AD_____

Award Number: W81XWH-05-2-0012

TITLE: Trauma Pod/Operating Room of the Future

PRINCIPAL INVESTIGATOR: Delbert Tesar, Ph.D.
Chetan Kapoor, Ph.D.
Chalongarh Pholsiri
Edwin Jung
Greg Giem
Jonathan Knoll

CONTRACTING ORGANIZATION: The University of Texas at Austin
Austin, TX 78758

REPORT DATE: February 2006

TYPE OF REPORT: Annual

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
01-02-2006	Annual	15 Jan 2005 – 14 Jan 2006

4. TITLE AND SUBTITLE

Trauma Pod/Operating Room of the Future

5a. CONTRACT NUMBER

5b. GRANT NUMBER
W81XWH-05-2-0012

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S) Delbert Tesar, Ph.D
. Chetan Kapoor, Ph.D.
Chalongarh Pholsiri
Edwin Jung, Greg Giem, Jonathan Knoll

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The University of Texas at Austin
Austin, TX 78758

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES
Original contains colored plates: ALL DTIC reproductions will be in black and white.

14. ABSTRACT

The University of Texas (UTA) has played a central role in the trauma pod project and has been responsible for: 1. systems engineering and design from a robotics perspective, 2. high fidelity 3D simulator, 3. motion planning software for the scrub nurse systems, and 4. the supervisory control system (SCS). In systems engineering and design, UTA specified a hierarchical control architecture for the trauma pod and also recommended a communications architecture that allowed peer-to-peer communications. Detailed task analysis for tool change, supply dispense and calibration was performed. This led to the identification of timing bottlenecks and modifications to trauma pod design. Another result of the task analysis was the identification of subsystem functional interfaces. Detailed layout and timing analysis was also performed in Q3 and Q4. This led to the specification of all subsystem position and orientation along with other coordinates related to robot motion. A high fidelity 3D simulator was delivered by Q2. This simulator supports integrated collision detection, multiple camera views, and software interfaces for other subsystems to send sensor data to it. First version of the motion planning software was done in Q1. This supported kinematic control functionality along with limit avoidance capability. Since then, obstacle avoidance and collision detection capability has been added and this software is currently controlling robot hardware. The supervisory control system is central to the operation of the trauma pod. It accepts commands from the user interface system. These commands are translated into subsystem specific commands by the SCS using a task execution engine that is built into the SCS. Till Q4, two versions of the SCS have been released. These included scripted task execution capability. Over the next year, pod level collision detection, automated task execution, alarm handling and cancellation, and calibration functionality will be added.

NOT PROVIDED

16. SECURITY CLASSIFICATION OF:				18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
					USAMRMC
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
U	U	U	UU	113	

Table of Contents

INTRODUCTION

The University of Texas (UTA) has played a central role in the trauma pod project. Specifically, it has been responsible for:

5. systems engineering and design from a robotics perspective,
6. high fidelity 3D simulator,
7. motion planning software for the scrub nurse systems, and
8. the supervisory controller.

This report describes the technical developments, including current status, limitations, and issues in each of the areas listed above.

BODY

UTA's work has been in four areas as outlined in the previous section. Accomplishments in each of these areas are discussed below:

Systems Engineering and Design

UTA has played a critical role in this area, providing design and engineering expertise leading to the development of the control architecture of the trauma pod, the detailed task analysis and functional interfaces for trauma pod subsystems, and the layout and timing analysis for all subsystems in the pod. These are further discussed below:

Control Architecture

This defines the command and control hierarchies between the various subsystems. The primary purpose of this architecture is to simplify the overall control system design by *separation of concerns*. This partitioning can be achieved based on functionality and timing requirements. For example, the low level servo control of a manipulator is generally the lowermost part of a manipulator control system versus the task planning system that runs at a slower rate and is at the highest level.

The two major options available for the control architecture were: hierarchical control system and a reactive, agent based control system. The former is better suited for well structured environments where there is a clear hierarchy of tasks to be performed and master-slave relationships are obvious. A reactive control system is better suited for highly autonomous systems that are sensor driven and where most subsystems have a peer-to-peer relationship. Based on the structured design of the trauma pod and little autonomy of the robotic systems, UTA proposed a hierarchical control architecture. In this architecture, the top most layer is the surgeon interface. Below that is the Supervisory Control System (SCS) and below the SCS are the other subsystems.

One limitation of the hierarchical control system is that it restricts peer-to-peer communication. As such, all communications have to be through by the SCS, thereby limiting efficiency. To work around this problem, the communication architecture that was proposed allowed for peer-to-peer communications. As such, if the Scrub Nurse System (SNS) wanted to communicate with the Tool Rack System (TRS), it could do so. This does break the hierarchy of the system, but

allowed efficiencies to be added. Overall, the system is hierarchical with some peer-to-peer communication where necessary. This architecture is shown in Figure 1.

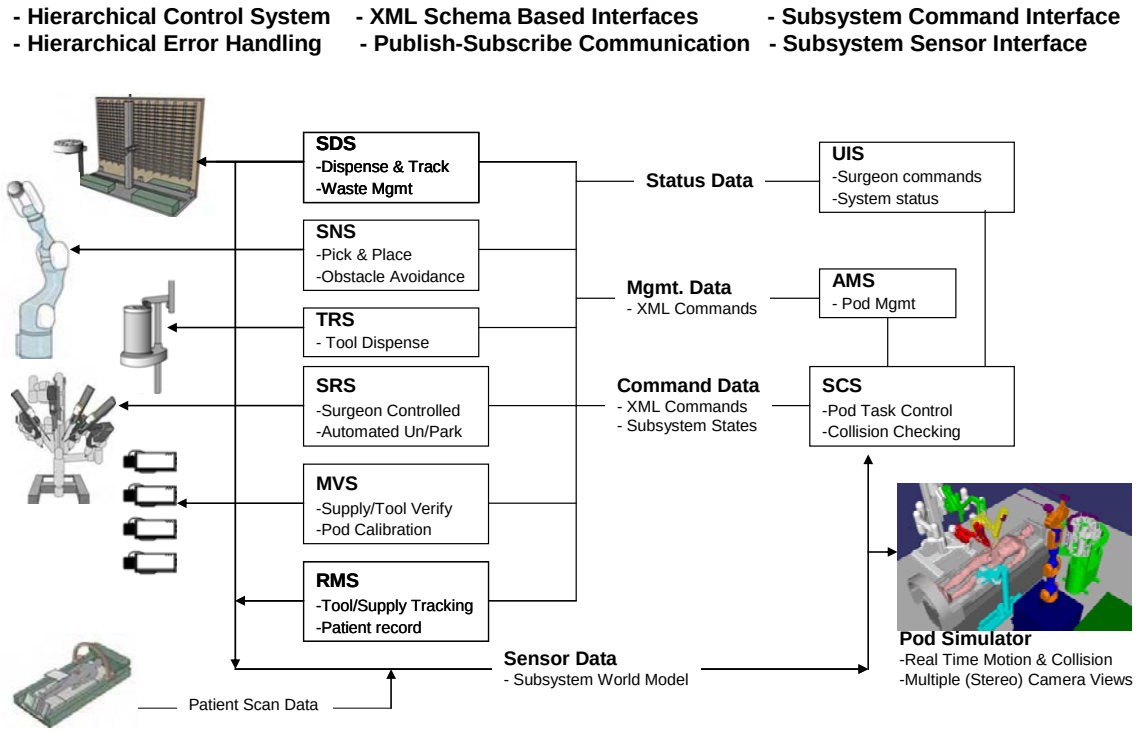


Figure 1: Trauma Pod Control, Communications and Software Architecture.

Task Analysis and Function Interfaces

This task involved studying the interactions between all subsystems that are required to meet the two primary tasks of the trauma pod. These are "tool request by the surgeon" and a "supply request by the surgeon". Besides these two surgeon level tasks, the setup task of trauma pod calibration also was analyzed.

The analysis of these tasks started with the critical requirement that the surgeon's request for tool or supply be met within 10 seconds. When the task analysis showed that it would be near impossible to meet the 10 second request, it was decided to make physical design changes to the trauma pod to meet the 10 second requirements. These changes included 2 grippers on the SNS and the addition of a Fast Cache (FC) to the Supply Dispensing System (SDS). These changes led to reanalyzing of the tasks and the development of new interaction diagrams. This was all done in Q2 and Q3 of this project. In Q4, specific emphasis was placed on the User Interface Subsystem (UIS) and as its details were worked out, the task analysis had to be redone to take into account complex interactions between the UIS and the Surgical Robot System (SRS).

In addition to the task analysis, this process also identified the syntax and semantics of the function interface for each subsystem. For example, to command the SNS to move to a certain Cartesian position, the command was identified to be *MoveToEEPose(….)*. This type of function

interface specification was done for the SNS, SDS, TRS, FC, etc. Results of the interaction diagram are shown in Appendix A.

Layout and Timing Analysis

This task involved coming up with a logical layout of the trauma pod and then performing a detailed kinematic analysis to identify Cartesian positions and orientations for each subsystem. This included the joint position configurations for the SRS, FC, and the SNS. This analysis had to be performed keeping in mind the speed of operations for the FC and the SNS, the clearance between each subsystem, the potential for collisions, and the ability of the SNS to meaningfully interact with relevant subsystems.

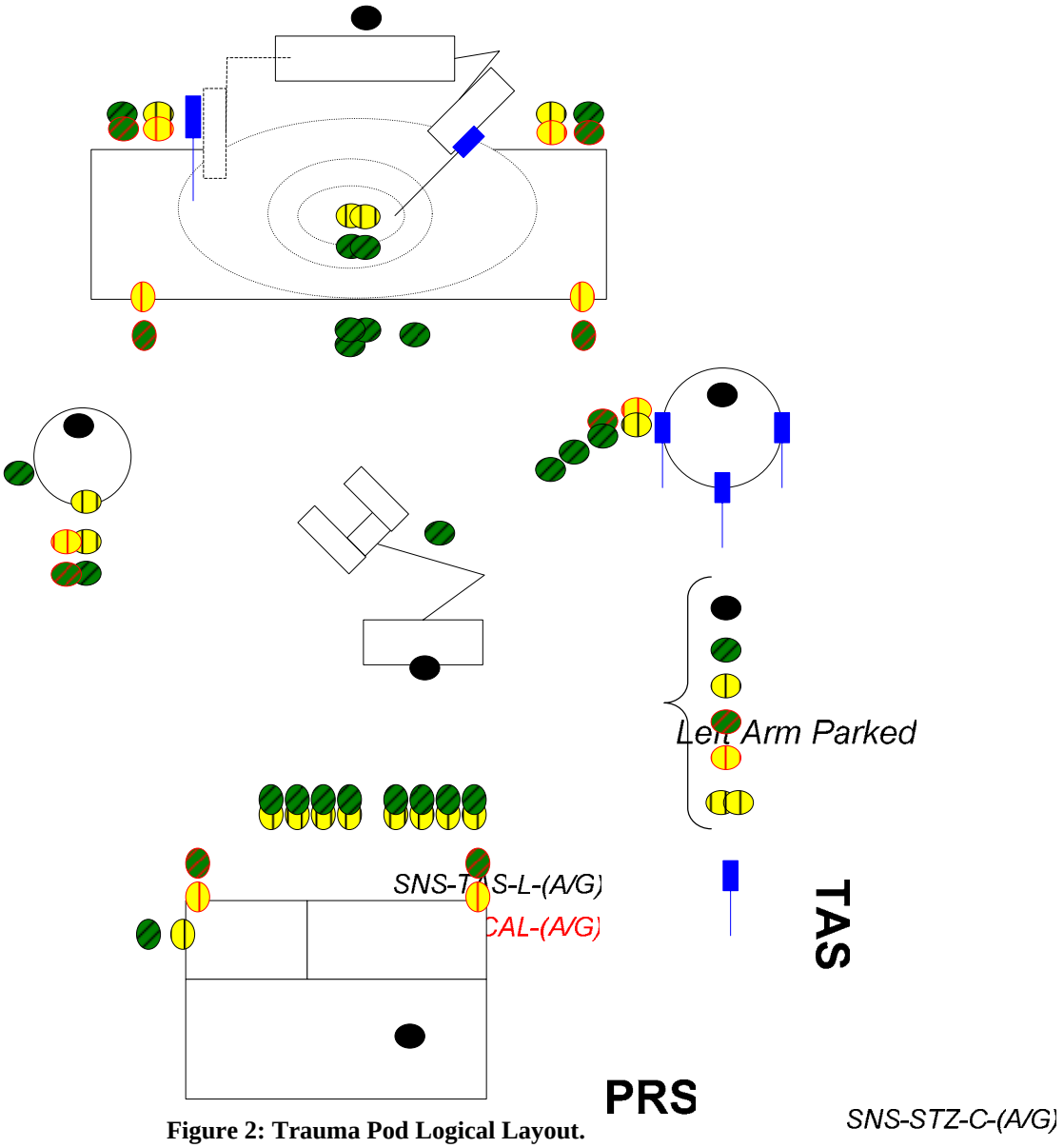


Figure 2: Trauma Pod Logical Layout.

This analysis was started in Q1 and reached its completion in Q4. Q1 basically identified the logical layout (shown in Figure 2). Detailed kinematic analysis was started in Q3 and it was iterated over as there were design changes made to the subsystems. Finally, all open issues were addressed by end of Q4. In addition to this analysis, it was noted that the kinematics of the FC were not clearly defined. UTA took the lead in this and further developed this in Q3 and Q4.

Summary of the timing analysis is given in Figure 3. Detailed results of the layout and timing analysis are given Appendix B. Details of FC kinematics are given in Appendix C.

Origin	Destination	Approx. Move time (s)
SNS-TRS	SNS-TOOL-W	0.8
SNS-TOOL-W	SNS-TAS-L	0.6
Left gripper	Right gripper	0.8
SNS-TAS-L	SNS-TRS	1.2
SNS-SDS-TZ	SNS-SUPP-W	1.4
SNS-SDS-SC	SNS-SUPP-W	1.1
SNS-SUPP-W	SNS-STZ-R	0.7
SNS-SUPP-W	SNS-SDS-W	1.6
SNS-SDS-TZ	SNS-FC	1.0

Note:
1. **Move times include move from approach to grab points (A-G) and/or G-A, where applicable. In these examples, move time for each of A-G and G-A is 0.2 second. Gripper opening/closing, force times not included.**
2. **Some trajectories involve move via which has not been optimized yet.**
3. **Obstacle free paths were used.**

Figure 3: SNS Move Times.

World Model

The purpose of the world model is to provide a geometric description of all the physical subsystems in the trauma pod and also provide configuration data for all relevant points in the trauma pod. This information is critical to the functioning of the trauma pod and is used by the following:

1) Calibration software.
2) SNS control software.
3) Collision detection software.
4) Path planning software.
5) Supervisory control system.
6) Fast cache control software.
7) SRS control software.
8) User interface software.
9) Simulator software.

The world model data is generated from multiple sources. First is the CAD model of the trauma pod. This provides geometric data associated with each subsystem. This geometric data is used to build the collision detection model, the obstacle avoidance model for the SNS, and the 3D

display for the simulator. The second source of world model data is the Layout and Timing Analysis that was discussed in the previous section. This analysis leads to the generation of Cartesian coordinates for each subsystem location and also for all the relevant points in space at which subsystems interact with each other. The third source of world model data is the specifications for each of the robot system in the trauma pod. Specifically, robot kinematic data (called DH parameters), their position, velocity, and acceleration travel limits, their joint positions at home configuration, etc. are all captured in the world model.
For the trauma pod, the functionality provided in the UTA developed OSCAR software was used to create the world model. OSCAR basically provides a means of describing a generic robotic workcell, of which the trauma pod is an instance. The basic structure in OSCAR to do this is hierarchical and is shown below:

Workcell (*1*)
- Manipulator (*0* through *n*)
  - DH Parameters
  - BasePose
  - Position, Velocity, Acceleration, Torque, Current Limits
  - Configurations of Interest
  - Frames of Interest
  - Obstacle Model
- Tool (*0* through *n*)
  - Tool Tip Pose
  - Obstacle Model
- Environment Obstacle Model (*1*)
- Frames of Interest (*0* through *n*)

There are detailed specifications under each of the headings shown above. For example, Obstacle Model has significant detail under it that allows the specification of geometric shapes and polytopes.

The world model is implemented using XML with a well defined schema that is used for validation. OSCAR provides C++ classes that allow the parsing and editing of the world model. The specific instance of the world model for the trauma pod is given in Appendix D.

3D Simulator

The trauma pod simulator serves three purposes. First, it provides a virtual environment for design and software testing, prior to the availability of the hardware. Second, it provides an alternative display with views from various vantage points of the trauma pod, and third, it is an excellent tool for training the surgeon on using the trauma pod. The purposes outlined above led to the development of the simulator that is described below:

Requirements: The simulator must provide a geometrically and kinematically correct animation of the pod and all subsystems. It must support an interface through which the individual subsystems can communicate to completely graphically describe any motions within the scene. It should provide a stereo output so the surgeon can practice working with the pod before the

hardware design is complete. Since the design of all the hardware is done in CAD, the simulation environment must also be able to import CAD models.

To meet the above requirements, we evaluated various commercial and research simulation environments. This analysis is summarized in Table 1. OpenSceneGraph software was selected for developing the trauma pod simulator. Its main advantages are that it's a package of C++ libraries and as such is much more flexible than any of the other possibilities. It is also significantly faster than the alternatives. Its biggest downside is the fact that it requires higher programming effort than the other packages

Evaluation Criteria / Software	Graphics Update Rate	Applications Programming Interface	Stereo	Development Cost-Time	Purchase Cost
Solidworks	unacceptable	Visual Basic	Supported	High	High
Roboworks	adequate	Network Interface	Not Supported	Minimal	Low
OpenSceneGraph	good	C++ Libraries	Supported	Medium	Free
Webots	unacceptable	C++	Not Supported	Medium	Low

Table 1: Analysis of Simulator Choices.

Design: The basic OpenSceneGraph framework is hierarchical – a tree structure with one root node with any number of children, each of which also has children, etc. The simulator makes use of this framework by adding Robot and Camera nodes used within the scene. The robot models are all generated automatically from the world model data file using the included robot DH-Parameters and STL file paths. The locations for all cameras are also included within the world model and can be switched using the user interface. All geometry in the environment is defined in the world model file and loaded automatically by the simulator. The frames of interest and obstacle models are also included within the world model file and loaded automatically, and each can be toggled using the user interface.

The simulator has 3 parallel software threads running at all times, and a fourth if collision detection is being used. The dispatcher thread receives XML messages on the ModelUpdateIf's and converts them into a current snapshot of the subsystem that sent the message. This data is posted to a shared data object that is constantly being accessed by the 3D graphics update thread, and the collision detection thread if active. There is also the main thread which starts each of the other threads and provides a user interface for manipulating the view and changing the simulator options. This setup is shown in Figure 4.

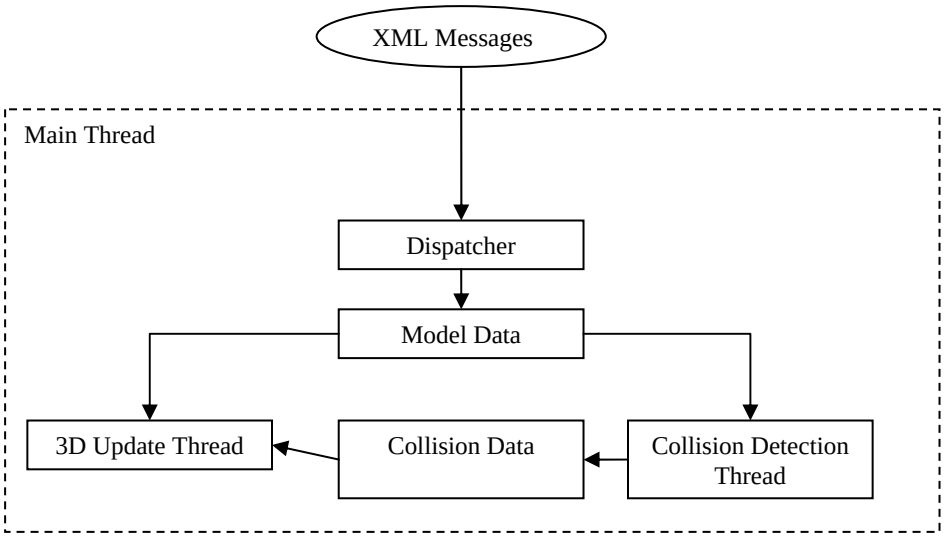


Figure 4: Simulator High Level Software Design.

Implementation: The simulator was written entirely in C++ using libraries from OpenSceneGraph (openscenegraph.org) and OSCAR (www.robotics.utexas.edu). OpenSceneGraph was used for the graphics while OSCAR was used for reading and parsing the world model data file and to provide basic math computation functionality. The collision detection code was included in the simulator project to make integrated collision detection an available runtime option. The entire project was about 10,000 lines of code, not including the test application. The test application used OSCAR code to plan some simple motions, and then send appropriate XML messages using spread communications infrastructure.

Performance: The simulator updates at a rate that varies depending on the density of triangles in the current view. In high density areas, it updates at 20Hz; and at low density areas it updates at about 60Hz. This is unaffected by the usage of the collision detection thread on the computer used for the tests, a dual xeon machine with 2gb ram and an NVIDIA quatro video card. Some screenshots of the simulator are shown in Figure 5 through Figure 11.

Status: Initially, two versions of simulators were developed. These were a RoboWorks based simulator and then the OSG based simulator that is described in this section. The RoboWorks based simulator was ready in Q1 and OSG based simulator was delivered in Q2. Since then, regular updates to the OSG simulator have been provided. These include changes to the simulator every time there is a CAD model change and discovery and fixing of software bugs.

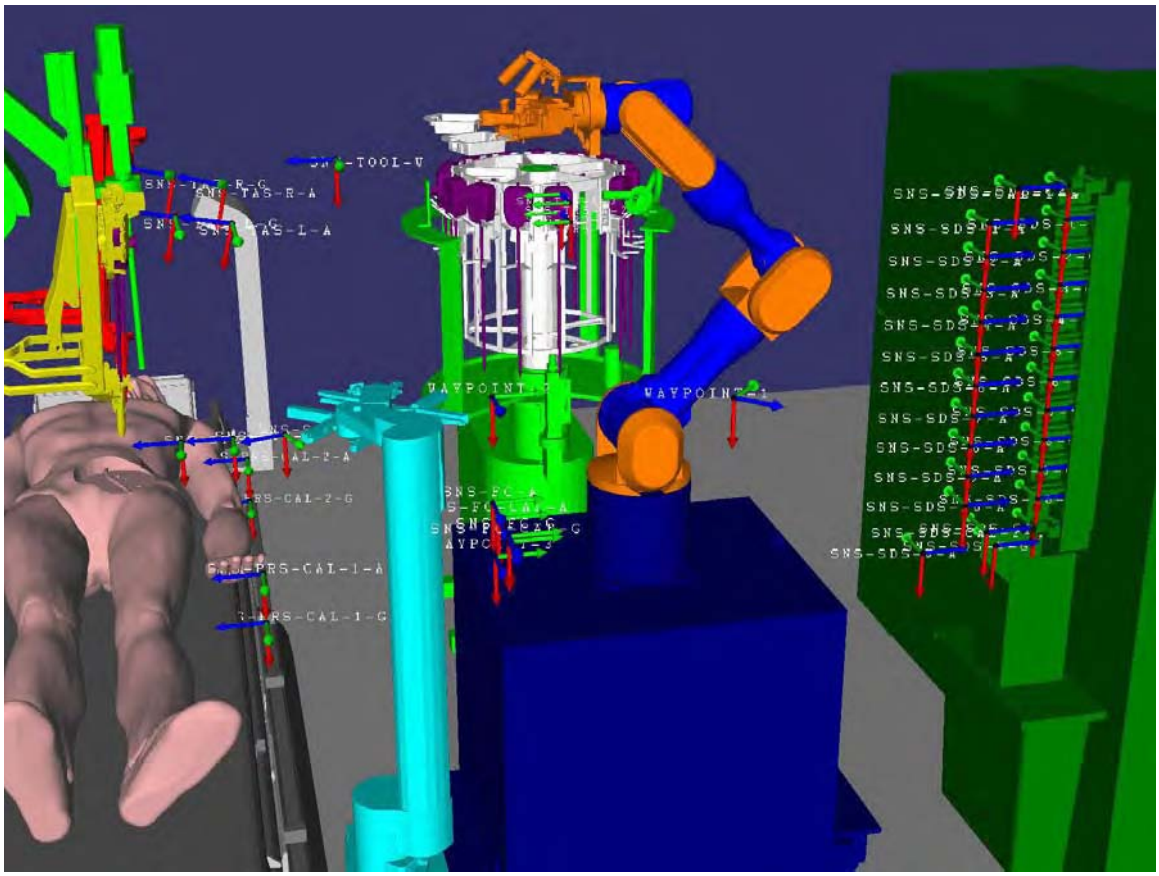


Figure 5: Frames of Interest with Labels.

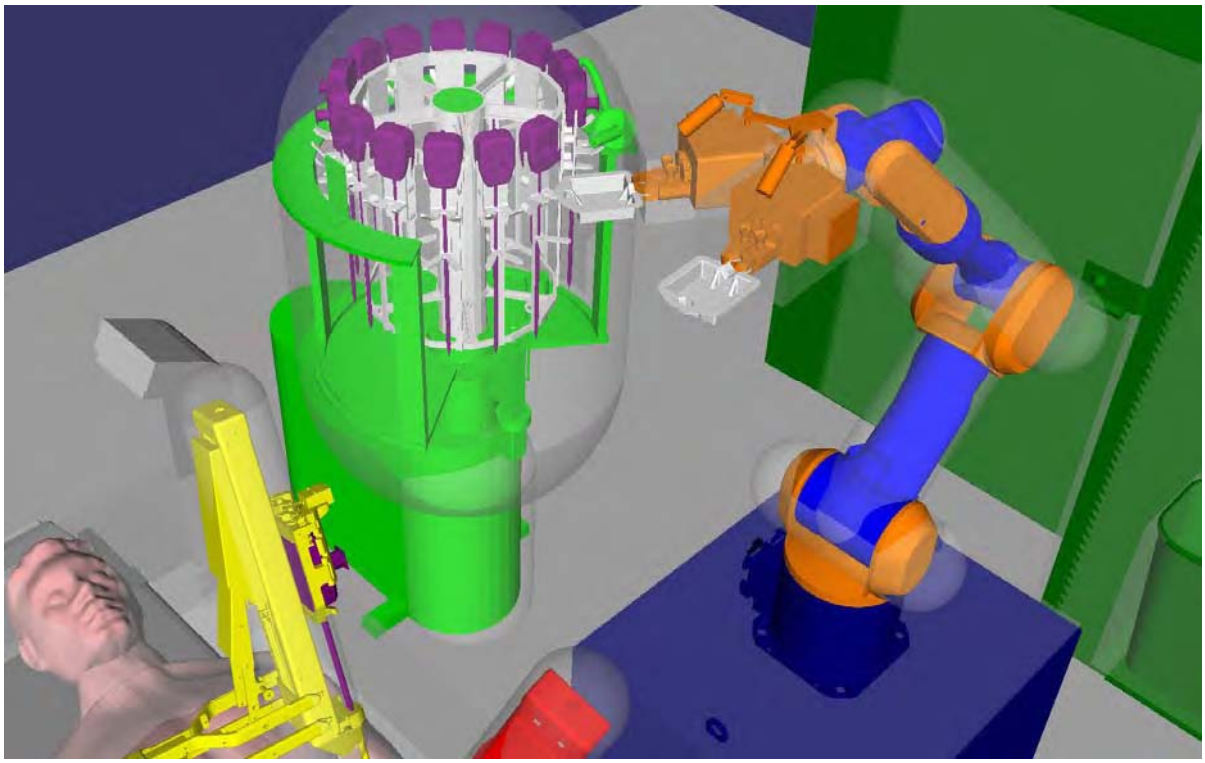


Figure 6: Obstacle Model Envelopes.

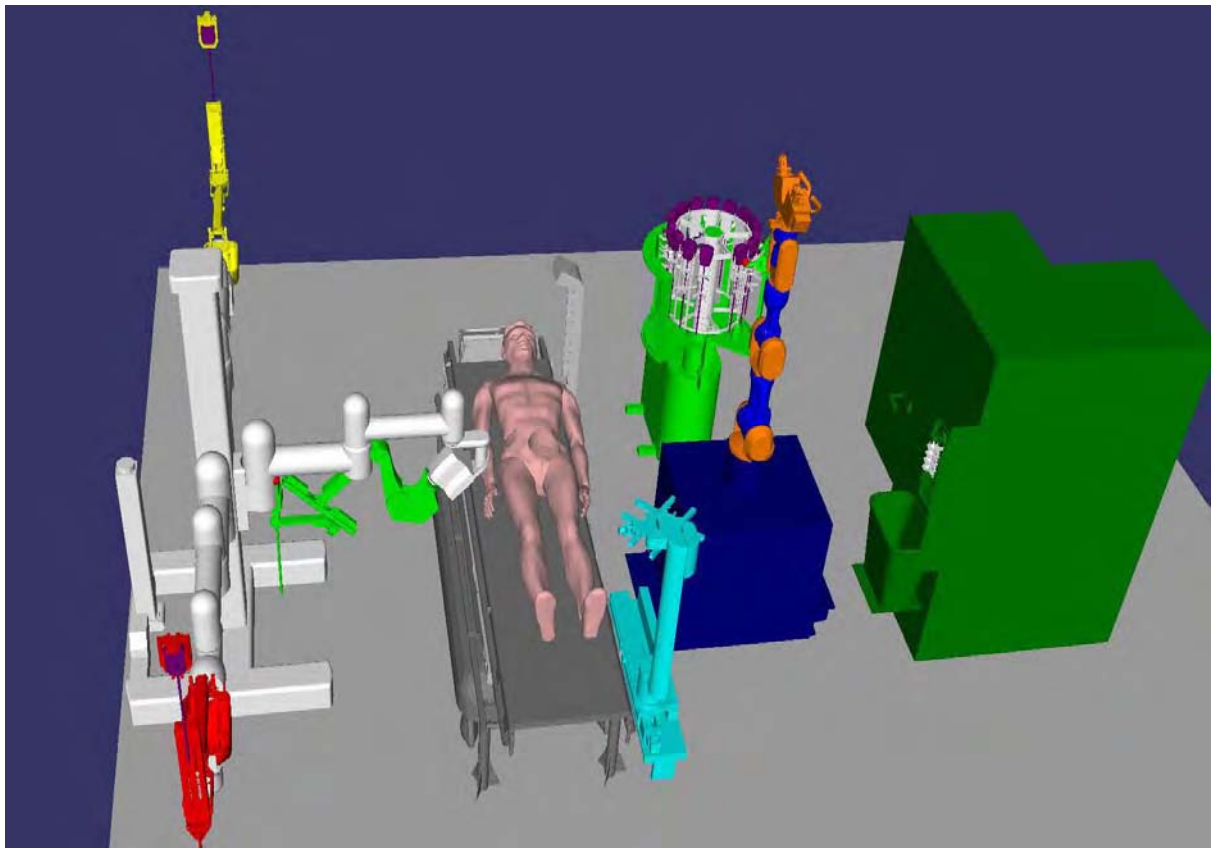


Figure 7: Default Scene Position.

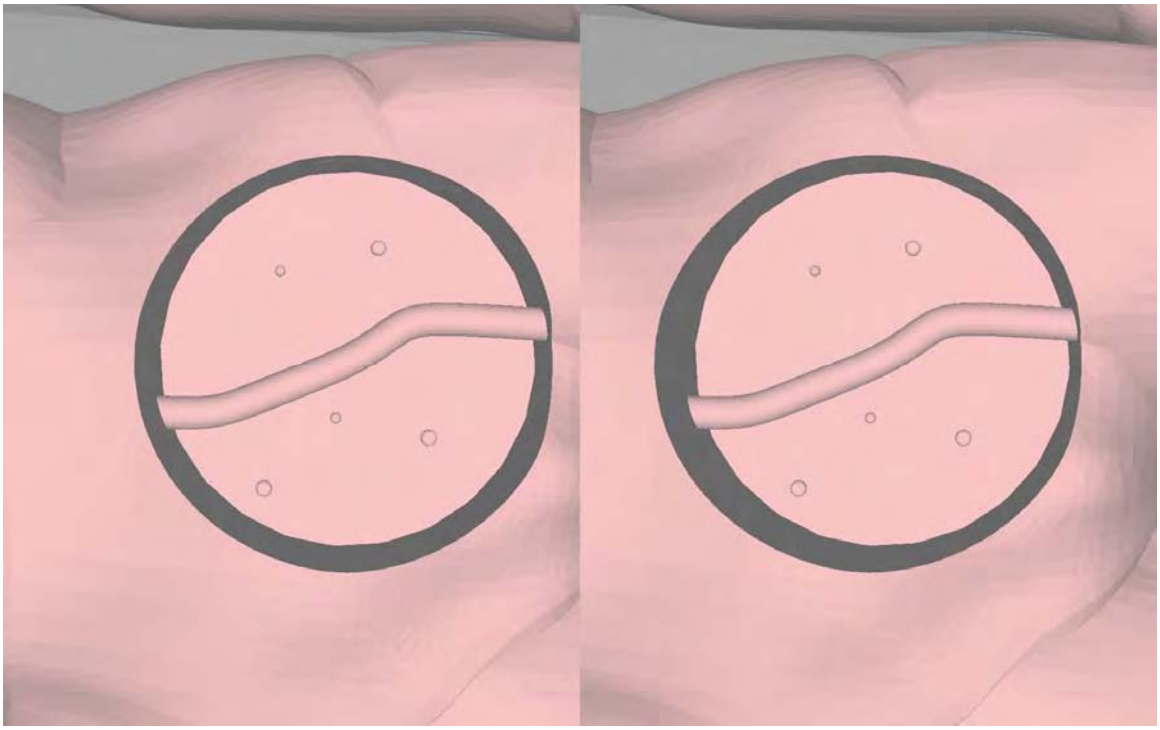


Figure 8: Split-Screen Stereo View of Surgical Zone.

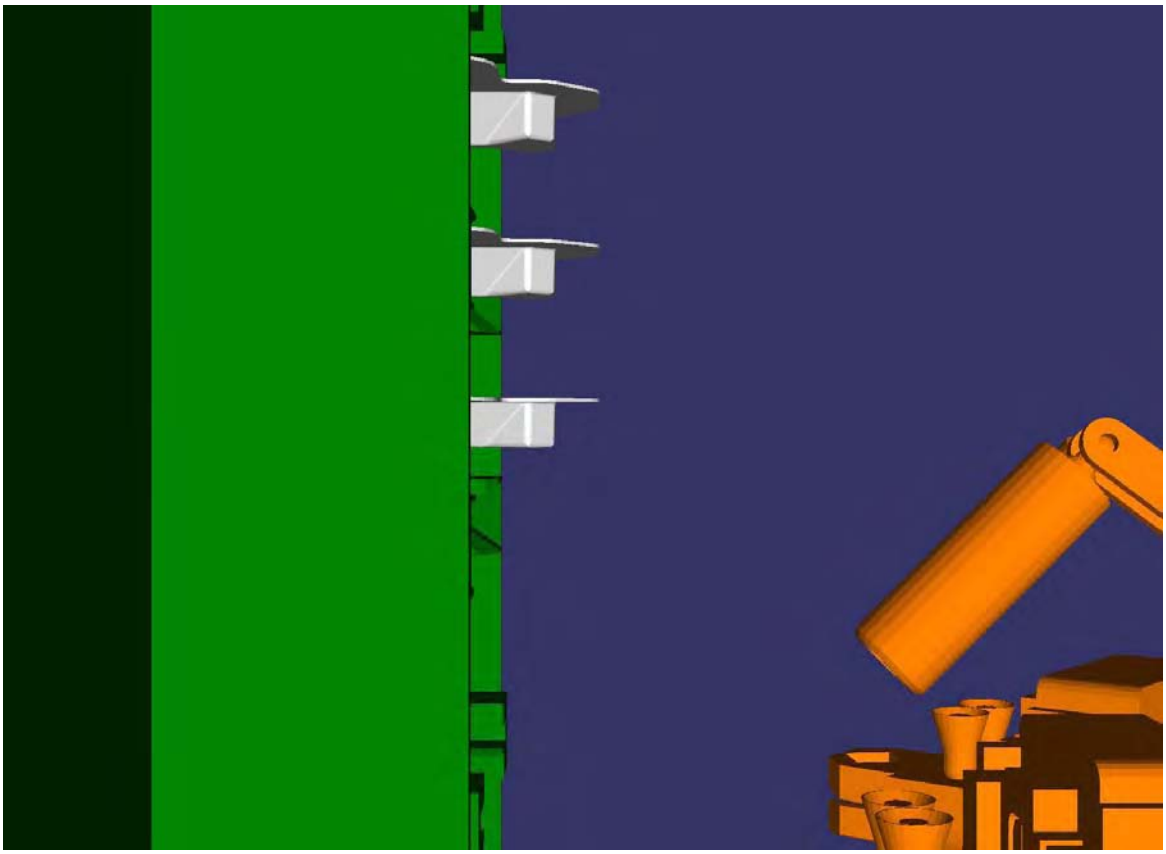


Figure 9: View from SDS Camera.

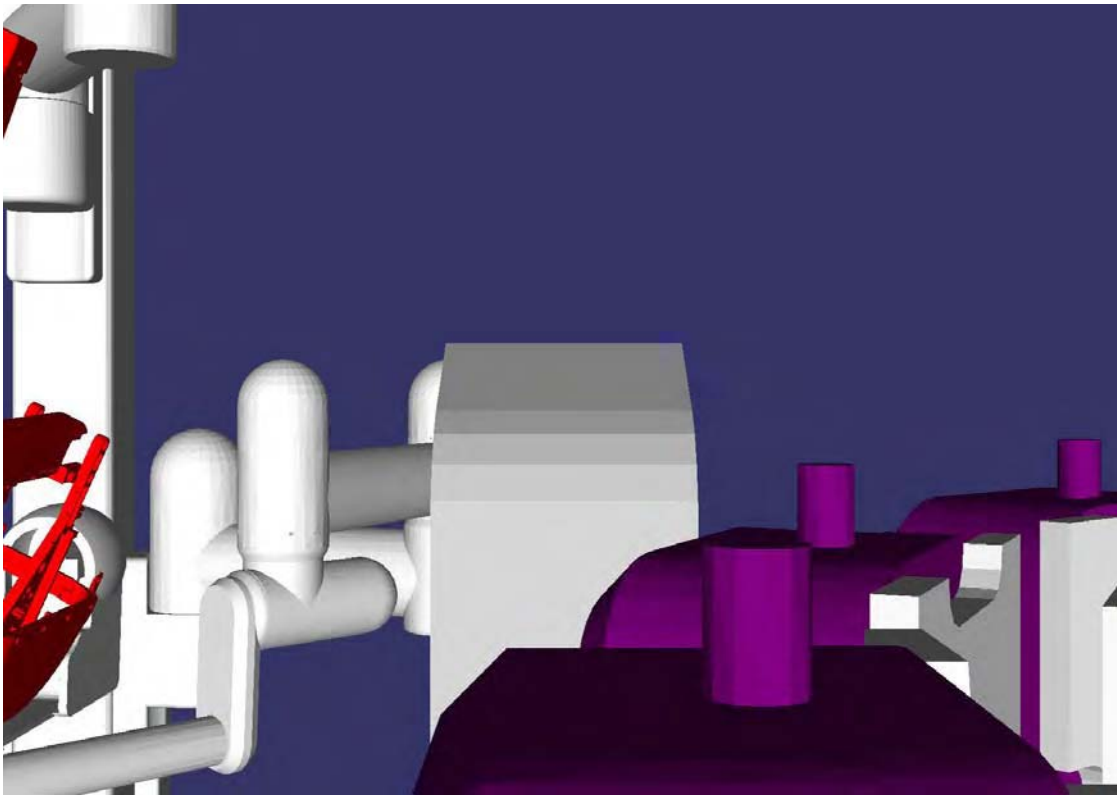


Figure 10: View from TRS Camera.

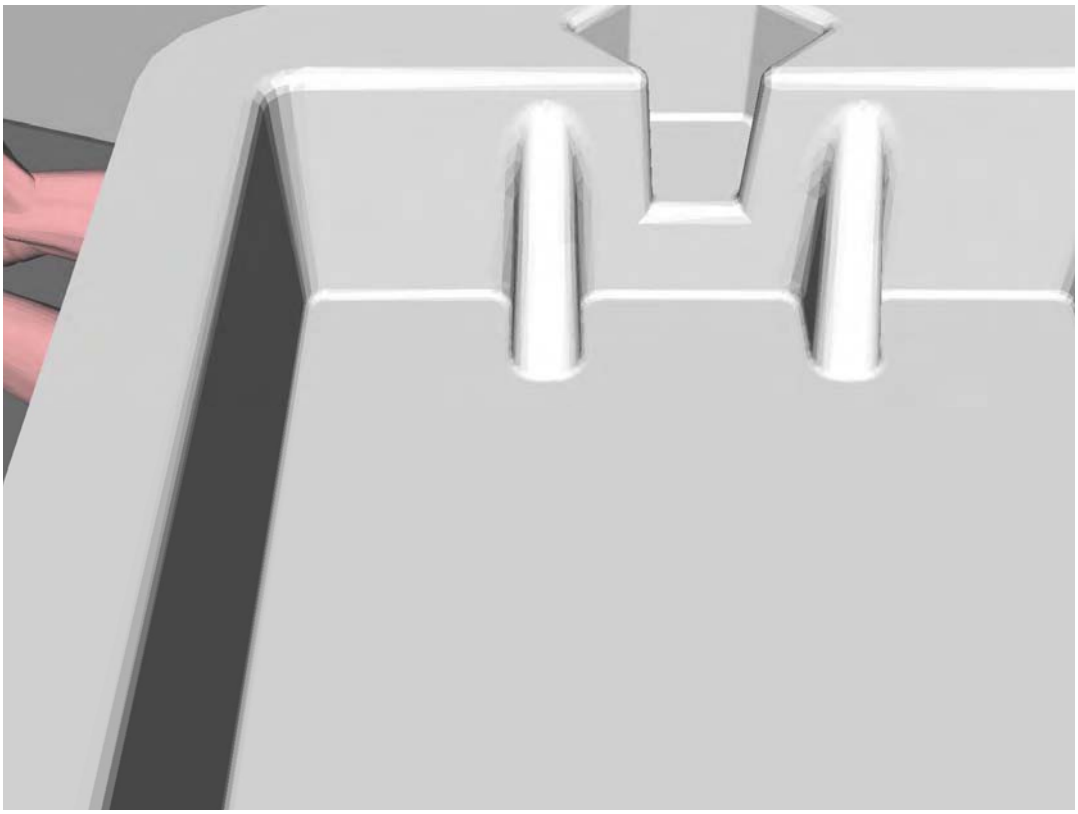


Figure 11: View of Tray from SNS Camera.

Robot Motion Planning Software

Motion Planner (MP) is a major component of the robot control software for trauma pod. It is used to plan and generate collision-free motion of the SNS robot. The current selection of the SNS robot is the 7-DOF Mitsubishi PA-10 manipulator.

Requirements: The SNS robot is responsible for delivering tools and supplies to the SRS. This must be done in a timely, safe, and accurate fashion.

1. MP must generate trajectories in such a way that allows SNS to complete the tool change task within 10 seconds.
2. MP must generate trajectories in such a way that allows SNS to complete the supply dispense task within 10 seconds.
3. The joint trajectories generated by MP must be within the SNS robot's physical (position, velocity, and acceleration) limits.
4. The trajectories generated by MP must be free of collision with any other subsystem or itself.

Design: The main task of MP is to provide coordinated control and to generate collision-free trajectories and send trajectory set-points to the robot controller at a sufficient rate (100 Hz). MP also has the ability to determine the minimum or near-minimum time to move the robot to a desired location from the current location, based on the robot's physical (position, velocity, and acceleration) limits. The MP module consists of several components working together to help meet the above requirements.

1. Trajectory Generation (TG): TG can generate trajectories in both joint and Cartesian spaces. Not only can TG generate trajectories between two end points but also trajectories that pass through intermediate points. In addition, TG takes into account the robot's velocity and acceleration limits to ensure that the generated trajectories are executable by the robot.
2. Kinematics consisting of Forward Kinematics (FK) and Inverse Kinematics (IK): After trajectories have been generated, IK is used to compute the next joint command to be sent to the robot controller. The computation rate of obstacle avoidance and collision detection must be greater than the rate required by the robot controller (100 Hz). There are two types of IK employed in MP, namely a closed-form IK and an iterative IK solution. FK is called upon whenever a need to compute the end-effector's location arises. FK is also used in the iterative IK.
3. Obstacle Avoidance (OA): OA uses distances computed from a geometrically simplified model of trauma pod to help guide the robot motion to avoid any obstacle or collision during run time. The simplified model consists of primitives whose distance to other primitives are simple to compute. These primitives are cylispheres (cylinders with two hemispheres at both ends), spheres, boxes, and planes.
4. Collision Detection (CD): CD is used as a safeguard ensuring that trajectories generated by MP will not cause any collision. Unlike OA, CD is based on a precise CAD model of trauma pod. Also unlike OA, CD does not guide the robot away from any obstacle or collision. It can only prevent collision. Before being sent to the robot controller, all the joint commands computed with TG, IK, and OA will be checked for collision. If a collision is detected, MP will issue an error and cancel the trajectory. Further details of CD are given in the next section.
5. Model Data Update: This is a component that is used to provide real-time sensory data to OA and CD. It receives the data (mostly locations) from all the subsystems, stores them, and lets OA and CD retrieve them in a timely and thread-safe manner.

The basic analytics on which the MP software is based on are described in Appendix G. Figure 12 shows a block diagram of these components pieced together.

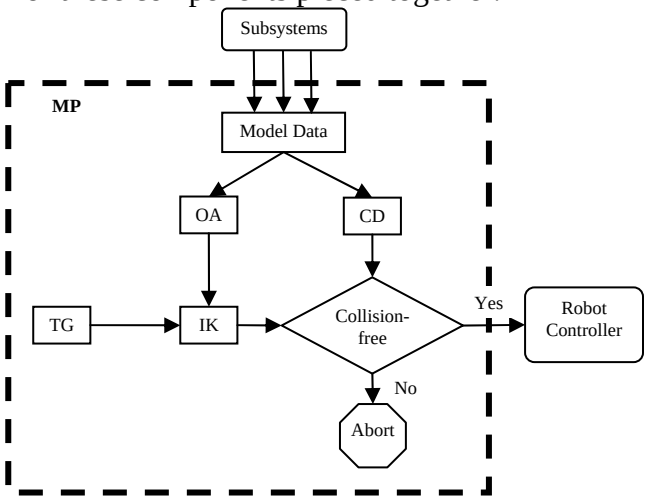


Figure 12: Block diagram of Motion Planner.

Implementation: The MP module is implemented in C++ using various OSCAR libraries, especially Forward Kinematics, Inverse Kinematics, and Obstacle Avoidance. Prior to the development of the MP software, UTA provided detailed OSCAR specifications as they related to the trauma pod and SNS in particular. These are given in Appendix E.

Excluding Collision Detection, the code written specifically for trauma pod contains approximately 20 C++ classes and roughly 10,000 lines of code. In this implementation, OA runs in its own thread at the user-specified rate (usually 50 or 100 Hz).

MP is deployed on the SNS controller as a non-real-time process in a Real-Time Linux (RTLinux) computer. Because of this, MP occasionally misses sending the joint command to the robot controller at the specified rate (100 Hz). A workaround was to make MP send a buffer of joint commands (typically 10 commands) so that if it is late sending the command at the next cycle, the robot controller can use the next command in the queue. This means that during each cycle, MP must compute up to 10 joint commands.

Performance: Studies showed that the move times for all trajectories generated by MP are under 2 seconds and most are about 1 second. Therefore, the requirements of 10 seconds for tool change and supply delivery tasks should be achievable. Another performance of interest of MP is the sufficient computational rate. The MP speed was tested on a computer with Pentium 4, 1.8 GHz and 512 MB of RAM running SuSE Linux 10 operating system. In this test, we assume that MP had to generate 5 new joint commands on average in each cycle. The test results summary is given below:

1. Without OA and CD, MP runs at around 1460 Hz.
2. With CD and OA (running at 100 Hz), the computational rate is approximately 170 Hz.
3. If we run OA at a slower rate of 50 Hz, then the cycle rate goes up to 224 Hz.

The conclusion here is that on average, MP with OA and CD is fast enough to meet the requirement of 100 Hz cycle rate.

Status: The first version of the MP software was ready at month 4. However, due to integration issues, the first version of MP was successfully integrated and demonstrated on the SNS hardware at month 8. Since then, the MP has been regularly updated, including the addition of CD, OA and buffering capability. Future developments in the MP are related to performance improvement of *MoveVia* commands and any other issues or problems that may arise.

Collision Detection Software

The main purpose of Collision Detection (CD) software, as implemented in the trauma pod, is to support a collision free and thus safe (for man and machine) surgery environment. Collision detection is vital to three main trauma pod areas: simulation/design, path planning, and real-time motion planning.

A simulation environment equipped with collision detection capabilities was a valuable tool for the design of multiple collaborative subsystems. Using such a simulation environment during the design phase of the trauma pod aided in layout analysis and manipulator design. As layout of the trauma pod was analyzed, collision detection helped resolve subsystem positioning conflicts

and also helped determine optimal placements of manipulators to reduce obstructions with other subsystems. Manipulator design was also influenced by collision detection such as when an end-effector design feature resulted in limited collision-free movement of the end-effector.

Collision-free path planning is also dependent on collision detection. In the case of the trauma pod, paths can be checked to be free from collision before or during execution. If collisions exist then obstacle avoidance techniques can be applied until the optimal path is found. Collision detection does not supply any information on how to avoid obstacles but is does provide position data to algorithms within the obstacle avoidance domain that do.

The SCS uses collision detection to ensure that all systems are operating safely during real-time execution. If proximity queries between two objects show that they are closer than an acceptable margin the SCS can then take appropriate action.

Analysis of CD Software Libraries: In choosing the method of collision detection to be used for the trauma pod, the following requirements must be considered:

- Support of complex geometric CAD data
- Acceptable computational performance
- Low-Level control of collision pairs

The geometric data is expressed in the form of stereolithography (STL) data files which are complex triangle meshes derived from CAD data. Acceptable performance for the trauma pod was defined as a rate of 100 Hz or higher. Low-level control of collision pairs means that the user must be able to control which collision pairs are calculated at a given time.

All methods of collision detection consist of calculating distances between every object in a given space. These distances are computed either analytically or by iterative algorithms. Analytical calculations exist for distances between simple primitives such as spheres, cylinders, and planes. However, for complex geometry, only iterative algorithms provide the best results in terms of the first two requirements, geometric complexity and speed.

Software Library for Interference Detection (SOLID) was chosen to perform distance computations in the trauma pod. Developed by G. van den Bergen, it uses a popular iterative algorithm, the Gilbert Johnson Keerthi algorithm (GJK). It was chosen because it fulfilled the above requirements, was well documented, and was thoroughly tested.

SOLID maintains high performance rates by computing distances between successively higher resolution objects. Distances between object bounding boxes are first computed and if any two bounding boxes intersect then distances are computed between relevant portions of triangle meshes. Position data is cached between each collision query so that time is not wasted by computing distances between non-moving objects or objects moving in opposite directions.

Implementation: The collision detection software for the trauma pod was written in C++. All trauma pod data is represented by one world model data file. The collision detection software reads this file and sets the position and geometric data accordingly. The collision detection software and all other software modules of the trauma pod use the same world model data file to prevent discrepancies. The collision detection software uses the SOLID library for distance

calculations and for robot kinematics it uses the OSCAR software developed by UTA. These two libraries along with approximately 6,000 lines of code provide the following interface functionality:

- Check collisions in trauma pod
- Get data from collision occurrence
- Update position of any moveable subsystem.
- Ignore collisions between any two objects
- Get distance between any two objects
- Set the bounding margin of any object

Task Example: As an example of this functionality, these are the steps that could be followed while the SNS performs the simple task of moving to the tool rack and picking up a tool.

1. As the SNS moves toward the tool rack, repeatedly update its position in the collision detection software and check for collisions.
2. As the SNS moves through space, check the distance between it and other objects of interest for use by the path planning software.
3. On approach to the tool of choice, turn off the end-effector and tool collision pair.
4. Pull the tool off the rack (still updating position repeatedly) and set the bounding margin of the tool to 2 cm. This will cause the rest of the trauma pod to see the tool as being 2 cm larger in all directions.

In addition to the above implementation, another configuration of the software was created which is only concerned with collisions which involve the SNS. This further improves performance of the collision detection software because distances between other subsystems are not computed.

Performance: The trauma pod geometric data is represented by about 700,000 triangles. Using that data, Table 2 shows the performance of the collision detection software at different instances. The software configurations refer to when collisions are being detected between everything in the trauma pod (Full) or when collisions are detected in which the SNS is involved (SNS). For each of these configurations the rate is computed when there are 0 and 1 collisions.

Software Config.	# of Collisions	Rate (Hz.)
Full	0	320
Full	1	115
SNS	0	6200
SNS	1	300

Table 2: Performance Data for Collision Checker.

As expected, the SNS configuration is much faster than the Full configuration. Also, when one collision occurs the performance deteriorates because deeper collision queries must be computed to compute intersecting triangle meshes.

Status: The first prototype of the CD software was available in Q1. It was successfully integrated with the simulator in Q2 and was integrated with the MP software in Q3. The CD

software will be integrated with the SCS in Q5. For now, no additional functionality is planned for the CD software. Primary focus will be on resolving any issues/problems that may come up.

Supervisory Control System

The primary responsibility of the supervisory control system is the command and coordination of other subsystems within Trauma Pod to complete requested tasks requested by the User Interface Subsystem (UIS). Additionally, the SCS is responsible for monitoring subsystems to detect collisions and basic subsystem failures.

The SCS, in combination with the AMS subsystem, is also used to execute various tasks for calibration, setup and initialization, and shutdown of the Pod subsystems.

Requirements: Detailed subsystem requirements are available in Functional Subsystem Requirements (CTL-001-R01-System_Requirements.xls) document available from www.traumapod.org. The major functional requirements from this document are summarized in Table 3.

Reqmt #	Requirement
SCS FUN 1.0	The SCS shall initiate, monitor, and terminate all subsystem primary tasks.
SCS FUN 1.1	The SCS shall provide the mechanism through which Trauma Pod tasks between interacting subsystems are planned and scheduled.
SCS FUN 1.2	The SCS shall provide the mechanism through which Trauma Pod tasks are executed.
SCS FUN 1.3	The SCS shall be responsible for coordinating the actions of each subsystem and ensuring the safety of the Trauma Pod system as a whole.
SCS FUN 1.4	The SCS shall be responsible for ensuring that Trauma Pod robotic elements do not collide. If any subsystems are about to collide, the SCS shall immediately stop the motion of the relevant subsystems to prevent collision.
SCS FUN 2.0	The SCS shall keep a repository of system tasks. Tasks shall be created and edited by an application on the SCS. Task definitions shall be decomposed into subsystem tasks.
SCS FUN 3.0	The SCS shall insure that subsystem primary tasks are executed with the necessary timing and order to complete system-level tasks (e.g. change SRS tool).
SCS FUN 4.0	The SCS shall detect error conditions related to surgical tasks, either by monitoring or by being alerted, and take corrective action.
SCS FUN 5.0	The SCS shall accommodate interruptions, such as the surgeon canceling a task in the middle of execution. The SCS shall provide recovery commands to return the system to an appropriate state to continue activities.
SCS FUN 6.0	The SCS shall provide software interlocks to prevent incompatible actions from being executed. A defined recovery procedure shall be provided to return the system to a previous state after an interlock is triggered.
SCS FUN 7.0	The SCS shall store the geometry definitions for all physical elements of the system including: subsystem outlines/footprints, robots, and patient registration. The SCS shall provide a data file server accessible by subsystems (e.g. via HTTP).
SCS FUN 11	The SCS shall recognize and respond to an Emergency Stop signal.

Table 3: SCS Functional Requirements.

Design: The design of the SCS involves system engineering efforts as well as subsystem design for the SCS. The design is described in the following sections on *Task Analysis*, *Interfaces and Communications*, and *Task Execution*.

Task Analysis

Because the SCS is central to executing surgeon level tasks and coordinating other subsystems, early efforts in Q1 through Q3 were spent on task analysis, interface design, and systems engineering. During Q4 these interactions were finalized and subsystem interfaces were frozen.

During Q1-Q2, logical points of relevance (see Figure 2. Trauma Pod Logical Layout) were first identified and named. These points defined locations to which the SNS robot would need to move. The sequence of SNS movements and subsystem commands which would then be required to complete a surgical task such as changing a tool were then identified and documented in interaction diagrams. Figure 13 is a representative diagram. Appendix A contains high level diagrams for Tool Change, Supply Dispense, and Calibration.

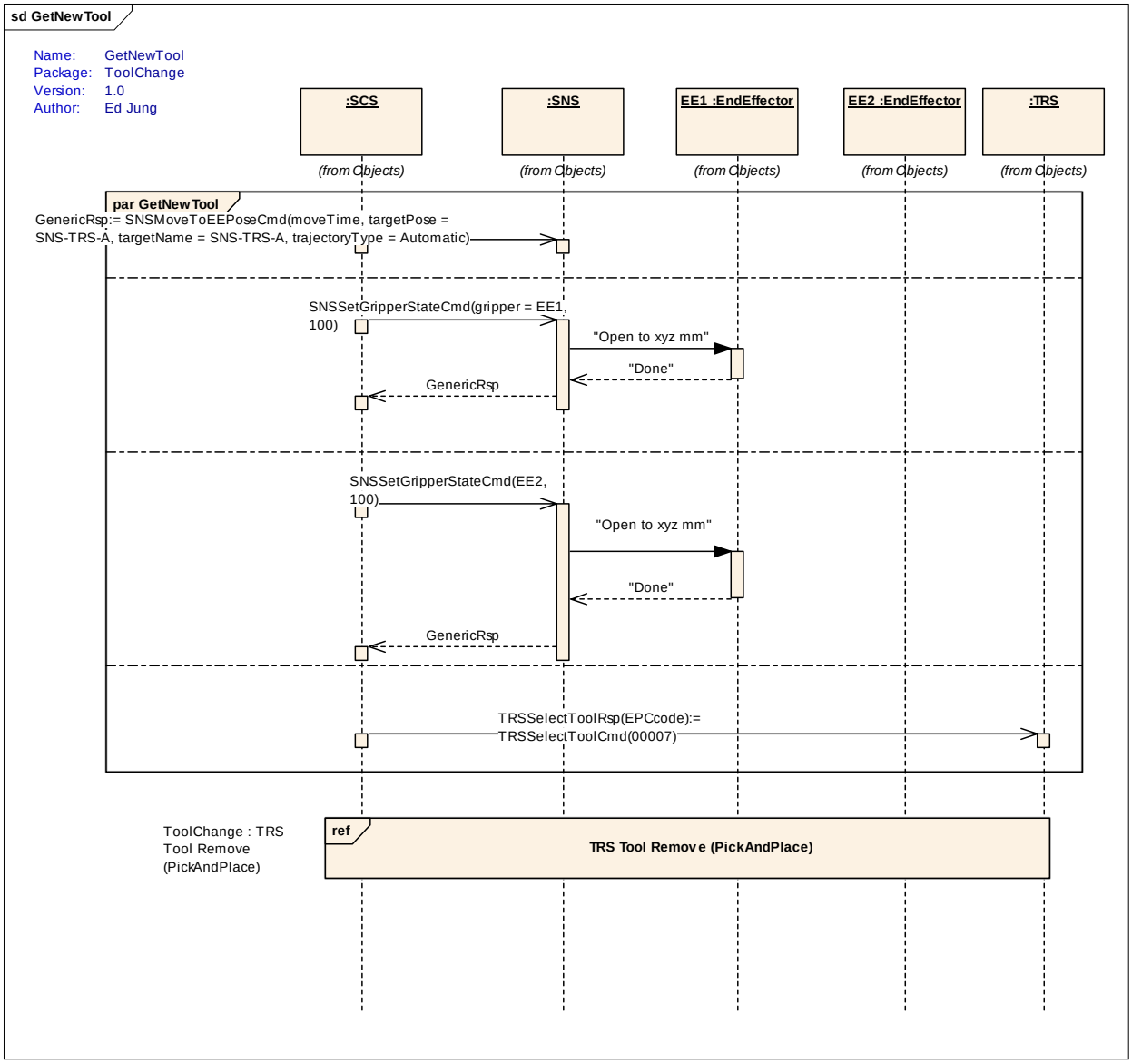


Figure 13: "Get New Tool" Interaction Diagram.

Interfaces and Communications

During Q2-Q4, based on the interaction diagrams messages that each subsystem needed to accept were defined. These messages are grouped according to *interface types*, such as SCSIf, SNSIf, TRSIf, etc.

XML strings were chosen as the format for the messages, which provides platform independence and a human readable format. XML schemas for each interface type formally define the messages which are accepted on an interface. These schemas may then be used to generate code that automatically transforms an XML string into a language specific format, such as C++ or Java.

For example, the TRSIf defines the following message interface for the TRS:

1. TRSSelectToolCmd
2. TRSSelectToolRsp
3. TRSSelectEmptySlotCmd
4. TRSSelectCalibrationLugCmd
5. TRSAcquireToolCmd
6. TRSAcquireToolRsp
7. TRSSurrenderToolCmd
8. TRSSurrenderToolRsp
9. TRSGotoBayCmd
10. TRSSetGripperStateCmd

These messages are formally specified in the TRSIf.xsd schema. For example, the TRSSelectToolCmd is specified as:

```
<xs:element name="TRSSelectToolCmd">
        <xs:annotation>
                <xs:documentation>
                        Receiver must lookup "type" in the local inventory to
                        find a bay containing the requested tool. If a tool is available,
                        the tool magazine is rotated to present the tool at the TRSTransferZone.
                        If the requested tool is not available, a Nak message is sent back.
                </xs:documentation>
        </xs:annotation>
        <xs:complexType>
                <xs:complexContent>
                        <xs:extension base="MessageType">
                                <xs:sequence>
                                        <xs:element name="type" type="ProductType"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
</xs:element>
```

and a specific instance of this message string is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<TRSSelectToolCmd>
  <src>SCS</src>
  <dst>TRS</dst>
  <ts>1138909101965</ts>
  <xid>89</xid>
  <type>00006</type>
</TRSSelectToolCmd>
```

Several other interfaces such as NodeIf, AlarmMonitorIf, AlarmMonitorIf, and others are also defined and supported by subsystems. These are intended for use by the AMS. Additionally, ModelIf (e.g., SNSModelIf, TRSModelIf) interfaces are defined for use by the simulator and collision checking software. These interfaces may be used by the SCS, but are not fundamental to its task execution capability.

The XML messages defined by the schema are sent to subsystems over Ethernet using the *Spread Group Communication* software (www.spread.org). Spread is a freely available publish-subscribe software package.

Task Execution

The primary functional responsibilities of the SCS include the planning and scheduling of tasks as defined by the interaction diagrams, and ensuring the safety of Trauma Pod subsystems.

Each of the commands in the interaction diagrams previously may be described in terms of their *pre-conditions* and *post-conditions*. *Pre-conditions* are conditions which must be true before a command can execute, and *post-conditions* are conditions which must be true after a command executes. Conditions can be compared against *extended state variables* for each subsystem to ensure they are met, where e*xtended state variables* describe the status of a subsystem.

For example, in Table 4 below, the tool held by the SRS is an extended state variable, the pre-condition for the Surgical Robot Subsystem (SRS) is that its end-effector is empty, and the post-condition for the SRS is that its end-effector has scissors.

Task: *Insert scissors in surgical robot's end-effector* *Initial State*: Surgical robot's end-effector is empty AND Nurse has scissors *Goal State*: Surgeon robot's end-effector has scissors AND Nurse does not have scissors

Table 4: Example of Pre-conditions, Post-conditions, and Extended State Variables.

Pre and post conditions for tasks may be checked by the SCS to ensure the safety of pod subsystems. Before executing a command, the SCS should check pre-conditions to ensure that they are met, and after executing a command, the SCS should compare the expected post-conditions against the actual state of the Trauma Pod to ensure that the command has completed successfully.

These conditions may additionally be used to plan and schedule surgeon tasks. By comparing pre and post conditions of commands, it should be possible to automatically assemble an optimal sequence of commands to accomplish any surgeon level goal. This capability, however, is beyond the scope of Phase I goals for the SCS. Appendix F analyzes the feasibility of automated planning and scheduling for the SCS. Phase I implementation will use pre and post conditions, as well as a constrained set of pre-defined tasks to handle surgeon commands.

Extended state variables for Trauma Pod Subsystems were specified in Q4 and are summarized in Table 5 below.

SRS Variables	SNS Variables
SRS.SlaveLeftArm.State = [Lock, Clutch, Follow, Float]	SNS.ActiveGripper = [EE1, EE2]
SRS.SlaveRightArm.State = [Lock, Clutch, Follow, Float]	SNS.Gripper.EE1.Contents = [Tool, Tray, Empty]
SRS.MasterLeftArm.State = [Lock, Clutch, Follow, Float]	SNS.Gripper.EE2.Contents = [Tool, Tray, Empty]
SRS.MasterRightArm.State = [Lock, Clutch, Follow, Float]	SNS.Gripper.EE1.State = [0..100]
SRS.SlaveLeftArm.ToolType = productType	SNS.Gripper.EE1.Error = [true, false]
SRS.SlaveRightArm.ToolType = productType	SNS.Gripper.EE2.State = [0..100]
	SNS.Gripper.EE2.Error = [true, false]
SDS Variables	SNS.TAS.Left.IsLatched= [true, false]
SDS.FC.Location = [Home, Park, SNSAccess, STZ]	SNS.TAS.Left.HasTool = [true, false]
SDS.FC.IsMoving = [true, false]	SNS.TAS.Right.IsLatched = [true, false]
SDS.SC.slot.<int>.empty = [true, false, unknown]	SNS.TAS.Right.HasTool = [true, false]
SDS.FC.slot.<int>.empty = [true, false, unknown]	SNS.IsMoving = [true, false]
SDS.dispenser.slot.<int>.empty = [true, false, unknown]	
	TRS Variables
	TRS.CurrentBay.Index= [0..14]
	TRS.CurrentBay.epc = epcCode
	TRS.CurrentBay.GripperState? = [0..100]
	TRS.inventory.hasProductType.<ToolType> = [true, false]

Table 5: Extended State Variables.

Calibration

Calibration involves identifying the position and orientation of all subsystems in the trauma pod based on the actual physical layout. The data from the CAD model provides a starting point. This data is analyzed for layout and timing issues and is captured in the world model. During calibration, the design data captured in the world model is modified to reflect actual configuration of the trauma pod.

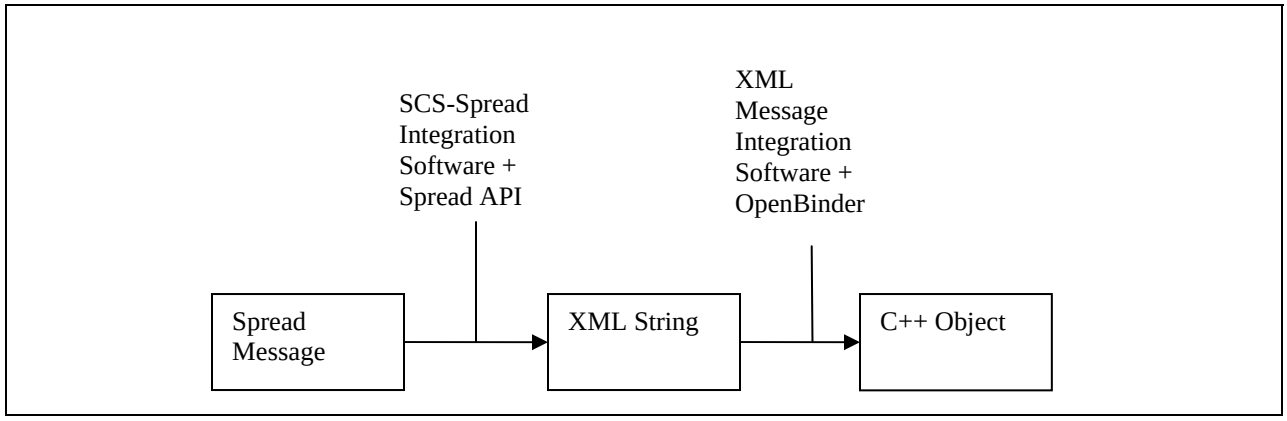
Two different calibration processes will be implemented in the trauma pod. These are manual and automatic calibration. In manual calibration, an operator will move the SNS robot to all the calibration lugs. During this process, the actual position of the calibration lugs as seen by the SNS will be captured. This information will be compared to the position of the lugs as defined in the world model. The delta error between the two will then be used to modify the position and orientation of the relevant subsystems. During automatic calibration, the SNS robot will be commanded to move to the calibration lugs in a sequence. If there is a failure in any of these moves, automatic calibration will be cancelled and manual calibration sequence will follow.

Currently, the calibration task and calibration software module have not been implemented. These will be done in Q5.

SCS Implementation and Status
The following sections outline the status of the SCS as currently implemented.

Communications and Interfaces
Because the SCS must communicate with numerous subsystems and halt subsystems in case of faults, the SCS communications software should be scalable to a large number of interfaces and handle messaging errors by other subsystems. Also, the interfaces defined by the XML schema are expected to change numerous times throughout the project, and introduce another source of bugs into the system. Efforts in Q1 and Q2 were directed at implementing scalable and robust communications software.

The communications software for the SCS is composed of two major components—SCS-Spread integration software and XML message integration software. Although the Spread software package provides a ready-to-use software library, bridging the gap between Spread communications, XML messages, and C++ objects usable by the SCS still requires significant effort. These Spread integration and XML message integration software bridge this gap.



The SCS-Spread integration software consists of several classes for simplifying the sending, receiving and dispatching Spread messages on the SCS. This software was also reused for communications on the Simulator.

The XML message integration software consists of several C++ classes and several XSL stylesheets to automatically generate code from the interface schemas for transforming Spread messages into C++ objects. OpenBinder software provided by ORNL was also used.

This approach reduces the potential errors that might be introduced by manually writing the software for transforming a raw Spread message into a C++ object usable by the SCS. Changes in interface schemas can also be automatically reflected in the SCS software. Additionally, the addition of new interfaces to the SCS can be easily achieved.

Basic Subsystem Functions
The SCS implements all basic behaviors and functions required of all Trauma Pod subsystems, as defined by the NodeIf interface and the Node State Machine diagram in Figure 14. These functions encompass activities such as setting/getting of configuration parameters, alarm

management, bootup and initialization, and logging on the SCS subsystem. These basic functions were delivered for Q2, with additional updates and revisions for Q3 and Q4.

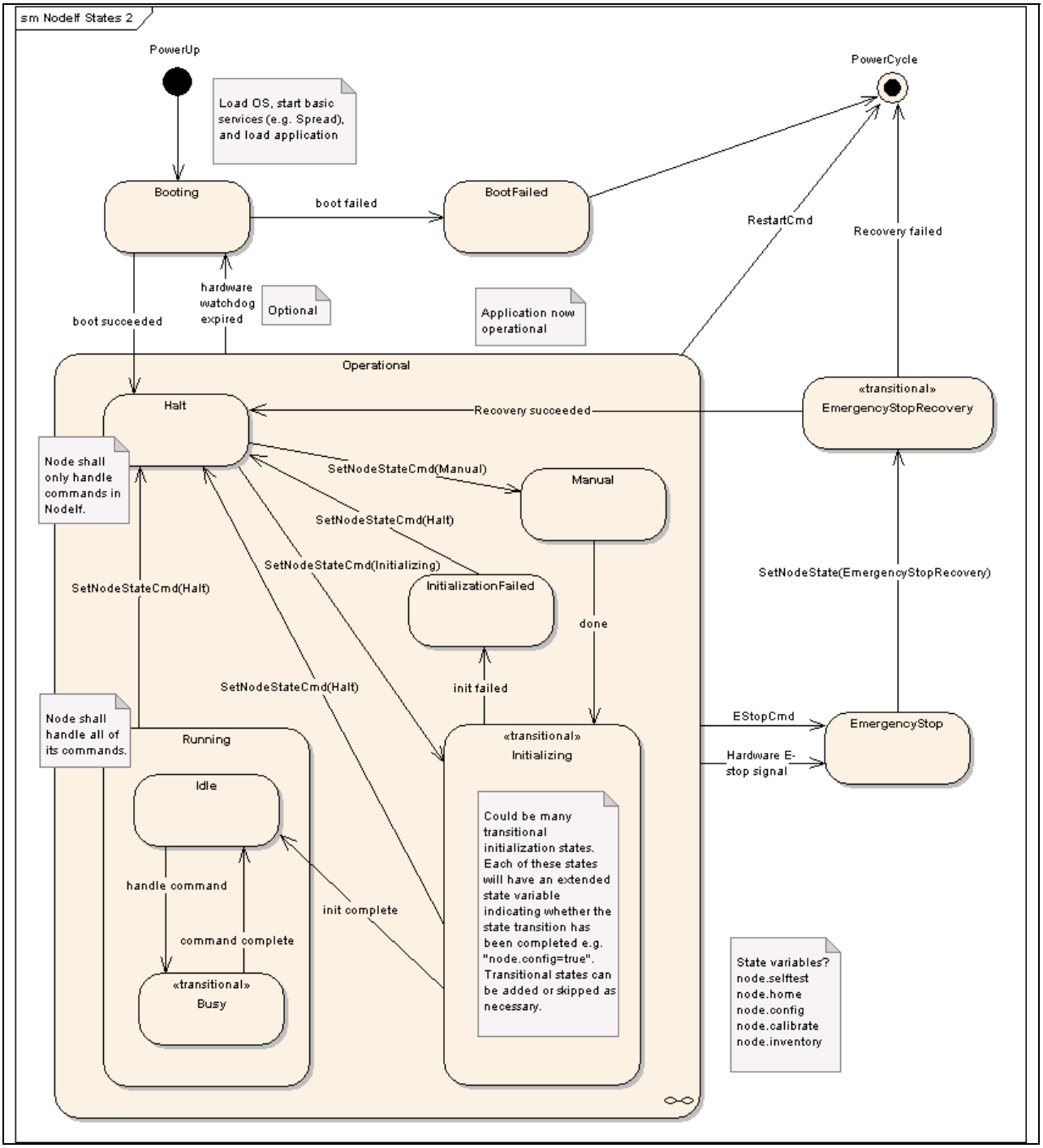


Figure 14: Node State Machine.

Task Execution Functions

The SCS currently supports basic task execution functionality, including the ability to execute a task, queue tasks, cancel tasks, and query task status. Communications with each subsystem has also been demonstrated.

The basic Tool Change task on the SCS was delivered at the end of Q3, but could not be tested or demonstrated until Q4, due to the dependencies on the correct functioning of other subsystem emulators in the Trauma Pod.

A framework for executing any surgeon task is currently in place, but the following task execution functions remain to be implemented in the following quarters.

- Supply Dispense Task: The surgeon requested supply dispense task will be implemented and debugged in Q5-Q6. Because Supply Dispense involves the coordination of several robots, supply counting, and two-way interactions between subsystems, its implementation is more complex than Tool Change.

- Subsystem monitoring: Implementation of the extended state variables is required for the SCS to perform subsystem monitoring. Once these variables are implemented by subsystems in Q5, the SCS will add the capability to monitor subsystems for faults and status.

- Cancellation, Pause, and Recovery: Though the SCS is currently capable of queuing and canceling tasks, it does not yet have the ability to recover from a cancelled task. During Q4 the strategy for handling and recovering from a cancelled task was identified.

- Debug tasks: Various debugging tasks are needed to run the actual hardware subsystems. These include tasks such as inserting and removing fast cache/slow cache trays from the SDS, or tools from the SRS and TRS.

Additional functionality which remains to be implemented includes:

- Collision detection: Collision detection software has already been implemented and tested on the SNS. The SCS will share this code for its collision detection implementation, and will be added for Q5.

- Manual mode tasks and GUI: A manual mode administrative GUI for the SCS will be added. This GUI will allow a user to access various debug tasks and functions on the SCS.

- Task Updates: As actual timing data and experience with Trauma Pod hardware is gained, it is expected that surgeon level tasks will have to be revised or rewritten to accommodate changes.

KEY RESEARCH ACCOMPLISHMENTS

- Developed a generalized world modeling format that can be used as input to task planning algorithms, 3d simulators, robot controllers, and collision detection software.
- Developed a kinematic controller that supports a variety of trajectory types along with the ability to pass through kinematic singularities while avoiding obstacles.
- Resolved issues related to integration of collision checking with obstacle avoidance. Basically, obstacle avoidance is a real-time function of the robot controller that requires constant time distance computation. Due to this requirement, using hi-fidelity CAD models for obstacle avoidance is very compute prohibitive. On the other hand, collision detection is more of a binary check to see whether there are any collisions or not. This does not require distance computations between shapes. However, collision detection does require a high-fidelity model for correctness. Our research led to the separation of these two tasks, as in essence they both have different functions. Simply put, obstacle avoidance guides the robot motions whereas collision detection simply provides an alarm.
- Showed the feasibility of a high bandwidth robot controller that implements obstacle avoidance and high fidelity collision detection at rates greater than 100 Hz.
- Developed a generalized simulator for robot surgical environments. From a robotics perspective, this simulator offers significantly higher functionality then previous attempts. If tissue models can be integrated with this simulator, it could provide a better simulation environment for surgeon training on the Da Vinci and the use of the trauma pod.

REPORTABLE OUTCOMES

Awards

- Three engineering researchers from the Robotics Research Group at The University of Texas at Austin won an award for a paper they co-produced, “Real-Time Robot Capability Analysis. (see Appendix G)” Dr. Chalongrath "Josh" Pholsiri was the principle researcher and author, and won the “MSC Simulation Software Award” along with colleagues, Dr. Chetan Kapoor and Dr. Delbert Tesar. They received the award in September 2005 at the Mechanisms and Robotics Conference of the American Society of Mechanical Engineers. The conference featured papers and workshops in the field of mechanical systems, which includes mechanisms, robots and machines. Pholsiri’s paper documented results of a study on robotic manipulators used in complex environments. Specifically, the research results were demonstrated on the Mitsubishi PA-10 manipulator. This is the same manipulator that is being used in the trauma pod. Josh is a post-doc at the Robotics Research Group and is working full time on the trauma pod project.

- Dr. Delbert Tesar was awarded the Engelberger Robotics Award in the area of education at the 36th International Symposium in Robotics held in Tokyo, Japan. Active in robotics for some 40 years, Tesar leads the largest university-based robotics research group in mechanical engineering in the United States. To date, the program has graduated 53 Ph.Ds and 129 Masters of Science. Tesar has written 90 position papers, 215 refereed conference and journal papers and given more than 500 invited lectures. He also holds several U.S. patents.

Degrees

Student Name	Degree	Date	Research Topic	Trauma Pod Role
Jeremy Sevier	Masters	9/1/03 - 1/15/06	Resource Optimization for Industrial Robotic Systems	Kinematics development and software testing
John Hall	Masters	01-2003 to 08-2005	A Test Bed Framework for Actuator Management Operating Software (AMOS)	Software testing

CONCLUSION

The trauma pod project involves significant integration with research challenges in the area of manipulator control, world modeling and task planning and execution. The integration challenges is further heightened by the fact that the prototype subsystems are in different stages of developments, based on different computing technologies and are from a set of geographically diverse vendors. This integration challenge has been addressed rather well by the trauma pod team. The reasons for this are:

1) A control architecture that relies on *separation of concerns*.
2) A communication architecture that allows flexible communication between any two subsystems and that uses a descriptive language that can be used to formalize interfaces.
3) The effective communication between teams due to web based collaboration.
4) The development of emulators and simulators to accelerate integration prior to hardware availability.

Despite the success of the above, the robustness of this integration and also its ability to work in a real-time environment still needs to be evaluated. Specifically, the networking system in the trauma pod has to be stress tested with the full amount of sensor data and control and command data.

A high fidelity simulator with integrated collision detection was developed and delivered on time. This simulator can be further enhanced by improving the dynamic behavior of the mechanical systems in the trauma pod and integrating tissue modeling into it. Such a simulator can be an invaluable tool for surgeon training and "what if" analysis.

On the robotics front, the feasibility of a complex controller in a distributed environment has been demonstrated. Additionally, the ability of modern computers and software to successfully compute complex geometric models in real-time has also been demonstrated. However, the potential of this development is not fully exploited as the trauma pod consists primarily of a

single robot that is under automatic control (scrub nurse). The fast cache does add some complexity, but it is more of a discrete device due to the limitations of its controller. In the future trauma pod, multiple manipulators can be expected and the ability of UTA's robot software to work in this environment has to be evaluated.

A format for defining the world model of the trauma pod was also developed. This development leveraged significant previous work at UTA in robotic workcell modeling and control. Based on the world model format, an instance file for trauma pod has been created. This contains all geometric information related to the trauma pod and is being used by the collision detection software, the simulator, the robotic software (SNS, FC, SRS), and the SCS.

Collision detection software developed for the trauma pod provides hi-fidelity collision detection based on the CAD models of the subsystems. This software computes at rates above 100 Hz and is being used by the simulator, the SNS software, and the SCS.

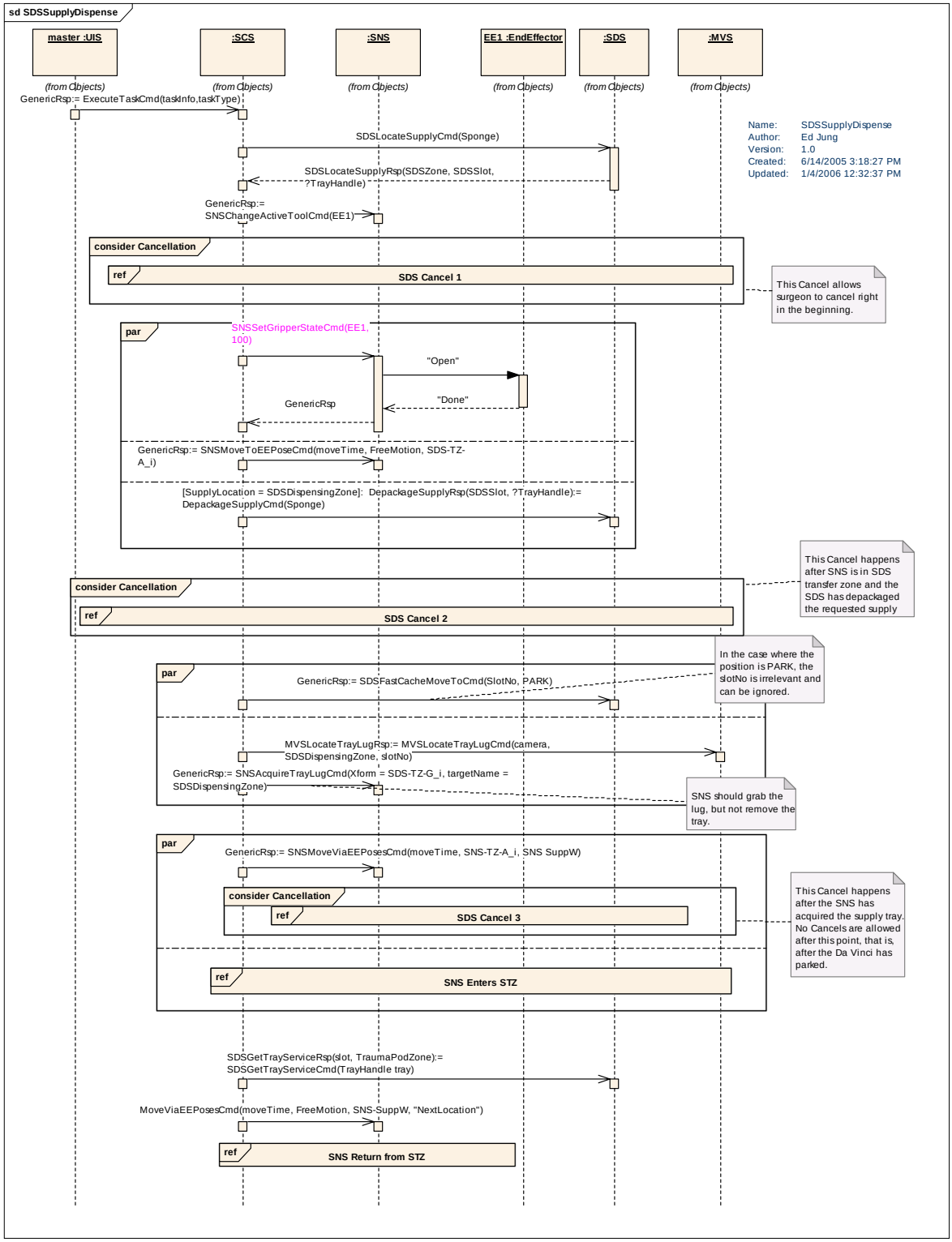
The SCS is central to the functioning of the trauma pod. As the SCS integrates and manages all subsystems, its software design was primarily driven by issues related to scalability, error handling, and interface evolution. This design and implementation was completed by Q3 and is now functioning as a part of the emulator network. The task planning and execution capability of the SCS is probably the most challenging part of this project. The capability deployed in Phase I will be limited to task execution with minimal planning. The development of fully automated planning will require significant new research that blends the discipline of artificial intelligence with the model-based approaches of mechanical systems.

REFERENCES

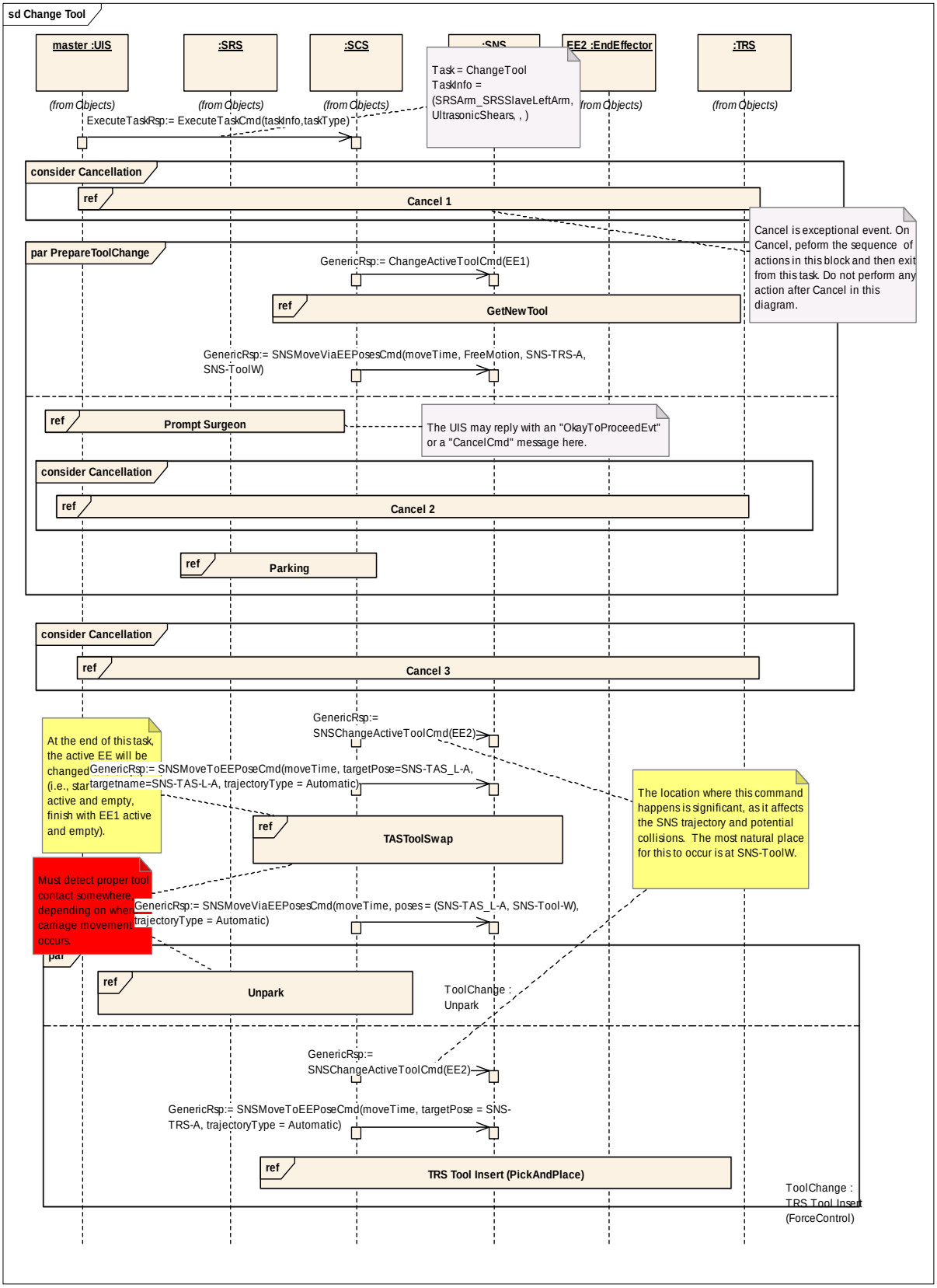
1. Bass, L., Clements, P., and Kazman, R., 2003, "*Software Architecture in Practice*," Addison-Wesley Professional, Boston, MA.
2. Chien, S., Hill, R., Wang, X., Estlin, T., Fayyad, K., and Mortenson, H., 1996, "Why Real-World Planning is Difficult: A Tale of Two Applications," *New Directions in AI Planning*, IOS Press, Washington, D.C., pp. 287-298.
3. Elmer G. Gilver, Daniel W. Johnson, S. Sathiya Keerthi, "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space", *IEEE Journal of Robotics and Automation*, Vol 4, No 2, April 1988, pp. 193-203.
4. Harden, T. Kapoor, C. Tesar, D., "Experimental Evaluation of a Criteria-based Obstacle Avoidance Scheme," *Proceedings of the 1999 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, September 12-16, 1999, Las Vegas, Nevada.
5. Harden, T. Kapoor, C. Tesar, D., "Obstacle Avoidance Influence Coefficients for Manipulator Motion Planning," DETC2005-84223, *Proceedings of the ASME 2005 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Long Beach, CA, September 24-28, 2005.
6. Kapoor, C. Tesar, D., "A Reusable Operational Software Architecture for Advanced Robotics," *Proceedings of the Twelfth CSIM-IFToMM Symposium on theory and Practice of Robots and Manipulators*, Paris, France, July 1998.
7. March, P. Kapoor, C. Tesar, D., "Motion Planning of Robotic Systems for Applications in Nuclear Facilities Clean-Up," *Proceedings of the 2002 ANS Spectrum Conference*, August 2002.
8. Miyawaki, F., Masamune, K., Suzuki, S., Yoshimitsu, K., and Vain, J., 2005, "Scrub Nurse Robot System-Intraoperative Motion Analysis of a Scrub Nurse and Timed-Automata-Based Model for Surgery," *IEEE Transactions on Industrial Electronics*, 52(5), pp. 1227-1235**.**
   Taylor, R.H., 2003, "Medical Robotics in Computer-Integrated Surgery," *IEEE Transactions on Robotics and Automation*, 19(5), pp. 765-781.
9. Myers, K. L.,1996, "Strategic Advice for Hierarchical Planners", *Proc. of 5th International Conference of Principles of Knowledge Representation and Reasoning*, Aiello, L. C. et al., eds., Morgan Kaufmann Publishers Inc., San Francisco, CA, pp. 112–123.
10. Pholsiri, C. Kapoor, C. Tesar, D., "Manipulator Task-Based Performance Optimization," *Proceedings of ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Salt Lake City, UT, Sept. 28 - Oct. 2, 2004.
11. Pholsiri, C. Kapoor, C. Tesar, D., "Real-Time Robot Capability Analysis," DETC2005-84223, *Proceedings of the ASME 2005 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Long Beach, CA, September 24-28,2005.
12. Russell, S., and Norvig, P., 2003, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, pp. 375-461.
13. Tisius, M. S., and Tesar, D., 2004, "An Empirical Approach to Performance Criteria and Redundancy Resolution", M.S. Thesis, The University of Texas at Austin, Austin, TX.
14. Van den Bergen, G., "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects" *Journal of Graphics Tools*, 4(2):7-25, 1999.
15. Van den Bergen, G., *Collision Detection in Interactive 3D Environments*, Elsevier, Inc., San Fransisco, CA, 2004.

APPENDICES

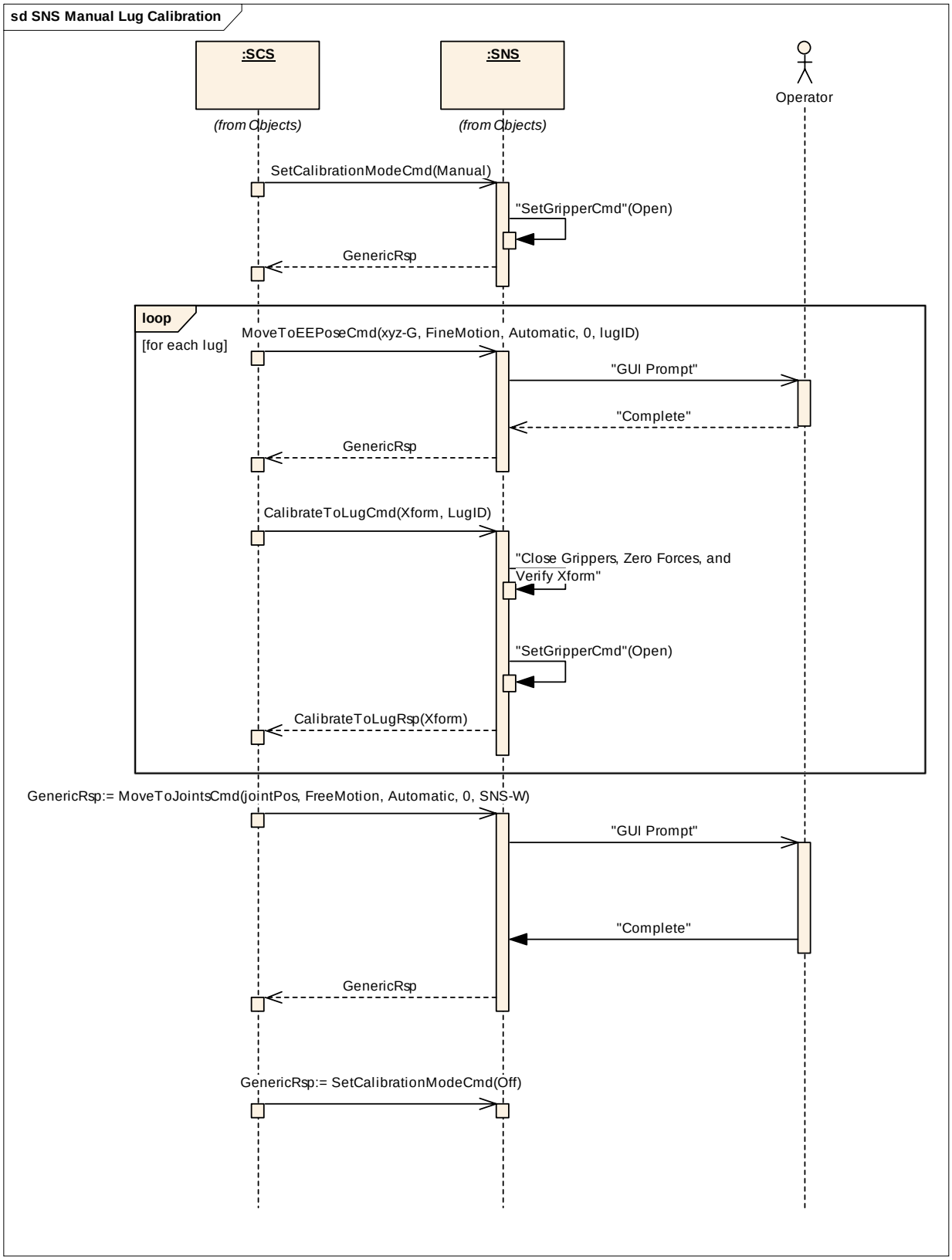
Appendix A: Interaction Diagrams



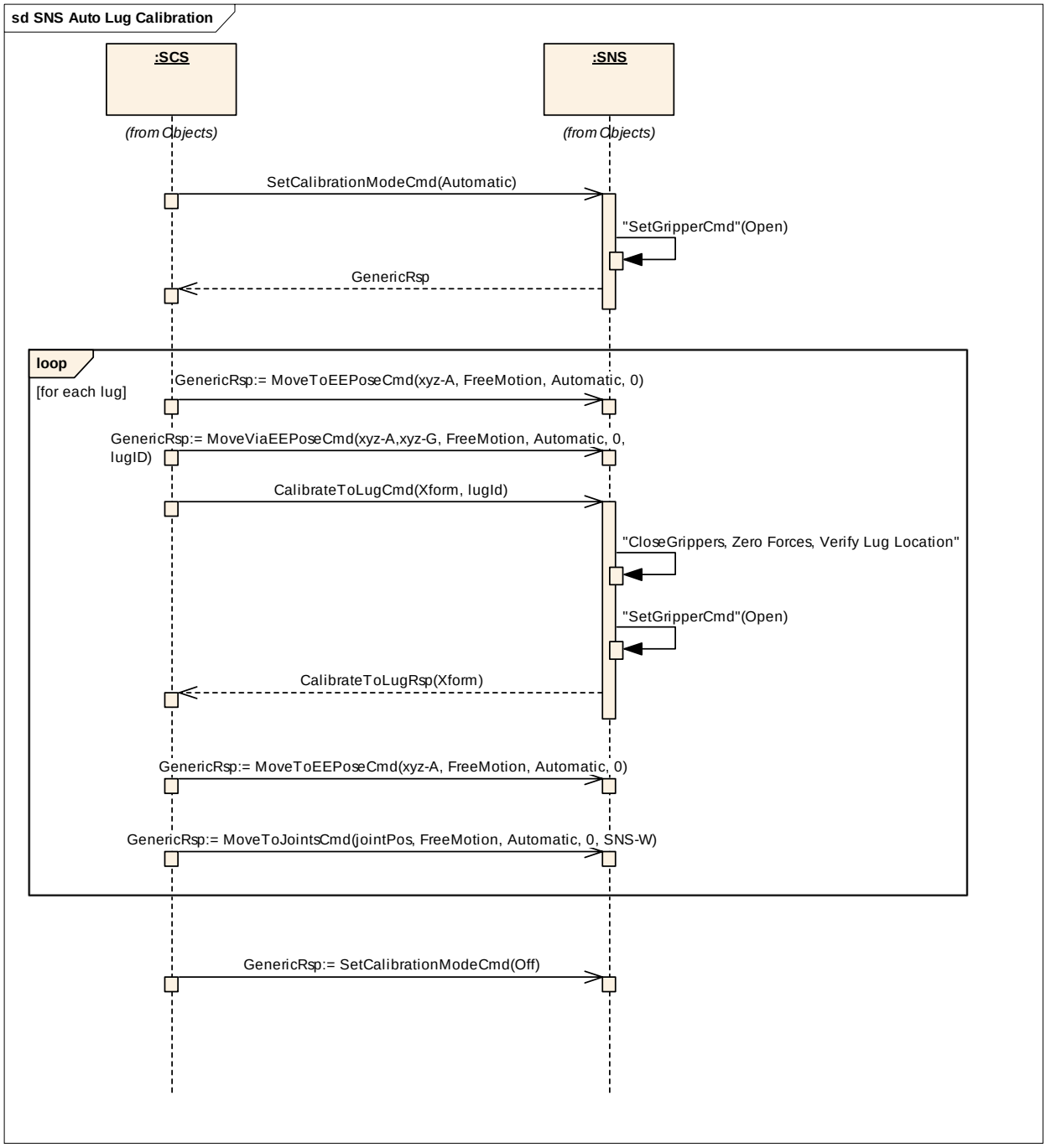
Tool Change Interaction



Supply Dispense Interaction



Manual Calibration Interaction



Automatic Calibration Interaction

Appendix B: Layout and Timing Analysis

Doc. No. CTL-038-R01

SUBSYSTEM INTERFACE LOCATIONS

Trauma Pod Phase I

Date: September 15, 2005

Prepared by

Josh Pholsiri

Chetan Kapoor
University of Texas

for

USA Medical Research Acquisition Activity
820 Chandler St.
Fort Detrick, MD 21702-5014

Layout Analysis and Interface Locations

The purpose of this document is to identify the physical locations of the interfaces for all subsystems and the physical locations of all the points of interest as defined in the logical layout below. For points that are also the SNS grab points, the approach points are also identified. Grab locations are designated by G and approach points as A in the logical layout illustrated in Figure 15.

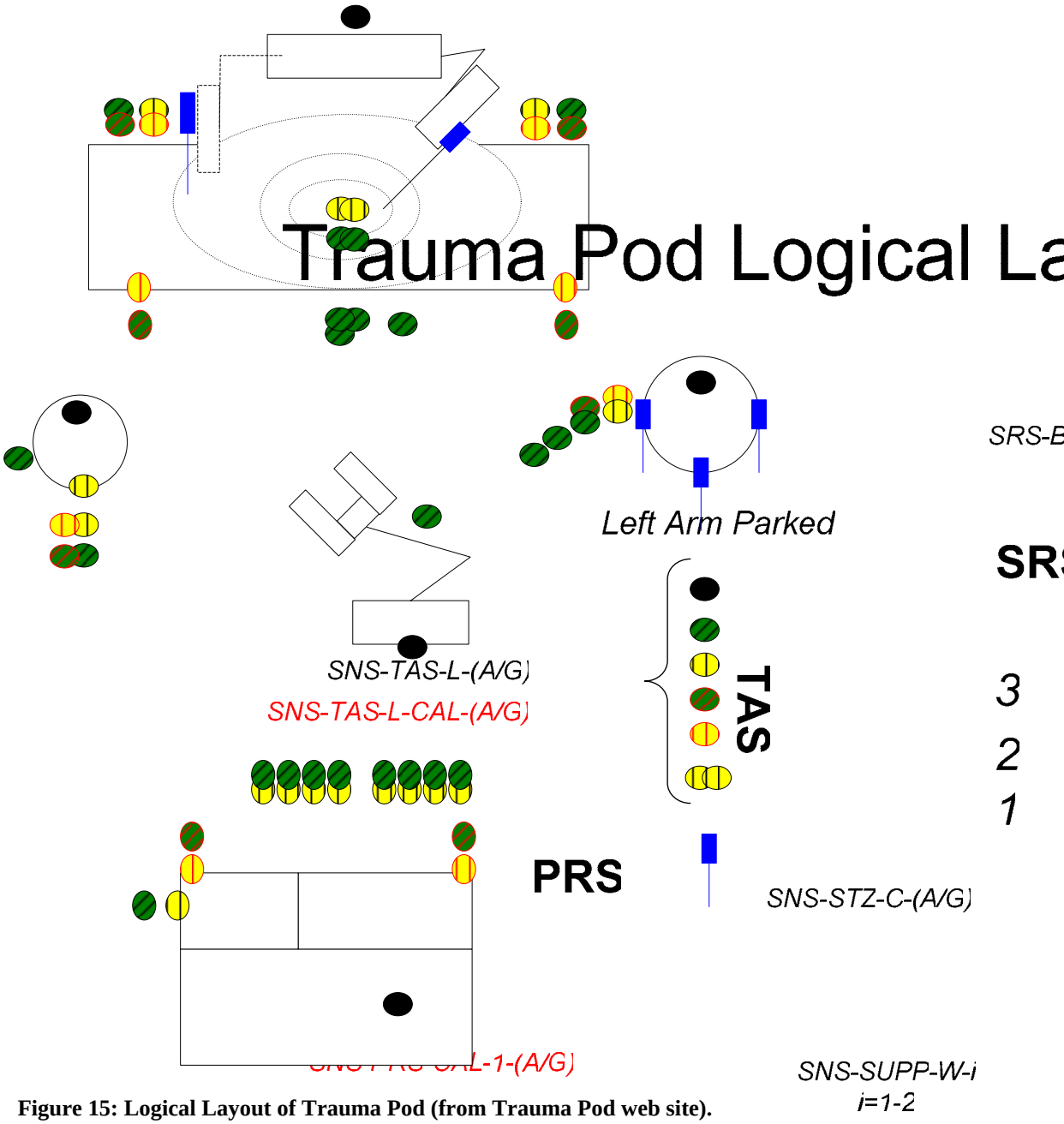


Figure 15: Logical Layout of Trauma Pod (from Trauma Pod web site).

Figure 16 depicts the physical layout of trauma pod as of 12/02/2005. This layout has been achieved through motion and workspace analysis of all the relevant subsystems.

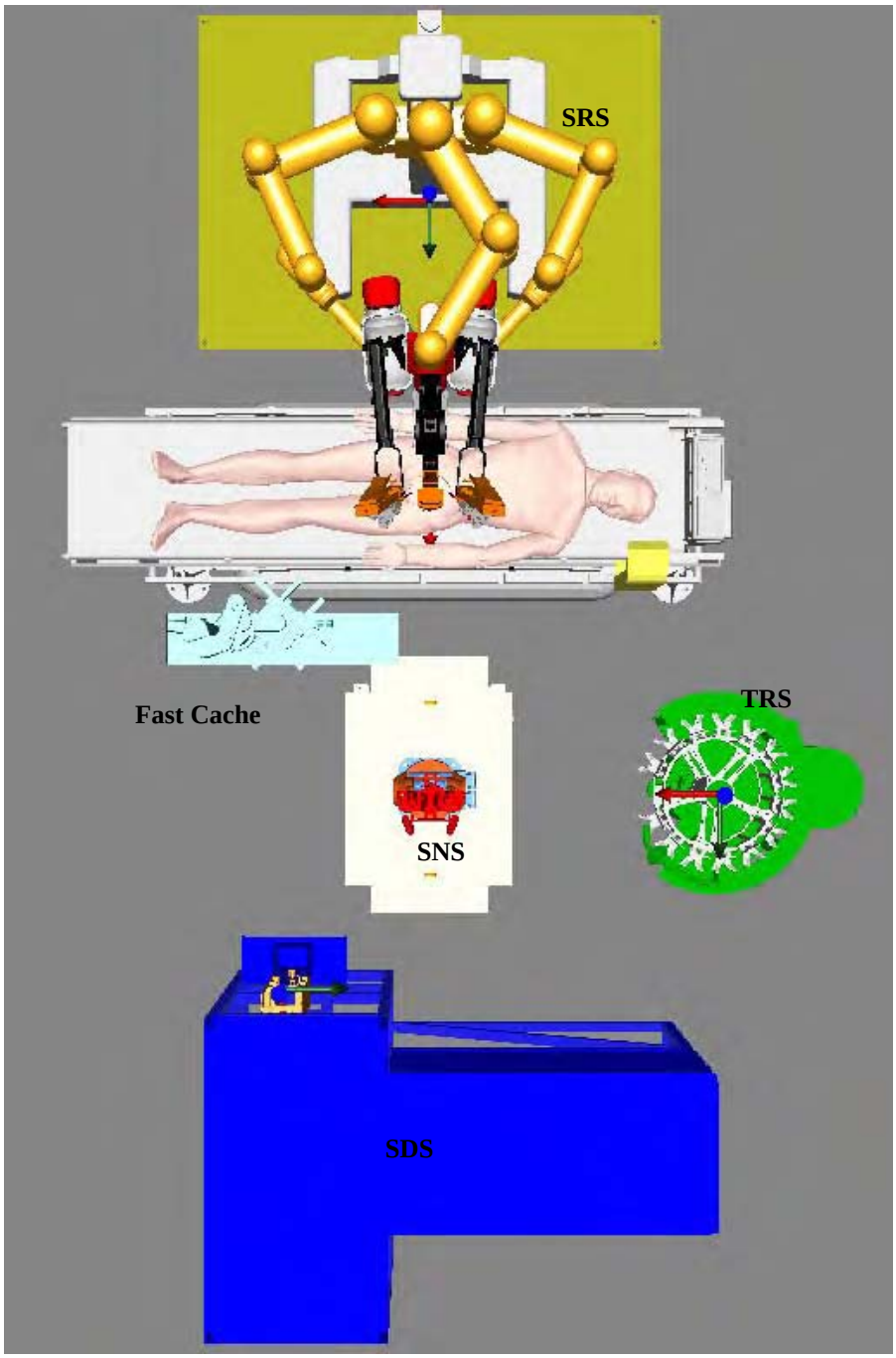


Figure 16: Physical layout of Trauma Pod.

SNS and Global Coordinates

It is agreed upon that all coordinates should be measured with respect to SNS, so we define the global coordinates as follow. The global origin is located directly below the center of the SNS base (inline with the first SNS joint) on the floor and the X and Y axes1 are shown in Figure 17. The Z direction points upward. A location consists of a position and an orientation. All the positions in this document are measured in *meters* with respect to this frame. Since we're operating on a patient who lies on PRS, its height dictates the Z-location of SNS. It was determined that *SNS pedestal should be approximately 0.718 m high.* Note also that the cable connectors of SNS robot faces towards TRS. Note that, the joint angles of the SNS robot are all zeros in the configuration shown in Figure 17.

Figure 17: SNS and Global Coordinate Frame

1 In this document, the coordinate axes are represented as follows: X is red, Y is green, and Z is blue, unless otherwise stated.

The SNS at home position is shown in Figure 18. The joint angles of the SNS at home position are 0.0, 0.7854, 0.0, -1.5708, 0.0, -0.7854, and 0.0 radians from Joint 1 to Joint 7, respectively (or 0, 45, 0, -90, 0, -45, and 0 degrees). Also shown in the figure are the definitions of the grippers EE1 and EE2.

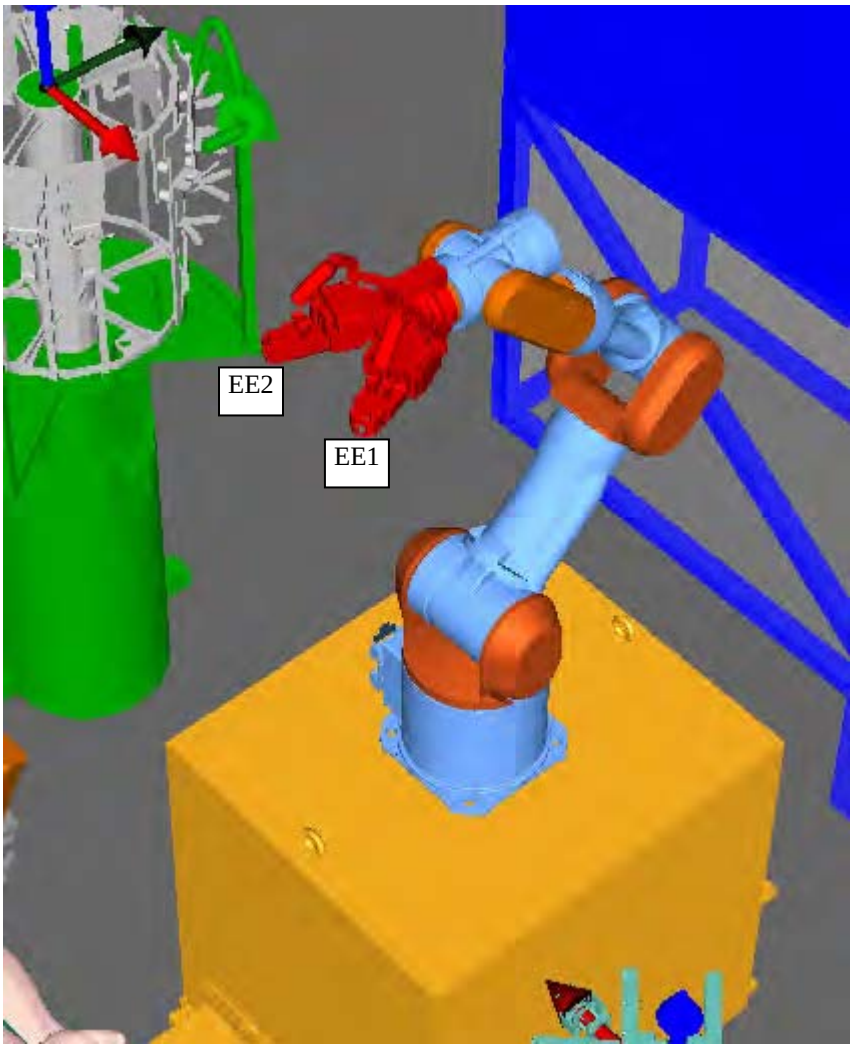


Figure 18: SNS at Home Position.

Grab Locations

Pictures in Figure 19 (a) show the grab frames of the SNS grippers. The origins of these frames are in the middle of the two fingers and should be where the center of the lug should be when SNS grabs something (tray or tool). The physical locations (positions and orientations) to be defined later in this document are essentially the locations of one of these grab frames in the 3D workspace with respect to the global coordinates defined above.

Figure 19: Grab Frames of the Grippers.

SDS

When positioning SDS, we need to ensure that SNS can reach the following points: SNS-SDS-1 to 10, SNS-SDS-W, SNS-SDS-CAL-1, and SNS-SDS-CAL-2 (top and bottom SDS calibration lugs). The SDS must be placed in such a location that the points of interest in Table 6 can be achieved. Figure 20 shows SDS and its origin, which is located right at the top calibration lug.

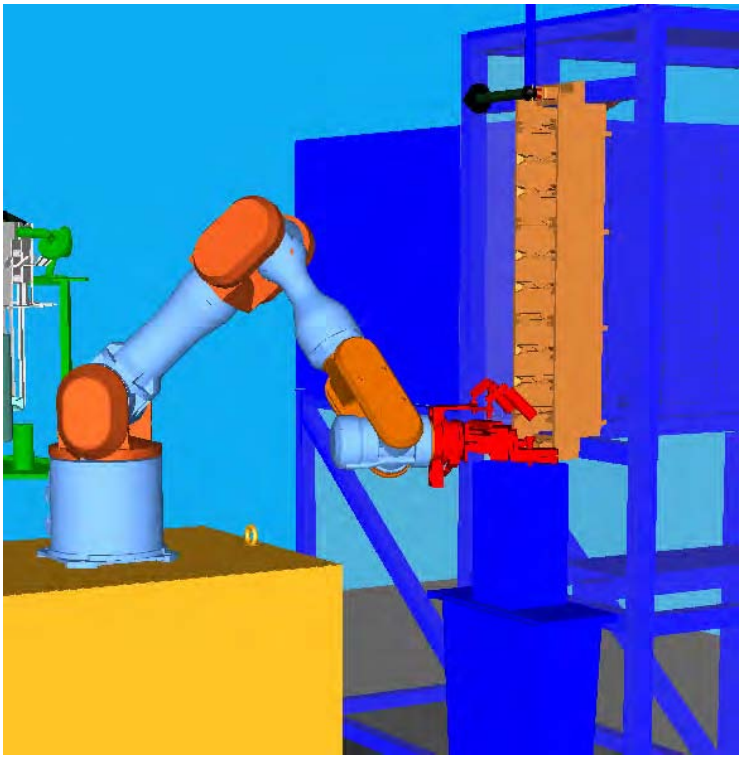


Figure 20: Location of SDS.

Location	Description	Coordinates (X,Y,Z)

SDS-BASE	Origin of SDS	(-0.65, 0.50, 1.61)
SNS-SDS-CAL-1-G	Grab point for top SDS calibration lug	(-0.65, 0.50, 1.61)
SNS-SDS-CAL-1-A	Approach point for top SDS calibration lug	(-0.55, 0.50, 1.61)
SNS-SDS-CAL-2-G	Grab point for bottom SDS calibration lug	(-0.65, 0.50, 0.95)
SNS-SDS-CAL-2-A	Approach point for bottom SDS calibration lug	(-0.55, 0.50, 0.95)
SNS-SDS-1-G	Grab point at the topmost slot2	(-0.65, 0.50, 1.55)
SNS-SDS-1-A	Approach point at the topmost slot	(-0.50, 0.50, 1.55)
SNS-SDS-10-G	Grab point at the bottommost slot	(-0.65, 0.50, 1.01)
SNS-SDS-10-A	Approach point at the bottommost slot	(-0.50, 0.50, 1.01)
SNS-SDS-W-G	Grab point above the waste basket	(-0.57, 0.50, 0.92)
SNS-SDS-W-A	Approach point above the waste basket	(-0.42, 0.50, 0.92)

Table 6: SDS Positions of Interest in Global Coordinates.

The orientations of all these locations are the same and can be described by the following rotation matrix: [0 0 -1, 0 -1 0, -1 0 0]3. This is basically the orientation the SNS gripper needs to be in for a successful grab. *Note that these points are contingent upon the background plate being moved inside closer to the SDS slots. The background plate should be moved to -0.1 m in the Y-direction of the SDS coordinates and should be limited to 0.09 m wide (off the SDS wall) if possible.*

Fast Cache

For Fast Cache (FC), we are interested in where it is placed in the workspace, where the SNS-FC grab point is, whether or not it can reach the two STZ's, all of these without interfering with other subsystems, especially the PRS and SNS. To simply FC positioning, the FC is mounted on a linear track. For easy identification, we'll use the top left corner of the surface of the linear track plate as the origin of the reference frame of FC. This frame is shown in Figure 21. The location of this frame in the global coordinates is the position of (0.668, 1.017, 0.1813) m with the orientation of [0 1 0, -1 0 0, 0 0 1]. Figure 21 also shows FC at the home position proposed by GDRS.

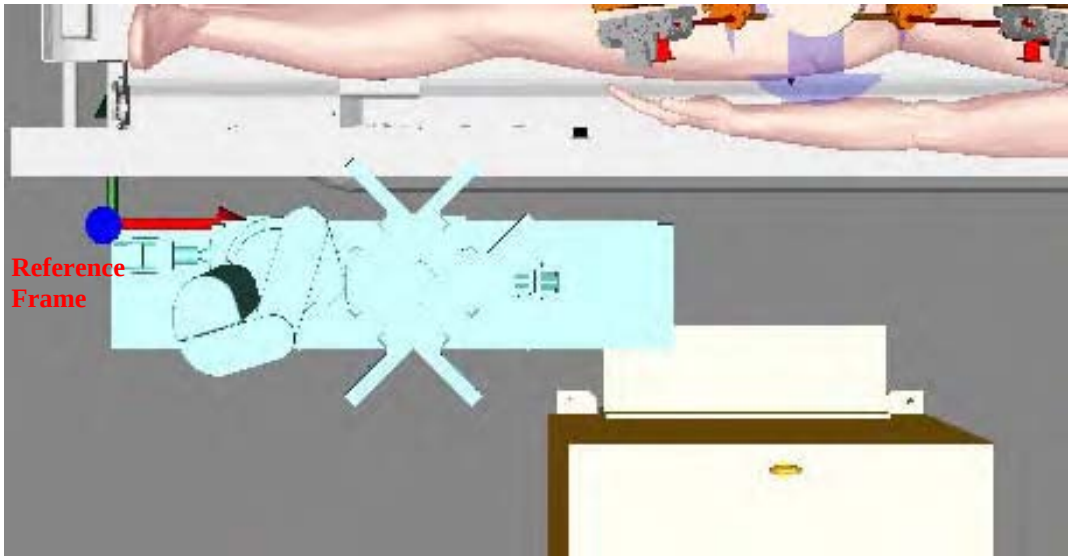


Figure 21: FC Location.

Table 7 contains the positions of interest of FC in the global coordinates from an SNS perspective. The orientation of the locations in Table 7 (except FC-BASE) is [0 -1 0, 0 0 1, -1 0 0].

2 The grab and approach points for the other 8 slots are evenly distributed between the topmost and the bottommost slots.

3 To be able to write a rotation matrix on one line, all the rotation matrices are represented as row-major vectors, with commas separating each row.

Location	Description	Coordinates (X,Y,Z)
FC-BASE	**Origin of FC**	**(0.668, 1.017, 0.1813)**
SNS-FC-CAL-G	**FC calibration lug**	**(0.393, 0.596, 1.071)**
SNS-FC-CAL-A	**FC calibration lug approach point**	**(0.393, 0.496, 1.071)**
SNS-FC-G	**Supply tray exchange between SNS and FC**	**(0.393, 0.596, 1.081)**
SNS-FC-A	**Approach point for SNS-FC interaction**	**(0.393, 0.446, 1.081)**

Table 7: FC Positions of interest in Global Coordinates.

With FC at the proposed location, Table 8 shows the joint displacements of FC defined for the FC's points of interest. Please refer to the document "CTL-039-R01-Fast Cache Kinematics" for detailed descriptions of these joints. These FC configurations are depicted visually in Table 9.

Point of interest	Joint 1 (m)	Joint 24 (deg)	Joint 3 (m)	Joint 4 (deg)	Joint 5 (deg)	Joint 6 (deg)	Joint 7 (deg)
FC-HOME	**0**	**0**	**0**	**-34**	**-34**	**-17**	**0**
FC-STZ	**0.452**	**105**	**-0.012**	**-130**	**130**	**-65**	**5**
FC-W	**0.452**	**54**	**-0.012**	**-90**	**90**	**-45**	**56**
FC-SNS	**0**	**-83**	**-0.012**	**-34**	**34**	**-17**	**-35**
FC-PARK	**0**	**-83**	**-0.312**	**-34**	**34**	**-17**	**-35**
FC-SNS-CAL	**0**	**-83**	**-0.012**	**-34**	**34**	**-17**	**10**

Table 8: Proposed FC Joint Configurations for Various Positions.

Note that the wrist angle (Joint 7) depends on which slot is currently being used (except for FC-SNS-CAL point in which the wrist angle must be at the specified value). So it could be the values specified in the table plus or minus -90/90/180 degrees (-1.5708/1.5708/ 3.1416 radians). To increase the effectiveness of calibration, the Calibration configuration is chosen such that it is as close to FC-SNS position as possible. So basically, FC-SNS-CAL is the same as FC-SNS except for the wrist joint. This allows SNS to move to essentially the same location during both supply exchange and calibration. Table 9 also shows the locations of the wrist of FC at different configurations in the local FC coordinates. These locations are essentially the locations of the coordinate frame shown in Figure 22. The SCS will command the FC controller to go these locations at appropriate times.

4 Note that Joint 2 angle is made up of two parts. One part is the rotation that contributes to the rotation of the entire arm and the other part is the rotation that contributes to the arm extension.

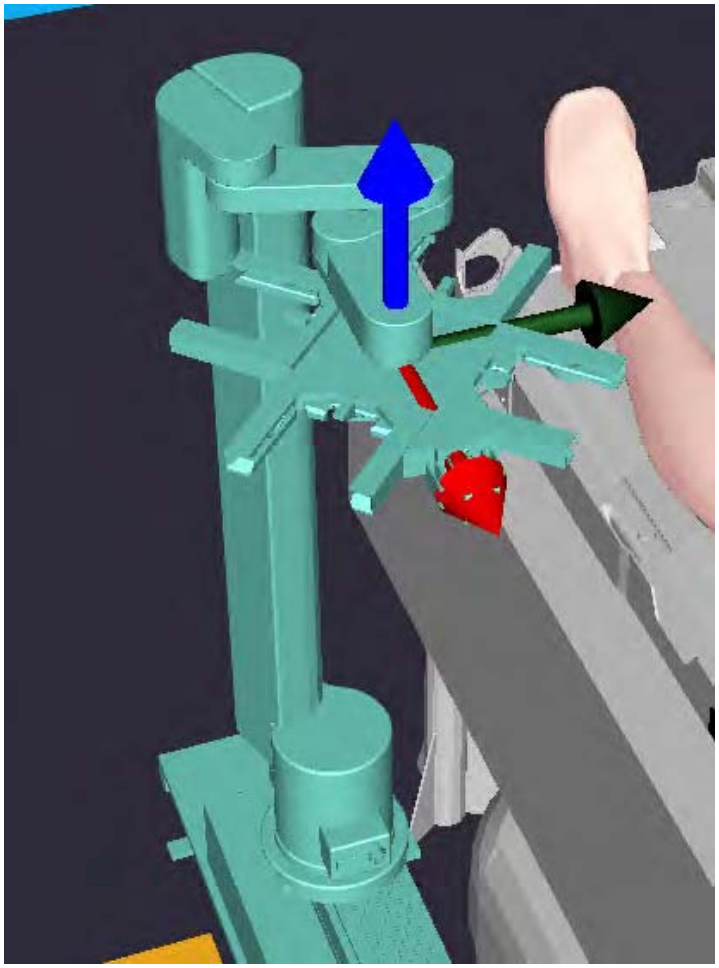
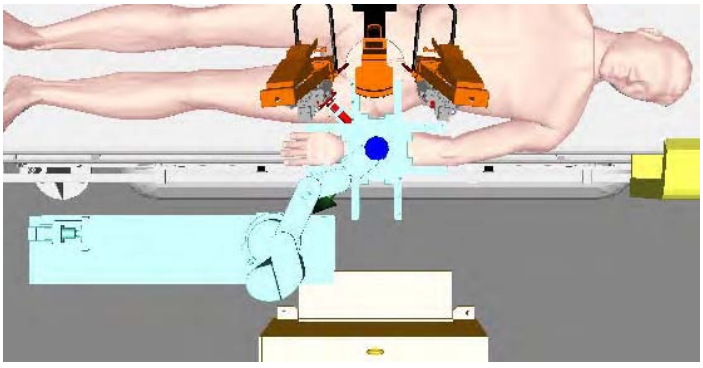


Figure 22: Coordinate frame at wrist of FC.

Point of interest	FC Configuration	Location of Wrist in Local Coordinates[5]
FC-STZ		**Position:** **(1.0159, 0.1849, 0.8894)** **Orientation:** **[-0.70711 -0.70711 0, 0.70711 -0.70711 0, 0 0 1]**

5 Note that these locations are copied from the document titled "CTL-039-R01-Fast Cache Kinematics." Please refer to that document for updated values of these locations.

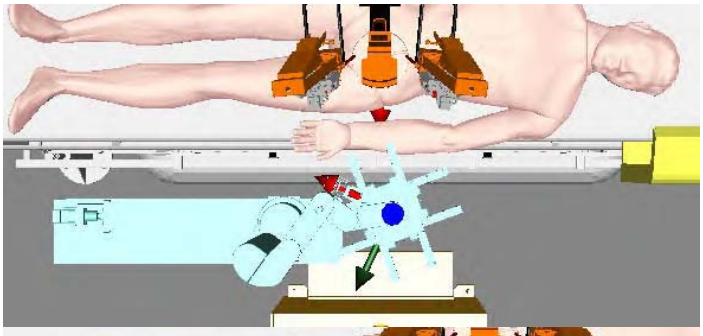
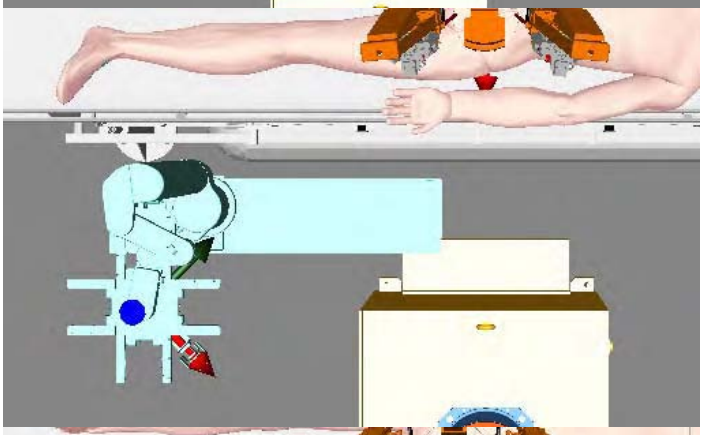
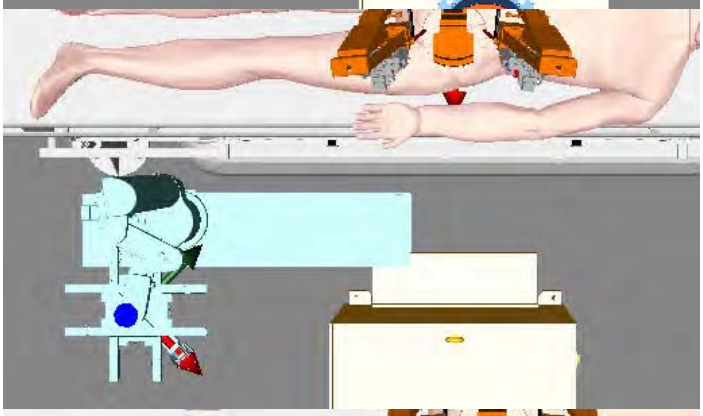
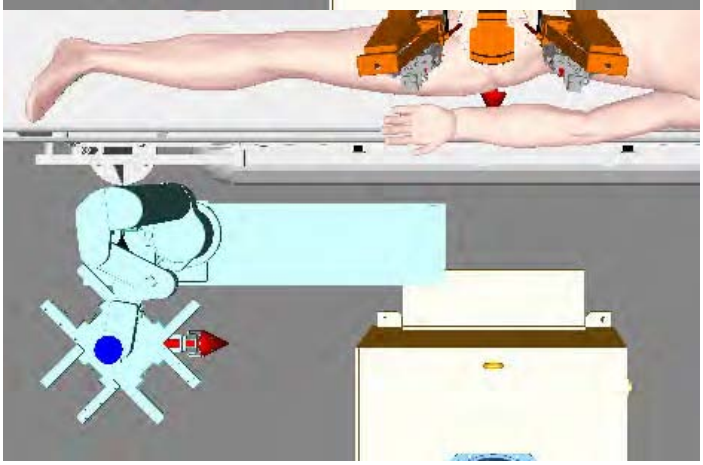
Point of interest	FC Configuration	Location of Wrist in Local Coordinates5
FC-W		**Position: (1.0443, -0.0161, 0.8894) Orientation: [-0.90631 -0.42262 0, 0.42262 0.90631 0, 0 0 1]**
FC-SNS		**Position: (0.2255, -0.2752, 0.8894) Orientation: [0.70711 0.70711 0, -0.70711 0.70711 0, 0 0 1]**
FC-PARK		**Position: (0.2255, -0.2752, 0.5894) Orientation: [0.70711 0.70711 0, -0.70711 0.70711 0, 0 0 1]**
FC-SNS-CAL		**Position: (0.2255, -0.2752, 0.8894) Orientation: [1 0 0, 0 1 0, 0 0 1]**

Table 9: Visual Representations and Locations of Various FC Configurations.

Because FC almost always operates at the top of its vertical axis, which is also the home position, the homing sequence is simplified. Assuming that when the vertical axis homes, **it always moves UP**, here's a proposed homing sequence.

1. Home the vertical axis (if necessary).
2. Home the extension arm.
3. Home the shoulder joint.
4. Home the linear track.
5. Home the wrist joint.

TRS

SNS needs to retrieve and return tools to TRS, so the SNS-TRS Grab point must be reachable by SNS. The SNS-TRS Grab point is essentially the locations of the lug on the tool when the tool is to be retrieved/ returned by SNS from/to the slot in TRS. There is one Grab point and three associated Approach points, all of which are shown in Figure 24. The two scenarios for the SNS-TRS exchange involving these Grab and Approach points are described below.

1. **SNS retrieving a tool:** Here, the active gripper does not currently hold a tool. Thus, it can come directly at SNS-TRS-G (via SNS-TRS-A-1). Once, it grabs the tool, it needs to move up to SNS-TRS-A-3 and then out to SNS-TRS-A-2.
2. **SNS returning a tool:** Here, the active gripper currently holds a tool. Therefore, it needs to pass through SNS-TRS-A-2 and SNS-TRS-A-3. It can then move down to SNS-TRS-G. Once, it lets go of the tool, it can move straight out of TRS through SNS-TRS-A-1.

The TRS calibration lug is assumed to be at the same location as SNS-TRS-G-2. It has been determined that the origin of TRS, which is at the top of the spool as shown in Figure 23, should be located at (0.0, -1.0, 1.38) m with the orientation [-1 0 0, 0 -1 0, 0 0 1]. All points relevant to the TRS are listed in Table 10. Note that the orientation of all locations (except TRS-BASE) is [0 1 0, 0 0 -1, -1 0 0]. The approach points A-1 and A-2 are basically offset a distance of 0.1 m from the grab point in the global Y direction.

Location	Description	Position (X,Y,Z)
TRS-BASE	**Origin of TRS**	**(0.0, -1.0, 1.38)**
SNS-TRS-G	**Grab point for SNS**	**(0.0, -0.73, 1.325)**
SNS-TRS-A-1	**Approach point if the active gripper does not hold a tool**	**(0.0, -0.63, 1.325)**
SNS-TRS-A-2	**Approach point if the active gripper already holds a tool**	**(0.0, -0.63, 1.37)**
SNS-TRS-A-3	**Approach point right above SNS-TRS-G**	**(0.0, -0.73, 1.37)**
SNS-TRS-CAL-G	**Calibration lug grab point for SNS**	**(0.0, -0.73, 1.325)**
SNS-TRS-CAL-A	**Calibration lug approach point for SNS**	**(0.0, -0.63, 1.325)**

Table 10: TRS grab and approach points in global coordinates

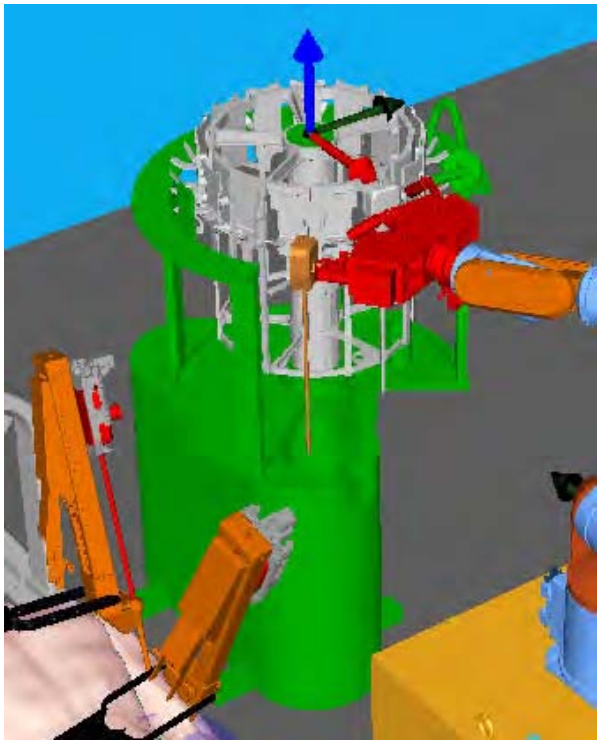


Figure 23: Origin of TRS.

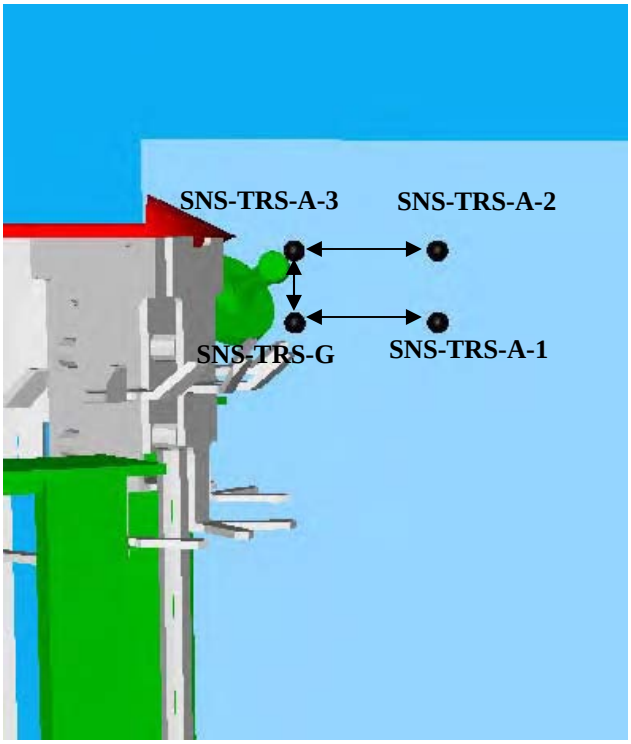


Figure 24: Grab and Approach Points for the TRS.

PRS

The location of PRS determines the locations of the Surgical Transfer Zones (STZ), which are the area where SRS and SNS exchange supplies with each other. Another important location is the area outside STZ at which SNS waits for a go-ahead command from the surgeon that he is ready for supply delivery.

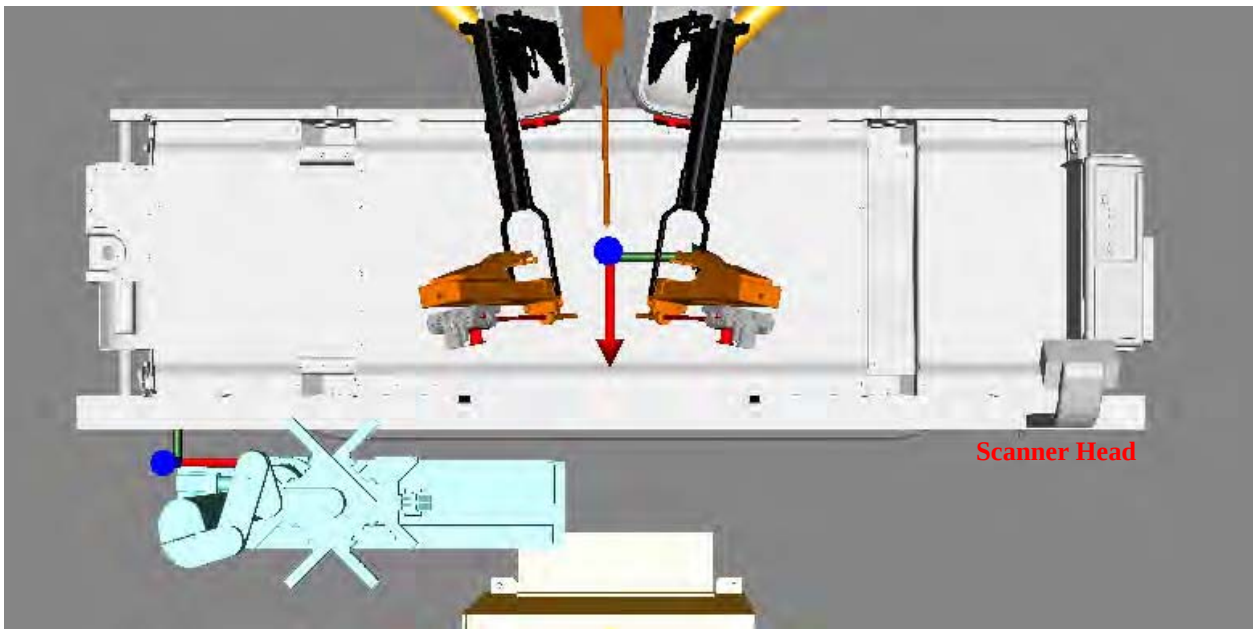


Figure 25: Origin of PRS.

Figure 25 shows the origin of PRS, which is located at the center of the top surface. The X and Y axes are shown in red and green, respectively, with the Z axis pointing out of the page. The location of the PRS origin in global coordinates: (1.1, 0.0, 0.82) with an orientation of [-1 0 0, 0 -1 0, 0 0 1]. Note that the Z location (0.8 m) is

actually whatever the height of the bed is. The scanner head, when parked, should be at 0.8 m from this origin in the Y-direction of the PRS coordinates. The locations of the calibration lugs *in the PRS coordinates* are shown in Table 11.

Location	Description	PRS Coordinates (X,Y,Z)
PRS-SNS-CAL-1	**Calibration lug 1 on PRS**	**(0.3, -0.3, 0.0)**
PRS-SNS-CAL-2	**Calibration lug 2 on PRS**	**(0.3, 0.3, 0.0)**

Table 11: PRS Calibration Lugs in PRS Coordinates.

The locations of the points of interest for PRS are shown in Table 12. The orientation of these locations (except PRS-BASE) are [0 0 1, 0 1 0, -1 0 0].

Location	Description	Position (X,Y,Z)
PRS-BASE	**Origin of PRS**	**(1.1, 0.0, 0.82)**
SNS-PRS-CAL-1-G	**Calibration lug 1 grab location on PRS**	**(0.8, 0.3, 0.82)**
SNS-PRS-CAL-1-A	**Calibration lug 1 approach location on PRS**	**(0.8, 0.3, 0.92)**
SNS-PRS-CAL-2-G	**Calibration lug 2 grab location on PRS**	**(0.8, -0.3, 0.82)**
SNS-PRS-CAL-2-A	**Calibration lug 2 approach location on PRS**	**(0.8, -0.3, 0.92)**
SNS-STZ-G	**Grab location for supply exchange between SNS and SRS**	**(0.93, 0, 1.081)**
SNS-STZ-A	**Approach location for supply exchange between SNS and SRS**	**(0.83, 0, 1.081)**

Table 12: Points of interest of PRS in Global Coordinates.

Since there will be a camera to check the supplies before they are delivered to STZ, it is desirable to have the waiting areas for both SNS and FC at the same location. With FC-W configuration provided in Table 8 and the SNS-SUPP-W location given below, this goal is accomplished.

Location	Description	Location
SNS-SUPP-W	**Waiting area before delivering supply to SRS**	**Position: (0.73, 0.0, 1.081) Orientation: [0 -0.33660 0.94165, 0 0.94165 0.33660, -1 0 0]**

Table 13: Waiting point for supply delivery

SRS

SNS needs to retrieve or insert tools to or from SRS. It thus needs to be able to reach left and right TAS. For this, we have determined that the base of SRS, which is frame F0 of SRS in the da Vinci kinematics document, should be positioned at (2.275, 0, 0.168) m with the orientation of [0 -1 0, 1 0 0, 0 0 1] as expressed in the global coordinate frame.

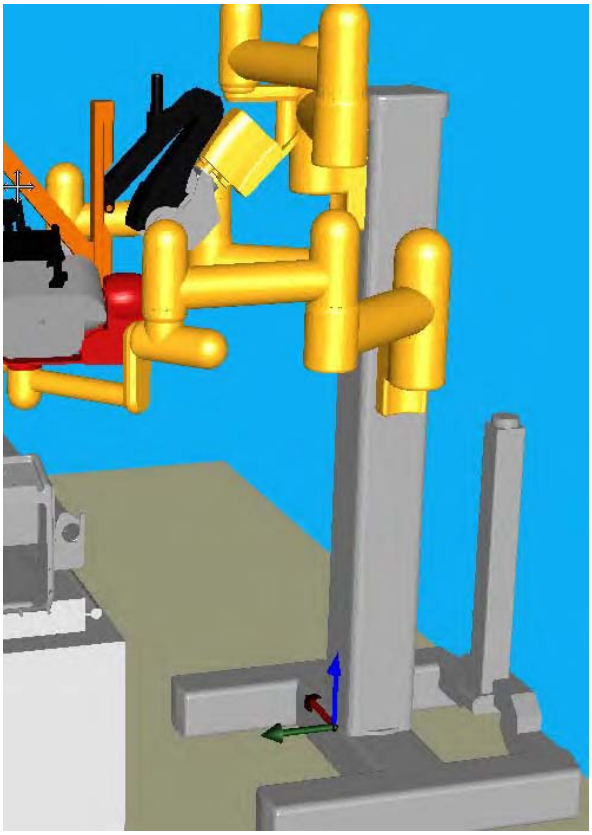


Figure 26: Base of SRS.

During a tool change, the left and right arms of SRS are commanded to move to predetermined configurations. These configurations are detailed in Table 14. SUJ stands for set up joint (or passive joint) and SLV stands for slave joint (or active joint).

Joint	SRS-Left	SRS-Right	SRS-Camera
SUJ-1 (m)	**0.1008**	**0.1008**	**0.8428**
SUJ-2 (rad / deg)	**0.3927 / 22.5**	**-0.3927 / -22.5**	**0.5441 / 31.18**
SUJ-3 (rad / deg)	**1.5882 / 91.0**	**-1.5882 / -91.0**	**-1.0884 / -62.36**
SUJ-4 (rad / deg)	**0.3229 / 18.5**	**-0.3229/ -18.5**	**-2.5974 / -148.82**
SUJ-5 (rad / deg)	**0.0550 / 3.15**	**-0.0550 / -3.15**	**N/A**
SUJ-6 (rad / deg)	**-0.6411 / -36.73**	**0.6411 / 36.73**	**N/A**
SLV-1 (rad / deg)	**-0.1747 / -10.01**	**0.1747 / 10.01**	**0.0**
SLV-2 (rad / deg)	**0.1747 / 10.01**	**0.1747 / 10.01**	**-0.7854 / -45.0**
SLV-3 (m)	**0.0**	**0.0**	**0.0**

Table 14: Joint configurations of SRS during tool change.

Based on the joint configurations in Table 14, the locations of SNS-TAS-L and SNS-TAS-R are determined and shown in Table 15.

Location	Description	Location

SRS-BASE	**Origin of SRS**	**Position:** **(2.275, 0, 0.168)** **Orientation:** **[0 -1 0,** **1 0 0,** **0 0 1]**
SNS-TAS-L-G **SNS-TAS-L-CAL-G**	**Tool change between SNS and left TAS. Same applies to calibration lug position.**	**Position:** **(0.902, 0.181, 1.567)** **Orientation:** **[0.2242 -0.08326 0.9709,** **-0.1938 0.9726 0.1281,** **-0.9550 -0.2169 0.2019]**
SNS-TAS-L-A **SNS-TAS-L-CAL-A**	**Approach points for tool change between SNS and left TAS. Same applies to calibration lug position.**	**Position:** **(0.805, 0.168, 1.547)** **Orientation:** **[0.2242 -0.08326 0.9709,** **-0.1938 0.9726 0.1281,** **-0.9550 -0.2169 0.2019]**
SNS-TAS-R-G **SNS-TAS-R-CAL-G**	**Tool change between SNS and right TAS. Same applies to calibration lug position.**	**Position:** **(0.902, -0.181, 1.567)** **Orientation:** **[0.2242 0.08326 0.9709,** **0.1938 0.9726 -0.1281,** **-0.9550 0.2169 0.2019]**
SNS-TAS-R-A **SNS-TAS-R-CAL-A**	**Approach points for tool change between SNS and right TAS. Same applies to calibration lug position.**	**Position:** **(0.805, -0.168, 1.547)** **Orientation:** **[0.2242 0.08326 0.9709,** **0.1938 0.9726 -0.1281,** **-0.9550 0.2169 0.2019]**
SNS-TOOL-W	**Waiting area before delivering tools to SRS**	**Position:** **(0.6, 0.0, 1.62)** **Orientation:** **[0 0 1,** **0 1 0,** **-1 0 0]**

Table 15: Tool-related points of interest in Global Coordinates.

Open Issues

None

Resolved Issues

1. Evaluate SNS motion with new SRS configurations (as of Dec. 1, 2005) for tool changing procedure.

   New SRS configurations (as of Dec. 1, 2005) for tool change work fine.
2. Change STZ from dual zones to a single zone and re-evaluate SNS and FC motion and configuration.

   For both the issues above, the patient seems to be pretty high above the bed, and as such, the tool tips come close to the patient during a tool change. The STZ also had to be raised to 1.055 m above the floor (global frame) because of the thickness of the patient. A lower STZ will be fine from the SNS perspective, provided the patient is thinner. At this point, we should really replace the patient CAD with the CAD of the actual phantom that we are going to use.
3. Move the top calibration lug on SDS to 1.61 m from 1.31 m and re-evaluate SNS motion.
   a. *The upper calibration lug at (-0.65, 0.50, 1.61) m in the global coordinate is fine. SNS can reach it.*
   b. *The lower calibration lug, which is now at (-0.65, 0.50, 0.95) m in the layout document, should be move up a little bit because when SNS removes trays to waste bin (the grab point at (-0.55, 0.50, 0.92) m), the tray could hit the lower cal lug. Note, however, that we don't have CAD with the lower cal lug, so this is just an educated guess. Basically, 0.92 m is the lowest SNS can go comfortably, so the waste bin can't be moved down. I'd suggest moving the lower cal lug to 0.98 m just to be safe. Of course, it could go higher now that the upper cal lug is probably at 1.61 m.*
4. Evaluate SNS motion over FC. Preliminary analysis tells us that, with motion coordination with FC, the SNS can safely move over the FC, and after a move over the FC, it can then move down to the SNS-SUPP-W-1 and SNS-SUPP-W-2 points.

   The motion coordination between SNS and FC should be like this. When FC is at FC-W (in front of STZ), SNS is most likely at SNS-HOME. If the surgeon needs a supply that is in SC, then the following sequence occurs.
   a. *SNS moves to SNS-SDS-SC while FC moves to FC-PARK at the same time.*
   b. *SNS get a tray from SC but must wait until FC reaches FC-PARK.*
   c. *Once FC is at FC-PARK, SNS can move to SNS-SUPP-W, awaiting command from the surgeon before bringing supply to SNS-STZ.*

   General rule: *FC is not allowed to move while SNS is in motion and vice versa, except for the above scenario.*
      1. *For supply exchange between SNS and FC, SNS is not allowed to move until FC rests at FC-SNS.*
5. Interference of the mounting bracket for SNS cameras and the next-to-last link of PA-10 robot. The height of the bracket has been increased and it limits the range of motion of the wrist joints (last 3 joints) of PA-10. Occasionally, this could prevent SNS from executing some of the motions.

   With the new smaller gripper, the height of the camera mounting bracket has been lowered and no longer interferes with the PA-10 link.

Appendix C: Fast Cache Kinematics

Doc. No. CTL-039-R01

FAST CACHE KINEMATICS

Trauma Pod Phase I

Date: October 21, 2005

Prepared by

Jonathan Knoll
Chetan Kapoor
University of Texas

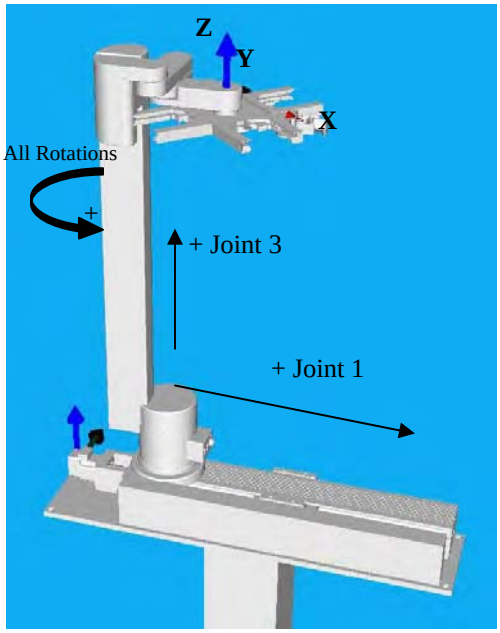
for

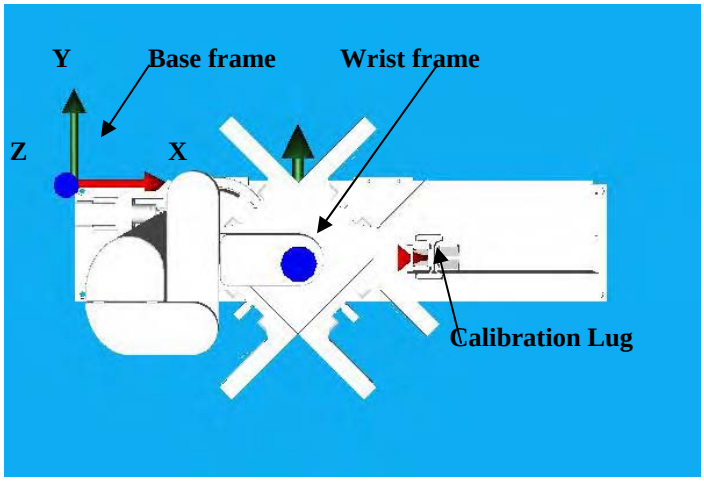
USA Medical Research Acquisition Activity
820 Chandler St.
Fort Detrick, MD 21702-5014

Zero Position of FC and +/- Directions

This model was created assuming 7 joints so that each part could be properly located. The figure below shows the FC at its zero position, i.e, all joints are at 0 value. Based upon the physical understanding of the FC, limits can be assigned to each joint. Plus and minus directions are assigned as indicated in the figure. Transformation matrices representing the Wrist frame relative to the Base frame for this and other configurations are listed in the last section of this document. Note also that these matrices are only estimates (albeit fairly accurate) since the exact location of the center of the shoulder joint at its home position is not known.

Zero Position





Forward Kinematic Analysis (going from the ground to the end effecter):

Joint 1 is the prismatic motion along the base track. **+ left to right**

Joint 2 is the rotation about the cylindrical base. **+ CC (looking from above)***

Joint 3 is the prismatic motion along the vertical beam. **+ up**

Joint 4 is the rotation of the first link attached to the vertically translating part. **+ CC (looking from above)**

Joint 5 is the rotation of the second link attached to the vertically translating part. **+ CC (looking from above)**

Joint 6 is the rotation of the link that is attached to the carousel. **+ CC (looking from above)**

Joint 7 is the rotation of the carousel. **+ CC (looking from above)**

*Note: All rotations are measured with respect to the previous link and are positive in the counter-clockwise direction.

Coupled Joint Values:

The Fast Cache can be modeled kinematically with 5 joints if the rotations that make up Joints 4, 5, and 6 are replaced with one prismatic joint. This one prismatic joint has a value of 0 when the FC is fully retracted (see zero position). This displacement variable, D, is related to the rotation, θ, of Joints 2, 4, 5, and 6 by the following equation:

$$\theta = \arcsin\left(\frac{D}{0.3}\right)$$, where D is the distance between Joint 6 and Joint 2.

Once θ is calculated:

Joint $2_{\text{prismatic part}}$* = +θ
Joint 4 = -2θ
Joint 5 = +2θ
Joint 6 = -θ

* It is important to note that each Joint 2 value is made up of two parts. One part is the rotation that contributes to the rotation of the entire arm and the other part is the rotation that contributes to the translational distance, D.

Appendix D: Trauma pod.xml

<rrgOSCAR:Workcell xmlns:rrgOSCAR="http://www.robotics.utexas.edu" xmlns:xsd="http://www.w3.org/2001/XMLSchema" Name="**Trauma Pod Version 1.0**">
<Manipulator Name="**SNS**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>0</alpha>
<a>0
<d>0</d>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>0</d>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<d>0.450</d>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>0</d>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<d>0.480</d>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>0</d>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<d>0</d>
</Frame>
</DHParameters>
<BasePose>-1 0 0 0 0 -1 0 0 0 0 1 1.035 0 0 0 1</BasePose>
<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0.07 0 0 0 1</ToolPlatePose>
<Limits>
<Position>
- <!-- These are all in degrees -->
<Frame>
<Min>-177</Min>
<Max>177</Max>
</Frame>
<Frame>
<Min>-94</Min>
<Max>94</Max>
</Frame>
<Frame>
<Min>-174</Min>
<Max>174</Max>
</Frame>
<Frame>
<Min>-137</Min>
<Max>137</Max>
</Frame>
<Frame>
<Min>-255</Min>
<Max>255</Max>

</Frame>
<Frame>
<Min>-165</Min>
<Max>165</Max>
</Frame>
<Frame>
<Min>-255</Min>
<Max>255</Max>
</Frame>
</Position>
<Velocity>
- <!-- These are all in radians/sec -->
<Frame>
<Min>-1.4</Min>
<Max>1.4</Max>
</Frame>
<Frame>
<Min>-1.4</Min>
<Max>1.4</Max>
</Frame>
<Frame>
<Min>-2.25</Min>
<Max>2.25</Max>
</Frame>
<Frame>
<Min>-2.25</Min>
<Max>2.25</Max>
</Frame>
<Frame>
<Min>-6.28</Min>
<Max>6.28</Max>
</Frame>
<Frame>
<Min>-6.28</Min>
<Max>6.28</Max>
</Frame>
<Frame>
<Min>-6.28</Min>
<Max>6.28</Max>
</Frame>
</Velocity>
<Acceleration>
- <!-- These are all in radians/sec2 -->
<Frame>
<Min>-7</Min>
<Max>7</Max>
</Frame>
<Frame>
<Min>-5</Min>
<Max>5</Max>
</Frame>
<Frame>
<Min>-10</Min>
<Max>10</Max>
</Frame>
<Frame>
<Min>-7.5</Min>
<Max>7.5</Max>
</Frame>
<Frame>
<Min>-68</Min>
<Max>68</Max>
</Frame>
<Frame>
<Min>-36</Min>
<Max>36</Max>
</Frame>
<Frame>

<Min>-40</Min>
<Max>40</Max>
</Frame>
</Acceleration>
</Limits>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>.080</Radius>
<Point>0 .125 -.025</Point>
<Point>0 -.125 -.025</Point>
</Cylisphere>
<Polytope>
<STLFile>/SNS/Frame1.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>.080</Radius>
<Point>0 0 0</Point>
<Point>0 -.400 0</Point>
</Cylisphere>
<Polytope>
<STLFile>/SNS/Frame2.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>.080</Radius>
<Point>0 .125 -.025</Point>
<Point>0 -.125 -.025</Point>
</Cylisphere>
<Polytope>
<STLFile>/SNS/Frame3.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 4**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>.090</Radius>
<Point>0 0 0</Point>
<Point>0 -.5 0</Point>
</Cylisphere>
<Polytope>
<STLFile>/SNS/Frame4.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 5**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SNS/Frame5.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 6**">

<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>.060</Radius>
<Point>0 -.125 0</Point>
<Point>0 .075 0</Point>
</Cylisphere>
<Polytope>
<STLFile>/SNS/Frame6.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 7**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Box>
<XDimension>.1</XDimension>
<YDimension>.145</YDimension>
<ZDimension>.2</ZDimension>
<Pose>1 0 0 0 0 .945519 .325568 .095 0 -.325568 .945519 .21 0 0 0 1</Pose>
</Box>
<Box>
<XDimension>.1</XDimension>
<YDimension>.145</YDimension>
<ZDimension>.2</ZDimension>
<Pose>1 0 0 0 0 .945519 -.325568 -.095 0 .325568 .945519 .21 0 0 0 1</Pose>
</Box>
<Polytope>
<STLFile>/SNS/Frame7.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**Gripper**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SNS/SimpleGripper.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**SNS-HOME**">
<JointValue>0 45 0 -90 0 -45 0</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**FastCache**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>-90</alpha>
<a>0
<theta>0</theta>
<Offset>-0.255229</Offset>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<d>0</d>
<Offset>0</Offset>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.0750
<theta>0</theta>

<Offset>0.7815576</Offset>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0
<d>0</d>
<Offset>180</Offset>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.15
<d>-0.027</d>
<Offset>180</Offset>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.075
<d>-0.027</d>
<Offset>90</Offset>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.1222375
<d>-0.03987</d>
<Offset>0</Offset>
</Frame>
</DHParameters>
<BasePose>0 1 0 .51723 -1 0 0 -.068453 0 0 1 .21374 0 0 0 1</BasePose>
<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/FastCache/Frame0.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/FastCache/Frame1.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>0.064</Radius>
<Point>0.01 -0.084 -0.024</Point>
<Point>0.01 -0.084 0.806</Point>
</Cylisphere>
<Polytope>
<STLFile>/FastCache/Frame2.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>0.064</Radius>
<Point>-0.02 -0.084 -0.0814</Point>
<Point>-0.02 -0.084 0</Point>
</Cylisphere>
<Polytope>

<STLFile>/FastCache/Frame3.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 4**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>0.035</Radius>
<Point>0 0 -0.013</Point>
<Point>0.15 0 -0.013</Point>
</Cylisphere>
<Polytope>
<STLFile>/FastCache/Frame4.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 5**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>0.035</Radius>
<Point>0 0 -0.013</Point>
<Point>0.075 0.000000 -0.013</Point>
</Cylisphere>
<Polytope>
<STLFile>/FastCache/Frame5.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 6**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>0.05</Radius>
<Point>0 0 -0.013</Point>
<Point>0.12 0 -0.013</Point>
</Cylisphere>
<Polytope>
<STLFile>/FastCache/Frame6.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 7**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Box>
<XDimension>0.35</XDimension>
<YDimension>0.35</YDimension>
<ZDimension>.02</ZDimension>
<Pose>0.70710678 -0.70710678 0 0 0.70710678 0.70710678 0 0 0 0 1 0 0 0 0 1</Pose>
</Box>
<Polytope>
<STLFile>/FastCache/Frame7.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**FC-HOME**">
<JointValue>0 0 0 -34 34 -17 0</JointValue>
</Configuration>
<Configuration Name="**FC-STZ**">
<JointValue>0.452 105 -0.012 -130 130 -65 95</JointValue>
</Configuration>
<Configuration Name="**FC-W**">
<JointValue>0.452 54 -0.012 -90 90 -45 146</JointValue>
</Configuration>

<Configuration Name="**FC-SNS**">
<JointValue>0 -83 -0.012 -34 34 -17 55</JointValue>
</Configuration>
<Configuration Name="**FC-PARK**">
<JointValue>0 -83 -0.312 -34 34 -17 55</JointValue>
</Configuration>
<Configuration Name="**FC-SNS-CAL**">
<JointValue>0 -83 -0.012 -34 34 -17 100</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**SRS Left SUJ**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>0</alpha>
<a>0.0896
<theta>0</theta>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0
<d>0.4166</d>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.4318
<d>0.1429</d>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.4318
<d>-0.1302</d>
<Offset>90</Offset>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<d>0.4089</d>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>-0.1029</d>
<Offset>-90</Offset>
</Frame>
</DHParameters>
<BasePose>1 0 0 0.1016 0 1 0 -.1016 0 0 1 .43 0 0 0 1</BasePose>
- <!-- Identity -->
<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame1.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame2.STL</STLFile>
</Polytope>
</Primitives>
</Frame>

<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame3.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 4**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame4.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 5**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame5.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 6**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame6.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**TOOL-PARK**">
<JointValue>0.1008 22.5 91 18.5 3.15 -36.73</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**SRS Left PSM**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>90</alpha>
<a>0
<d>0</d>
<Offset>90</Offset>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>0</d>
<Offset>-90</Offset>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<theta>0</theta>
<Offset>-0.4318</Offset>
</Frame>
</DHParameters>
<BasePose>0 1 0 0.4864 -1 0 0 0 0 0 1 0.1524 0 0 0 1</BasePose>
- <!-- Identity -->
<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>

<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame7.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**4BarA**">
<LocalXform>1 0 0 0 0 0 1 0 0 -1 0 0.52776 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/4BarA.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**4BarB**">
<LocalXform>1 0 0 0.039444015 0 1 0 0.147207097762 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/4BarB.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**4BarC**">
<LocalXform>1 0 0 0.039444015 0 1 0 0.18531 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/4BarC.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame8.STL</STLFile>
</Polytope>
<Cylisphere>
<Radius>.050</Radius>
<Point>0.040000 0.550000 0.000000</Point>
<Point>0.040000 0.100000 0.000000</Point>
</Cylisphere>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/TAS.STL</STLFile>
</Polytope>
<Box>
<XDimension>0.05</XDimension>
<YDimension>0.1</YDimension>
<ZDimension>0.15</ZDimension>
<Pose>1 0 0 0 0 1 0 0 0 0 1 -0.02 0 0 0 1</Pose>
</Box>
</Primitives>
<Children>
<Node Name="**Tool2**">
<LocalXform>0 0 1 0 0 -1 0 0 1 0 0 0 0 0 0 1</LocalXform>
<Primitives>
<Box>
<XDimension>0.104</XDimension>
<YDimension>0.065</YDimension>
<ZDimension>0.035</ZDimension>
<Pose>1 0 0 -0.012 0 1 0 0 0 0 1 0.025 0 0 0 1</Pose>
</Box>

<Cylisphere>
<Radius>.01</Radius>
<Point>0.040000 0.000000 0.025000</Point>
<Point>0.430000 0.000000 0.025000</Point>
</Cylisphere>
</Primitives>
</Node>
</Children>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**TOOL-PARK**">
<JointValue>-10.01 10.01 0</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**SRS Right SUJ**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>0</alpha>
<a>0.0896
<theta>0</theta>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0
<d>0.4166</d>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.4318
<d>0.1429</d>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.4318
<d>-0.1302</d>
<Offset>90</Offset>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<d>0.4089</d>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>-0.1029</d>
<Offset>-90</Offset>
</Frame>
</DHParameters>
<BasePose>-1 0 0 -.1016 0 -1 0 -.1016 0 0 1 0.43 0 0 0 1</BasePose>
- <!-- +180 Z -->
<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame1.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>

<Polytope>
<STLFile>/SRS/ToolArms/Frame2.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame3.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 4**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame4.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 5**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame5.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 6**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame6.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**TOOL-PARK**">
<JointValue>0.1008 -22.5 -91 -18.5 -3.15 36.73</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**SRS Right PSM**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>90</alpha>
<a>0
<d>0</d>
<Offset>90</Offset>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>0</d>
<Offset>-90</Offset>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<theta>0</theta>
<Offset>-0.4318</Offset>
</Frame>
</DHParameters>
<BasePose>0 1 0 0.4864 -1 0 0 0 0 0 1 0.1524 0 0 0 1</BasePose>
- <!-- -90 Z -->

<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame7.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**4BarA**">
<LocalXform>1 0 0 0 0 0 1 0 0 -1 0 0.52776 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/4BarA.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**4BarB**">
<LocalXform>1 0 0 0.039444015 0 1 0 0.147207097762 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/4BarB.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**4BarC**">
<LocalXform>1 0 0 0.039444015 0 1 0 0.18531 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/4BarC.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>.050</Radius>
<Point>0.040000 0.550000 0.000000</Point>
<Point>0.040000 0.100000 0.000000</Point>
</Cylisphere>
<Polytope>
<STLFile>/SRS/ToolArms/Frame8.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/TAS.STL</STLFile>
</Polytope>
<Box>
<XDimension>0.05</XDimension>
<YDimension>0.1</YDimension>
<ZDimension>0.15</ZDimension>
<Pose>1 0 0 0 0 1 0 0 0 0 1 -0.02 0 0 0 1</Pose>
</Box>
</Primitives>
<Children>
<Node Name="**Tool1**">
<LocalXform>0 0 1 0 0 -1 0 0 1 0 0 0 0 0 0 1</LocalXform>
<Primitives>
<Box>

<XDimension>0.104</XDimension>
<YDimension>0.065</YDimension>
<ZDimension>0.035</ZDimension>
<Pose>1 0 0 -0.012 0 1 0 0 0 0 1 0.025 0 0 0 1</Pose>
</Box>
<Cylisphere>
<Radius>.01</Radius>
<Point>0.040000 0.000000 0.025000</Point>
<Point>0.430000 0.000000 0.025000</Point>
</Cylisphere>
</Primitives>
</Node>
</Children>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**TOOL-PARK**">
<JointValue>10.01 10.01 0</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**SRS Camera SUJ**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>0</alpha>
<a>0.0896
<theta>0</theta>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0
<d>0.4166</d>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.4318
<d>0.1429</d>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0.4318
<d>-0.3432</d>
<Offset>90</Offset>
</Frame>
</DHParameters>
<BasePose>0 -1 0 0 1 0 0 0 0 0 1 0.43 0 0 0 1</BasePose>
- <!-- +90 Z -->
<ToolPlatePose>0 -1 0 0 0.707107 0 .707107 -.047164 -.707107 0 .707107 .047164 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame1.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame2.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">

<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Frame3.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 4**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/CameraArm/Frame4A.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**Frame 4B**">
<LocalXform>0 -1 0 0 0.707107 0 .707107 0 -.707107 0 .707107 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/CameraArm/Frame4B.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Frame>
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**TOOL-PARK**">
<JointValue>0.8428 31.18 -62.36 -148.82</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<Manipulator Name="**SRS Camera ECM**">
<DHParameters>
<FrameLocationMethod>Paul</FrameLocationMethod>
<Frame>
<alpha>90</alpha>
<a>0
<d>0</d>
<Offset>90</Offset>
</Frame>
<Frame>
<alpha>-90</alpha>
<a>0
<d>0</d>
<Offset>-90</Offset>
</Frame>
<Frame>
<alpha>90</alpha>
<a>0
<theta>0</theta>
<Offset>-0.3822</Offset>
</Frame>
<Frame>
<alpha>0</alpha>
<a>0
<d>0.3639</d>
</Frame>
</DHParameters>
<BasePose>0 1 0 0.6126 -1 0 0 0 0 0 1 0.1016 0 0 0 1</BasePose>
- <!-- -90 Z -->
<ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolPlatePose>
<ObstacleModel>
<Frame Name="**Frame 0**" />
<Frame Name="**Frame 1**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>

<STLFile>/SRS/CameraArm/Frame5.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**4BarCamA**">
<LocalXform>1 0 0 0 0 0 1 0 0 -1 0 0.34036 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/CameraArm/4BarCamA.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**4BarCamB**">
<LocalXform>1 0 0 0.1042416 0 1 0 0.28642055 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/CameraArm/4BarCamB.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Frame>
<Frame Name="**Frame 2**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/CameraArm/Frame7.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 3**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/CameraArm/CameraAndBoom.STL</STLFile>
</Polytope>
</Primitives>
</Frame>
<Frame Name="**Frame 4**" />
</ObstacleModel>
<ConfigurationsOfInterest>
<Configuration Name="**TOOL-PARK**">
<JointValue>0 -45 0 0</JointValue>
</Configuration>
</ConfigurationsOfInterest>
</Manipulator>
<EnvironmentObstacleModel>
<Node Name="**PRS**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Box>
<XDimension>0.75</XDimension>
<YDimension>2.23</YDimension>
<ZDimension>0.79</ZDimension>
<Pose>1 0 0 0 0 1 0 0 0 0 1 -0.44 0 0 0 1</Pose>
</Box>
<Box>
<XDimension>.64</XDimension>
<YDimension>1.84</YDimension>
<ZDimension>.24</ZDimension>
<Pose>1 0 0 0.05 0 1 0 -0.05 0 0 1 .11 0 0 0 1</Pose>
</Box>
<Polytope>
<STLFile>/PRS/LSTAT.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**Gurney**">

<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/PRS/Gurney.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**Scanner**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/PRS/SurgeryScanner.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**Patient**">
<LocalXform>1 0 0 0.0 0 1 0 0.17881 0 0 1 0.09707 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/PRS/Body.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Node>
<Node Name="**Floor**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 -0.015 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/PRS/Floor.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**SDS**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Box>
<XDimension>1.03</XDimension>
<YDimension>0.63</YDimension>
<ZDimension>1.64</ZDimension>
<Pose>1 0 0 0.595 0 1 0 0.1 0 0 1 -0.26 0 0 0 1</Pose>
</Box>
<Box>
<XDimension>0.59</XDimension>
<YDimension>1.05</YDimension>
<ZDimension>1.24</ZDimension>
<Pose>1 0 0 0.68 0 1 0 0.91 0 0 1 -0.455 0 0 0 1</Pose>
</Box>
<Box>
<XDimension>0.06</XDimension>
<YDimension>0.144</YDimension>
<ZDimension>0.38</ZDimension>
<Pose>1 0 0 0.05 0 1 0 -0.004 0 0 1 -0.14 0 0 0 1</Pose>
</Box>
<Box>
<XDimension>0.23</XDimension>
<YDimension>0.38</YDimension>
<ZDimension>0.38</ZDimension>
<Pose>1 0 0 -0.045 0 1 0 -0.004 0 0 1 -0.62 0 0 0 1</Pose>
</Box>
<Polytope>
<STLFile>/SDS/SDS.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**Plating**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>

<Polytope>
<STLFile>/SDS/Plating.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**Slow Cache**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SDS/SlowCache.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Node>
<Node Name="**TRS**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Cylisphere>
<Radius>0.35</Radius>
<Point>0 0 -0.605</Point>
<Point>0 0 -0.135</Point>
</Cylisphere>
<Cylisphere>
<Radius>0.16</Radius>
<Point>0 0 -0.935</Point>
<Point>0 0 -1.335</Point>
</Cylisphere>
<Polytope>
<STLFile>/TRS/TRSBase.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**TRS Spool**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/TRS/TRSSpool.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Node>
<Node Name="**SRS-BASE**">
<LocalXform>0 -1 0 2.2752 1 0 0 0 0 0 1 0.1684 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/Frame0.STL</STLFile>
</Polytope>
</Primitives>
</Node>
<Node Name="**SNS Base**">
<LocalXform>-1 0 0 0 0 -1 0 0 0 0 1 1.035 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SNS/Frame0.STL</STLFile>
</Polytope>
</Primitives>
<Children>
<Node Name="**SNS Pedestal**">
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Box>
<XDimension>0.72</XDimension>
<YDimension>0.61</YDimension>
<ZDimension>0.72</ZDimension>
<Pose>1 0 0 0 0 1 0 0 0 0 1 -0.67 0 0 0 1</Pose>
</Box>

<Polytope>
<STLFile>/SNS/SNSBasePedestal.STL</STLFile>
</Polytope>
</Primitives>
</Node>
</Children>
</Node>
</EnvironmentObstacleModel>
<Tool Name="**EE1 Gripper**">
<ToolTipPose>-1 0 0 -0.0223 0 -0.941647 -0.336602 -0.08886 0 -0.336602 0.941647 0.20412 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**EE2 Gripper**">
<ToolTipPose>-1 0 0 -0.0223 0 -0.941647 0.336602 0.08886 0 0.336602 0.941647 0.20412 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SRS Left Tool**">
<ToolTipPose>1 0 0 -0.0352 0 1 0 0 0 0 1 0 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SRS Right Tool**">
<ToolTipPose>1 0 0 -0.0352 0 1 0 0 0 0 1 0 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SRS-PSM-RC-TOOL**">
<ToolTipPose>1 0 0 0 0 1 0 0 0 0 1 0.4318 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SNS EE1 Tool**">
<ToolTipPose>0 0 -1 -0.0223 -.336602 .941647 0 -0.08886 .941647 .336602 0 0.20412 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SNS EE2 Tool**">
<ToolTipPose>0 0 -1 -0.0223 .336602 .941647 0 0.08886 .941647 -.336602 0 0.20412 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SNS EE1 Tray**">
<ToolTipPose>1 0 0 -0.0223 0 .941647 -.336602 -0.08886 0 .336602 .941647 0.20412 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SNS EE2 Tray**">
<ToolTipPose>1 0 0 -0.0223 0 .941647 .336602 0.08886 0 -.336602 .941647 0.20412 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**Generic Tool**">
<ToolTipPose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolTipPose>
<ObstacleModel>
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SRS/ToolArms/Tool.STL</STLFile>
</Polytope>
</Primitives>
</ObstacleModel>
</Tool>
<Tool Name="**Generic Supply Tray**">
<ToolTipPose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</ToolTipPose>
<ObstacleModel>
<LocalXform>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</LocalXform>
<Primitives>
<Polytope>
<STLFile>/SDS/SupplyTray.STL</STLFile>
</Polytope>
</Primitives>
</ObstacleModel>
</Tool>
<Tool Name="**TRS Tool Pose**">
<ToolTipPose>-1 0 0 0.275789 0 1 0 0 0 0 -1 -0.0516 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 1**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.038608 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 2**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.0751205 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 3**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.111633 0 0 0 1</ToolTipPose>

</Tool>
<Tool Name="**SDS Pose 4**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.1481455 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 5**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.184658 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 6**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.2211705 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 7**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.257683 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Pose 8**">
<ToolTipPose>0 0 1 0.006429 0 -1 0 0 1 0 0 -.2941955 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**FC Tool Pose**">
<ToolTipPose>0 0.7071 -.7071 0.13819 0 .7071 .7071 -0.13819 1 0 0 0.011 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**Left Camera**">
<ToolTipPose>0 0 -1 -0.02454 -1 0 0 .04273 0 1 0 0.1185 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**Right Camera**">
<ToolTipPose>0 0 -1 -0.02454 -1 0 0 -.04273 0 1 0 0.1185 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SRS Camera**">
<ToolTipPose>1 0 0 0 0 0.5 -.866025 0 0 .866025 0.5 0.1 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**TRS Camera**">
<ToolTipPose>-1 0 0 0.250508 0 -1 0 0.107944 0 0 1 0.0225 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SNS Left Camera**">
<ToolTipPose>0 -.70711 .70711 .054449 .941647 -.23801 -.23801 -0.0953805 .336602 .665845 .665845 0.1778178 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SNS Right Camera**">
<ToolTipPose>0 -.70711 .70711 .054449 .941647 .23801 .23801 0.0953805 -.336602 .665845 .665845 0.1778178 0 0 0 1</ToolTipPose>
</Tool>
<Tool Name="**SDS Camera**">
<ToolTipPose>-1 0 0 -0.006607 0 -1 0 0.40708 0 0 1 -0.152115 0 0 0 1</ToolTipPose>
</Tool>
<FramesOfInterest>
<Frame Name="**SNS-TRS-G**">
<Pose>0 1 0 0 0 0 -1 -0.73 -1 0 0 1.325 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TRS-A-1**">
<Pose>0 1 0 0 0 0 -1 -0.63 -1 0 0 1.325 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TRS-A-2**">
<Pose>0 1 0 0 0 0 -1 -0.63 -1 0 0 1.37 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TRS-A-3**">
<Pose>0 1 0 0 0 0 -1 -0.73 -1 0 0 1.37 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-CAL-1-A**">
<Pose>0 0 -1 -0.55 0 -1 0 0.5 -1 0 0 1.61 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-CAL-1-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.61 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-CAL-2-A**">
<Pose>0 0 -1 -0.55 0 -1 0 0.5 -1 0 0 0.95 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-CAL-2-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 0.95 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-1-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.55 0 0 0 1</Pose>
</Frame>

<Frame Name="**SNS-SDS-1-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.55 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-2-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.49 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-2-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.49 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-3-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.43 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-3-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.43 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-4-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.37 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-4-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.37 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-5-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.31 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-5-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.31 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-6-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.25 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-6-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.25 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-7-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.19 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-7-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.19 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-8-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.13 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-8-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.13 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-9-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.07 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-9-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.07 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-10-A**">
<Pose>0 0 -1 -0.5 0 -1 0 0.5 -1 0 0 1.01 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-10-G**">
<Pose>0 0 -1 -0.65 0 -1 0 0.5 -1 0 0 1.01 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-W-A**">
<Pose>0 0 -1 -0.42 0 -1 0 0.5 -1 0 0 0.92 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SDS-W-G**">
<Pose>0 0 -1 -0.57 0 -1 0 0.5 -1 0 0 0.92 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-STZ-A**">
<Pose>0 0 1 0.83 0 1 0 0.0 -1 0 0 1.08069 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-STZ-G**">
<Pose>0 0 1 0.93 0 1 0 0.0 -1 0 0 1.08069 0 0 0 1</Pose>

</Frame>
<Frame Name="**SNS-TAS-L-A**">
<Pose>0.224222 -0.0832619 0.970975 0.804999 -0.193817 0.97263 0.128161 0.168401 -0.955071 -0.216928 0.201947 1.54728 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TAS-L-G**">
<Pose>0.224222 -0.0832619 0.970975 0.902096 -0.193817 0.97263 0.128161 0.181217 -0.955071 -0.216928 0.201947 1.56747 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TAS-R-A**">
<Pose>0.224222 0.0832619 0.970975 0.804999 0.193817 0.97263 -0.128161 -0.168401 -0.955071 0.216928 0.201947 1.54728 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TAS-R-G**">
<Pose>0.224222 0.0832619 0.970975 0.902096 0.193817 0.97263 -0.128161 -0.181217 -0.955071 0.216928 0.201947 1.56747 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-TOOL-W**">
<Pose>0 0 1 0.6 0 1 0 0 -1 0 0 1.62048 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-SUPP-W**">
<Pose>0 -0.33660 0.94165 0.73 0 0.94165 0.33660 0.0 -1 0 0 1.08069 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-PRS-CAL-1-A**">
<Pose>0 0 1 0.8 0 1 0 0.3 -1 0 0 0.920478 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-PRS-CAL-1-G**">
<Pose>0 0 1 0.8 0 1 0 0.3 -1 0 0 0.820478 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-PRS-CAL-2-A**">
<Pose>0 0 1 0.8 0 1 0 -0.3 -1 0 0 0.920478 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-PRS-CAL-2-G**">
<Pose>0 0 1 0.8 0 1 0 -0.3 -1 0 0 0.820478 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-FC-CAL-A**">
<Pose>0 -1 0 0.39326 0 0 1 0.49626 -1 0 0 1.07069 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-FC-CAL-G**">
<Pose>0 -1 0 0.39326 0 0 1 0.59626 -1 0 0 1.07069 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-FC-A**">
<Pose>0 -1 0 0.39326 0 0 1 0.44626 -1 0 0 1.08069 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-FC-G**">
<Pose>0 -1 0 0.39326 0 0 1 0.59626 -1 0 0 1.08069 0 0 0 1</Pose>
</Frame>
<Frame Name="**WAYPOINT-1**">
<Pose>0 -0.7071 -0.7071 0.0 0 -0.7071 0.7071 0.6 -1 0 0 1.3 0 0 0 1</Pose>
- <!-- <Pose>0 -0.94165 -0.3366 0.0 0 -0.3366 0.94165 0.6 -1 0 0 1.15 0 0 0 1 </Pose> -->
</Frame>
<Frame Name="**WAYPOINT-2**">
<Pose>0 -1 0 0.4 0 0 1 0.5 -1 0 0 1.3 0 0 0 1</Pose>
</Frame>
<Frame Name="**WAYPOINT-3**">
<Pose>0 -1 0 0.4 0 0 1 0.5 -1 0 0 1 0 0 0 1</Pose>
</Frame>
<Frame Name="**PRS-BASE**">
<Pose>-1 0 0 1.1 0 -1 0 0 0 0 1 0.820478 0 0 0 1</Pose>
</Frame>
<Frame Name="**SDS-BASE**">
<Pose>-1 0 0 -0.65 0 -1 0 0.5 0 0 1 1.61 0 0 0 1</Pose>
</Frame>
<Frame Name="**FC-BASE**">
<Pose>0 1 0 0.66847 -1 0 0 1.01723 0 0 1 0.18126 0 0 0 1</Pose>
</Frame>
<Frame Name="**SNS-BASE**">
<Pose>1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1</Pose>
</Frame>
<Frame Name="**TRS-BASE**">
<Pose>0 -1 0 0 1 0 0 -1 0 0 1 1.38 0 0 0 1</Pose>
</Frame>

<Frame Name="**SRS-LEFT-PSM-BASE**">
<Pose>-0.995168 0.0910389 -0.0367688 1.51548 -0.0925207 -0.994873 0.0408359 0.140142 -0.0328627 0.0440405 0.998489 1.02576 0 0 0 1</Pose>
</Frame>
<Frame Name="**SRS-RIGHT-PSM-BASE**">
<Pose>-0.995168 -0.0910389 -0.0367688 1.51548 0.0925207 -0.994873 -0.0408359 -0.140142 -0.0328627 -0.0440405 0.998489 1.02576 0 0 0 1</Pose>
</Frame>
<Frame Name="**SRS-ECM-BASE**">
<Pose>-0.707107 0 -0.707107 1.49392 0 -1 0 0 -0.707107 0 0.707107 1.70466 0 0 0 1</Pose>
</Frame>
<Frame Name="**SRS-ECM-RC**">
<Pose>-0.707107 0 -0.707107 1.49392 0 -1 0 0 -0.707107 0 0.707107 1.68466 0 0 0 1</Pose>
</Frame>
<Frame Name="**SRS-LEFT-PSM-RC**">
<Pose>0.970975 0.0832619 0.224222 1.02583 0.128161 -0.97263 -0.193817 0.101363 0.201947 0.216928 -0.955071 1.16194 0 0 0 1</Pose>
</Frame>
<Frame Name="**SRS-RIGHT-PSM-RC**">
<Pose>0.970975 -0.0832619 0.224222 1.02583 -0.128161 -0.97263 0.193817 -0.101363 0.201947 -0.216928 -0.955071 1.16194 0 0 0 1</Pose>
</Frame>
<Frame Name="**SURGICAL-SITE**">
<Pose>-0.995168 0.0910389 -0.0367688 1.51548 -0.0925207 -0.994873 0.0408359 0.140142 -0.0328627 0.0440405 0.998489 1.02576 0 0 0 1</Pose>
</Frame>
</FramesOfInterest>
</rrgOSCAR:Workcell>

Appendix E: OSCAR Specifications for Trauma Pod

February 1, 2005

I. Introduction

Operational **S**oftware **C**omponents for **A**dvanced **R**obotics (**OSCAR**) is an object-oriented framework for the development of control programs for robotic manipulators. OSCAR is developed in C++ and is independent of the application software architecture. A detailed overview and relevant publications can be found here. OSCAR has been used by developers at the Univ. of Texas, ORNL, NASA/Ames, and NASA/JSC. RRGKinematix, a single software library built upon OSCAR that performs generalized kinematics and collision detection for serial chain robots, has been accessed by over 500 users. OSCAR is compatible with Windows and Linux platforms and has been compiled using Visual Studio 6.0, .NET, and GNU C++ (a.k.a. g++) compilers. There are some platform dependent hardware interfacing components in OSCAR, primarily due to lack of platform independent device drivers.

The remainder of this document introduces OSCAR components relevant to the trauma pod Project; specifically the operation and control of the Surgical Nurse Subsystem (SNS). Although OSCAR is well documented on the web, this document directs the reader to material deemed most critical to this project. It contains:

- a general overview of OSCAR,
- a review of OSCAR domains relevant to this project including component functionality, interfaces, and usage,
- a list of resources for developers,
- a listing of components or functionality that must be added to OSCAR during this project, and
- appendices that contain
    - an example application for the SNS,
    - an example control block diagram illustrating a typical robot control system as developed using OSCAR components, and
    - a list of robot modeling parameters that OSCAR can use for control, modeling and optimizing the operation of the Mitsubishi PA10.

II. OSCAR Relevant Domains

A summary of OSCAR domains can be found here and the software implementation is documented here. Using the example control system block diagram in the appendix, relevant OSCAR modules are identified.

- **Support Domains**
    - OSCAR contains numerous support domains to handle mathematical operations (Math6), communications (Communications), File I/O (FileData), error checking (Base::OSCARError), etc.
- **Forward Kinematics**

---

6 The OSCAR Reference Pages are constantly updated. Therefore, pages for specific links or methods cannot be provided in this paper. Instead, all references are written to direct the reader from the main module page to the relevant library and class. For example, (Math) directs the reader to the Math module on this page and ForwardKinematics::FKPosition directs the reader to the class RRFKPosition in the ForwardKinematics module. OSCAR class names are preceded by RR to avoid namespace pollution. The 'RR' is removed from this documentation for readability.

    - ForwardKinematics::FKPosition – Generalized solution for the location of a tool tip, global and local joint transformations (Math::Xform using DH parameters (FileData::DHData or FileData::XMLDHData). (see Appendix E.2 for a sample file format).
    - Other generalized advanced forward kinematic functionality:
        - ForwardKinematics::FKJacobian, FKVelocity, and FKAcceleration
    - ForwardKinematics::FKPositionMitsubishi – Efficiently calculates the tool point for the Mitsubishi PA-10 manipulator. Allows both the base pose and the tool tip location relative to the last plane to be modified. This is a closed form solution.
    - ForwardKinematics::FKJacobianMitsubishi – Jacobian determination customized for the Mitsubishi PA-10 for computation performance. This is a closed form solution.
- **Inverse Kinematics**
    - OSCAR contains generalized InverseKinematics to determine Joint Positions (::IKPosition), and velocities (::IKVelocity) for a desired tool motion.
    - Closed form inverse kinematic solutions also exist for many robots including the Mitsubishi PA-10 (::IKMitsubishi).
- **Redundancy Resolution Techniques (RRT)**
    - In addition to the methods shown above, OSCAR can resolve redundancies and improve performance defined by a large and varied set of performance criteria.
    - InverseKinematics::IKJGenerateOptions – Generate set of kinematically valid joint configurations for desired EEF Position
    - PerformanceCriteria::PerformanceCriteria7 – Metrics used to compare the generated joint configurations. Some example criteria relevant to this effort are:
        - PerformanceCriteria::JointRangeAvailability – That monitors the joint state relative to joint limits. Limit criteria also exist for velocity, acceleration, and torque. Limit criteria take use exponential curve formulations and critical boundaries to avoid underutilization of the work space.
        - PerformanceCriteria::SingularityAvoidance – Avoid configurations where the Jacobian has no inverse.
        - PerformanceCriteria::ConservativeMotion – Avoid problems associated with joint drift in the workspace.
        - PerformanceCriteria::GeneralizedStiffness – minimize compliance at the EE.
        - PerformanceCriteria::Fusion – A weighted set of other criteria.
        - and Over 20 others implemented.
    - PerformanceCriteria::Repository – Store data generated by criteria in order to avoid repeating calculations. Implements a blackboard architecture for criteria computations.
- **Obstacle Avoidance/Collision Checking**
    - An ObstacleAvoidance::Obstacle (both manipulators and workspace) is modeled from primitives using formatted text files or XML File Data.
    - Collision Detection is performed by determining the Smallest Minimum Distance (ObstacleAvoidance::PCSmallestMinDist) in a between obstacles defined in a Robotic Work Cell (ObstacleAvoidance::Workcell) object.
    - Obstacle Avoidance is performed using the RRTs discussed earlier and Performance Criteria such as, PCSmallestMinDist, PCAverageDistReciprocal,

7 Criteria are traditionally categorized into performance criteria and obstacle avoidance criteria in OSCAR.

and PCDistToForce criteria that determine virtual forces between obstacles and move the robot away from such obstacles via self motion.

- **Path Planning**
  - MotionPlanning::MotionPlanning – Generating EE Paths between desired locations in either joint space or EE space. For example,
  - Path Blending (::PathBlend) – Generates constant velocity paths for a set of points.
  - 5^{th} order Polynomial Curves (::FifthOrderPoly) – for smooth trajectories from based on the initial and final conditions for position, velocity, and acceleration.
- **Hardware Interfaces**[8]
  - Device::RobotServoInterface – Abstract interface for sending appropriate OSCAR joint command (positions, velocities, currents or torques) to a robot's joint servomotors. Includes optional limit checking.
    - RobotServoInterface::RoboworksInterface – Example interface to graphical simulation.
  - Device::Tool – Interfacing for abstract grippers and tools..
  - Device::Sensor – Base class for all Sensors
    - Sensor::ATISensor – Interface to an ATI Force/Torque Sensor that uses the serial port.[9]

III. Developer Resources

- **RRG Points of Contact**

Dr. Mitch Pryor	mpryor@mail.utexas.edu	(512) 471-5182
Dr. Chetan Kapoor	chetan@mail.utexas.edu	(512) 471-7098
Mr. Ed Jung	ed.jung@mail.utexas.edu	(512) 471-6930
Dr. Chalongrath Pholsiri	longrath@mail.utexas.edu	(512) 471-6930

- **OSCAR Intranet Resources**
  - Intranet resources will require you to have an account on the OSCAR server. Developers can get an account by contacting Mitch Pryor.
  - Concurrent Version Systems (CVS) is used with OSCAR at the University of Texas. There are tutorials and implementation instructions for using CVS on the RRG Intranet.
  - An active Developer's Forum is also available on the RRG Intranet.
- **3^{rd} Party Libraries used by OSCAR**
  - These libraries are embedded in OSCAR and therefore their API is not a burden on application developers, but the libraries must be available for certain methods to be fully functional.
  - MATLAB C++ Libraries – Some higher level mathematical operations such as Singular Value Decomposition (SVD) employ MATLAB C++ Libraries. For this project, these operations will not be used and OSCAR math libraries are compiled in versions with and without these MATLAB based methods.
  - Xerces Libraries – These libraries are used for platform independent parsing, generating, and manipulating XML data. Classes handling data in OSCAR will use the open-source Xerces to guarantee platform independence.

---

8 The Device library in OSCAR is primarily an abstract set of classes with a few hardware interfaces currently implemented that are required used in the RRG laboratories.

9 This implementation of the ATI sensor is for an older serial bus sensor which will probably not contribute directly to the trauma pod effort. This class can be modified to use a data acquisition card for this project.

- ACE Libraries – These libraries are used to provide "developing high-performance, distributed real-time and embedded systems." It is open-source and platform independent and its primary use in OSCAR is for error logging and multithreaded support. If you are using single threaded OSCAR, ACE will not be required.

IV. Computational Performance Benchmarking

These control rates are based on OSCAR running on AMD Athlon-M 2400+ CPU with 512 MB of RAM running Windows XP. All programs were single threaded with no parallelism. Here are the speeds of calculations for some major OSCAR components. The robot geometry used was the Mitsubishi PA10.

- **Forward kinematics**
  - Position (Only EE location determined)
    - Closed-form ~ 350 KHz
    - Generalized ~ 95 KHz
  - Position and Jacobian (Both EE location and current Jacobian determined)
    - Closed-form ~ 140 KHz
    - Generalized ~ 70 KHz
- Inverse kinematics
  - Closed-form Position ~ 190 KHz
  - Resolved-rate with these parameter values
    - 1 mm increment steps at the EE
    - 0.01 mm error and 1000 rotation scale (default values)10
    - IKJacobian using FKJacobian ~ 37 KHz (generalized method)
    - IKJacobian using FKJacobianMitsubishi ~ 39 KHz11 (customized method)
- Distance calculations
  - Mitsubishi robot is modeled with 6 cylispheres (no self-collision).
    - Envi. with 6 cyl., 3 spheres, and 1 plane (10 total) ~ 5900 Hz
    - Envi. with 12 cyl., 6 spheres, and 2 plane (20 total) ~ 3400 Hz
    - Envi. with 5 cyl. ~ 12-13 KHz

Overall performance: Here is a typical scenario where the robot is instructed to follow a certain path and we use redundancy resolution with 3 criteria to solve the inverse kinematics problem.

- Kinematics component used
  - FKJacobian
  - IKJReconfig
  - IKJGenerateOptions
- OA and criteria
  - JRA - Avoid physical travel limits of the joints
  - Smallest Minimum Distance – makes sure there are no collisions.
  - Average Distance Reciprocal – does obstacle avoidance using redundancy.
- Models
  - PA10 - 6 cylispheres

---

10 OSCAR Inverse Kinematics uses an allowable tolerance error to determine when a successful iterative inverse solution has been found. The tolerable error is defined as a Cartesian error and scaling from the Cartesian error for the rotational values. In this case, the computational kinematics are determining solutions more accurate than most joint servo controllers can follow.

11 The geometry of the PA10 was analyzed and any mathematical short cuts to the generalized solution were implemented in a derived class.

    - Obstacles - 4 cylispheres, 3 spheres, and 1 plane
- Perturbation type: Simple (15 options)12
    - 270 Hz
    - 230 Hz including simple motion planning

From this preliminary analysis, it is clear that the operational components of OSCAR is expected to meet the necessary control rates anticipated in the trauma pod cell. These numbers are preliminary since the code was a 'bare bone' operational application that will need to be considered as a component in the hardware control system. The performance of these components will also increase as they are further customized for SNS use.

V. Future OSCAR Components

It is already recognized that additional functionality will need to be added to OSCAR in order to meet the requirements for the Phase I demonstrations. This listing is for the operational components of OSCAR and does not consider hardware implementations or code associated with other software components such as the Supervisory Controller.

- A global path planner
- Real-time trajectory generation
- Multi-thread safe libraries
    - Integration with the ACE Framework.
- Polyhedral Obstacle Modeling

12 RRIKGenerateOptions Method generated 15 discreet solutions for the EE location that could be compared in each cycle in terms of the selected criteria.

E.I. Sample Programs

An Online OSCAR Tutorial walks new OSCAR developers through its foundation classes (i.e. Math, FileData, etc.), Kinematics, Motion Planning, Dynamics, and Redundancy Resolution.

The following example program demonstrates the kinematic operation of the Mitsubishi PA10 using the closed-form inverse kinematic object.

```
////////////////////////////////////////////////////////////////
// File Name: main.cpp
//
// This program was used compare cycle rates of the
// closed form inverse kinematics of the Mitsubishi
// PA-10.
////////////////////////////////////////////////////////////////
#include "ForwardKinematics\FKPositionMitsubishiPA-10.h"
#include "ForwardKinematics\FKJacobian.h"
#include "InverseKinematics\IKMitsubishiPA-10.h"
#include "InverseKinematics\IKJacobian.h"
#include "FileData\ManipulatorData.h"
#include "Base\Timer.h"

using namespace OSCAR::FileData;
using namespace std;
int main()
{
  // Create an OSCAR error object for error checking in program
  RROSCARError  err(noError);

  // Collect the XML schemas and robot parameter data
  err = XMLData::SetSchemaFile(RRString("\\\\server\\... \\Schemas\\ModelTypes.xsd"));
  if(err != noError){
    DisplayError(err);
    return 0;
  }
  ManipulatorData pa10Data(RRString("MitsubishiPA7C-ManipulatorModel.xml"), err);
  if(err != noError){
    DisplayError(err);
    return 0;
  }
  if(!pa10Data.LoadData()){
    DisplayError(pa10Data.GetError());
    return 0;
  }
  const XMLDHData*  pa10DH = pa10Data.GetDHParameters();

  // Create joint vectors of the correct length.
  RRJointVector initialJoints(pa10DH->GetDOF()), invSolution(pa10DH->GetDOF());
  initialJoints[0] = 0.0;  initialJoints[1] = 45.0;  initialJoints[2] = 0.0;
  initialJoints[3] = 45.0; initialJoints[4] = 0.0;   initialJoints[5] = 45.0;
  initialJoints[6] = 0.0;
  initialJoints *= DegToRad; //All OSCAR calculations use radians.

  // Create the forward kinematics object using the XML Data
```

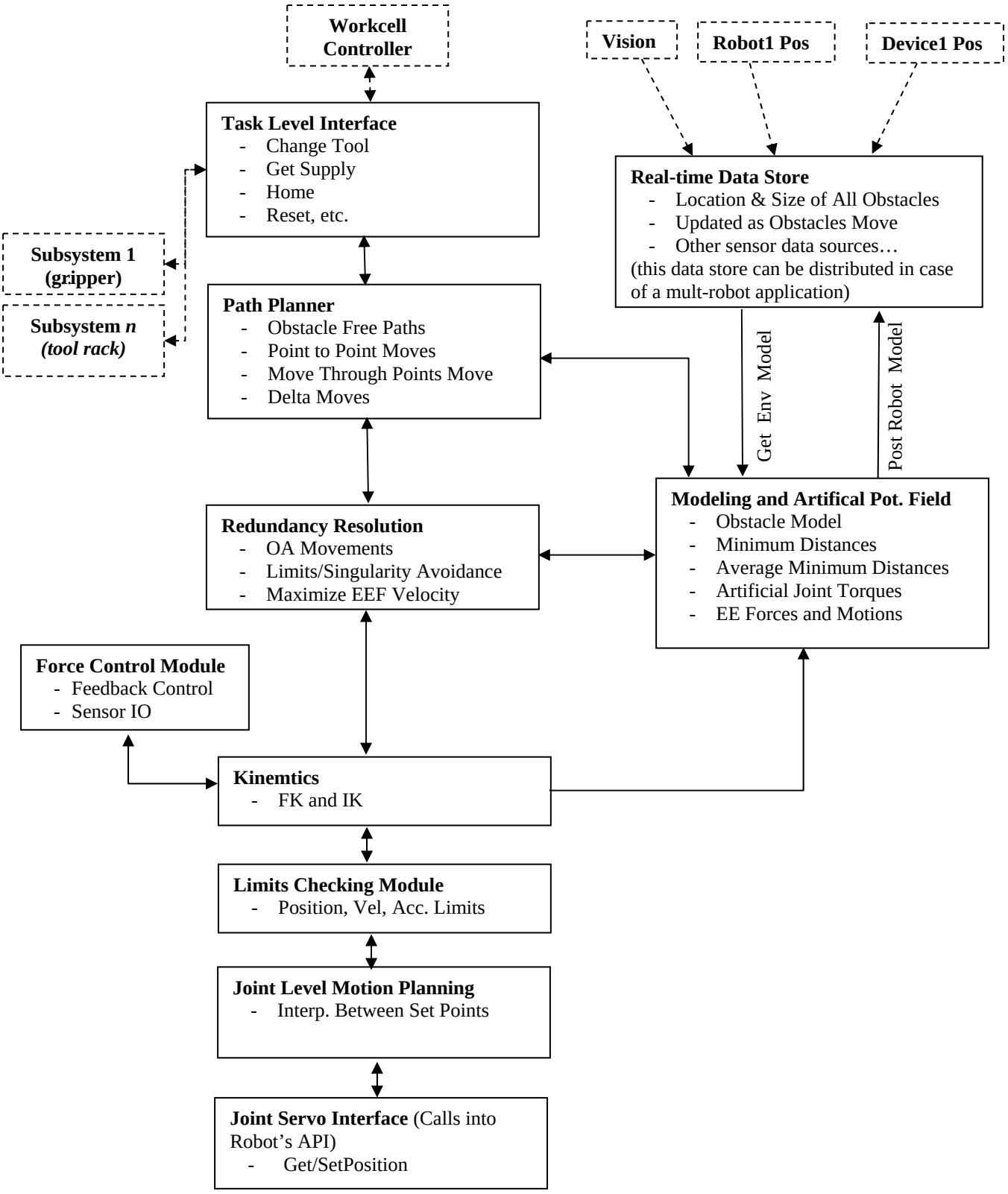
```
  RRFKPositionMitsubishi fk(*pa10DH, err);
  if(err != noError){
    DisplayError(err);
    return 0;
  }

  // Determine the initial tool location
  RRXform hand = *fk.GetHandPose(initialJoints);

  RRIKMitsubishi ik(*pa10DH, err);
  if(err != noError){
    DisplayError(err);
    return 0;
  }

  bool  retVal;  // used to determine the success of the IK Object
  RRTimer clock;
  unsigned int cycles = 1500; //# of steps in the path
  unsigned int loops = 10; //# of times path is repeated

  // Begin moving tool tip along a simple path.
  clock.Start();
  for(unsigned int j = 0; j < loops; j++){
    for(unsigned int i = 0; i < cycles; i++){
      retVal = ik.GetJointPosition(hand, invSolution);
      if(!retVal)
        DisplayError(ik.GetError());

      hand.at(0,3) -= 1.0;  // decrement X by 1 mm
    }
    for(i = 0; i < cycles; i++){
      retVal = ik.GetJointPosition(hand, invSolution);
      if(!retVal)
        DisplayError(ik.GetError());

      hand.at(0,3) += 1.0;  // increment X by 1 mm
    }
  } //end of for(unsigned int j = 0; j < loops; j++)
  clock.Stop();

  cout << "IK Frequency = ";
  cout << (2*loops*cycles)/clock.ElapsedTime() << endl;
  return 0;
}
```

E.II. Manipulator Data for PA-10. Sample XML File

Input to OSCAR components is primarily through formatted text data and lately through XML. XML is used to specify manipulator geometry (see below), obstacle models, tool models, workcell configurations (location of key devices), etc. The XML format is based on well defined schemas (xsd) that is used to check the correctness of the data provided. Example below shows the PA-10 kinematic model.

```
<rrgOSCAR:ManipulatorModel xmlns:rrgOSCAR="http://www.robotics.utexas.edu"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <Name>Mitsubishi Pa-10 C</Name>
        <DHParameters>
                <FrameLocationMethod>Paul</FrameLocationMethod>
                <Row>
                        <alpha>0</alpha>
                        <a>0</a>
                        <d>0</d>
                        <MinLimit>-177</MinLimit>
                        <MaxLimit>177</MaxLimit>
                </Row>
                <Row>
                        <alpha>-90</alpha>
                        <a>0</a>
                        <d>0</d>
                        <MinLimit>-94</MinLimit>
                        <MaxLimit>94</MaxLimit>
                </Row>
                <Row>
                        <alpha>90</alpha>
                        <a>0</a>
                        <d>450</d>
                        <MinLimit>-174</MinLimit>
                        <MaxLimit>174</MaxLimit>
                </Row>
                <Row>
                        <alpha>-90</alpha>
                        <a>0</a>
                        <d>0</d>
                        <MinLimit>-137</MinLimit>
                        <MaxLimit>137</MaxLimit>
                </Row>
                <Row>
                        <alpha>90</alpha>
                        <a>0</a>
                        <d>480</d>
                        <MinLimit>-255</MinLimit>
                        <MaxLimit>255</MaxLimit>
                        </Row>
                <Row>
                        <alpha>-90</alpha>
                        <a>0</a>
                        <d>0</d>
                        <MinLimit>-165</MinLimit>
                        <MaxLimit>165</MaxLimit>
                </Row>
                <Row>
                        <alpha>90</alpha>
                        <a>0</a>
                        <d>0</d>
                        <MinLimit>-255</MinLimit>
                        <MaxLimit>255</MaxLimit>
                </Row>
        </DHParameters>
        <BasePose>1 0 0 -5 0 1 0 12 0 0 1 10 0 0 0 1</BasePose>
        <ToolPlatePose>1 0 0 0 0 1 0 0 0 0 1 5 0 0 0 1</ToolPlatePose>
</rrgOSCAR:ManipulatorModel>
```

E.III. Example Control System Block Diagram



- Blocks with Solid lines reside on the robot controller. Only exception being the Real-Time Data Store that may reside on the network for a truly scalable application where multiple robots need to perform OA.
- Force Control Module also reads a force/torque sensor (not shown) and may perform averaging of force data to reduce noise. It can also be run asynchronously from the main control loop.

E.IV. Manipulator Parameters Required/Desired by OSCAR

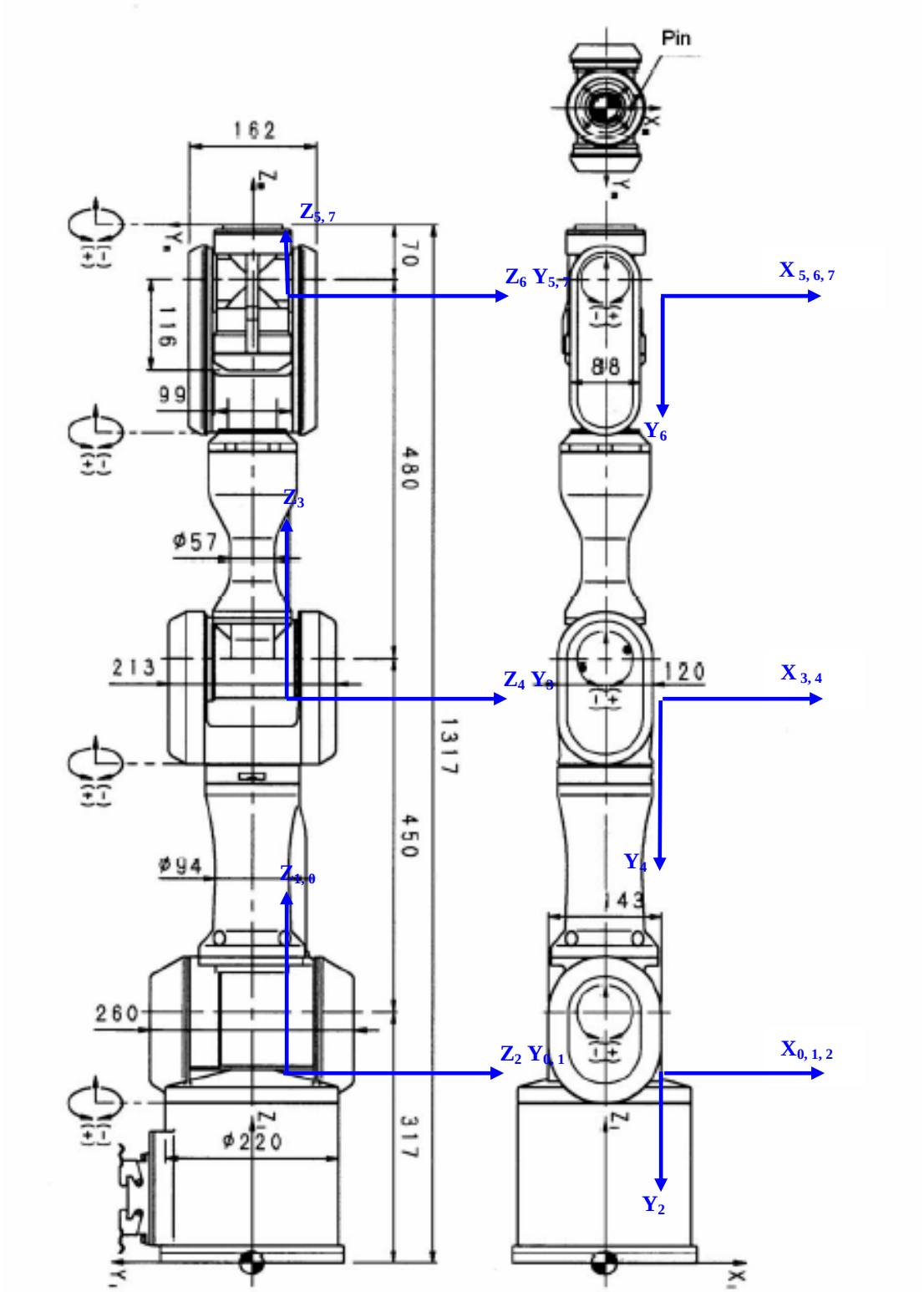
The table below shows the parameters that are required/desired and the impact capability that can be provided in the software based on their availability. Accurate knowledge of some these parameters will be critical to the success of the Phase I demonstrations. The desired parameters (based on their accuracy) can be used to further optimize the manipulator motion w.r.t. to its torque limits and accuracy.

Parameters	Controller Capability
Kinematic Parameters (required) **Dimensions** **Axis locations/directions** **Zero Position for axes** **Position, Velocity, Acceleration** **Travel Limits** **Joint Position Resolution/Accuracy**	- **Forward and Inverse position and velocity capability** - **Avoiding limits, maximizing speed** - **Redundancy resolution** - **Collision detection, obstacle avoidance**
Dynamic Parameters (desired) **Part Inertias** **Center of Mass Locations for each part** **Mass of each part** **Torque limits** **Joint Stiffness**	- **Optimizing motions w.r.t. torques** - **Minimizing inertia loads, leading to lower vibrations and shocks** - **Improved accuracy due to deflection modeling**
Friction Properties (desired)	- **Optimizing motions w.r.t. torques as friction can account for 15% of torques**

Details of these and specifics on how these parameters should be specified are presented on the following pages.

Kinematic Properties

Based on available drawings (see below), we have already figured out the dimensions and DH parameters. The DH frame assignment has also been done (see below):



Frames for setting DH Parameters for the PA-10 Robot

Link	Alpha(i-1)	A(i-1)	d(i-1)	Theta(i-1)
1	0	0	0	Var
2	-90	0	0	Var
3	90	0	450	Var
4	-90	0	0	Var
5	90	0	480	Var
6	-90	0	0	Var
7	90	0	0	Var

Dimensions are in **mm** and angles in **degrees**

BasePose w.r.t. to the mounting plate of the robot is at 317 mm. The tool plate pose w.r.t the last DH frame (at the wrist) is 70 mm.

Position Travel Limits (degrees) - these are software limits as specified by ORNL

```
Joint 1 = +,- 177
Joint 2 = +,- 94
Joint 3 = +,- 174
Joint 4 = +,- 137
Joint 5 = +,- 255
Joint 6 = +,- 165
Joint 7 = +,- 255
```

Joint Position Resolution and Accuracy (degrees)

Velocity Limits

```
Joint 1 = 1.4 rad/sec
Joint 2 = 1.4 rad/sec
Joint 3 = 2.25 rad/sec
Joint 4 = 2.25 rad/sec
Joint 5 = 6.28 rad/sec
Joint 6 = 6.28 rad/sec
Joint 7 = 6.28 rad/sec
```

Acceleration Limits

```
Joint 1 = 7 rad/sec-sec
Joint 2 = 5 rad/sec-sec
Joint 3 = 10 rad/sec-sec
Joint 4 = 7.5 rad/sec-sec
Joint 5 = 68 rad/sec-sec
Joint 6 = 36 rad/sec-sec
Joint 7 = 40 rad/sec-sec
```

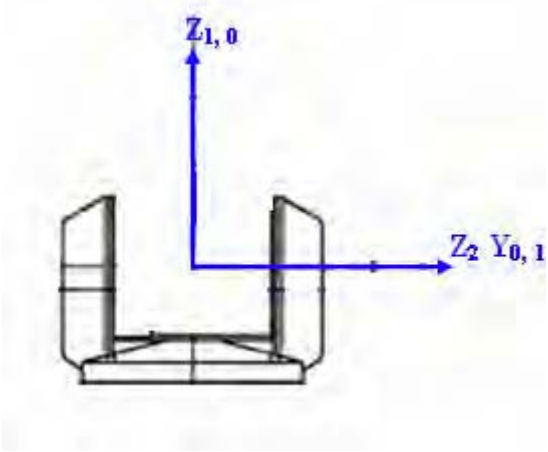
Operational Offsets

- Often the operational zero location for each joint is different that dictated by the DH Parameters. The can be caused by either the arbitrary orientation of some DH frames or the zero position from the encoder was selected to put the robot into a more compact or safer configuration. If the zero position of the manipulator is offset from the zero position derived from the DH parameters, this information will also be needed.

Dynamic Parameters

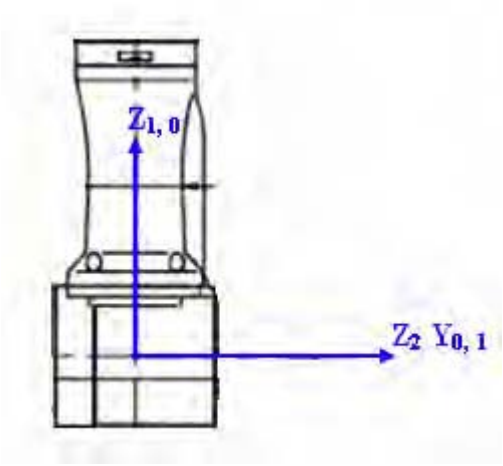
We have broken the PA-10 into various parts and would like the mass of each part, the location of the C.G. of each part w.r.t. the specified frame, and the inertia of each part about the C.G. for that part. The C.G. location can be

expressed as vector pointing from the origin of the specified frame to the C.G. Please specify in S.I. units. Below, ***I*** represents the inertia matrix and ***r*** represents the vector from the specified frame to the C.G.



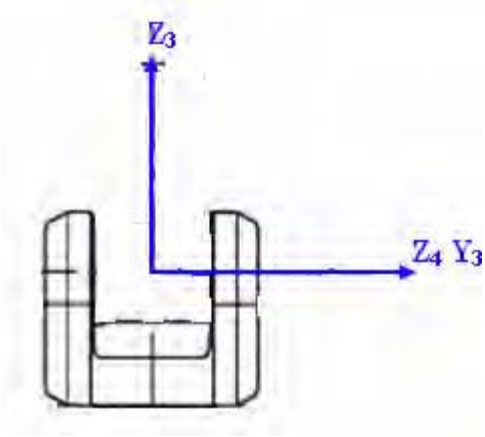
$$\mathbf{I_1} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_1 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

The part above in frame 1



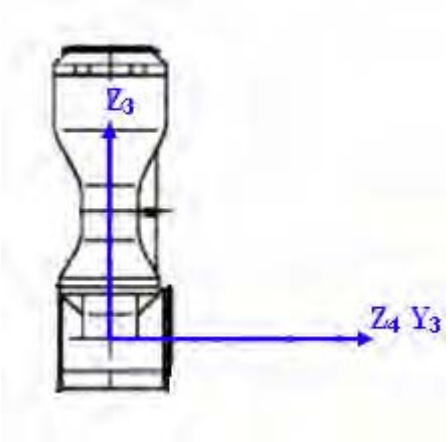
$$\mathbf{I}_2 = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_2 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

The part above in frame 2



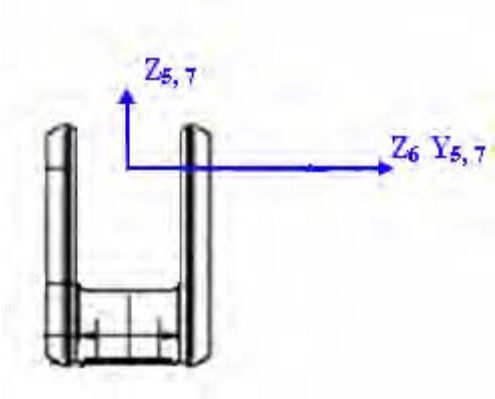
The part above in frame 3

$$\mathbf{I}_3 = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_3 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$



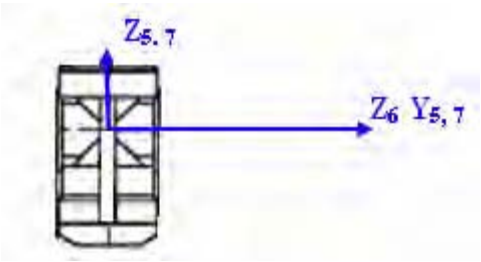
The part above in frame 4

$$\mathbf{I}_4 = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_4 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$



The part above in frame 5

$$\mathbf{I}_5 = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_5 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$



The part above in frame 6

$$\mathbf{I}_6 = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_6 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

The part above in frame 7

$$\mathbf{I}_7 = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{xy} & I_{yy} & I_{xx} \\ I_{xz} & I_{xz} & I_{zz} \end{bmatrix} \vec{\mathbf{r}}_7 = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix}$$

Friction Model:

It would also be nice to have a friction model of the robot if possible, as friction can account for up to 15% of the torques. So, for fully optimized motions, this may be necessary. A method we are aware of for getting the friction model uses least squares estimation techniques.

A commonly used model when using least squares estimation is:

$$f_j = f_{cj} sign(\theta_j) + f_{vj}\dot{\theta}_j$$

Where f_j is the total friction torque for joint j, f_{cj} is the Coulomb friction constant for joint j and f_j is the viscous friction constant for joint j.

Appendix F: Task Planning Techniques for Trauma Pod

Author: Shilpa Gulati

INTRODUCTION

Robot assisted surgery has helped reduce patient trauma and recovery time due to increased precision of surgical robot systems compared to humans [1, 2]. In general, a robot assisted surgery cell consists of a tele-operated surgical robot that performs the surgery. The support functions like handing over the surgical instruments to the surgical robot, monitoring the patient etc. are carried out by trained humans such as nurses. Attempts are being made to automate the support functions in robot assisted surgery in view of the demonstrated effectiveness of robotic systems and their ability to extend the surgeon's capabilities [1, 3]. Various factors need to be considered for full automation of the support functions. An important consideration is *Task Planning*, that is, the ability to convert the surgeon's high level commands into appropriate actions of the support devices [4, 5].

Many techniques have been proposed for automated task planning. An important subset of these techniques is classical task planning. Literature survey and analysis shows that classical task planning techniques are not adequate for automated task planning of fully automated support functions in a tele-robotic surgery cell.

BACKGROUND

Definitions of key terms

World: World is the system of interest [6].
Examples: Chess board during a chess game, automated surgery cell.

State: A state is a condition of the world where certain properties hold true. These properties are uniquely determined at any given instant of time [7]. Thus, state is a snapshot of the world at a given time.
Examples: The location of all pieces on a chess board at a give instant represents its state completely at that instant. For an automated surgery cell the properties of its inhabitants (humans and mechanical systems) represent its state.

Action: An action is an intentional event that changes the state of the system [7]. An action is performed by the inhabitants of the world and transforms one state into another.

- **Primitive Action**: A primitive action is one that cannot be further broken down into other actions [6]. In theory, we can break down an action into smaller actions endlessly. In practice, we *define* a certain action as primitive when that action can be completely performed by an inhabitant at a single command.
  Examples: Moving a pawn from one square to another in a game of chess is a primitive action. For an automated surgery cell, we can define a primitive action as the movement of surgical robot from one orientation to the other.

- **Compound Action:** A compound action is one that can be further decomposed into other actions [6]. An inhabitant cannot perform a compound action directly. This action must be broken down into primitive actions before it can be performed. When a compound action is decomposed it results in a sequence of primitive and/or compound actions [6] called *decomposition sequence* or simply *decomposition*. There might be more than one decomposition sequence for a compound action because there may be more than one way of accomplishing the same task.
  Example: (a) Surrounding the queen in a game of chess is a compound action. It may involve moving multiple pieces in a particular sequence. There may be many sequences that will attain the same result. (b) For an automated surgery cell, a compound action can consist of a sequence: (1) Nurse: pick up scissors (2) Surgical robot: lock end-effector such that force in direction *s* produces only a small deflection (3) Nurse: Slide scissors in surgical robot's end-effector in direction *s*.

Deterministic vs. Non-deterministic world: A deterministic world is one in which the next state is completely determined by the present state and the action taken. A non-deterministic world is one in which it is not possible to know the next state exactly even if the current state and the action taken are known [6].

Example: (a) A chess board is a deterministic world. Given current location of all pieces, and knowing that a pawn (say) is moved, the resulting state is completely known. (b) Automated surgery cell containing one surgical robot is a non-deterministic world because of the inaccuracy inherent in the robot. For example, if the robot is moved from point *A* to point *B*, it does not end up exactly at point *B*. Thus, even though we knew the initial state (robot at point *A*) and the action we commanded (Move robot from *A* to *B*) we cannot predict the exact state after the action (robot is near *B* with some error *e*).

Task Planner

A task planner takes a high level task description as input and produces a sequence of lower level of actions or a *plan* as output [6]. The schematic of a task planner is shown in Figure 1.

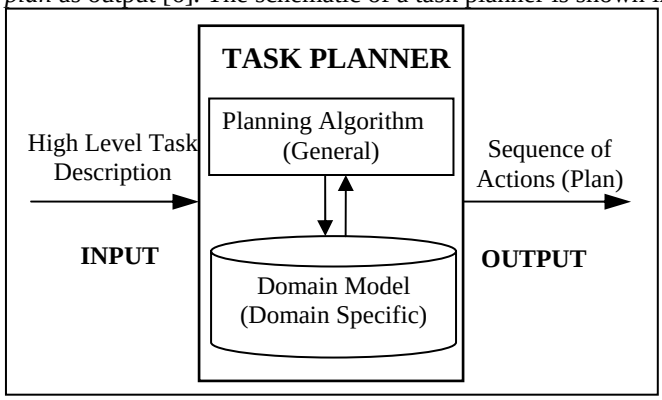


Figure **1**: Schematic of a Task Planner.

A task planner consists of:

1. **Planning Algorithm:** A planning algorithm is a general reasoning procedure that is designed to work for any environment that satisfies certain properties.
2. **Domain Model:** This contains domain specific knowledge that is pre-stored by the designer. It maybe possible to augment the domain model during plan generation.

A general planning algorithm can use the domain model to generate plans particular to that domain. There could be multiple planning algorithms that work for a given environment. The domain model could also be in various forms. For example, in an automated surgery cell, the domain model could consist of all possible features and actions of the devices of the surgery cell. Then a planning algorithm could combine suitable actions in a logical manner to form a plan.

Classical Task Planning

A planning algorithm is designed based on certain assumptions about the kind of world it can work in. *Classical Task Planning refers to the subset of planning algorithms that work in a deterministic world* [6].

Classical Task Planning algorithms are further divided into two categories depending on the kind of domain model they use [8] and are summarized in Table 1.

Planning Algorithm	Input to the algorithm	Domain Model	Adding new knowledge to domain model
Primitive Action (PA)	**Initial and Final State**	**All possible primitive actions of the devices.**	**Add new primitive actions**
Knowledge Based (KB)	**Initial State and Goal State**	**1. All possible primitive actions of the devices.** **2. A repository of compound actions and their decompositions.** **3. A list of criteria for selecting a particular decomposition for a compound action.** **4. Other rules such as time and resource constraints.**	**1. Add new primitive actions.** **2. Add new compound actions** **3. Add new criteria for selecting decompositions of compound actions.** **4. Add new rules.**

Table 1: Summary of the PA and KB planning techniques

Both PA and KB algorithms take an initial state and a final state as input, and then use the knowledge stored in the domain model to find a sequence of actions or "path" that leads from the initial state to the goal state [6]. An example input is shown in Figure 2.

Task: ***Insert scissors in surgical robot's end-effector***
***Initial State*: Surgical robot's end-effector is empty AND Nurse has scissors**
***Goal State*: Surgeon robot's end-effector has scissors AND Nurse does not have scissors**
Figure **2:** A sample input to the planning algorithms

As seen from Table 1, *the essential difference between PA and KB techniques is the kind of knowledge in their domain model*. While PA algorithms use a domain model consisting only of primitive actions, KB algorithms use a domain model consisting of primitive and compound actions, criteria for selecting the decompositions of compound actions and other rules [8].

THE REPRESENTATIVE SURGERY CELL

The representative surgery cell is chosen to perform minimally invasive surgery since it is one of the most widely performed types of robot-assisted surgery [1, 2, 5].The surgery is performed by a human from a remote location with a tele-operated robot. The support functions are fully automated and no human is present at the surgery site. The cell consists of various subsystems that are controlled either directly by the surgeon or by a central computer called *supervisor*. The commands issued by the surgeon are processed by a *planner* that breaks the commands into subsystem level actions. The supervisor then commands the subsystems to execute those actions.

The subsystems in the cell are (Fig 3):
Surgical Robot: The surgeon robot consists of three arms with one end-effector each. Surgical tools are mounted on two of the arms. The third arm carries a camera.
Nurse Robot: This is a 7 DOF robot with one gripper and performs the function of handing over surgical tools and supplies to the surgical robot.
Tool Rack: The tool rack holds the sterile tools (like scissors, forceps etc.) needed for surgery. Its function is to present the appropriate surgical tool to the nurse robot.
Supply Dispenser: The supply dispenser places the appropriate surgical supplies (like swabs) in a supply tray and presents the tray to the nurse robot.

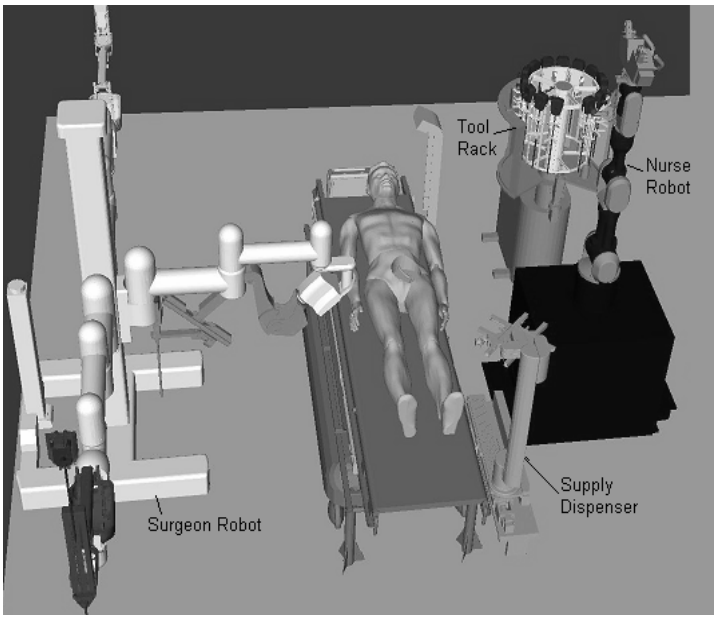


Figure 3: The representative surgery cell.

A small subset of the domain model of the representative surgery cell for PA and KB planning techniques is shown in Table 2. In particular, note the compound action "Place new tool in surgical robot's end-effector" in KB domain model. There are two ways of achieving the same result. In the first, the nurse robot moves through an intermediate point in such a way that the Torque Limit Avoidance criterion, $TLA \rightarrow 0$. This means that torque in every joint is much less than the maximum allowable actuator torque. Details of this criterion13 can be found in [9]. In the second, the nurse robot moves directly to the surgical robot's end-effector and at least one joint torque approaches its maximum allowable torque, that is, $TLA \rightarrow 1$. The first sequence is safer (because we are operating

13 Many other criteria could be used to extend the KB domain model. For example, we could use system level criteria [9] such as Velocity Limit Avoidance, Joint Range Availability, Force Transmission Ratio, Effective EEF stiffness; obstacle avoidance criteria such as Smallest Minimum Distance and Average Minimum Reciprocal Distance; actuator level criteria relating to torque, speed, precision, noise etc. (under development at Robotics Research Group) etc. to enhance KB planners.

all actuators within safe torque limits), but the second is faster (because it does not have an intermediate point). The planning algorithm or the surgeon has the *choice* of selecting one of these depending on the requirements of the surgery.

Planning technique	Domain Model
PA planning	**Primitive actions** **1 Nurse Robot: Move from point *X* to point *Y*.** **2 Nurse Robot: Orient and lock tool in surgical robot's end-effector.** **3. Surgical Robot: Make an incision in patient at point (*x, y, z*)**
KB planning	**Primitive actions** **Same as PA planning.** **Compound actions and criteria** Place new tool in surgical robot's end-effector ***Decomposition Sequence 1: TLA→ 0. Safer but slower of the two.*** **1. Nurse Robot: Move from current position to intermediate point *A*.** **2. Nurse Robot: Move from point *A* to surgical robot's end-effector.** **3. Nurse Robot: Orient and lock tool in surgical robot's end-effector.** ***Decomposition Sequence 2: TLA→ 1. Faster of the two but less safe.*** **1. Nurse Robot: Move from current position to surgical robot's end-effector.** **2. Nurse Robot: Orient and lock tool in surgical robot's end-effector.** **Additional rules or heuristics** **1. Do not add any action to a plan that takes more than 20 seconds to execute.** **2. Do not add any action to a plan that results in collisions.** **3. If the patient's weight is less than 250 lbs the incision point should be at location (*x, y, z*) on the body otherwise it should be at location (*u, v, w*).**

TABLE 2: PARTIAL PA AND KB DOMAIN MODEL OF THE REPRESENTATIVE SURGERY CELL.

ANALYSIS

In this section, we analyze the requirements of a task planner for fully automated support functions and then evaluate the two classical planning techniques against the requirements.

Requirements of a Task Planner for Fully-Automated Support Functions

A task planner for fully-automated support functions in a surgery cell must meet two kinds of requirements: (1) those related to the plan generation process (how it is generated, how fast it is generated, etc.) and (2) those related to the kind of plan generated (does it have concurrent actions, is it optimized etc.). This analysis focuses on the first kind of requirements.

1. **Meaningful interaction with surgeon:** Surgery is a complex procedure, and the surgeon often makes decisions and takes actions depending on the situation [5]. For example, the location of incision for inserting the tools in the body cavity may depend on the patient's weight and body shape [3]. The surgeon may want the incision to be in a different location than that specified in rule 3 in Table 2 after observing the patient or may want to add a new rule based on his experience. Thus, a planner must be able to accept new knowledge and guidance from the surgeon to make plans.
2. **Execution monitoring and continuous plan modification**: Safety is one of the most important considerations in a surgery cell [5]. To ensure safety, the actions of the subsystems must be monitored by sensing the environment and checking the actual state against the expected state [3]. If the actual state of the environment is different from the state expected according to the plan, the task may have to be paused or cancelled or modified. Thus, a planner must be able to monitor execution and modify plans continuously [8, 10].
3. **Anticipatory planning:** In a non-robotic surgery cell, the human nurse is able to anticipate the next step of surgery and obtain a tool even before the surgeon asks for it, resulting in speeding up of the surgery process [3]. An ideal planner must do the same and generate plans for tasks even before the surgeon's requests.
4. **Meet time constraints:** Various contingencies like excessive bleeding, irregular heartbeat of the patient etc. might arise during surgery and it is possible that a plan might not exist for handling all such situations. Hence, in addition to the plans that have been generated offline and stored, new plans may have to be generated on-site.

The time taken for on-site plan generation must be of the order of a few seconds [3]. Slow plan generation may result in loss of concentration and frustration in the surgeon [3] and serious consequences for the patient.

Evaluation of classical planning techniques against the requirements

In this section, we evaluate the two classical planning techniques against the requirements. To see if the techniques meet the time requirements, we will look at one representative algorithm from each technique.

Evaluation of PA planning techniques against the requirements

Meaningful interaction with Surgeon

The surgeon can interact with a PA planner in only one way, that is, by specifying new tasks in the form of initial state and goal state.

However, much of the knowledge of the surgeon is in terms of various rules or heuristics [1, 2, 3] such as those shown in Table 2. The domain model for primitive action planners does not contain any knowledge of this kind and so cannot benefit from the surgeon's input.

Thus, PA planners cannot interact with the surgeon in a meaningful way because they contain only primitive actions and do not allow knowledge in the form of rules and criteria.

Execution monitoring and continuous plan modification

PA planning techniques assume that the world is deterministic. Therefore, these algorithms first generate a plan and then the supervisor blindly executes it assuming all goes well [6, 8]. In reality, the surgery cell domain (like most other domains with mechanical systems) is not completely deterministic because of inaccuracy in mechanical devices and unexpected events like jamming and malfunctioning of devices.

For example, in the compound action **"**Place new tool in surgical robot's end-effector**"** in Table 2, the nurse robot is expected to lock the tool in the surgical robot's end-effector in a particular orientation. It is possible that due to slight orientation or positioning errors of the robot's end-effectors and/or arms the tool may break or may be inserted in a slightly different orientation than what is expected by the plan. An ideal planner should be able to get this information from the environment and modify the plan to take this into account.

Execution monitoring and continuous plan modification is not supported by PA planners because they assume a deterministic world and fully automated surgery cell is not a deterministic world.

Anticipatory planning

A human nurse learns from experience and is able to anticipate the surgeon's demands. For example, when the surgeon is performing incisions in the patient's body the nurse observes him carefully and gets ready with the scalpel. As soon as the surgeon puts down the scissors, the nurse is ready to hand over the scalpel to the surgeon [3].

Thus, observation and learning are two key features needed for anticipatory planning. We saw in the previous analysis that PA planners do not monitor or observe the environment.

Thus, anticipatory planning cannot be done by PA planners because they do not observe and learn.

Meet time requirements

We look at the time requirements for a representative PA algorithm. First we define two terms:

Branching factor: Only some actions can be applied to transform a state into some other state. a

For example, we cannot apply an action "Nurse, release scissors from gripper" when the gripper is empty. The average number of actions applicable per state, *b*, is called *branching factor* [6].

Minimum number of steps in the solution: There may be more than one path from the initial state to the goal state. The number of steps in the shortest path [6] is called *d.*

A simplistic algorithm for finding a solution works like this (Fig 4): Apply all applicable actions to the initial state resulting in *b* states. Then, apply all possible actions to each of the resulting states. This would give us *b* x *b* states at the next level. This process is continued till the goal state is reached [6].

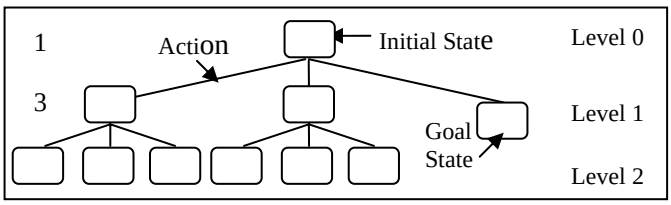


Figure **4**: A simplistic planning tree for $b = 3$ and $d = 2$

GRAPHPLAN: This algorithm works in a clever way so that it produces only one state at each level (Fig. 5). The state at level 0 is represented by all the conditions that are true in the initial state. The state at level 1 is obtained by including the initial state and effect of all actions that are applicable in state at level 0 [11, 12]. *Thus, the application of b actions does not result in b states but only one state at the next level.* This is shown in Figure 5. This state may

have inconsistent conditions because two applicable actions may have effects that are negative of each other. The algorithm keeps track of this by storing additional data about inconsistent conditions [11, 12].

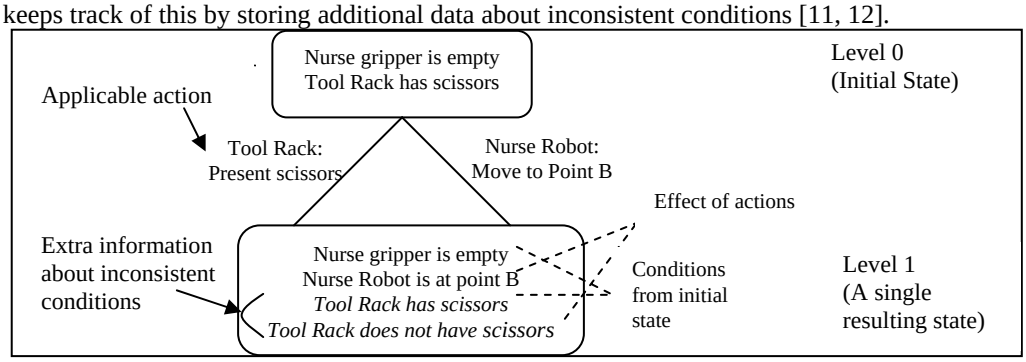


Figure **5**: GRAPHPLAN produces only one state after applying actions

GRAPHPLAN was shown [11] to have a time complexity that is a low order polynomial in the number of actions at a given level. In a simple analysis [6] we can represent this as $O(p^q)$ where p is the maximum number of actions at any level and q is a small number. q is the maximum number of conditions that are needed to describe a particular aspect of the state (for example, "Nurse Robot is at point A" has only one condition; "Tool Rack has scissors AND scissors are sterile" has 2 conditions).

Let us now get a rough estimate of the time taken by the algorithm (the exact analysis is beyond the scope of this paper, and can be found in [11]). Assuming q = 4 and the number of action nodes that can be generated [6] per second $=10^6$, a rough estimate of the time taken by the algorithm is shown in Table 3.

p	**Maximum number of actions at any level**	**Estimated time taken**
10	10^4	**0.01 sec**
50	50^4	**6.25 sec**
100	100^4	**100 sec**

Table **3**: Rough estimate of time taken by GRAPHPLAN for q = 4

Now, the value of *p* depends on the average number applicable actions, which in turn depends on (1) the number of subsystems in the cell and (2) the number of actions per subsystem. The number of subsystems in the representative surgery cell is 4. The number of actions per subsystem is limited because each subsystem is expected to perform a certain well-defined finite set of functions. Thus, *p* may have a reasonable value (about 50) if the number of subsystems is small and each subsystem performs a small number of actions (about 100). *To summarize, GRAPHPLAN has polynomial time complexity and it may satisfy the time requirements if the problem size p is not very large.*

Evaluation of KB techniques against the requirements

Meaningful interaction with Surgeon

The surgeon can interact with a KB planner in three ways:

1. **New input**: Like PA planners, the surgeon can ask the KB planner to make new plans by specifying a new initial state and final state.
2. **Augmenting the domain model**: The surgeon can easily understand and augment the domain model of KB planners since it is in the form of criteria and rules. For example, in Table 2 the surgeon could add a new rule: "The distance between the nurse robot and the patient should be greater than 10 cm while orienting and locking a tool in the surgical robot's end-effector". This ensures that the nurse robot never accidentally hurts the patient during the compound action "Place new tool in surgical robot's end-effector".
3. **Advising the planner during plan generation phase**: The surgeon can provide "advice" in selecting the decomposition sequence for a particular action [13]. For example, in Table 4, the surgeon could advise the planner to choose Sequence 2 for the compound action "Insert tool in surgical robot's end-effector" if the surgeon wants to speed up execution of this action.

Thus, KB planners are able to interact with the surgeon in a meaningful manner because they use a domain model in the form of criteria and rules that a human user can easily understand, augment and modify.

Execution monitoring and continuous plan modification

A similar argument as the one provided for PA planners holds for KB planners because KB planners also assume a deterministic world.

Anticipatory planning

A similar argument as the one provided for PA planners holds for KB planners because KB planners also do not observe the world and learn from it.

Meet time requirements

We look at the time and space requirements for a representative KB algorithm.

HTN (Hierarchical Task Network) planning: This algorithm [6, 14] starts with an initial, high-level task specification. It then finds a suitable decomposition sequence for this task. The decomposition results in a plan with primitive and/or compound actions. Each compound action in the resulting plan is then decomposed. This process is continued till the plan consists only of primitive actions [14] as shown in Figure 6.

At every step, the algorithm needs to (1) Either find a decomposition sequence for each action or (2) Make such a sequence for that action if it does not exist. *The problem is thus divided into a number of smaller problems each of which can be solved in much lesser time than the original problem and then combined.*

To see this, assume that the input problem consists of a single high level goal.

Number of primitive actions (that is, the number of steps) in the final plan = *d*.

Number of actions at the next level into which each compound action decomposes = *k*

Now, start from the high level goal at level 0 and pick one decomposition sequence (say sequence 1 in Figure 6) resulting in *k* actions at the next level. For each of these *k* actions, pick one decomposition sequence and continue till a sequence of only primitive actions is obtained. A single tree is formed in the process as shown in Figure 6.

If the number of levels below level 0 is *h*, we have,

$k^h = d \Rightarrow h = \log_k d$.

Number of actions per tree

$= 1 + k + k^2 + .. + k^h$

$= (k^h - 1)/(k - 1) = (d - 1)/(k - 1)$

We can select any of the *b* possible sequences for each action. Hence,

Total number of trees possible,

$n = b^{(d-1)/(k-1)}$

Thus, the algorithm needs to construct *n* trees to find a solution.

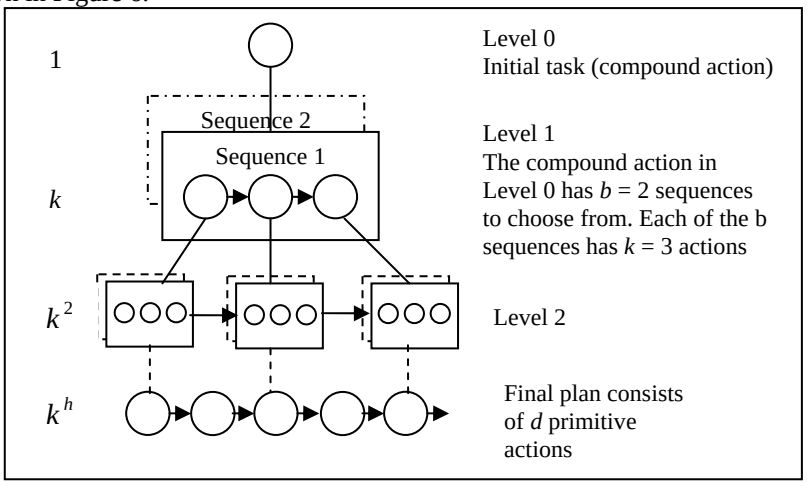


Figure **6**: HTN planning tree with $k = 3$ and $b = 2$

Choosing $d = 10$ and $k = 5$, we get the number of actions per tree to be 2.25. Assuming that 10^6 action nodes can be generated per second [6], a rough estimate of the time taken by the algorithm for various values of *b* is shown in Table 4.

b	**No. of trees constructed**	**Total no. of actions**	**Estimated time taken**
10	**177**	**400**	**0.0004 sec**
50	**6650**	**15000**	**0.015 sec**
100	**31650**	**72000**	**0.07 sec**

Table **4**: Rough estimate of time taken by HTN planning for d =10 and k = Z

The time complexity of HTN algorithm is orders of magnitude smaller than GRAPHPLAN because it makes use of hierarchical action decompositions. From Table 4, we see that HTN algorithm meets the time constraints of the surgery cell.

Conclusions

The conclusions drawn in section 4 are summarized in Table 5.

Criterion	Primitive Action Planning		KB Planning	
Meaningful interaction with Surgeon	**No**		**Yes**	
Execution monitoring and continuous plan modification	**No**		**No**	
Anticipatory Planning	**No**		**No**	
Planning to meet time constraints	**GRAHPLAN**	**May if problem size is small**	**HTN Planning**	**Yes**

Table **5**: Summary of the evaluation of planning algorithms against the requirements

The currently existing classical planning techniques do not meet the requirements of a planner for fully automated support functions in a tele-operated surgery cell. PA planning methods cannot interact with the surgeon in a meaningful way because the domain model used by them does not contain criteria and rules. KB planning methods use a domain model that contains rules and criteria which a human can understand and augment. Hence, KB planers are able to interact with the surgeon in a meaningful way. Both PA and KB planning methods do not meet two of the important requirements, namely (1) Execution monitoring and continuous plan modification, and (2) Anticipatory planning, because they assume a deterministic world and do not observe and learn from the world.

In general, *all* classical planning techniques are unsuitable for fully automated task planning in surgery cells because these techniques assume a deterministic world and fully automated surgery cell is not a deterministic world.

References

[1] Bove, P., Stoianovici, D., Micali, S., Patriciu, A., Grassi, N., Jarrett, T.W., Vespasiani, G., and Kavoussi, L.R., 2003, "Is Telesurgery a New Reality? Our Experience with Laparoscopic and Percutaneous Procedures," *Journal of Endourology*, **17(3)**, pp. 137-142.

[2] Meadows, M., May-June 2002, "Robots Lend a Helping Hand to Surgeons," *U.S. Food and Drug Administration: FDA Consumer magazine*, Retrieved October 30, 2005, from http://www.fda.gov/fdac/features/2002/302_bots.html

[3] Miyawaki, F., Masamune, K., Suzuki, S., Yoshimitsu, K., and Vain, J., 2005, "Scrub Nurse Robot System-Intraoperative Motion Analysis of a Scrub Nurse and Timed-Automata-Based Model for Surgery," *IEEE Transactions on Industrial Electronics*, **52(5),** pp. 1227-1235.

[4] LeGoullon, A.P., and Tesar, D., 1997, "Configuration Management of Robotic Workcells," M.S. Thesis, The University of Texas at Austin, Austin, TX.

[5] Taylor, R.H., 2003, "Medical Robotics in Computer-Integrated Surgery," *IEEE Transactions on Robotics and Automation*, **19(5),** pp. 765-781.

[6] Russell, S., and Norvig, P., 2003, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey, pp. 375-461.

[7] Georgeff, M. P., 1987, "Planning," *Annual Review of Computer Science*, Annual Reviews Inc., Palo Alto, CA, **2,** pp. 69-74, 359-400.

[8] Wilkins, D.E., and desJardins, M., 2001, "A Call for Knowledge-based Planning," *AI Magazine*, **22,** pp. 99-115.

[9] Tisius, M. S., and Tesar, D., 2004, "An Empirical Approach to Performance Criteria and Redundancy Resolution", M.S. Thesis, The University of Texas at Austin, Austin, TX.

[10] Chien, S., Hill, R., Wang, X., Estlin, T., Fayyad, K., and Mortenson, H., 1996, "Why Real-World Planning is Difficult: A Tale of Two Applications," *New Directions in AI Planning*, IOS Press, Washington, D.C., pp. 287-298.

[11] Blum, A.L., and Furst ,M.L., 1997, "Fast Planning through Planning Graph Analysis," *Artificial Intelligence*, **90**, pp. 281–300.

[12] Weld, D.S., 1999, "Recent Advances in AI Planning," *AI Magazine*, **20(2)**, pp. 93-123.

[13] Myers, K. L.,1996, “Strategic Advice for Hierarchical Planners”, *Proc. of 5th International Conference of Principles of Knowledge Representation and Reasoning,* Aiello, L. C. et al., eds., Morgan Kaufmann Publishers Inc., San Francisco, CA, pp. 112–123.
[14] Erol, K., Hendler, J., and Nau, D., 1994, “UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning,” *Proc. 2nd International Conference on AI Planning Systems*, Hammond, K., ed., AAAI Press, Menlo Park, CA, pp. 249-254.

Appendix G: Real Time Robot Capability Analysis

Proceedings of IDETC/CIE 2005
ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference
September 24 – 28, 2005, Long Beach, California, USA

DETC2005-84353

REAL-TIME ROBOT CAPABILITY ANALYSIS

Chalongrath Pholsiri
Postdoctoral Fellow
Robotics Research Group
Department of Mechanical Engineering
The University of Texas at Austin
longrath@mail.utexas.edu

Chetan Kapoor
Chief Scientist
Robotics Research Group
Department of Mechanical Engineering
The University of Texas at Austin
chetan@mail.utexas.edu

Delbert Tesar
Professor, Director
Robotics Research Group
Department of Mechanical Engineering
The University of Texas at Austin
tesar@mail.utexas.edu

ABSTRACT

Robot Capability Analysis (RCA) is a process in which force/motion capabilities of a manipulator are evaluated. It is very useful in both the design and operational phases of robotics. Traditionally, ellipsoids and polytopes are used to both graphically and numerically represent these capabilities. Ellipsoids are computationally efficient but tend to underestimate while polytopes are accurate but computationally intensive. This article proposes a new approach to RCA called the *Vector Expansion (VE)* method. The VE method offers accurate estimates of robot capabilities in real time and therefore is very suitable in applications like task-based decision making or online path planning. In addition, this method can provide information about the joint that is limiting a robot capability at a given time, thus giving an insight as to how to improve the performance of the robot. This method is then used to estimate capabilities of 4-DOF planar robots and the results discussed and compared with the conventional ellipsoid method. The proposed method is also successfully applied to the 7-DOF Mitsubishi PA10-7C robot.

1. INTRODUCTION

Evaluation of force/motion capabilities of a manipulator is very important in all phases of robotics. In the design phase, it can be used to determine the structure and size of a manipulator, including the specifications of the main components such as actuators. Before a robotic operation, RCA can for example be used to plan a trajectory that can be executed by a robot without violating its joint speed or torque constraints. Note that, in these two cases, the evaluation of robot capabilities need not be in real time.

RCA is also useful during a robotic task execution. For a robotic task to be performed successfully, we must satisfy all the task requirements, which are defined in terms of desired speed, force, and accuracy at the end-effector (EEF). RCA can then be used to estimate the robot's *achievable* speed, force, and accuracy and compare them to the desired values. If the robot in operation is kinematically redundant, the null space motion can be exploited to place the robot in a configuration that satisfies the task requirements [11]. For non-redundant robots, one can continuously monitor robot capabilities to determine whether or not the robot would be able to successfully execute the task at hand. If the robot seems unable to, then it could alert an external path planner to generate a new trajectory with less stringent task requirements. For RCA to be applicable during task execution, it must be performed in real time. This is the kind of application that the proposed RCA technique in this article is aimed at.

This paper discusses how to accurately and quickly estimate robot capabilities from the robot properties, joint capacities, and robot configuration using a novel method called Vector Expansion (VE). Like polytopes, joint capability (torque, speed, etc.) limits are represented and used in their

simplest form – a hypercube – thus giving more accurate estimates than the ellipsoid method, which uses a hypersphere. Since VE is intended for computing robot capabilities only in a direction of interest in the task space, it can be made to compute much faster than polytopes, which are meant to represent robot capabilities in all directions. The VE method is based on the ellipsoid expansion method proposed by Bowling and Khatib [1].

2. ISSUES ON ROBOT CAPABILITY ESTIMATIONS

This section discusses two primary issues that arise when attempting to develop the formulations for estimating robot capabilities.

2.1 Homogeneity of the Jacobian Matrix

One important aspect that has to be pointed out here is that the measures whose values depend upon the manipulator Jacobian *J*, JJ^T, or the pseudoinverse of *J* suffer from possible inconsistency deriving from improper use of vector norm and from dependency on change of scale and coordinate frame [5]. Chiacchio and Concilio [3] avoided this problem when formulating a dynamic manipulability ellipsoid by assuming that all the joints are of the same kind (prismatic or revolute) and that the task space is composed by either linear or angular motion. Yoshikawa [14] decomposed the task space and the Jacobian matrix into the translation and rotation parts and proposed the translational and rotational manipulability measures. Combining this Jacobian decomposition with normalizing the input vector gives us *homogeneous* Jacobian matrices (i.e. Jacobian matrices whose elements are of the same units). Consider the following example. Suppose that the robot has two joints with the first joint being revolute and the second prismatic. Furthermore, the task space has two outputs; the first is translation and the other rotation. The relationship between the task space velocity and the joint space velocity is:

$$\dot{x} = J\dot{\theta}. \tag{1}$$

where $\dot{x}$ is an *m*-dimensional EEF velocity vector, J the $m \times n$ Jacobian matrix, and $\dot{\theta}$ the *n*-dimensional joint velocity vector. The dimension analysis of Eq. (1) yields[1]

$$\begin{bmatrix} length/time \\ 1/time \end{bmatrix} = \begin{bmatrix} length & 1 \\ 1 & 1/length \end{bmatrix} \begin{bmatrix} 1/time \\ length/time \end{bmatrix}. \tag{2}$$

Obviously, the Jacobian contains mixed units. Now, let's decompose the Jacobian into the translation and rotation parts such that Eq. (1) becomes

$$\begin{aligned} \dot{x}_T &= J_T\dot{\theta} \\ \dot{x}_R &= J_R\dot{\theta} \end{aligned}. \tag{3}$$

where $\dot{x}_T$ and $\dot{x}_R$ are, respectively, the m_T-dimension translational velocity and the m_R-dimension rotational velocity of the end-effector; J_T and J_R are the translational and rotational Jacobian matrices, respectively. Now the dimension analysis yields

$$\begin{aligned} length/time &= \begin{bmatrix} length & 1 \end{bmatrix} \begin{bmatrix} 1/time \\ length/time \end{bmatrix} \\ 1/time &= \begin{bmatrix} 1 & 1/length \end{bmatrix} \begin{bmatrix} 1/time \\ length/time \end{bmatrix} \end{aligned}. \tag{4}$$

These decomposed Jacobian matrices (J_T and J_R) still contain mixed units. The next step is to normalize the input vector (joint velocities) by their speed limits. Let's define the normalized joint velocity vector as

$$\tilde{\dot{\theta}} = L_{\dot{\theta}}^{-1}\dot{\theta}, \tag{5}$$

where $L_{\dot{\theta}} = diag\left\{\dot{\theta}_1^{\max}, \dot{\theta}_2^{\max}, \ldots, \dot{\theta}_n^{\max}\right\}$ is an $n \times n$ diagonal matrix consisting of the joint speed limits. Note that the normalized input vector $\tilde{\dot{\theta}}$ is dimensionless. This normalization process is not only necessary for a mixed-joint robot but also useful for a same-joint (either revolute or prismatic) robot with each joint possessing different capability limit. It is assumed here that the joint speed upper and lower limits are of equal magnitude but in the opposite direction. Substituting $\dot{\theta}$ from Eq. (5) into Eq. (3) yields

$$\begin{aligned} \dot{x}_T &= J_T L_{\dot{\theta}} \tilde{\dot{\theta}} \triangleq \tilde{J}_T \tilde{\dot{\theta}} \\ \dot{x}_R &= J_R L_{\dot{\theta}} \tilde{\dot{\theta}} \triangleq \tilde{J}_R \tilde{\dot{\theta}} \end{aligned} \tag{6}$$

The new dimension analysis yields

$$\begin{aligned} length/time &= \begin{bmatrix} length & 1 \end{bmatrix} \begin{bmatrix} 1/time & 0 \\ 0 & length/time \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} length/time & length/time \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ 1/time &= \begin{bmatrix} 1 & 1/length \end{bmatrix} \begin{bmatrix} 1/time & 0 \\ 0 & length/time \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 1/time & 1/time \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \end{aligned}. \tag{7}$$

The new Jacobian matrices ($\tilde{J}_T$ and $\tilde{J}_R$) now have consistent units and therefore any measures derived from these matrices do not suffer the same potential inconsistency as those derived directly from the manipulator Jacobian *J*.

2.2 Ellipsoids and Polytopes

The concept of ellipsoids is perhaps the most popular tool used to analyze the robot's manipulability and kinetostatic capability due to its simplicity and ease of interpretation. In the ellipsoid concept, the set of realizable EEF velocities from the set of joint velocities whose Euclidean norm is less than or equal to one can be obtained by transforming the space $Q_e = \left\{\dot{\theta} : \left\|\dot{\theta}\right\|_2 \le 1\right\}$ via Eq. (1). Note that this transformation changes the Euclidean norm which is a unit hypersphere in the joint space to an ellipsoid in the task space. Figure 1 illustrates the ellipsoid concept for a 2-DOF planar manipulator with the task space of dimension 2. $\sigma_{\min}$ and $\sigma_{\max}$ are, respectively, the minimum and maximum singular values of the Jacobian, which correspond to the directions of minimum and maximum task space velocities, respectively.

[1] Note that prismatic joints do not contribute to rotational motion and thus the corresponding Jacobian elements should be zero. However, for consistency in the dimension analysis, their dimensions are shown here.

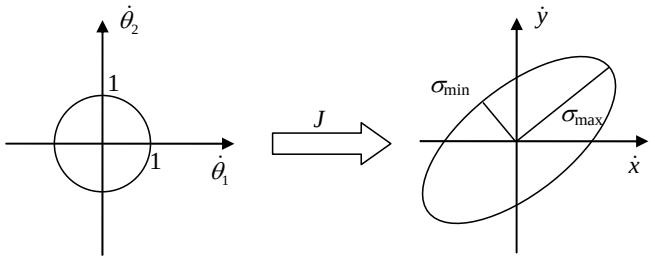


Figure 1: A 2-DOF example of ellipsoid transformation.

The ellipsoid transformation can be used to estimate the robot EEF capabilities given the joint (or actuator) capabilities as will be shown shortly. However, it is noteworthy that an ellipsoid only approximates a more accurate *polytope*. A polytope is the result of transformation from a hypercube via a linear transformation in the same way an ellipsoid is from a hypersphere. Since actuators have maximum or minimum performance values (speed limits, torque limits, etc.), a more accurate representation of their capabilities is by the ∞-norm (or a hypercube) not the Euclidean norm (or a hypersphere). Ellipsoids can be considered *conservative* approximations of polytopes because they always yield smaller EEF capabilities. Figure 2 compares the ellipsoid and the polytope of a 2-DOF planar robot. In addition, Lee [10] also showed that the directions of minimum and maximum velocity transmission ratios obtained via the ellipsoid method may not be the same as those obtained from the polytope method.

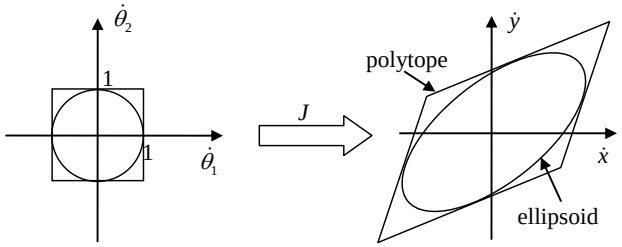


Figure 2: A 2-DOF example of ellipsoid and polytope.

Although polytopes are a more accurate representation than ellipsoids, they are also complex and considerably more expensive to compute, especially for redundant manipulators. Using a new recursive dimension-growing algorithm, Hwang *et al*. [8] showed that the computation time of a velocity polytope of a 10-DOF robot on to a 3D task space was improved from 427.52 seconds using a conventional method to 6.64 seconds on an HP 700 system. Even with the remarkable improvement in computation speed, it is still far from being acceptable for real-time applications. Many researchers including Chiacchio *et al.* [2], Lee [10], and Finotello *et al.* [6] have applied the concept of polytopes to analyzing kinetostatic capabilities of manipulators.

In their investigation of the acceleration characteristics of non-redundant manipulators, Bowling and Khatib [1] used the *ellipsoid expansion* approach. In this approach, "the linear and angular accelerations are considered as two separate entities." This decomposition of the task space was discussed in detail in the previous section. A unique trait of this approach, however, is that quantities of interest are mapped from the task space to the joint space whereas other studies do the opposite.

Figure 3 illustrates how the ellipsoid expansion method works. The purpose here is to find the largest possible magnitude of acceleration in all directions that the manipulator can provide. This isotropic acceleration is represented by a hypersphere with radius *a* as shown in the right hand side of Figure 3. This hypersphere is then mapped from the task space to the joint space as an ellipsoid as shown in the left hand side of Figure 3. The isotropic acceleration magnitude *a* is determined by expanding/contracting the ellipsoid until it lies within and is tangent to one or more of the torque bounds.

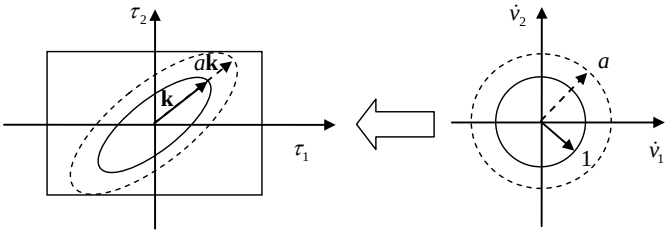


Figure 3: Ellipsoid expansion (Adapted from [1]).

With this reverse mapping, the computationally expensive process of creating polytopes in the task space is eliminated. Yet, bounds on joint capabilities are represented and used in its simplest and exact form as a hypercube. Thus, the ellipsoid expansion method has a benefit of being more accurate than the ellipsoid model without the computational overhead of the polytope. Moreover, an important piece of information obtained via this method that is not possible in the ellipsoid method is the limiting joint, i.e. the joint whose capability limits the EEF capability. In Figure 3 above, Joint 2 is the limiting joint because the expanded ellipsoid touches the torque bound of Joint 2. The limiting joint information is extremely valuable in the design phase of the robot. However, this approach does not directly provide the visualization of the robot capabilities in the task space that ellipsoids or polytopes can. Perhaps more importantly, it fails to impart the general characteristics of the robot capabilities in the task space such as the directions of maximum or minimum transmission ratios, etc. Nonetheless, for our purpose of estimating the robot capabilities in a given direction, the ellipsoid expansion approach may be more appropriate than either the ellipsoid or the polytope method.

In the development of the estimation of robot capabilities, the normalized decomposed Jacobian matrices will be used whenever appropriate to promote the consistency of the results. In addition, polytopes will not be utilized due to their unacceptable computation time as pointed out earlier. We will opt instead for the ellipsoid model and the vector expansion method, which is adapted from the ellipsoid expansion method proposed by Bowling and Khatib [1]. As its name implies, this method expands or contracts a vector, instead of an ellipsoid, to determine the robot capabilities. Like the ellipsoid expansion method, the vector expansion method has a benefit of being more accurate than the ellipsoid model without the

computational overhead of the polytope. Also, the limiting joint information is still available as it is in the ellipsoid expansion method. The main difference between the vector and ellipsoid expansion methods is that the latter uses isotropic properties, which aim at determining robot capabilities in *all* directions. This is appropriate when we wish to analyze the overall robot capability, for example, in the design stage of the robot. However, during robot operations in which the requirements in some directions are more demanding than those in others, isotropic characteristics may not be desirable.

3. ROBOT CAPABILITY ESTIMATIONS

3.1 Achievable EEF Velocity

Based upon the joint speed limits, this section presents two methods that can be used to estimate the EEF speed in the task space from a purely kinematic point of view.

Ellipsoid Formulation

This section formulates the EEF achievable speed estimation using the ellipsoid concept. Based upon the decomposed Jacobian matrices in Eq. (3), Yoshikawa [14] defined the Translational Velocity Ellipsoid (TVE)[2] *in the weak sense* as "the set of all translational velocities that are realizable under the constraint $\left\|\dot{\theta}\right\|_2 \le 1$." The TVE *in the strong sense* adds another constraint that the EEF orientation is kept constant ($\dot{x}_R = 0$). The Rotational Velocity Ellipsoids (RVEs) in the weak and strong senses are defined similarly. However, as we have shown previously, the decomposed Jacobian matrices in Eq. (3) are still not homogeneous. It is better to use the normalized decomposed Jacobian matrices in Eq. (6).

Because the analyses are identical, we will only develop the formulations on the translational EEF velocity and will deduce the results for the rotational EEF velocity. The translation part of Eq. (6) is repeated here for convenience.

$$\dot{x}_T = J_T L_\theta \tilde{\dot{\theta}} \triangleq \tilde{J}_T \tilde{\dot{\theta}} \tag{8}$$

Using the pseudoinverse, we obtain

$$\tilde{\dot{\theta}} = \tilde{J}_T^\dagger \dot{x}_T \,. \tag{9}$$

Then,

$$\left\|\tilde{\dot{\theta}}\right\|_2 = \tilde{\dot{\theta}}^T \tilde{\dot{\theta}} = \dot{x}_T^T \left(\tilde{J}_T^\dagger\right)^T \tilde{J}_T^\dagger \dot{x}_T \,. \tag{10}$$

Therefore, the TVE in the weak sense is described by

$$\left\{\dot{x}_T : \dot{x}_T^T \left(\tilde{J}_T^\dagger\right)^T \tilde{J}_T^\dagger \dot{x}_T \le 1 \text{ and } \dot{x}_T \in R\left(\tilde{J}_T\right)\right\} \tag{11}$$

where $R\left(\tilde{J}_T\right)$ denotes the range of $\tilde{J}_T$. If the manipulator is not in a singular configuration, then the condition $\dot{x}_T \in R\left(\tilde{J}_T\right)$ is not necessary. Rearranging a few terms and Eq. (11) can be rewritten as

$$\left\{\dot{x}_T : \dot{x}_T^T \left(J_T L_\theta^2 J_T^T\right)^{-1} \dot{x}_T \le 1 \text{ and } \dot{x}_T \in R\left(\tilde{J}_T\right)\right\}. \tag{12}$$

[2] In Yoshikawa [1991], the term Translational Manipulability Ellipsoid (TME) was used but we think the term Translational Velocity Ellipsoid is more appropriate and thus will be used here.

Let $\dot{x}_T = v\hat{t}$ where $\hat{t}$ is the unit vector in the direction of interest in the task space. Then, the inequality in Eq. (12) becomes

$$\left[\hat{t}^T \left(J_T L_\theta^2 J_T^T\right)^{-1} \hat{t}\right] v^2 \le 1 \,. \tag{13}$$

Thus, the maximum achievable translational EEF speed in the $\hat{t}$ direction can be estimated by

$$v_{\max} = \pm \frac{1}{\sqrt{\hat{t}^T \left(J_T L_\theta^2 J_T^T\right)^{-1} \hat{t}}} \,. \tag{14}$$

Similarly, the maximum achievable rotational EEF speed in the $\hat{t}$ direction is

$$\omega_{\max} = \pm \frac{1}{\sqrt{\hat{t}^T \left(J_R L_\theta^2 J_R^T\right)^{-1} \hat{t}}} \,. \tag{15}$$

Vector Expansion Formulation

This section discusses how to estimate the manipulator's speed capabilities using the *vector expansion* method. Note from Eq. (8) that the bounds on $\tilde{\dot{\theta}}$ can be written as

$$-\mathbf{1} \le \tilde{\dot{\theta}} \le \mathbf{1} \tag{16}$$

where **1** is a vector of *n*-dimension with each element equal to one. Combining Eqs. (9) and (16) gives

$$-\mathbf{1} \le \tilde{J}_T^\dagger \dot{x}_T \le \mathbf{1} \,. \tag{17}$$

Again, let $\dot{x}_T = v\hat{t}$ where $\hat{t}$ is the unit vector in the direction of interest in the task space. The maximum achievable speed is conceptually determined by expanding/contracting the vector $v\tilde{J}_T^\dagger \hat{t}$ (changing *v*) until it touches one of the joint speed limits as depicted in Figure 4.

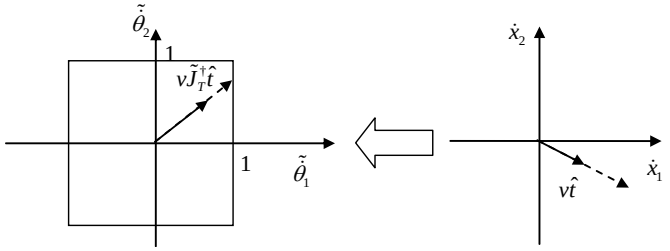


Figure 4: Joint speed limits and velocity vector expansion.

In determining $v_{\max}$ (by expanding or contracting $v\tilde{J}_T^\dagger\hat{t}$), we can compare; the vectors $\tilde{J}_T^\dagger\hat{t}$ and **1** element by element. By inspecting Eq. (17), the maximum achievable translational EEF speed in the $\hat{t}$ direction can be expressed as

$$v_{\max} = \min_i \frac{1}{\left|\tilde{J}_T^\dagger \hat{t}\right|_i} \,, \quad i = 1, \ldots, n \tag{18}$$

where $\left|\tilde{J}_T^{\dagger}\hat{t}\right|_i$ is the absolute value of the *i*th-component of the vector $\tilde{J}_T^{\dagger}\hat{t}$. Similarly, the maximum achievable rotational EEF speed in the $\hat{t}$ direction is

$$\omega_{\max} = \min_i \frac{1}{\left|\tilde{J}_R^{\dagger}\hat{t}\right|_i}, \quad i = 1,\ldots,n. \tag{19}$$

3.2 Achievable EEF Acceleration

The previous section discussed the achievable EEF speed from a purely kinematic point of view. The true achievable EEF speed also depends on how much the robot can accelerate its EEF. In this section, we wish to determine the achievable EEF acceleration, which is a function of the robot's joint torque limits and its dynamic properties.

Ellipsoid Formulation

The formulations here are adapted from the work done on dynamic manipulability by Chiacchio and Concilio [3]. The second-order differential kinematics of a manipulator can be written as

$$\ddot{x} = J\ddot{\theta} + \dot{J}\dot{\theta} \tag{20}$$

The robot dynamics equation can be written as

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + g(\theta) + J^T F_e = \tau \tag{21}$$

where $\tau \in \mathcal{R}^n$ is a generalized joint torque vector, $M(\theta) \in \mathcal{R}^{n\times n}$ a symmetric and positive definite joint inertia matrix, $C(\theta,\dot{\theta})\dot{\theta} \in \mathcal{R}^n$ the Coriolis/centripetal torque, $g(\theta) \in \mathcal{R}^n$ the torque due to gravity, and $F_e \in \mathcal{R}^m$ the contact force exerted by the EEF.

The robot's acceleration capability is defined as the ability of the robot to move its EEF from stationary ($\dot{\theta} = 0$) with no external force acting on the EEF (F_e = 0). Applying these two conditions, Eq.(20) becomes

$$\ddot{x} = J\ddot{\theta}, \tag{22}$$

and Eq. (21) is reduced to

$$M(\theta)\ddot{\theta} + g(\theta) = \tau. \tag{23}$$

Define the normalized joint torque vector

$$\tilde{\tau} = L_\tau^{-1}\tau \tag{24}$$

where $L_\tau = diag\left\{\tau_1^{\max}, \tau_2^{\max}, \ldots, \tau_n^{\max}\right\}$ is a $n \times n$ diagonal matrix consisting of the joint torque limits. Combining Eqs. (22)-(24) yields

$$\ddot{x}_T = B_T\tilde{\tau} + \ddot{x}_{gT} \tag{25}$$

where $B_T = J_T M^{-1} L_\tau$ and $\ddot{x}_{gT} = -J_T M^{-1} g$. Using the pseudoinverse of B_T in Eq. (25),

$$\tilde{\tau} = B_T^{\dagger}\left(\ddot{x}_T - \ddot{x}_{gT}\right) \tag{26}$$

$\tilde{\tau}$ in Eq. (26) can be thought of as a set of joint torques required to generate the translational (linear) acceleration $\ddot{x}_T$ in the task space. Therefore, it is technically more correct to denote it as $\tilde{\tau}_T$. Similarly, the joint torques required to generate the rotational (angular) acceleration $\ddot{x}_R$ can be expressed as

$$\tilde{\tau}_R = B_R^{\dagger}\left(\ddot{x}_R - \ddot{x}_{gR}\right) \tag{27}$$

where $B_R = J_R M^{-1} L_\tau$ and $\ddot{x}_{gR} = -J_R M^{-1} g$. The total normalized torque is then the sum of the two $\tilde{\tau} = \tilde{\tau}_T + \tilde{\tau}_R$.

The unit sphere in the space of normalized joint torques $\tilde{\tau}^T\tilde{\tau} \le 1$ can now be mapped to the task space to give the translational acceleration ellipsoid as

$$\left(\ddot{x}_T + J_T M^{-1} g\right)^T B_T^{\dagger T} B_T^{\dagger}\left(\ddot{x}_T + J_T M^{-1} g\right) \le 1. \tag{28}$$

Defining $Q = ML_\tau^{-2}M$, Eq. (28) can be rewritten as

$$\left(\ddot{x}_T + J_T M^{-1} g\right)^T \left(J_T Q^{-1} J_T^T\right)^{-1}\left(\ddot{x}_T + J_T M^{-1} g\right) \le 1. \tag{29}$$

Let $N_T \equiv \left(J_T Q^{-1} J_T^T\right)^{-1}$ and $\ddot{x}_T = a_T\hat{t}$ where $\hat{t}$ is the unit vector in the direction of interest in the task space. Then Eq. (29) can be rewritten as

$$\alpha_T f^2 + 2\beta_T f + \gamma_T \le 0 \tag{30}$$

where

$$\alpha_T = \hat{t}^T N_T \hat{t}$$
$$\beta_T = -\hat{t}^T N_T \ddot{x}_{gT}$$
$$\gamma_T = \ddot{x}_{gT}^T N_T \ddot{x}_{gT} - 1$$

Solving Eq. (30) yields the estimation of the achievable translational EEF acceleration given by

$$\frac{-\beta_T - \sqrt{\beta_T^2 - \alpha_T\gamma_T}}{\alpha_T} \le a_T \le \frac{-\beta_T + \sqrt{\beta_T^2 - \alpha_T\gamma_T}}{\alpha_T}. \tag{31}$$

The rotational EEF acceleration capability can be estimated using Eqs. (29)-(31) with J_T being replaced with J_R.

Vector Expansion Method

The bounds on the normalized joint torques can be written as

$$-\mathbf{1} \le \tilde{\tau} \le \mathbf{1} \tag{32}$$

where **1** is a vector of *n*-dimension with each element equal to one. Substituting $\tilde{\tau}$ from Eq. (26) in Eq. (32) yields

$$-\mathbf{1} \le B_T^{\dagger}\left(\ddot{x}_T - \ddot{x}_{gT}\right) \le \mathbf{1}. \tag{33}$$

Rearranging a few terms gives the governing equation as

$$\tau_{lower} \le B_T^{\dagger}\ddot{x}_T \le \tau_{upper} \tag{34}$$

where $\tau_{upper} = \mathbf{1} + B_T^{\dagger}\ddot{x}_{gT}$ and $\tau_{lower} = -\mathbf{1} + B_T^{\dagger}\ddot{x}_{gT}$. τ_{upper} and τ_{lower} can be thought of, respectively, as vectors of upper and lower limits of normalized joint torques shifted by the robot links' weights (gravity effects).

Let $\ddot{x}_T = a\hat{t}$ where $\hat{t}$ is the unit vector in the direction of interest in the task space. Then, Eq. (34) can be rewritten as

$$\tau_{lower} \le aB_T^{\dagger}\hat{t} \le \tau_{upper} \tag{35}$$

The vector expansion model describing Eq. (35) is illustrated in Figure 5.

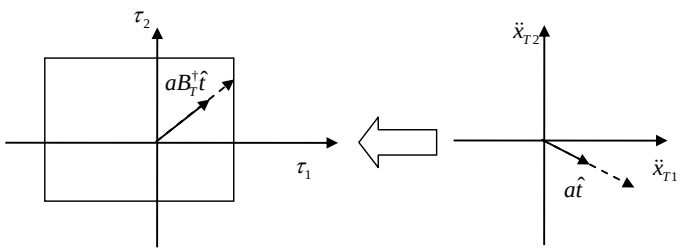


Figure 5: Joint torque bounds and acceleration vector expansion.

Note here that, unlike the joint speed bounds, the torque bounds in Figure 5 are not symmetrical due to the weight of the robot. If the bounding box is shifted so far that it does not contain the origin, it means the robot cannot support its own weight. This should never happen in real operations. Mathematically, this means that all elements of τ_{lower} have to remain negative and τ_{upper} positive. But this situation could arise if we use this analysis as a tool in designing robots. By expanding/contracting the vector $aB_T^{\dagger}\hat{t}$, the magnitude of the achievable translational acceleration is determined by

$$a \le \min_i \left\{ \max \left(\frac{\tau_{upper,i}}{\left[B_T^{\dagger}\hat{t}\right]_i}, \frac{\tau_{lower,i}}{\left[B_T^{\dagger}\hat{t}\right]_i} \right) \right\}, \quad i = 1,\ldots,n \qquad (36)$$

where $\tau_{lower,i}$, $\tau_{upper,i}$, and $\left[B_T^{\dagger}\hat{t}\right]_i$ are the i^{th} elements of vectors τ_{lower}, τ_{upper}, and $B_T^{\dagger}\hat{t}$, respectively. For each joint *i*, note that one of $\frac{\tau_{upper,i}}{\left[B_T^{\dagger}\hat{t}\right]_i}$ and $\frac{\tau_{lower,i}}{\left[B_T^{\dagger}\hat{t}\right]_i}$ will be positive and the other negative. The max operation will choose the positive one between them, which will present the maximum acceleration allowed by joint *i*. Then, the min operator will choose the maximum acceleration achievable by the whole robot.

Similarly, the rotational EEF acceleration capability can be expressed as

$$\alpha \le \min_i \left\{ \max \left(\frac{\tau_{upper,i}}{\left[B_R^{\dagger}\hat{t}\right]_i}, \frac{\tau_{lower,i}}{\left[B_R^{\dagger}\hat{t}\right]_i} \right) \right\}, \quad i = 1,\ldots,n\,. \qquad (37)$$

3.3 Achievable EEF Static Force Capability

Many studies found in the literature omit the effect of gravity when computing force capability (see [2], [6], and [9] for example). As Hernandez and Tesar [7] pointed out that the weight of the manipulator is usually the most dominant load, especially for industrial robots, it is not wise to ignore the gravity effect when estimating the robot force capability. The impact of the gravity on the force capability will be demonstrated shortly.

Ellipsoid Formulation

Considering the EEF force and gravity, the joint torques can be expressed as

$$\tau = J^T F + g \qquad (38)$$

Decomposing the Jacobian and the EEF force yields

$$\tau = \begin{bmatrix} J_T^T & J_R^T \end{bmatrix} \begin{bmatrix} F_T \\ F_R \end{bmatrix} + g = J_T^T F_T + J_R^T F_R + g\,. \qquad (39)$$

Define the normalized joint torque vector

$$\tilde{\tau} = L_\tau^{-1}\tau \qquad (40)$$

where $L_\tau = diag\left\{\tau_1^{\max}, \tau_2^{\max}, \ldots, \tau_n^{\max}\right\}$ is a $n \times n$ diagonal matrix consisting of the joint torque limits. The force ellipsoid is then described by

$$\tilde{\tau}^T\tilde{\tau} = \left(J_T^T F_T + J_R^T F_R + g\right)^T L_\tau^{-2}\left(J_T^T F_T + J_R^T F_R + g\right) \le 1\,. \qquad (41)$$

Consider the case where F_R = 0 (i.e. the external moment is neglected). Eq. (41) becomes

$$\tilde{\tau}^T\tilde{\tau} = \left(J_T^T F_T + g\right)^T L_\tau^{-2}\left(J_T^T F_T + g\right) \le 1\,, \qquad (42)$$

which describes the Translational Force Ellipsoid (TFE) in the weak sense. Let $F_T = f\hat{t}$ where $\hat{t}$ is the unit vector in the direction of interest in the task space. Then Eq. (42) can be rewritten as

$$\alpha_T f^2 + 2\beta_T f + \gamma_T \le 0 \qquad (43)$$

where

$$\alpha_T = \hat{t}^T J_T L_\tau^{-2} J_T^T \hat{t}$$
$$\beta_T = g^T L_\tau^{-2} J_T^T \hat{t}$$
$$\gamma_T = g^T L_\tau^{-2} g - 1$$

Solving Eq. (43) yields the estimation of the achievable translational EEF force capability given by

$$\frac{-\beta_T - \sqrt{\beta_T^2 - \alpha_T\gamma_T}}{\alpha_T} \le f \le \frac{-\beta_T + \sqrt{\beta_T^2 - \alpha_T\gamma_T}}{\alpha_T}\,. \qquad (44)$$

The rotational EEF force (moment) capability can be estimated using Eqs. (42)-(44) with J_T being replaced with J_R.

Vector Expansion Formulation

The bounds on τ can be written as

$$-\tau_{\max} \le \tau \le \tau_{\max}\,. \qquad (45)$$

Combining Eqs. (39)-(40) with (45) yields

$$-\mathbf{1} \le L_\tau^{-1}\left(J_T^T F_T + J_R^T F_R + g\right) \le \mathbf{1} \qquad (46)$$

where **1** is a vector of *n*-dimension with each element equal to one. Rearranging a few terms gives the governing equation as

$$\tau_{lower} \le L_\tau^{-1} J_T^T F_T + L_\tau^{-1} J_R^T F_R \le \tau_{upper} \qquad (47)$$

where $\tau_{upper} = \mathbf{1} - L_\tau^{-1} g$ and $\tau_{lower} = -\mathbf{1} - L_\tau^{-1} g$.

Consider the case where F_R = 0 (i.e. the external moment is neglected) and let $F_T = f\hat{t}$ where $\hat{t}$ is the unit vector in the direction of interest in the task space. Then, Eq. (47) can be rewritten as

$$\tau_{lower} \le f L_\tau^{-1} J_T^T \hat{t} \le \tau_{upper} \qquad (48)$$

The vector expansion model describing Eq. (48) is illustrated in Figure 6.

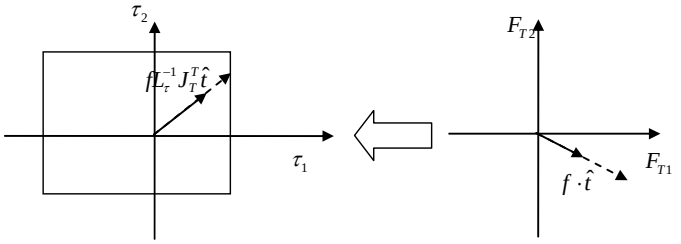


Figure 6: Joint torque bounds and force vector expansion.

Note here that, unlike the joint speed bounds, the torque bounds in Figure 6 are not symmetrical due to the weight of the robot itself. The largest magnitude of translational force that still fits the transformed force vector in the torque bounds is determined by

$$f \le \min_i \left\{ \max \left(\frac{\tau_{upper,i}}{\left[L_\tau^{-1} J_T^T \hat{t} \right]_i}, \frac{\tau_{lower,i}}{\left[L_\tau^{-1} J_T^T \hat{t} \right]_i} \right) \right\}, \quad i = 1, \ldots, n \quad (49)$$

where $\tau_{lower,i}$, $\tau_{upper,i}$, and $\left[L_\tau^{-1} J_T^T \hat{t} \right]_i$ are the i^{th} elements of vectors τ_{lower}, τ_{upper}, and $L_\tau^{-1} J_T^T \hat{t}$, respectively.

Similarly, the rotational EEF force (moment) capability can be found using Eqs. (48) and (49) with J_T being replace with J_R as

$$m \le \min_i \left\{ \max \left(\frac{\tau_{upper,i}}{\left[L_\tau^{-1} J_R^T \hat{t} \right]_i}, \frac{\tau_{lower,i}}{\left[L_\tau^{-1} J_R^T \hat{t} \right]_i} \right) \right\}, \quad i = 1, \ldots, n . \quad (50)$$

Let's reconsider Eq. (47) but without assuming that either the force or moment is zero. Let $F_T = f\hat{t}_T$ and $F_R = m\hat{t}_R$ then Eq. (47) becomes

$$\tau_{lower} \le fL_\tau^{-1} J_T^T \hat{t}_T + mL_\tau^{-1} J_R^T \hat{t}_R \le \tau_{upper} . \quad (51)$$

This equation is illustrated graphically in Figure 7. Note that there is no unique solution to Eq. (51). One can only find *f* given *m*, or vice versa.

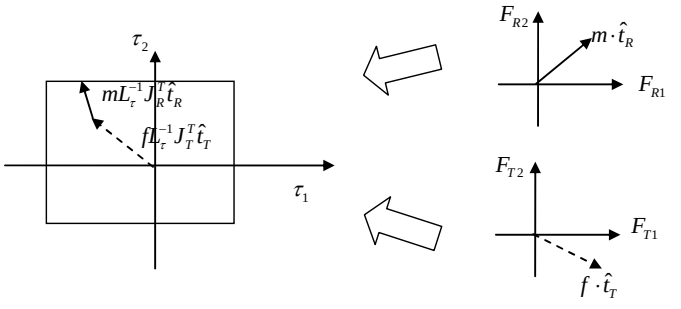


Figure 7: Translation and rotation vectors added.

The process of determining *f* given *m* (or vice versa) is straightforward. For a given *m*, Eq. (51) can be rewritten as

$$\tilde{\tau}_{lower} \le fL_\tau^{-1} J_T^T \hat{t}_T \le \tilde{\tau}_{upper} \quad (52)$$

where $\tilde{\tau}_{lower} = \tau_{lower} - mL_\tau^{-1} J_R^T \hat{t}_R$ and $\tilde{\tau}_{upper} = \tau_{upper} - mL_\tau^{-1} J_R^T \hat{t}_R$. Eq. (49) can then be used to determine *f*.

One intriguing application of the estimation of robot force capability is in the area of fault tolerance and failure recovery. Cocca and Tesar [4] proposed a Partial Failure Torque Minimization criterion for dealing with joint partial failure. Joint partial failure is referred to when one or more joints can no longer supply the torques at their full capacities. The proposed estimation of robot force capability is readily equipped to cope with this circumstance. Once the system detects joint partial failure and determines the reduced torque capabilities for the failed joints, it can set these new values in the joint torque limit matrix L_τ. The new force capability based on the reduced joint torques can then be computed and used in the redundancy resolution process or to create a sense of margin of failure.

3.4 Maximum EEF Position Error

If the joint errors $\Delta\theta$ are assumed to be small, the EEF error can then be estimated by

$$\Delta x = J \Delta\theta . \quad (53)$$

This is the same equation that governs the relationship between the joint and EEF velocities. Thus, the process of estimating the maximum EEF position error is identical to that of estimating the achievable EEF speed.

Ellipsoid Formulation

Similar to the EEF speed, the translational EEF error Δx_T is constrained by

$$\Delta x_T^T \left(J_T L_{\Delta\theta}^2 J_T^T \right)^{-1} \Delta x_T \le 1 . \quad (54)$$

where $L_{\Delta\theta} = diag\left\{ \Delta\theta_1^{\max}, \Delta\theta_2^{\max}, \ldots, \Delta\theta_n^{\max} \right\}$ is a $n \times n$ diagonal matrix consisting of the maximum joint errors. Similar to the achievable EEF speed in the previous section, the maximum translational and rotational EEF position errors can then be estimated as

$$\left| \Delta x_T \right|_{\max} = \frac{1}{\sqrt{\hat{t}^T \left(J_T L_{\Delta\theta}^2 J_T^T \right)^{-1} \hat{t}}}$$
$$\left| \Delta x_R \right|_{\max} = \frac{1}{\sqrt{\hat{t}^T \left(J_R L_{\Delta\theta}^2 J_R^T \right)^{-1} \hat{t}}} . \quad (55)$$

Vector Expansion Formulation

Using the results from Eqs. (18) and (19), the maximum translational and rotational EEF position errors using the vector expansion method can then be estimated as

$$\left| \Delta x_T \right|_{\max} = \min_i \frac{1}{\left| \tilde{J}_T^\dagger \hat{t} \right|_i}, \quad i = 1, \ldots, n$$
$$\left| \Delta x_R \right|_{\max} = \min_i \frac{1}{\left| \tilde{J}_R^\dagger \hat{t} \right|_i}, \quad i = 1, \ldots, n \quad (56)$$

Although there are many more types of errors, in this work, we assume that each maximum joint error can be estimated by

combining the joint encoder resolution with the joint deflection due to joint flexibility.

$$\Delta\theta_i^{\max} = \varepsilon_i + [C\tau]_i \qquad (57)$$

where ε_i is the encoder resolution of joint i, C is the joint compliance matrix, and τ is the joint torque vector. In the static case, the joint torques can be obtained from the gravity torques and the external EEF force. Naturally, others joint errors can be easily added to this model.

4. APPLICATION

In this section, we apply the VE method to a 4-DOF planar robot with link lengths of 0.3, 0.24, 0.1, and 0.08 m. The joint speed limits are 50 deg/second for all joints and joint torque bounds are 100, 45, 35, and 15 N-m. The links weigh 7.5, 5, 2, and 1 kg with the centers of mass at 0.2, 0.15, 0.08, and 0.06 m from the joints. For acceleration and force estimations, gravity is assumed present at 9.8 m/s^2 in the downward direction. The manipulator configuration is at $\theta = [45 \quad -45 \quad -45 \quad -45]^T$ as shown in the figure below.

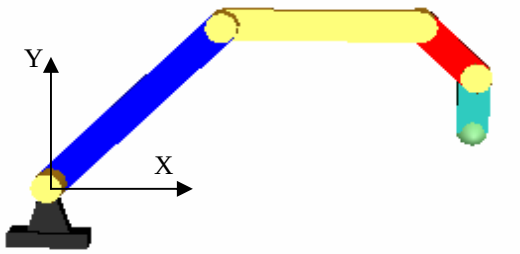


Figure 8: 4-DOF planar robot used in simulations.

Figure 9 shows the velocity ellipsoid (created by the ellipsoid method) and the velocity polytope (created by the vector expansion method) of the 4-DOF robot. The numbers beside the edges of the polytope indicate the limiting joints at those edges. Although the vector expansion method cannot directly create a polytope, the polytope here was generated by incrementing the angle of the unit vector $\hat{t}$ by one degree at a time and recording the resulting robot's speed capability. The ellipsoid was generated the same way even though we could have easily used Eq. (13) to plot the ellipsoid. As expected, the ellipsoid is smaller than the polytope. The ellipsoid gives a fairly good estimation in the Y direction (0.5260 vs 0.5756 m/s) whereas it underestimates the velocity in the X direction by more than 30% (0.2026 vs 0.3222 m/s). The numbers beside the edges of the polytope indicate the limiting joints at those edges. Interestingly, Joint 4 is never the limiting joint in any direction at this configuration.

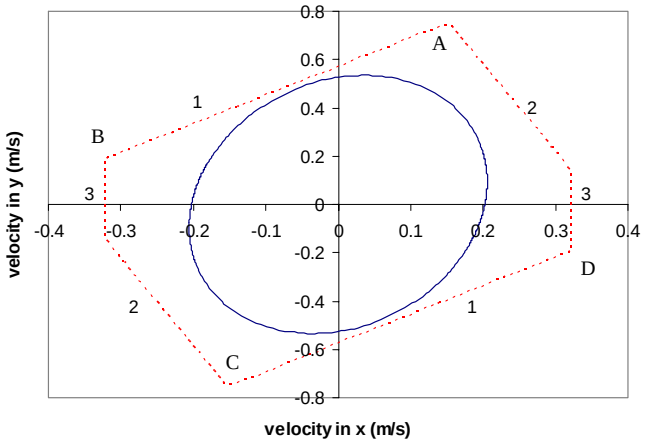


Figure 9: Velocity ellipsoid (solid) and polytope (dotted)

Figure 10 shows the acceleration ellipsoid and polytope of the Planar4R robot. Not surprisingly, both the ellipsoid and polytope are shifted downwards because of gravity (in fact, they are also slightly shifted to the right). Again, ellipsoid does a good job of approximating the acceleration capability in some directions but not as good in others. The numbers beside the edges of the polytope again denote the limiting joints. For this particular case, only Joints 2 and 4 are limiting joints.

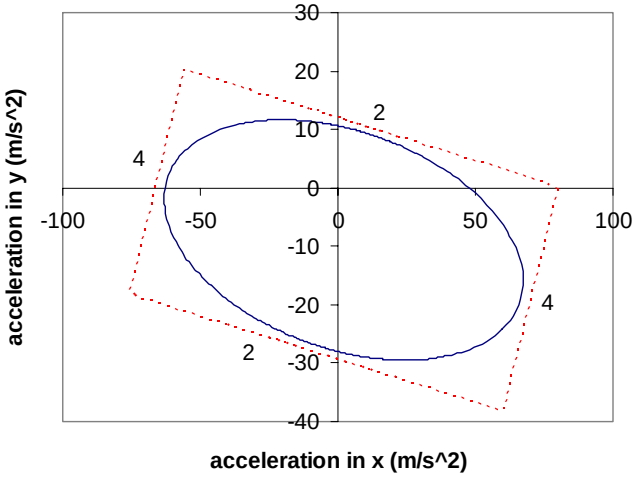


Figure 10: Acceleration ellipsoid (solid) and polytope (dotted)

Force ellipsoid and polytope are shown in Figure 11. Like the acceleration capability, gravity plays a significant role on the ability of the robot to exert force. According to the ellipsoid formulation, with gravity, the robot can exert the force downwards almost 3.5 times as much as it can upwards in this particular configuration. The effect of gravity on the polytope is even greater. Not only does gravity shift the center of the polytope downwards, it alters the shape of the polytope as well. Again, gravity enhances the robot ability to exert the force in the downward direction and reduces that in the upward direction. However, the polytope only shows the ratio of around 2 instead of 3.5 as indicated by the ellipsoid. The significant impact of

gravity on the robot force capability is, without question, illustrated by either model.

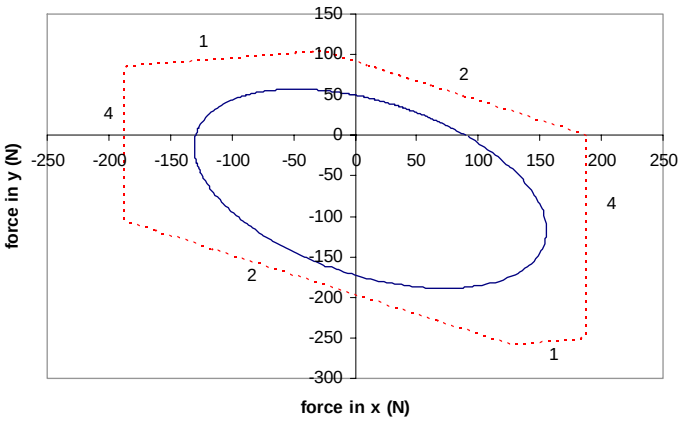


Figure 11: Force ellipsoid (solid) and polytope (dotted)

The proposed method has also been successfully applied to a commercial 7-DOF Mitsubishi PA10-7C robot. The result, which is a movie file showing the force capability of the robot when it moves around, can be found at [13]. The snapshot of the movie is illustrated in Figure 12. Here, the size of the arrow at the EEF is proportional to the amount of force the robot is able to apply in the direction of the arrow. The red color signifies which joint is currently the limiting joint (it is Joint 2 in this picture). The VE method is able to calculate robot capabilities and the limit joint information as the robot moves its EEF or performs self-motion in real time.

Figure 12: Snapshot of force capability of PA10-7C robot.

In addition, the VE method has also been used in a real-time task-based redundancy resolution scheme [11] [12] with great success. Using the 7-DOF Mitsubishi robot shown in Figure 12, the average computation speed of around 130 Hz was achieved on a computer with AMD Athlon-M 2400+ processor with 512 MB of RAM running Windows XP. During each cycle, the computation included several VE calculations, inverse kinematics and other performance criteria. This proved that the VE method is fast enough to be used in real-time task execution.

To prove the validity of the vector expansion method, we compared the polytope generated by this method with the force polytope algorithm proposed by Chiacchio *et al.* [2]. Note however that, in the development of the force polytope algorithm, two restrictive assumptions were made: all the joints are of the same kind and the task space of interest is composed by either forces or moments. Gravity was also omitted in their formulation. In addition, it was not clear whether or not their force polytope algorithm could provide the limiting joint information. We applied the VE method to a 3-DOF planar robot with the link lengths of 0.5, 0.3, and 0.2 m and with the torque bounds of 3, 2, and 1 N-m[3]. The manipulator configuration is $\theta = [60 \quad -60 \quad -60]^T$. The force polytope generated by the vector expansion method, shown in Figure 13, matches the one presented in [2].

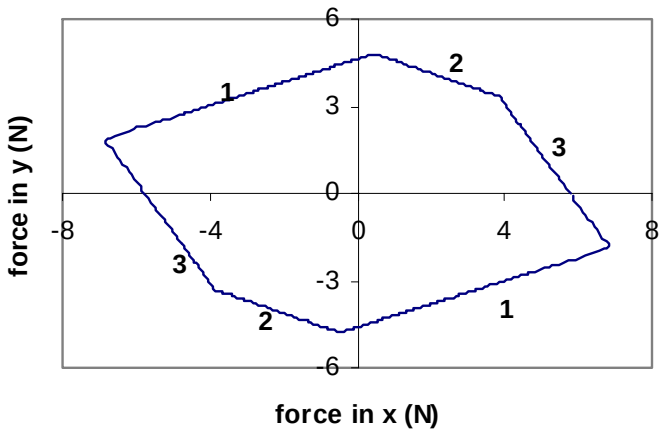


Figure 13: Force polytope for a 3-DOF planar robot.

5. CONCLUSION

The robot capabilities are functions of, among other factors, the joint capabilities and the joint configuration. In this paper, we presented a new approach to accurately estimating the robot capabilities in the task space called the *vector expansion method*. We developed formulations based on the vector expansion method for each of the robot capabilities (speed, accuracy, and force). We also gave numerical examples and compared the results with the widely accepted ellipsoid method.

As has been shown, the vector expansion method yields more accurate estimations than the ellipsoid method because it uses the infinity norms of the bounds of the joint capabilities instead of the Euclidean norm used in the ellipsoid method. Using the reverse mapping, the vector expansion method does not require the generation of the whole polytope and as a result is computationally fast enough to be used in a real-time redundancy resolution process [11] [12].

The robot capability analysis can be used as a tool in many applications including real-time task-based decision making,

[3] These are the same robot properties as those used in [2].

online and offline path planning, and design and/or assembly of robotic manipulators.

ACKNOWLEDGMENTS

This work was supported by funding from Department of Energy (grant no. DE-FG04-94EW37966) and Texas Higher Education Coordinating Board (grant no. ATP 003658-0034-2001).

REFERENCES

[1] Bowling, A. and Khatib, O., 1995, "Analysis of the Acceleration Characteristics of Non-Redundant Manipulators," *Proc. of IEEE/RSJ IROS Conf.*, pp. 323-328.

[2] Chiacchio, P., Bouffard-Vercelli, Y., and Pierrot, F., 1997, "Force Polytope and Force Ellipsoid for Redundant Manipulators," *Journal of Robotic Systems*, v. 14, no. 8, pp. 613-620.

[3] Chiacchio, P. and Concilio, M., 1998, "The Dynamic Manipulability Ellipsoid for Redundant Manipulators," *Proceedings of IEEE Int. Conf. on Robotics and Automation*, Leuven, Belgium, pp. 95-100.

[4] Cocca, C. Cox, D. Tesar, D., 1999, "Failure recovery in redundant serial manipulators using nonlinear programming," *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, May 10-15, pp. 855 – 860.

[5] Doty, K.L., Melchiorri, C., Schwartz, E.M., and Bonivento, C., 1995, "Robot Manipulability," *IEEE Transactions on Robotics and Automation*, v. 11, n. 3, pp. 462-468.

[6] Finotello, R., Grasso, T., Rossi, G., and Terribile, A., 1998, "Computation of Kinetostatic Performances of Robot Manipulators with Polytopes," *Proceedings of IEEE Int. Conf. on Robotics and Automation*, pp. 3241-3246.

[7] Hernandez, E. and Tesar, D., 1996, *Compliance Modeling for General Manipulator Structures*, Ph.D. Dissertation, University of Texas at Austin.

[8] Hwang, Y.-S., Lee, J., and Hsia, T.C., 2000, "A Recursive Dimension-Growing Method for Computing Robotic Manipulability Polytope," *Proceedings of IEEE Int. Conf. on Robotics and Automation*, pp. 2569-2574.

[9] Kim, H.S. and Choi, Y.J., 1999, "The Kinetostatic Capability Analysis of Robotic Manipulators," *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1241-1246.

[10] Lee, J., 1997, "A Study on the Manipulability Measures for Robot Manipulators," *Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1458-1465.

[11] Pholsiri, C., Kapoor, C., and Tesar, D., 2004, "Manipulator Task-Based Performance Optimization," *Proc. of ASME Design Engineering Technical Conference*, Salt Lake City, UT.

[12] Pholsiri, C., Kapoor, C., and Tesar, D., 2004, *Task-Based Decision Making and Control of Robotic Manipulators*, Ph.D. Dissertation, University of Texas at Austin.

[13] RRG Simulation web site http://www.robotics.utexas.edu/simulations/Subjects/Robotics/Mitsubishi 7DOF/index.htm

[14] Yoshikawa, T., 1991, "Translational and Rotational Manipulability of Robotic Manipulators," *Proceedings of Conference on Industrial Electronics, Control and Instrumentation IECON*, pp. 1170-1175.