

Design and Implementation Methodology for Autonomous Robot Control Systems

Eric Jackson & David Eddy
International Submarine Engineering, Ltd.
1734 Broadway St.,
Port Coquitlam, BC, V3C 2M8

Abstract

This paper describes a general design and implementation approach used for programming and controlling complex robotic systems such as remotely operated submersibles, autonomous submersibles, and robotic manipulator systems. A hierarchical approach to control system design is adopted. The hierarchical design is translated into a component-based software design. All real-time software components are rigorously unit-tested and can be reused in other control systems. The programming environment allows the design and evaluation of control systems that span several embedded processors. Graphical design tools are used for specifying component interconnections. The project intends to support the graphical methods of the process control industry's IEC 1131-3 Programmable Controller Programming Standard for defining robotic control systems and robotic control tasks. The paper also discusses some project management advantages using these approaches as well as some ongoing development work.

Introduction

This paper describes a design and implementation methodology used by the authors for developing supervised robotic systems. The methodology has been used in the development of a number of successful robotic systems ranging from teleoperated to highly autonomous systems.

The development process is split into two parts - design and implementation. These are two discrete phases in developing telerobotic systems. A hierarchical control structure, combined with a component-based software implementation approach serves to simplify and accelerate control system development.

Control System Design Methodology

This section defines the methodology used by the authors to design a telerobotic system. Each step is described in sequence as follows:

Define Hierarchy of Desired Behaviors

The first step is to define and name a hierarchy of desired Robot Behaviors. The hierarchy is based on time domain behavior. At the top are symbolic rule-based high level robot tasks, e.g. "Extract module" or "Transit to Start Location". At the bottom are low level, high speed servo functions, e.g. position, velocity, force, and impedance. A lexicon of these behaviours (Task Verbs) must be generated and maintained throughout the project.

Usually, only about four levels are required in the hierarchy. For the case of an autonomous submersible the levels from lowest to highest would be: control plane servos, heading/depth/attitude servos, line-following and homing servos with reactive obstacle avoidance, and mission command sequencing. For an autonomous manipulator system the levels from lowest to highest would be: actuator torque servos, position/velocity/impedance servos, operational space trajectory servos with reactive obstacle avoidance, collision-free trajectory planning, and work task command sequencing. The designers must ensure that the behaviours specified are sufficient to allow operations personnel to describe all robotic system operations.

Define the Level of Robot Autonomy

The levels in the behaviour hierarchy represent various levels of autonomy. That is, as the higher levels in the hierarchy become automated, the operator is required to provide less supervision to the robot. The level of autonomy to be implemented by the control system must be defined. Ensure that the Task

Verbs defined for the highest level of robot autonomy are sufficient to allow the operator to specify all desired robot behaviours.

The decision as to what level of autonomy should be implemented must be made based on considerations of safety, complexity, and the relationships between communications channel characteristics and robot controller stability/bandwidth requirements.

Define Local/Remote Partitioning

Automated functions can be split between the console (local) and the robot (remote), i.e. the robot will execute lower levels of control than the console. Since it is usually more expensive to put processing functions locally on the robot than on the Operator Workstation, the amount of onboard functionality needs to be determined through engineering analyses. For example, the bandwidth and latency of the communications link between the robot and the operator will determine whether or not any servos can be closed at the Operator Workstation. The reliability of the communications system is also a major factor in this analysis.

Define Behaviour Matrix

For each behaviour (Task Verb) to be executed by the Robot and the Operator Workstation the following must be defined:

- i) its nominal decomposition into lower level Task Verbs;
- ii) exception handling behaviour decomposition;
- iii) the static or quasi-static knowledge required to support the decomposition, e.g. manipulator geometry, kinematics, module grapple points and extraction rules;
- iv) sensory input requirements to support the decomposition, e.g. joint positions, optical targets, proximity sensors, limit switches;
- v) sensory processing outputs to other levels in the hierarchy, e.g. “extraction complete”, “manipulator stowed”, “grasp failed”;

Note that these functions correspond to the standard NASREM (NASA Standard REference Model for telerobots - Ref. 1) columns; i.e. i) and ii) are performed by the right-hand “task decomposition column, iii) is the center “world modeling” column, and iv) and v) are performed by the left-hand “sensory processing” column. This structure provides a general template for complex control system design. (See Figure 1).

Conflicting Behavior Resolution

The manner in which conflicting behaviors are to be resolved must be defined. For example, if an exception-induced behavior is triggered such as “avoid obstacle” or “freeze”, define how the robotic system will deal with the fact that it is also supposed to perform other operations such as following a trajectory or intercepting a target. Standard methods of resolving conflicting goals include logic-based approaches (e.g. subsumption) and cooperative approaches (e.g. potential fields and fuzzy logic).

Define Control System Modules

The overall control system modules (CSCIs - computer software configuration items, or “computer boxes”) and communications between them are then defined. The sensory input requirements, Task Verb decompositions, and partitioning approach can be used as a baseline.

Because communications between separate computer boxes is subject to communications errors, define the type of communications required between CSCIs. Commands sent asynchronously from the operator to the robot need to be sent using an error-free protocol using end-to-end acknowledgment (e.g. TCP). Feedback values from the robot to the operator, however, can usually be sent continuously without acknowledgment (e.g. UDP).

Define User Interface

- The User Interface must be defined at two levels:
- i) standard real-time on-line operator interface, and
 - ii) configuration, scripting, and behavior definition.

Define Network of Software Modules

Each CSCI should be broken down into logical groups, e.g. operator interface, trajectory generation, navigation algorithms, and position servos.

Define the lowest level software components required to implement each of these logical groups (CSUs - computer software units) and the data/event flow between them. The resulting design is a network diagram resembling a printed circuit board schematic.

In breaking the system down into low level components, the designer should attempt to eliminate two-way communications between components, e.g. remote procedure calls. Component execution should be completely deterministic, relying on timers and data flow to determine when components will run.

Control System Implementation

For many years, International Submarine Engineering, Ltd. (ISE) has been using its own event-driven, component-based software environment for all of its control applications. This software architecture has now been used on several telerobotic systems including teleoperated and autonomous submersibles, telerobotic manipulators, and autonomous service robots. This section describes this software environment and how it is used in the development of control systems for telerobots.

Component-based Software

All software is written in the form of “components” which behave like software integrated circuits. The function of a component is the smallest possible functional unit of a real-time system, and represents an atomic computation unit in a complex controller. Each component can be represented by a “black box” with well defined inputs, outputs and parameters.

The function of a component is general enough that the code can be written, thoroughly tested, documented, and reused as required. The code is developed in C or C++. Once tested, components are placed in a library under version control.

The current library contains hundreds of components including servo controllers, filters, linear and nonlinear transformations, logic and arithmetic functions, kinematic and dynamic calculations, device interfaces, and graphical/user interface elements. Usually, however, each new project requires some new components. These new components are then implemented, tested, and integrated into the library.

Events and Actions Execution Kernel

A component interacts with the outside world by connecting its inputs and outputs to other components. New modules can be added to an existing control system without changing verified modules or working configurations. These interconnections are called “events”, and the component procedures triggered by events are called “actions”. For this reason, this architecture is sometimes referred to as the “events and actions” architecture.

All scheduling of and communications between software components is “event-based,” meaning that the function of each component is entirely event-driven. Components interact with one another by “event propagation”. The concept of an “event” encompasses that of interrupt handling, task

synchronization, data sharing, and message passing in conventional multitasking systems. Events may or may not contain data. Events act as a medium between data providers and data consumers, and therefore the software for the providers and consumer is completely independent of each other.

Each component is assigned a preemption level. When an event is generated by a component, all consumers of that event are scheduled for processing on their respective preemption levels. The scheduler then processes whichever component has the highest priority.

Software Modules Integration

Constructing a real-time system from available components is a matter of choosing the required components from the library, specifying any required parameters (e.g. default servo gains), and defining the interconnections between them. The diagram of component interconnections generated during the design process (resembling a printed circuit board schematic) is a complete representation of the applications software configuration (see Figure 2). The process of implementing the control system application is therefore completely separated from that of implementing the software components.

Since the application development is independent of software component development, it is possible for non-programmers, such as system engineers, technicians and end-users, to configure their own real-time systems.

At the present time, there are three systems in use at ISE for configuring real-time control systems: declarative configuration, code-based configuration, and graphical configuration.

Declarative Configuration

Using declarative configuration, users describe all components and their interconnections in a text-based file. The configuration file is then read and interpreted by the program to initialize the data structures for real-time control. Since all components are pre-compiled, this approach is very fast and does not require the use of a compiler.

Code-Based Configuration

Code-based configuration is the C++ version of the declarative configuration approach. Instead of a simple text-based file representing component instantiations and interconnections, the configuration consists of a list of C++ constructors. The

configuration file is then compiled and linked before the application is run.

Graphical Configuration

The graphical configuration approach relies on the use of a computer-aided design tool to draw the components and their interconnections (See Figure 2), and then to generate a list describing these interconnections. The list can then be parsed into a text-based or a C++ configuration file.

Project Management Issues

Project management issues that are influenced by the described design and implementation approach include quality management, work scheduling, and personnel issues. A specific benefit of the described approach is the parallelization achievable in time and in personnel. It divides the project into a set of successive layers, and it facilitates the use of a multi-disciplinary development team.

The component-based software approach facilitates software quality management. Once components are developed and tested, they can be integrated into a library under version control. Formal procedures can be enforced as to how components are modified. Responsibility for software quality control belongs to software personnel.

The implementation schedule for the control system is determined by its hierarchical definition. Once the overall control system is outlined, the detailed design, implementation and testing proceeds from the lowest levels upward. In order to accelerate system development, implementation and testing of the low levels can proceed concurrently with the design of the higher levels of the control system.

The design and implementation tasks are partitioned into two parts - the control system application and the software components. The result is a separation into two disciplines - control systems engineering and software development. The software team acts as a supplier to the applications engineers. The software team is responsible for development and maintenance of all software components. Applications engineers are responsible for the application design, implementation, and testing.

Ongoing Developments

This section describes ongoing developments at ISE relevant to the design and implementation of telerobot control systems.

Graphical Programmer Interface

In the earlier section on software module integration, a graphical configuration technique was described. This graphical approach is a direct link between design and implementation. The designer can use the graphical tool to specify all of the component interconnections, which can then be translated directly into the configuration for an operational system. This approach is similar to that used in the process control industry for configuring programmable logic controllers.

The process control industry has generated a specification called IEC 1131-3 (Ref. 2) which describes methods for programming controllers. They include three graphical programming techniques, i.e.:

- i) sequential function charts - appropriate for specifying high level execution scripts,
- ii) ladder logic diagrams - appropriate for specifying logic, and
- iii) function block diagrams - appropriate for specifying control system structures.

A control system should be programmable using any combination of these three graphical techniques.

At present, ISE supports function block diagrams, and an effort is underway to support sequential function charts. The former shows control system software component interconnections directly. The latter would be used to specify high level sequencing commands for autonomous robots, i.e. mission scripts.

Hardware and Software Platforms

Most telerobot control systems consist of multiple CPU platforms; a minimum system usually includes an operator workstation and a dedicated robot controller. Many telerobot controllers consist of a number of CPUs, e.g. individual joint controllers, kinematics processors, and trajectory generators. Real-time control functions are distributed amongst these processors. It is desirable to be able to specify the integrated control system design across all of these processors simultaneously.

At present, ISE runs the same "events and actions" software across multiple processors, and the overall control system configuration is represented in a single place. Processors supported to date include Intel-based PCs, Motorola-based embedded computers, and digital signal processors. Future hardware platforms will include Motorola microcontrollers and SGI workstations.

Until recently, the software has provided all of its own kernel functions, running directly on the processor without any other operating system support

(except some PC-DOS file I/O). The newest version of the software environment runs under Windows NT, allowing it to coexist with other software packages, e.g. graphical user interface packages. Software packages communicate with each other using a number of protocols, but socket-based protocols are preferred. A Unix-based version will be developed next.

One hardware platform being used currently includes multiple C-40 digital signal processors. At present, the applications programmer must divide the software components amongst the multiple processors. However, a new version of the software environment under development is based on an object server model, whereby the process of loading the CPUs is done automatically, ensuring that CPU load-sharing is balanced.

Summary

A formal method for the design and development of robot control systems has been outlined. It is based on the concepts of hierarchical control and event-driven, component-based software. These practices simplify the development process for telerobot control systems.

Advantages of hierarchical control system design are that it provides a template for complex control system functional design and that it allows parallelism in the development process. Low level control system functions can be implemented and tested while the higher levels are still in the design phase.

Adoption of an event-driven, component-based software approach also has several advantages. Firstly, it facilitates software quality management - software components can be rigorously tested and then integrated into a version-controlled library for later reuse. Secondly, it divides the development process into separate disciplines of control systems engineering and software engineering, facilitating a multi-disciplinary development approach.

New developments underway include support for new hardware and software environments and more advanced graphical programming techniques. The former will allow the software environment to be more widely used. The second will further streamline the process of rapidly transforming designs into operational telerobot control systems.

References

1. J.S. Albus, H.G. McCain, ZR. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture", National Bureau of Standards Robot Systems Div., 12/4/86
2. International Electrotechnical Commission, "IEC 1131-3, Programmable Controllers - Programming Languages"

Acknowledgment

ISE would like to acknowledge the contribution of the Canadian Space Agency's Strategic Technologies in Automation and Robotics (STEAR) Program in the development of this control system design and implementation methodology.

Level	Sensor Processing	Knowledge	Task Decomposition
3	vehicle Latitude vehicle Longitude transponder Range transponder Bearing obstacle information estimated time till arrival device statuses	transponder radius potential functions for OAS maximum speed minimum maneuvering speed waypoint latitude, longitude, radius max depth min depth min altitude	targeting line following homing obstacle avoidance Outputs to Level 2: <ul style="list-style-type: none"> heading setpoint virtual rudder setpoint depth /altitude setpoint
2	pitch, pitch rate roll, roll rate heading, yaw rate depth, depth rate altitude, altitude rate doppler speed estimated speed VB tank levels distance traveled device statuses	pitch limit roll limit depth limit altitude limit maneuvering control modes zero speed threshold normal deceleration RPM extreme deceleration RPM VB tank level-to-weight parameters	maneuvering control speed control VB tank weight control Outputs to Level 1: <ul style="list-style-type: none"> planes setpoints planes enable setpoints propeller RPM setpoint VB tank fill/drain commands
1	planes angles thruster RPM VB tank fill status device statuses thruster overheat water alarms overcurrent alarms	PID control laws thruster gearbox ratio maximum allowable currents	PID control of all planes angles propellor RPM control VB fill/drain control device power switching Outputs to Hardware: <ul style="list-style-type: none"> planes motor voltages planes enable commands thruster RPM command thruster enable VB valve commands device power switches

Figure 1. NASREM-Style Behaviour Matrix

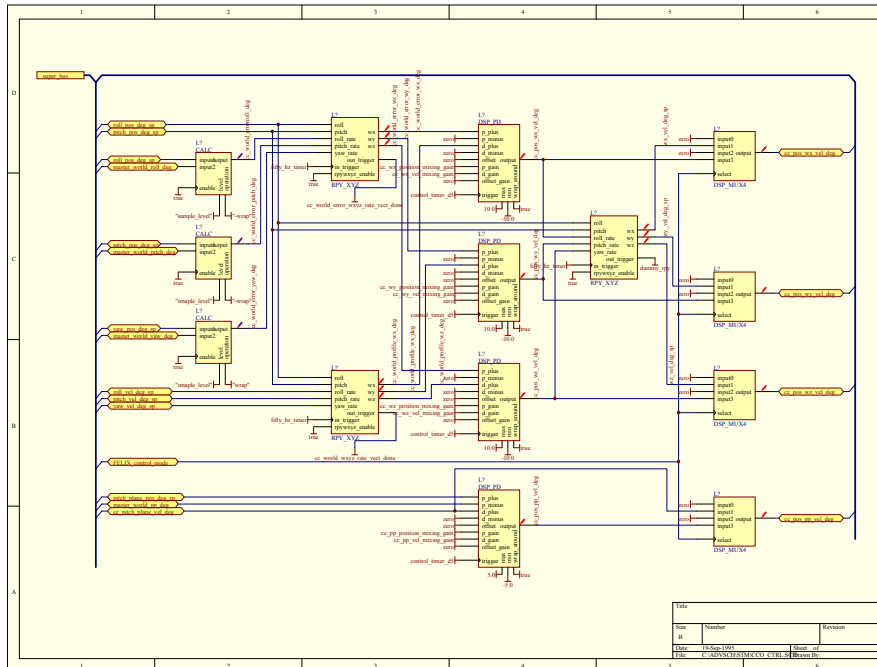


Figure 2. Sample Component Interconnection Diagram