

## **Customer Requirements For MRC-II System**

### **1. Objective**

This document defines the Customer Requirements for the MRC-II System (Modular Robot Controller – II). The MRC-II System will replace the MRC Software that is currently in use. Note that this project considers MRC-II to be a system (hardware and software) rather than just software, although the initial focus will be on the software.

The Customer Requirements can be considered a “marketing” document that drives the creation of an “engineering” Software Requirements Specification. In this context, the “customers” are the researchers (professors and students) that may use the system for their research, most likely in a Computer Integrated Surgery application.

These customer requirements are based on positive and negative experiences with the existing MRC Software and also some new feature requests.

### **2. General Requirements**

#### **2.1. Scope**

The MRC-II System will:

- Provide real-time control of robot systems and their associated sensors.
- Support open interfaces to other types of systems, including other robot systems and sensing systems (for example, cisTracker).
- Define a recommended hardware implementation that can be used to quickly develop new robot systems.

#### **2.2. Levels of Access**

The system must support different levels of access. Some customers would like the system to provide a “black box” position server, whereas at the other extreme, customers would like to change the servo control algorithm. These different levels of access imply the following:

- An API that provides an interface to the entire (black-box) MRC-II robot system.
- A configuration/customization mechanism that allows researchers to easily extend the MRC-II System to new hardware (e.g., robots, sensors).

- An interface and/or set of tools that allows researchers to add new real-time functions or replace existing ones (for example, add a new sensor-based motion or change the servo control algorithm).

### **2.3. Operating System Portability**

Because the MRC-II System must interface with other systems, it must be capable of running on different types of computers. For example, it may be necessary to run MRC-II and the companion system on the same computer to meet time-critical performance requirements for the integrated system.

Therefore, the MRC-II System must support the following operating systems that are currently in use (see the Constraints section for compiler information):

- Microsoft Windows (e.g., Windows NT)
- Unix (e.g., Linux)

### **2.4. Hardware Support**

Different types of interface hardware must also be supported. MRC includes support for the following hardware:

- Motion Engineering (MEI) PCX/DSP motion controller
- LARS proprietary shared memory interface
- ATI force sensor
- Various tracking devices (via cisTracker package)

Other hardware currently in use (but not yet supported by MRC) includes the following:

- Servo To Go
- Measurement Computing I/O boards
- Phantom haptic interface

The MRC-II System must be extensible to support new hardware, including a recommended hardware implementation that has yet to be defined.

### **2.5. Robot Support**

The MRC-II system must continue to support different robots. This is analogous to supporting different hardware – even though the system is isolated from the robot by the interface hardware, the overall system is generally affected by the physical properties of the robot (e.g., kinematics, dynamics).

### **3. Functional Requirements**

#### **3.1. Network Interface**

The network support (i.e., supporting robot clients and servers) is a key positive feature of the MRC Software and should be maintained in the MRC-II System. It should, for example, allow the construction of a teleoperated system with multiple masters and slaves.

The MRC-II Software should provide a minimal library for robot clients (i.e., a library that is composed of only the necessary components and does not require hardware-related include files). This would improve the ability to port the client software to other operating systems, such as Solaris.

There has been some discussion about a CORBA interface for the robot. This is currently not considered to be a requirement, but can be considered as an alternative to the current client/server implementation.

#### **3.2. Robot Kinematics**

The system must continue to provide forward and inverse robot kinematics but should address deficiencies in the current implementation, which include:

- Inconsistencies in the results produced by the forward and inverse kinematics (e.g., getting a different joint position when computing forward kinematics followed by inverse kinematics).
- Difficulty in specifying “user preferences” to manage kinematic redundancy, multiple configurations, singularities, etc.

The system should also provide the data required for graphical animation of the robot position.

#### **3.3. Robot Initialization and Calibration**

The system should provide methods for initializing (homing) the robot and for calibrating robot kinematic parameters, including joint offsets. In many cases, a quick calibration method would significantly improve the accuracy of a robot system that is frequently disassembled (for example, to install different tools). There are several possible aspects to this task:

- Provide hardware and software interfaces for additional sensors (e.g., limit switches, redundant sensors).
- Provide data collection and parameter estimation algorithms.
- Ensure that the kinematic models support the calibration parameters.

### **3.4. System Configuration**

The MRC Software uses a configuration file to define robot-specific parameters, such as kinematic parameters and control gains. The MRC-II System should also include a configuration mechanism. The configuration file format, however, should be revised so that it can be more robustly parsed.

### **3.5. Trajectory Control Loop**

The MRC Software lacks a trajectory control loop – at best, it provides coordinated joint-space motions using that functionality in the MEI motion controller. In several cases, such as for force control, users have implemented their own loops using timer events or foreground loops (possibly using the sleep command to affect timing). The MRC-II System should provide a trajectory control loop for customers and give them the ability to customize it for new types of motions.

### **3.6. Servo Control Loop**

The MRC Software does not currently contain any servo control loops because they are provided by the interface hardware, such as the MEI controller. The MRC-II System should continue to support servo control loops on external hardware, but should also provide servo control loops that customers can use (and modify if desired).

### **3.7. Error Handling**

The MRC-II System must improve the error handling; in particular, for a coordinated motion, the system should stop all axes if one axis stops due to an error. The system must also include a mechanism for error reporting so that (asynchronous) errors in the real-time system are propagated to the application.

### **3.8. Safety Systems**

The MRC-II System should provide safety systems such as external watchdogs that disable motor power if they are not refreshed within a specified time. Other things to consider are support for redundant sensors (e.g., redundant encoders) and checking of hardware status (e.g., amplifier errors).

### **3.9. Data Collection**

The system must include data collection functions that allow real-time data to be collected for on-line or off-line analysis. This should also help with the “tuning” of servo control loops.

### **3.10. Status Queries**

The system should include status query functions that report important information about the system, such as the following:

- whether the robot is powered on
- whether the robot has been zeroed
- whether there are any errors

## **4. Performance Requirements**

### **4.1. Loop Times**

There are no specific timing requirements imposed by customers. The general expectation for typical loop times is as follows:

- Trajectory Control Loop – about 10 milliseconds (100 Hz)
- Servo Control Loop – about 1 millisecond (1 KHz)

The system should be capable of providing these loop times while still maintaining enough capacity to handle any foreground tasks. Furthermore, there is no consensus on whether these loops require hard real-time performance (guaranteed execution within the time constraints) or soft real-time performance (execution usually within the time constraints). It is expected that some customers will prefer to accept soft real-time performance (e.g., servo control under Windows NT) rather than having to switch to a real-time operating system. The MRC-II System should therefore allow both types of performance.

### **4.2. Data Access**

The MRC-II System should allow efficient access to data. No specific timing requirements have been specified.

### **4.3. Accurate Velocity Determination**

The MRC-II System should support accurate determination of velocity from incremental encoder feedback, assuming proper hardware support. In most cases, velocity is determined by counting the number of encoder transitions over a fixed time (i.e., taking the difference in position between two samples). This works well at higher speeds when there are many encoder counts, but does not have good resolution at extremely low speeds. An alternate method is to measure the time between two consecutive encoder transitions. This works well at low speeds, but does not have good resolution at higher speeds and requires special hardware to make the measurement. The ultimate solution is to provide both methods for determining the velocity and then use the one that is most appropriate for the current speed.

## **5. Hardware/Software Constraints**

The MRC-II Software will use the CIS Library (e.g., cisVecs, cisClass, cisTracker, etc.).

As noted in the General Requirements, the MRC-II System must support multiple operating systems, interface hardware and robots.

The MRC-II Software should be supported for the following compilers:

- Visual C++ for Windows
- Gnu C++ (gcc) for Linux
- Intel C++ for Windows or Linux

## **6. Change History**

<b>Rev</b>	<b>Date</b>	<b>Description</b>
1	2/10/03	Initial Version (changes tracked by date for now)