A software infrastructure for sensors, actuators, and communication

Thomas Johansson, Thomas Hellström
Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden
{thomasj, thomash} @cs.umu.se

1. Background and motivation

Various tools for development of software for mobile robots have been proposed over the last two decades. These tools exist at many levels of abstraction, and are designed to support the development in different ways. At the highest level, both the hierarchical, reactive, and hybrid robotic paradigms are represented by a number of architectures that implement the general ideas within the respective paradigm. These architectures describe, on a high level, different ways to organize the main components to achieve the overall goal of designing and constructing an intelligent mobile robot. At the intermediate and low levels, there exists a number of general systems for development of robotic software.

However, it is our firm belief that all levels of tools mentioned above overlook important issues regarding target machines, actuators, and sensors. Some of the intermediate level systems do include handling of sensors and actuators, but do not pay enough attention to them in a more systematic and general way. The work presented here attempts to fill this gap, and provides a link between physical sensors/actuators (or rather their software counterparts: the driver routines) and the overall control program. Furthermore, the proposed system deals with multiple target machines and communication between software modules placed on the same, or different, computers. These issues become more and more important as the complexity of the developed robots increases, and support for them should, in our opinion, exist in the development process.

The system has been developed as part of an ongoing project for an autonomous path-tracking forest machine. The forest machine is intended to move autonomously along a recorded path through a changing environment. The machine has a number of sensors (e.g. position, obstacle, speed) and actuators (steering angle, throttle) to its aid. The numbers and types of sensors and actuators have varied during the development process, partly as a result of investigation of different sensor types, and partly as a result of the use of different vehicles. The project uses a small Pioneer robot for fast development, a software simulator for system testing and a Valmet 930 Forwarder for outdoor real-life testing of the system. This dynamic environment puts special demands on the software used during development, mainly demands for ease of configuration, quick exchange of vehicles, and replaceable sensors.

2. Primary criteria

The following set of criteria was deemed important at the outset of the design process:

Interchangeability of robot vehicles: We anticipated the need for different software modules that should represent different robot hardware, in particular software for our small Pioneer Robot and the 8-meter-long forest machine (Valmet 830). It should be easy to switch between the two machines without any code changes. This is a practical and efficient approach, especially when developing systems for large autonomous vehicles. Furthermore, support for this level of interchangeability will become more and more important as generic robotics systems for many types of tasks and platforms are being developed.

Interchangeability of sensors and actuators: In a complex system, sensors, and sometimes also actuators, often have to be replaced by similar, yet not identical, components. This kind of replacement is often a major part of the development and research process where different kinds of algorithms, sensors, and setups have to be evaluated and compared. Furthermore, this functionality is often needed in a completed robot. A satellite navigator e.g. may have to be replaced by odometry if the satellite signals are occluded, or a laser scanner used for obstacle detection may have to be replaced by a radar sensor due to weather changes.

Distributed processing: The initial specification placed one computer on the moving vehicle and another at a supervisor’s point, and freedom to place parts of the system on either machine was required. Placing the whole system on only one machine simplified debugging in the office. Freedom in the placement of software parts also means that extra computing capacity can be added to the system without changes to the code. The different parts communicate via a standard TCP/IP network, either Ethernet or a wireless LAN.

Event driven: The system should be driven by events, from either data obtained from a sensor, or control messages from a user or control software. There are no polling loops in the support system, even though there are some in the high-level control software.

3. Design

From these primary requirements a set of more detailed requirements was specified to facilitate the design and construction of the system:

Modularity: A software ‘module’ is the basic building block in the system. Modules exist in a type hierarchy, with subtypes being sensors, vehicles, and actuators for example. At the top level in the hierarchy all modules have a common interface, i.e. they have a common set of operations that can be performed by them. Examples are...
close, open, get status and so on. This property of modules is used by the system to load, start, stop, and interrogate modules on a high level without knowing the detailed function of a particular module. A common loader for all types, which executes an initialisation file with module names and actual types, loads the modules.

Extendibility and flexibility: New modules and module types should be easy to add to the system, without any changes to the existing software. An important feature of the system is the low coupling between modules, i.e. they are effectively isolated from one another with regard to internal representation of data and functions. Only the exposed external interface is shown, and if a new module of a certain type is added, it can be handled as any other module of this type. An example would be a Speed Sensor type, whose only function is getSpeed. This leaves it up to the implementers to design the module in any way they want, as long as the module delivers data via its getSpeed function. In the system there exist several speed sensors, which obtain data from the machine itself or from the GPS receiver, but the rest of the system does not know, and does not need to know, the actual sensor used at any specific moment. If a new type of speed sensor is installed, its corresponding speed sensor module can be added to the system without any changes to the existing code. Which sensor to use can be configured at start-up or dynamically during the run.

Cohesion: The modules are designed for high cohesion, i.e. a module does just one thing, but does it well. For example, a sensor that would require some form of filtering of its data. Instead of incorporating filter code into the sensor module (and thus into all sensor modules that require it), a special filter module is developed. The filter module would have the same type as the sensor, and in effect would be a virtual sensor, used in the normal sensor’s place. The filter module then uses the actual sensor as a source for its data. This virtual sensor will for all intents and purposes look exactly like a ‘real’ sensor to the users of the sensor data.

Multithreading: Every module should have its own execution thread, and thus run independently of other modules. Polling loops are discouraged, instead an event-driven system is used with multiple independently executing threads. The threads normally sleep and only wake up if a message arrives, either from another module or from the network or a user.

Object Orientation: The system should be written in an Object-Oriented language. Java’s ease of development, excellent development environments, support for modular software and multithreading, combined with our experienced staff members, settled the question of implementation language.

4. Current status and future work

At present the system is used daily for the development of the path tracking forest machine. It comprises some 100 modules divided into these groups:

Sensors – speed sensors, angle sensors, several GPS position sensors, heading, attitude, range, and obstacle sensors.

Actuators – throttle and steering angle actuators for different vehicles

Controls – control panels that make it possible to run the vehicle either autonomously or via teleoperation

Indicators – most sensor types have a corresponding indicator to show the current measured value on a screen, mainly for debugging purposes.

Proxies and servers – facilitates the communication over the network.

Simulators – simulate most sensor types, to ease debugging of communication systems

In a system of this size it is important to be able to get a view of the health of the modules. All modules can report a basic status plus, where appropriate, the delay time for the data from a sensor. A Health Monitor has the knowledge of all loaded modules on a computer, and also communicates health information with all other computers, so the overall health can be assessed on any machine.

The system relies heavily on configuration data stored in initialisation files. Every module has a file that specifies individual parameters for the module, for instance logging options, network port numbers or serial communication parameters, and most importantly, any other module used. There is also one large configuration file that specifies what modules to load when the system starts. To ease the problems of keeping track of hundreds of initialisation files there is a graphical tool, a Configuration Manager, that makes it possible to specify and connect modules to each other. At present this is a static tool used before the system is run, but plans exist of a dynamic version that can be used to change configuration in mid-run.

Also a more powerful inspection system is planned, that enables the user to see into every module while it is running, and also, with great care, change values of parameters.

So far, the developed system has proven to be a powerful tool that can serve as a generic framework when developing mobile robots and autonomous vehicles.

Acknowledgements

This work was financed by the Kempe Foundations, VINNOVA, Land Systems Hägglunds, Carl Tryggers stiftelse, LKAB and Komatsu Forest AB. We gratefully acknowledge their support.