

**Software Requirements Specification (SRS)
for MRC-II System**

Rev 1

Feb 10, 2003

**Johns Hopkins University
Center for Computer-Integrated Surgical
Systems and Technology (CISST)**

Software Requirements Specification (SRS) for MRC-II System

1.	Objective.....	3
2.	References.....	3
3.	System Description	3
3.1.	User Application.....	4
3.2.	Robot Client	4
3.3.	Robot Server.....	4
3.4.	Kinematics Module.....	4
3.5.	Trajectory Control Loop	5
3.6.	Servo Control Loop	5
4.	External Interface Requirements.....	5
4.1.	Operating System	5
4.2.	I/O Hardware.....	5
5.	Functional Requirements	6
5.1.	Robot Client/Server	6
5.2.	Kinematics Module.....	9
5.3.	Trajectory Control Loop	9
5.4.	Servo Control Loop	10
6.	Performance Requirements.....	11
6.1.	Robot Client/Server	11
6.2.	Kinematics Module.....	11
6.3.	Trajectory Control Loop	12
6.4.	Servo Control Loop	12
7.	Safety Requirements.....	12
7.1.	Robot Client/Server	12
7.2.	Kinematics Module.....	13
7.3.	Trajectory Control Loop	13
7.4.	Servo Control Loop	13
8.	Module Interface Requirements.....	14
8.1.	User Application and Robot Client/Server	14
8.2.	Robot Client and Robot Server	14
8.3.	Robot Server or Trajectory Control Loop and Kinematics Module.....	15
8.4.	Robot Server and Trajectory Control Loop	15
8.5.	Trajectory Control Loop and Servo Control Loop.....	16
9.	Design Constraints	16
9.1.	Operating Systems.....	16
9.2.	Programming Language.....	17
9.3.	Software Reuse.....	17
10.	Change History.....	17

1. Objective

This document defines the Software Requirements Specification (SRS) for the MRC-II System (Modular Robot Controller – II). The MRC-II System will provide a real-time robot control system for research in Computer-Integrated Surgery and will replace the current MRC software.

In general, the purpose of an SRS is to define software requirements, not software design details. In this project, a small amount of high-level design has been performed to decompose the software into logical modules, as described in the System Description.

The External Interface Requirements define the interaction of the entire MRC-II Software with external hardware and software.

For each module, the requirements are then separated into the following categories:

- Functional Requirements
- Performance Requirements
- Safety Requirements

The Module Interface Requirements define the internal interfaces between the software modules.

The Design Constraints specify certain conditions that are imposed on the design.

2. References

[Customer Requirements for MRC-II System](#)

3. System Description

The MRC-II System provides a real-time robot control system for research in Computer-Integrated Surgery. It is intended to control a number of different robots, using different hardware interfaces and operating systems. It will be implemented as a distributed system so that multiple robots and devices can be interconnected over a network. The software will provide an Application Programming Interface (API) to the entire robot system, but will also allow researchers to modify internal components, such as the high-level motion controller and the low-level control algorithm.

The following sections define the high-level modules that are shown in Figure 1.

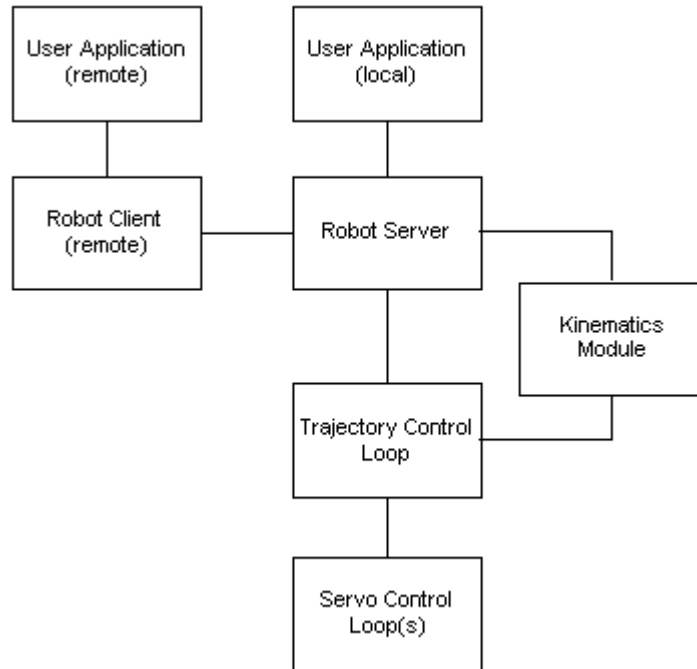


Figure 1. Module Interconnection Diagram

3.1. User Application

The User Application is the process (or processes) that makes requests to, and receives information from, the MRC-II System. This application may exist remotely (on a different machine) or locally (on the same machine).

3.2. Robot Client

The Robot Client allows a remote User Application to make requests to the Robot API. This could be implemented as a proxy object that packetizes the request, sends it to the Robot Server and then receives the result via the network.

3.3. Robot Server

The Robot Server provides the Robot API and associated non-real-time functions. This API is directly accessible to local User Applications and indirectly accessible (via the Robot Client) to remote User Applications.

3.4. Kinematics Module

The Kinematics Module implements the forward and inverse kinematics for the particular robot. It is shared between the real-time software (Trajectory Control Loop) and non-real-time software (Robot Server).

3.5. Trajectory Control Loop

The Trajectory Control Loop is a periodic real-time loop (on the order of 100 Hz) that coordinates all robot joints. It receives joint status and sensor information from the Servo Control Loop(s). It converts the specified trajectory (motion command) to intermediate goal positions for each joint.

3.6. Servo Control Loop

The Servo Control Loop is a periodic real-time loop (on the order of 1000 Hz) that controls one or more robot joints. There may be multiple Servo Control Loops, distributed among different processors, in cases where one processor does not have enough computational power or (more likely) enough I/O capability. The Servo Control Loop(s) receive joint goals from the Trajectory Control Loop and interpolate these goals to obtain joint setpoints. The Servo Control Loop could perform closed-loop control (e.g., PID) or open-loop control (e.g., stepper motors).

4. External Interface Requirements

4.1. Operating System

4.1.1. Portability

The software shall be designed to be as independent from the operating system as possible. Specific system details shall be encapsulated in a set of classes (derived from an abstract base class) that isolate them from the rest of the software.

4.1.2. Conditional Compilation

Conditional compilation directives shall be used to prevent the compilation of functions/modules that are not relevant to the current operating system.

4.2. I/O Hardware

4.2.1. Portability

Interface hardware shall be encapsulated in a set of classes that isolate hardware specifics from the rest of the software. In particular, interfaces for specific hardware boards shall be implemented as derived classes of an abstract class.

4.2.2. Multiple Access

Any component of the MRC-II System can potentially interface to a piece of hardware. If more than one software component requires

hardware access, it is necessary to provide mutual exclusion mechanisms.

One method that avoids mutual exclusion problems is to only allow the highest priority (fastest) process to directly interface with the hardware. Lower priority processes can make appropriate requests to the higher priority process or read data from shared memory.

5. Functional Requirements

5.1. Robot Client/Server

5.1.1. Distributed Model

The system shall allow distributed computing, where the robot may be local (on the same computer) or remote (via a network connection).

One possible implementation is to use a proxy object – the proxy object (client) would communicate with the actual object (server) via the network. This is similar to the current `mrcApiRemote` (proxy) and `mrcApiLocal` classes. Another possibility is to use CORBA.

5.1.2. Robot Definitions

The system shall support the concept of a “composite robot”, i.e., robot that is created from an ordered composition of other robots. For example, the 3 base translations and the RCM of the Steady Hand robot can be considered as two separate robots, as well as one combined robot.

5.1.3. Position/Velocity Query Commands

The API shall provide several position/velocity query commands that return the robot’s position/velocity in joint coordinates or as a frame. There may need to be more than one type of frame query command – for example, one to return the frame corresponding to the RCM point and one to return the frame corresponding to the tool (e.g., needle tip). Note that intermediate frames can be obtained by querying the position/velocity of a component robot (for example, to get the frame corresponding to the end of the 3 base translations).

There will also be different position/velocity query commands corresponding to positions/velocities computed from the joint setpoints, the primary sensor feedback or the redundant sensor feedback (if present).

5.1.4. External Sensor Query Commands

The API shall provide query commands for any external sensors, such as force sensors, that are configured with the robot (and are therefore read by the robot software).

5.1.5. External Sensor Update Commands

The API shall provide update commands for external sensors that are not directly read by the robot software. This supports real-time robot control based on sensor data that are read by an external software module (e.g., the User Application).

5.1.6. Status and Error Query Commands

The API shall provide query commands for the overall system status/error as well as for the status/error of individual joints. See the Real Time Error Handling section for additional specifications.

5.1.7. System Configuration Commands

The API shall include commands that allow a robot system to be configured from a configuration file. There shall also be a command to query a robot for its configuration data. Configuration data shall include the following items:

- Robot name and type
- External hardware definitions (e.g., board types)
- Kinematic parameters, including joint zero offsets
- Joint position, velocity and acceleration limits
- Servo control parameters
- Safety thresholds (e.g., max. tracking error)
- Sensor definitions, including conversion factors

5.1.8. Motion Commands

The API shall provide the following motion commands:

- Zero command to initialize the position
- Absolute and relative joint position/velocity motions
- Absolute and relative straight-line Cartesian position/velocity motions (vector interpolation for orientation changes)
- Force controlled motions (if force sensor present)

The API shall be easily extensible to other types of motion (e.g., circular motion, spline) and other types of sensor-based motion (e.g., visual servoing).

5.1.9. Real Time Commands

In addition to the Motion Commands, the following real-time commands shall be implemented:

- Pause to halt the current robot motion
- PowerOff to turn off motor power
- Resume to restart the paused robot motion
- WaitForDone to wait for motion to finish.

5.1.10. Kinematics Commands

The API shall include interface functions for the Kinematics Module (see below).

5.1.11. Data Collection Commands

The API shall include functions to setup real-time data collection (e.g., select signals of interest), start and stop data collection, and transfer the real-time data to a buffer. The API shall also allow the user to specify how a full buffer should be handled (e.g., overwrite oldest data or stop collecting data).

5.1.12. Real Time Error Handling

The system shall have a mechanism for obtaining error information from the real-time software and reporting these errors to the application. One possibility is to raise C++ exceptions.

5.1.13. System Performance Monitoring

The API shall include functions to query the system for performance monitoring information and (optionally) reset the data. For example, the real-time system may maintain statistics regarding loop execution times, control algorithm performance, measured physical parameters and other factors.

5.1.14. Message Logging System

The software shall support the current class-based message logging system.

5.2. Kinematics Module

5.2.1. Forward and Inverse Kinematic Functions

The Kinematics Module shall provide forward and inverse kinematic functions that are consistent with each other. These functions shall perform limit checking to ensure that all results are within the workspace.

5.2.2. Workspace Error Reporting

The Kinematics Module shall report workspace errors (i.e., result outside the robot's workspace) in a configurable manner. Options include:

- Raise an exception
- Leave the (out of workspace) result and return an error code.
- Set the result to the closest point in the workspace and return an error code.

5.2.3. Kinematic Configurations

The Kinematics Module shall provide functions to set "user preferences" to handle kinematic redundancy, multiple solutions, etc.

5.2.4. Kinematic Parameters

The Kinematics Module shall provide functions to query kinematic parameters and modify them based on calibration results.

5.3. Trajectory Control Loop

5.3.1. Loop Timing Information

The Trajectory Control Loop should measure its execution time and maintain statistical information (e.g., minimum, maximum and average execution times).

It can be difficult to obtain low-latency, high-resolution timing information on some platforms, such as Windows. One option to consider is the Pentium RDTSC instruction to read the chip's time stamp counter.

5.3.2. Sensor Data Acquisition

The Trajectory Control Loop shall obtain sensor data from all configured sources, including all joints being controlled by Servo Control Loops. The configured sources could also include external sensors such as force sensors.

The system design should ensure that sensor data is sufficiently synchronized. For example, it is important that joint encoder values be sampled at approximately the same time; otherwise, the robot's resolved position could be inaccurate. It may be less important for force data to be sampled at exactly the same time as the joint encoder data.

5.3.3. Motion Command Processing

The Trajectory Control Loop shall be responsible for the processing of motions specified by the Robot Server. Each invocation of the Trajectory Control Loop shall perform an iteration of the current motion until that motion is completed. The Trajectory Control Loop will then read the next motion from the motion queue and commence its execution.

The motion should be defined generally, so that standard position motions (e.g., joint-based or Cartesian motions) can be implemented as well as sensor-based motions (e.g., force control). Furthermore, it should be easy to add new motion types to the system, including motions that use sensor data that is periodically provided by external software.

5.3.4. Real Time Command Processing

The Trajectory Control Loop shall check a command mailbox for any Real Time Commands from the Robot Server. Examples of Real Time Commands include pausing motion and disabling motor power.

5.3.5. Data Collection

The Trajectory Control Loop shall, at the request of the Robot Server, buffer data at a specified rate (i.e., a factor of its loop frequency). For example, the Robot Server may direct the Trajectory Control Loop to store the commanded joint position and the measured joint position at each invocation.

5.4. Servo Control Loop

5.4.1. Loop Timing Information

The Servo Control Loop should measure its execution time and maintain statistical information (e.g., minimum, maximum and average execution times).

5.4.2. Sensor Data Acquisition

The Servo Control Loop shall obtain sensor data from all configured sources. The joint position data should be sampled at approximately the same time; otherwise, the robot's resolved position could be inaccurate.

5.4.3. Velocity Determination

If direct velocity feedback is not available, the Servo Control Loop shall compute it by differentiating successive position samples (and possibly applying a filter). This method works well at high speeds, when there are significant changes in successive positions.

If supported by the hardware, the Servo Control Loop shall also measure the time between two consecutive encoder transitions, which provides a more accurate estimate at low speeds. In this case, the software shall determine the best velocity value by considering both measurements.

5.4.4. Joint Goal Interpolation

The Servo Control Loop shall receive joint goals at the trajectory control rate, which is about an order of magnitude slower than the servo control rate. The Servo Control Loop shall therefore interpolate between the joint goals. This interpolation may be linear, quadratic or any other reasonable alternative.

5.4.5. Closed Loop Control

The Servo Control Loop shall provide a control algorithm for a closed loop control system, unless the system is configured for open loop control (e.g., for stepper motors). The software shall provide a default PID control algorithm, but it should be possible to add new types of control algorithms.

5.4.6. Data Collection

The Servo Control Loop shall, at the request of the Robot Server (via the Trajectory Control Loop), buffer data at a specified rate (i.e., a factor of its loop frequency). This data would be useful, for example, for tuning the control gains.

6. Performance Requirements

6.1. Robot Client/Server

None.

6.2. Kinematics Module

The kinematics functions must execute within the time constraint of the Trajectory Control Loop. In general, this should not be a problem unless an iterative solution is required for the inverse kinematics of a serial manipulator or the forward kinematics of a parallel manipulator.

6.3. Trajectory Control Loop

Execution of the Trajectory Control Loop shall be initiated by a periodic event, such as a timer interrupt. It is expected that this event will occur on the order of 100 Hz (e.g., in the 50 Hz – 200 Hz range). The loop shall be designed so that it completes execution before the next event occurs; if not, an overrun condition should be noted. The response to an overrun condition should be configurable – it could include ignoring the overrun or activating the safety response (e.g., disabling motor power).

One method for overrun detection is to define a flag that is set at the beginning of the loop and cleared at the end. If the flag is already set when the loop starts, then an overrun has occurred. Note, however, that this method only works if events are enabled during execution of the real-time loop.

6.4. Servo Control Loop

The Servo Control Loop has the same performance requirements as the Trajectory Control Loop (above), except that the expected frequency is on the order of 1000 Hz (e.g., in the 500 Hz – 2000 Hz range).

7. Safety Requirements

7.1. Robot Client/Server

7.1.1. Single Controller Client

The client/server architecture should ensure that no more than one client can specify motion commands for the robot server. This is intended to prevent unexpected motions of the system.

7.1.2. Multiple Observer Clients

Additional clients can act as observers (e.g., receive robot state). To extend the safety checking of the system, observers shall be capable of initiating a safety response (i.e., disabling motor power).

7.1.3. Configuration File Parsing

The system configuration file format shall ensure that the parser correctly interprets all fields and that it can detect errors (e.g., invalid data). For example, this can be implemented by a tagged file format.

7.2. Kinematics Module

7.2.1. Workspace Limit Checking

The kinematics functions shall perform limit checking to ensure that all results are within the robot's workspace. The error response shall be as specified in the Functional Requirements.

7.3. Trajectory Control Loop

7.3.1. Sensor-Based Safety Checks

The Trajectory Control Loop may include safety checks based on data that it receives directly, such as from a force sensor or a redundant position measurement system. The software shall maintain statistical information about the sensor values that it checks (e.g., maximum and average values).

7.3.2. Motion Command Output Check

Because the system will support the easy addition of new motions, it shall perform a sanity check on the output of the motion controller. In particular, it shall check that the motion controller output does not violate the physical constraints (e.g., maximum joint velocity). This check may alternately be implemented in the Servo Control Loop.

7.3.3. Coordinated Safety Response

The Trajectory Control Loop shall ensure that all joint motions are stopped (and motor power disabled if necessary) if an error occurs on one robot joint. This prevents the robot from performing an unexpected motion due to one joint stopping while all other joints continue to move.

7.4. Servo Control Loop

7.4.1. Motor Power Control

The Servo Control Loop shall contain a mechanism for disabling motor power to ensure that the system enters a safe state when a critical error is detected.

7.4.2. Sensor-Based Safety Checks

The Servo Control Loop shall include safety checks based on data that it receives directly. For example, it could check motor currents, power amplifier temperature, supply voltage, limit switches, invalid encoder transitions, redundant sensor feedback or any other parameter that can be measured (depending on the external hardware). The software shall

maintain statistical information about the parameters that it checks (e.g., maximum and average values).

7.4.3. Maximum Tracking Error

The Servo Control Loop shall compare the commanded position to the actual (measured) position and raise an error if the difference is greater than a specified threshold. The software shall also maintain statistical information about the tracking error (e.g., maximum and average error).

7.4.4. Missing Joint Goals

The Servo Control Loop shall handle a missed joint goal from the Trajectory Control Loop in a configurable manner. The response could be to hold the most recent (position or velocity) goal or to disable the motor power and raise an error. The latter option provides a higher level of safety – in this case, the Servo Control Loop acts as an external watchdog for the Trajectory Control Loop. Compromises between these solutions are also possible: for example, the Servo Control Loop could disable motor power and raise an error if there are a specified number of missed joint goals within a specified period.

7.4.5. External Watchdog

The Servo Control Loop shall refresh an external watchdog if one is provided by the hardware. If the watchdog is not refreshed within a specified time, it shall disable motor power. Motor power shall remain disabled until the watchdog is explicitly reset.

8. Module Interface Requirements

8.1. User Application and Robot Client/Server

The User Application can have a local or remote interface to the system. This can be implemented via Client/Server objects or a CORBA interface. The Application Programming Interface (API) shall be the same regardless of whether it is local or remote. It shall consist of one or more objects, such as an `mrcRobot` class, that are provided by a static or dynamic library.

The Robot Client, if present, shall be provided by a separately-compiled library (static or dynamic) that includes only the files that are needed for the client side functionality.

8.2. Robot Client and Robot Server

The interface between the Robot Client, if it exists, and the Robot Server shall be via a network connection (`mrcNetwork` class). Note that the Robot Client is not used if the User Application is local.

8.3. Robot Server or Trajectory Control Loop and Kinematics Module

The Kinematics Module is shared between the Robot Server and the Trajectory Control Loop. The following implementations can be considered:

- Design the module in a thread-safe manner so that it can be called from both processes (e.g., avoid internal state data).
- Create a kinematics class but instantiate two kinematics objects (one for each process).

8.4. Robot Server and Trajectory Control Loop

This is called the Real Time Interface because it serves as the interface between the non-real-time part of the system (Robot Client/Server and User Application) and the real-time part of the system (Trajectory Control Loop and Servo Control Loop). It can be assumed that the Robot Server and the Trajectory Control Loop will reside on the same computer and that the Trajectory Control Loop will be a higher priority task.

8.4.1. Data Consistency

The interface shall prevent inconsistent updates from occurring (e.g., prevent the Trajectory Control Loop from reading/writing data in the middle of a write/read by the Robot Server). Because the Trajectory Control Loop is a real-time process, the data transfer mechanism should never block it, as could be the case with standard mutual exclusion semaphores.

8.4.2. Real Time Commands

The Robot Server shall write Real Time Commands to a mailbox that is emptied (processed) by the next iteration of the Trajectory Control Loop. Therefore, Real Time Commands shall not be queued. Note, however, that some Real Time Commands, such as WaitForDone (wait for motion to finish), can be implemented as a Robot Server loop that submits CheckIfDone commands to the Trajectory Control Loop.

8.4.3. Motion Commands

The Robot Server shall perform a trajectory planning function (if appropriate) and then provide the Motion Command to the Trajectory Control Loop. Motion Commands will be queued to allow composition of complex path motions and to allow the User Application to perform other tasks while the robot is moving. The Motion Command queue could be filled directly by the Robot Server or indirectly via a “queue” Real Time Command. The queue would be emptied by the Trajectory Control Loop.

8.5. Trajectory Control Loop and Servo Control Loop

This interface is between two periodic real-time processes, which may reside on the same machine or on different machines. There may also be multiple Servo Control Loops, in which case the Trajectory Control Loop must interface with each of them.

8.5.1. Periodic Data Transfers

The interface shall support periodic data transfers between the two loops at the frequency of the (slower) Trajectory Control Loop. The Trajectory Control Loop should have a lower priority than the Servo Control Loop (of course, the priority relationship is only meaningful if they run on the same machine). In general, the data transfer mechanism should never block the higher priority process (Servo Control Loop). It is also desirable to minimize or eliminate any blocking (waiting) in the lower priority process (Trajectory Control Loop).

8.5.2. Background Data Transfers

The interface shall support background data transfers between the two loops at the frequency of the (slower) Trajectory Control Loop. A background data transfer is one that occurs occasionally, in response to a request that originates in the Robot Server. For example, background data transfers would be used to update control gains, receive packets of buffered data and receive statistical performance information. The background data transfer shall be at a lower priority than the periodic data transfers.

8.5.3. Synchronization of Loops (optional)

For optimal performance, the Trajectory Control Loop and Servo Control Loop(s) should be synchronized. For example, if the Trajectory Control Loop runs at 100 Hz and the Servo Control Loop runs at 1000 Hz, there should always be 10 Servo Control Loops for every Trajectory Control Loop. This can be achieved by having one loop provide a synchronization pulse (e.g., interrupt) to the other, but this can be difficult to implement in practice given hardware and operating system constraints. The MRC-II System shall provide loop synchronization mechanisms if supported by the specific hardware and operating system.

9. Design Constraints

9.1. Operating Systems

The MRC-II Software shall, at a minimum, support the following Operating Systems:

Software Requirements Specification (SRS) for MRC-II System

- Microsoft Windows
- Linux

9.2. Programming Language

The MRC-II Software shall be implemented in C++ and shall work with the following compilers:

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

9.3. Software Reuse

The MRC-II Software shall use the CIS library.

10. Change History

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