.

[11] V. L. Knoop and H. van Lint, “Sensor Intelligence for Mobility Systems - State of the Art in Coordinated Traffic Management”, Project Report, ICT Innovation Platform Sensor Networks, June 2011.

[12] D. Kortenkamp and R. G. Simmons, “Robotic Systems Architectures and Programming”, in “Springer Handbook of Robotics”, (edited by B. Siciliano and O. Khatib), pp. 187–206, Springer, 2008.

[13] J. J. Leonard *et al.*, “A perception-driven autonomous urban vehicle”, *Journal of Field Robotics*, volume 25(10):pp. 727–774, 2008.

[14] D. C. Moore, A. S. Huang, M. Walter, E. Olson, L. Fletcher, J. Leonard and S. Teller, “Simultaneous Local and Global State Estimation for Robotic Navigation”, in “Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)”, May 2009.

[15] E. Olson *et al.*, “Progress towards multi-robot reconnaissance and the MAGIC 2010 Competition”, *Journal of Field Robotics*, volume 29(5):pp. 762–792, September 2012.

[16] G. Pandey, J. R. McBride and R. M. Eustice, “Ford Campus vision and lidar data set”, *The International Journal of Robotics Research*, volume 30(13):pp. 1543–1552, 2011.

[17] I. Passchier *et al.*, “Sensor Intelligence for Mobility Systems - State of the Art in Cooperative Systems”, Project Report, ICT Innovation Platform Sensor Networks, September 2012.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A. Y. Ng, “ROS: an open-source Robot Operating System”, in “ICRA Workshop on Open Source Software”, 2009.

[19] P. Ranganathan, R. Morton, A. Richardson, J. Strom, R. Goeddel, M. Bulic and E. Olson, “Coordinating a Team of Robots for Urban Reconnaisance”, in “Proceedings of the Land Warfare Conference (LWC)”, November 2010.

[20] R. Rozeboom, “April Robotics Laboratory modules for USARSim”, Project Report, Universiteit van Amsterdam, August 2012.

[21] A. N. Steinberg and C. L. Bowman, “Rethinking the JDL data fusion levels”, in “Proceedings of National Symposium on Sensor Data Fusion (JHUAPL)”, June 2004.

[22] J. Strom and E. Olson, “Occupancy Grid Rasterization in Large Environments for Teams of Robots”, in “Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)”, October 2011.

[23] R. Touchton, D. Kent, T. Galluzzoa, C. D. Crane III, D. G. Armstrong II, N. Flann, J. Wit and P. Adsit, “Planning and modeling extensions to the Joint Architecture for Unmanned Systems (JAUS) for application to unmanned ground vehicles”, in “SPIE Defense and Security Symposium”, March 2005.

Acknowledgements

This research is funded by the Dutch IIP Cooperation Challenge 'Sensor Intelligence for Mobility Systems', Project 09-NROI-384. The author like to thank Richard Rozeboom for the initiation of this study and Camiel Verschoor and Leon Kester for proofreading this document.

IAS reports