AFRL-RX-WP-TR-2012-0344

iFAB SMART MANUFACTURING ADAPTING RAPIDLY TO PRODUCT VARIANTS (SMARTV)

David Bourne
Carnegie Mellon University

S.K. Gupta
University of Maryland

Kazu Saitou
University of Michigan

Prasanta Bose
Lockheed Martin Space Systems

Brandon Widmer
Pratt & Miller Engineering

MAY 2012
Final Report

AIR FORCE RESEARCH LABORATORY
MATERIALS AND MANUFACTURING DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7750
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the USAF 88th Air Base Wing (88 ABW) Public Affairs Office (PAO) and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RX-WP-TR-2012-0344 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

_____//SIGNED//_____	**_____//SIGNED//_____**
STEVEN A. TUREK, Program Manager	SCOTT M. PEARL, Branch Chief
AFRL/RXMS	AFRL/RXMS

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
May 2012	Final	20 May 2011 – 31 May 2012

4. TITLE AND SUBTITLE
iFAB SMART MANUFACTURING ADAPTING RAPIDLY TO PRODUCT VARIANTS (SMARTV)

5a. CONTRACT NUMBER
FA8650-11-C-7128

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)
David Bourne (Carnegie Mellon University)
S.K. Gupta (University of Maryland)
Kazu Saitou (University of Michigan)
Prasanta Bose (Lokheed Martin Space Systems)
Brandon Widmer (Pratt & Miller Engineering)

5d. PROJECT NUMBER
3000

5e. TASK NUMBER
00

5f. WORK UNIT NUMBER
M0129601

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Robotics Institute/Carnegie Mellon University
5000 Forbes Avenue, Carnegie Mellon University
Pittsburgh, PA 15218

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory
Materials and Manufacturing Directorate
Wright-Patterson Air Force Base, OH 45433-7750
Air Force Materiel Command
United States Air Force

Defense Advanced Research Projects Agency/ (DARPA)
3701 N. Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY ACRONYM(S)
AFRL/RXMS

11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)
AFRL-RX-WP-TR-2012-0344

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES
The U.S. Government is joint author of this work and has the right to use, modify, reproduce, release, perform, display, or disclose the work. PA Case Number and clearance date: 88ABW-2012-3761, 5 July 2012. This document contains color.

14. ABSTRACT
This one-year project developed and demonstrated a SMARTV iFAB Foundry, a single node within a larger, future iFAB Foundry network. Our work embodied the key idea of evolving a smart technology that can integrate design objectives with manufacturing realities and adapt rapidly to diverse product challenges. The SMARTV project focused on welding and assembly operations to support rapid manufacturing of custom military vehicles under the larger iFAB program. Our goal was to fabricate an HMMWV spaceframe with assembled attachments at least five times (5x) faster than the current best practices allow.

15. SUBJECT TERMS
Instant Foundry Adaptive through Bits (iFAB), Foundry, welding, assembly

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT:	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON (Monitor)
a. REPORT	b. ABSTRACT	c. THIS PAGE			Steven A. Turek
Unclassified	Unclassified	Unclassified	SAR	104	19b. TELEPHONE NUMBER *(Include Area Code)* N/A

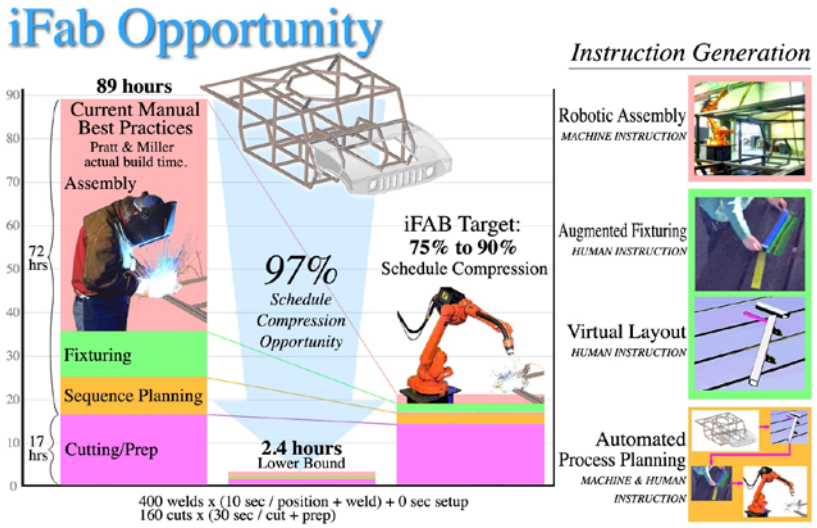
Contents

Executive Overview

This one-year project developed and demonstrated a SMARTV iFAB Foundry, a single node within a larger, future iFAB Foundry network. Our work embodied the key idea of evolving a smart technology that can integrate design objectives with manufacturing realities and adapt rapidly to diverse product challenges. The SMARTV project focused on welding and assembly operations to support rapid manufacturing of custom military vehicles under the larger iFAB program. Our goal was to fabricate an HMMWV spaceframe with assembled attachments at least five times (5x) faster than the current best practices allow.

Opportunity

Early in this project, the CMU team determined that many manufacturing processes require considerable setup time and offer a large potential for schedule compression. For example, Pratt & Miller constructed a spaceframe with best-practice methods, which took 89 hours — cutting square tubes, preparing them for welding, and then performing the final welding tasks to build the structure. On analysis, we discovered that the time actually spent on constructive processes was only 3% (slightly over two hours) of that total. Thus 97% of the overall time can potentially be eliminated.



Technical Approach

FANG Challenge designs for the iFAB Foundry will depend heavily on several key processes, each of which, like welding, is characterized by a "lower bound" process time. These processes include: assembly, machining, sheetmetal bending, injection molding, wire-harness layout, composite layups and several others. Each process entails considerable setup time, which in large-batch manufacturing can be ignored, since setup (and tooling) costs can be distributed over 1000's, 10,000's or even 100,000 sized lots. But in small-batch (i.e. custom) manufacturing, setup time dominates production time. In this case, SMARTV approaches and tools can compress the production schedule by partially automating setups and using machines and robots together to optimize manufacturing preparations. These concepts include:

- Automated process-planning for welding and assembly tasks
- Rapidly determining whether a task should be done by man or by machine, and then automatically assigning the task, generating instructions, and executing operations
- Employing virtual and augmented reality tools in human-assigned tasks to support operators with effective instructions that rapidly step them through the process while exploiting human flexibility to perform selected operations.

By radically cutting production time at the lowest process level, we can potentially gain stunning time compressions. But iFAB goals are vertical as well, meaning that all component processes — from part procurement, tool procurement, process setup, to part production, product assembly, part inventory, part inspection, and assembly inspection — must be considered to compress the total design-to-manufacturing process. In fact, many of these vertical tasks have potential for huge time compressions as well.

The SMARTV Foundry demonstrates such time compression for welding and assembly operations. SMARTV concepts dramatically reduce delivery time and, simultaneously, end-product costs by roughly the same amount, since labor costs dominate the cost structure.

Core Toolchain

In the SMARTV project we developed a core toolchain that can be applied to several different iFAB nodes. Each of our subcontractors participated in critical steps of building the toolchain.

1. Generate EBOM from design package (University of Michigan)
2. Match processes to parts (Lockheed Martin)
3. Determine optimal layout of welded components in stock supplies (Carnegie Mellon)
4. Order adequate stock to complete design fabrication (Carnegie Mellon)
5. Generate a partial order sequence based on tolerance requirements (University of Michigan)
6. Decompose design into flat subassemblies (University of Michigan)
7. Generate a total order sequence based on minimizing the number of setups (Carnegie Mellon)
8. Generate human instructions for welding operations (University of Maryland)
9. Generate augmented-reality instructions to guide welding operator during setup operations (Carnegie Mellon)
10. Sense final position of components to be welded and adjust robot programs accordingly (Carnegie Mellon)
11. Generate and execute welding-machine instructions (Carnegie Mellon)
12. Generate human instructions for assembly operations (University of Maryland)
13. Automatically plan mechanical-assembly sequence (University of Maryland)
14. Exercise SMARTV toolchain to fabricate a HMMWV spaceframe and hood assembly (Carnegie Mellon).

This toolchain describes the progression of plans and information for moving a welded product from design through to production. Mixed into the toolchain is a plan for mechanical assembly, which can be done as one logical flow or two separate nodes. In addition, we developed two

additional iFAB nodes for waterjet cutting (University of Maryland) and plasma cutting (Carnegie Mellon). We utilized both nodes to generate component parts that were later attached to the welded spaceframe assembly.

Lessons Learned

In the process of building the SMARTV iFAB Foundry, we gained several insights that should carry over to other iFAB manufacturing projects:

- *Matching parts to process* — Software that can accurately estimate part costs can be used for other critical applications in the manufacturing toolchain. For example, if you can query how much it would cost to produce a part using several manufacturing methods, the best choice may be to choose the cheapest process. *A Priori* ™ software is designed to cost parts and is effective at choosing the manufacturing process as well, as we have demonstrated in several applications.
- *Minimizing setup costs* — To speed the manufacturing of complex, near-custom products, automating the setup task, at least partially, is essential. The time to configure a part for a process dominates the production time, yet the industry has made little effort to improve this part of the process. Industry still has the mindset of mass manufacturing, where setup times don't matter. We have addressed this challenge and demonstrated an application of augmented reality in which a robot assists a human operator in a complex welding task with significant time savings (approaching 90%).
- *Achieving tolerance specifications* — Designs must carry enough information to transmit the designer's intent effectively. Consider, in particular, "key characteristics" representing a particular type of assembly tolerance that determines whether final the product will function as designed. Such features might include dimensional constraints, such as hinge-block colinearity, where all hinge axes must lie close to a theoretical straight line. Similar tolerance descriptions are critical when it is difficult to achieve component tolerances in complex assemblies (e.g. our spaceframe example). We have seen how to achieve the key characteristic tolerances even where individual component tolerances stack up beyond an acceptable limit. The solution is to ensure built in design flexibility so that assembly positions can absorb imprecision elsewhere.
- *Avoiding rework with robust designs and plans* — Hitting our eight-hour target in the spaceframe task resulted from everything going as planned. This situation emphasizes that plans must be highly robust to small errors and is also a primary reason to have design feedback introduce "slip planes" — assembly positions that can be adjusted to compensate for components on the "edge of a tolerance specification." While we can potentially save a great deal of manufacturing time, any saved time can easily vanish with even one manufacturing mistake. Effective SMARTV manufacturing designs and assembly plans must anticipate potential errors and incorporate appropriate risk-mitigation strategies.
- *Minimizing operator error* — Vast amounts of time are lost once mistakes occur, and automated assistance can help avoid them. For instance, using augmented reality, we have shown how to support a human operator in selecting the correct part at the correct time. Identifying components with machine-generated paper labels and projecting part numbers to target locations on the welding platform allows the operator to confirm part selection at assembly time.

- *Giving ongoing design feedback* — As each manufacturing process is systematized into a logical data flow, design elements that cause process problems emerge continuously, and the conventional concept of analyzing a "complete" design once to provide critical feedback no longer suffices. Until a part is operating in a functional product, the design should be considered a work in progress, informed by incremental discovery on the Foundry floor.
- *Increasing worker satisfaction* — We were pleasantly surprised to find that worker satisfaction is greatly enhanced in human-robot teams because the "teamwork" prevents many human mistakes, while allowing complex assembly projects to proceed smoothly.

We believe the principles that have been learned in this project can be applied in many other processes where there is high degree of customization and complex process setups are required.

1. Task Structure and Achievements

Under this SMARTV iFAB program, the Carnegie Mellon-led team of five collaborating organizations has successfully shown that, by focusing on setup processes, we can gain significant schedule compression in producing large, complex electromechanical systems. Our test case was an HMMWV welded spaceframe (Figure CMU-1). Using best available practice, the task required 89 hours. Replicating the effort with innovative digital tools and a tightly collaborating robot / human team took only eight hours. This final report describes those tools, how they work together, and the newly-demonstrated manufacturing processes.

As we pursued this task over the last year, we (along with DARPA) recognized that the overall objective of building custom military vehicles would necessarily involve purchasing almost 80% of the component parts. This observation highlighted the critical role of assembly processes (including welding) in achieving project success. Therefore, we added tasks that partially automate assembly-sequence planning for nut-and-bolt assemblies and tested these algorithms by extending the spaceframe to support a vehicle hood that requires mechanical assembly.

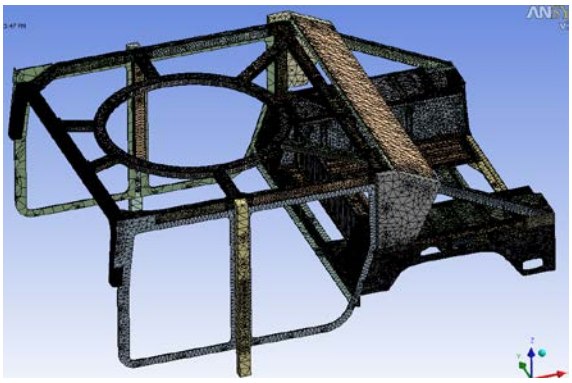


Figure CMU-1: Typical spaceframe structure

Hitting our eight-hour target in this task resulted from everything going as planned. This situation emphasizes that plans must be highly robust to small errors and is also a primary reason to have design feedback introduce “slip planes” — assembly positions that can be adjusted to compensate for components on the “edge of a tolerance specification.” While we can potentially save a great deal of manufacturing time, this saved time can be easily lost with even one manufacturing mistake. Effective SMARTV manufacturing designs and plans, therefore, must identify areas where errors are likely and must incorporate appropriate risk-mitigation strategies.

In addition to providing our own example, we participated significantly in DARPA’s group exercises, which focused mainly on mechanical assemblies such as those of Figure CMU-2.

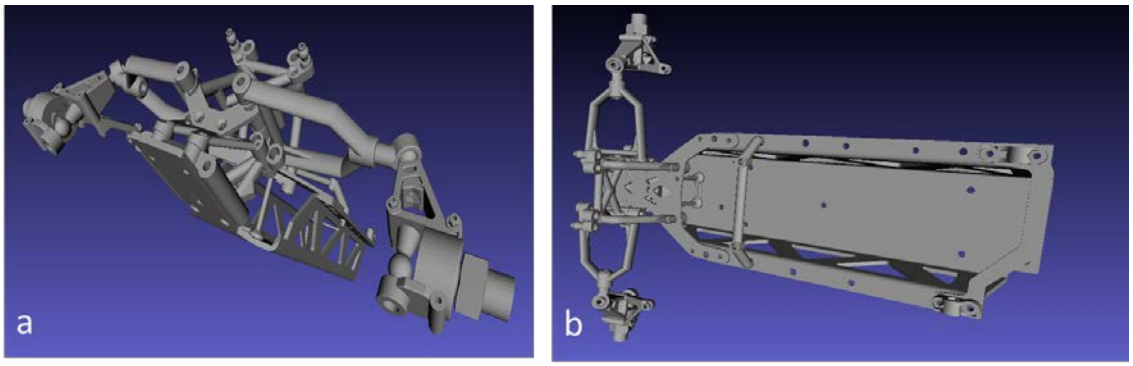


Figure CMU-2: Mechanical assembly design, supplied by Vanderbilt University under AVM's META program. This particular assembly represents 79 parts from a model RC car.

1.1 Versatile Software Tools

Our software tool designs thus addressed these two product classes (welded and mechanical assemblies), and we developed a toolchain, Figure CMU-3, that enables a manufacturer to process both product types.

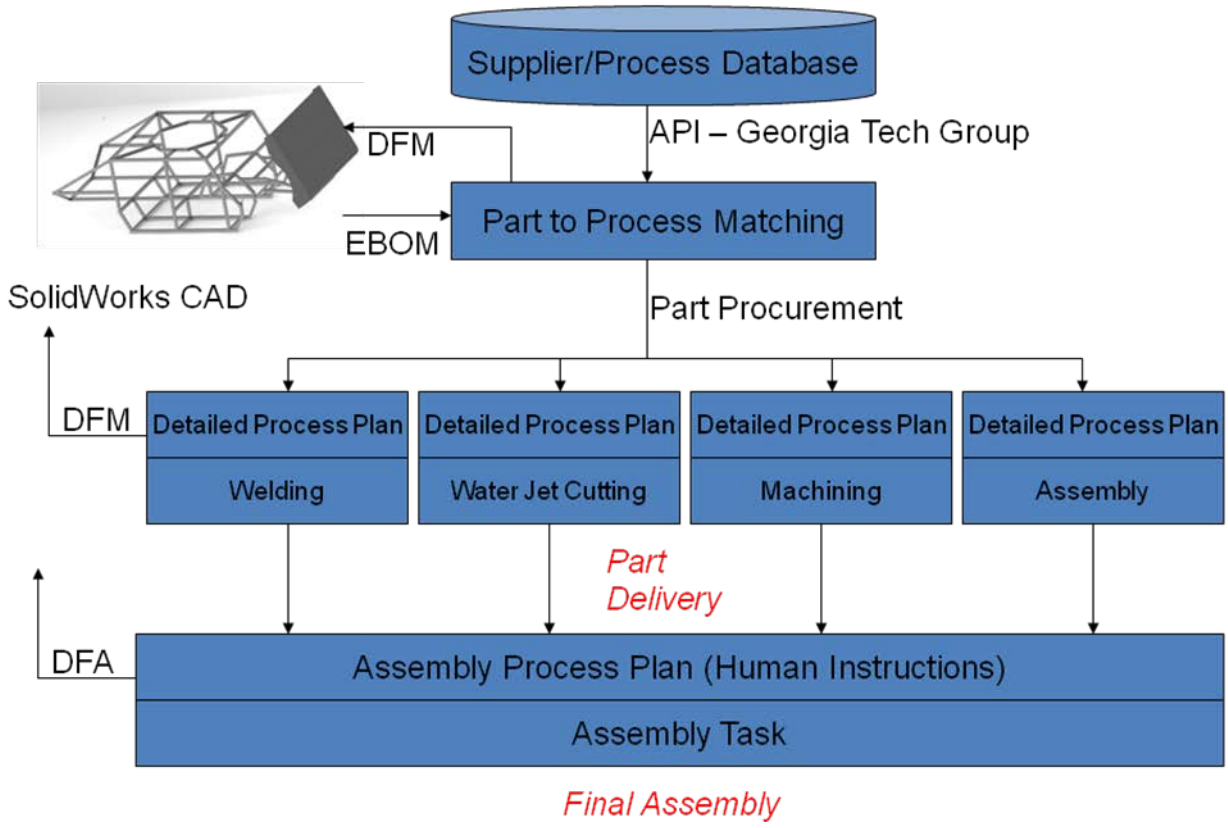


Figure CMU-3: Toolchain for mixed-process fabrication and assembly

While the toolchain's overall "shape" remained essentially stable throughout our research, each of the component tools evolved noticeably, with even the internal boxes changing:

1. **CAD Model/Design** (Pratt & Miller) **–** This team has continued to use SolidWorks because it is relatively inexpensive compared to Pro/ENGINEER and CATIA. The DARPA exercises, however, were developed at Vanderbilt University, which used those latter, commercial systems. Therefore, we used STEP as a transfer format between modeling systems and, in some cases, STL files to compute assembly-sequence plans and environment simulations.
2. **EBOM Extraction** (University of Michigan) **–** This team has developed a program that uses the SolidWorks API to extract components. These components are then listed in the bill of materials, along with simple computed and given attributes (bounding box, component tolerances and materials).
3. **Process Database** (external to our group) **–** We considered using the Process Library that Georgia Tech is developing under the iFAB program but found, unfortunately, that it is not yet ready for integration into a global toolchain. This resource is evolving rapidly, however, and may soon become suitable. Meanwhile, to move ahead in our own work, we employed a commercial product (a Priori) to develop Part-Process matching algorithms.
4. **Part and Process Matching** (Lockheed Martin**) –** Based on reasonable guesses at workable manufacturing processes, this team compute expected manufacturing costs. Based, in turn, on these estimates, we anticipate that the minimum cost provides the best process match.
5. **Process Planning for Machine Instructions** – In the course of this work, we have developed process-planning capabilities for three distinct processes:
   a. **Waterjet Cutting** (University of Maryland) **–** This team developed a process planner for waterjet cutting and delivered it to a job shop with significant water jet assets. The shop then employed the planner to plan and produce parts for our program automatically. Time from receipt of design to manufacturing was about an hour.
   b. **Welding** (Carnegie Mellon) **–** This team developed process planning and process execution plans for complex welding tasks (e.g., the spaceframe). This capability includes partially automated setups with augmented reality and setup inspection before welding commences. Other groups have also contributed to various aspects of the planning process.
   c. **Mechanical Assembly** (University of Maryland) **–** This team developed a complex planner to commute assembly sequences. Other groups have also contributed to various aspects of the planning process.
6. **Process Planning to Create Human Instructions** (University of Maryland) – This group developed a software environment to generate human instructions for welding and mechanical assembly tasks. This work builds on Tundra, an open-source game development system that also being used by Boeing, another iFAB performer.

1.2 iFAB SMARTV Foundry Test Case

To test the tools and concepts developed in this project, we formulated a test problem based on Pratt & Miller's spaceframe. The test procedure included replicating the structure using SMARTV concepts and techniques. In addition, we added key elements to challenge other processes, including waterjet cutting, machining, and assembly.

The first steps extracted part information from design files and ordered off-the-shelf parts or new parts fabricated by a SMARTV vendor. These vendors were, in fact, other iFAB performers (University of Delaware) and Cardinal Scientific (running UMD's Waterjet Planning Software).

The next steps decomposed the problem into tasks that could be easily accomplished on Carnegie Mellon's welding workstation and other tasks (with automatically-generated instructions) assigned to human partners. This decomposition began by finding the optimal "flats" hidden within the spaceframe structure, Figure CMU-4. These subassemblies incorporate numerous joints amenable to robotic welding. Human welders then completed the more difficult welds. Note that our goal is not to achieve 100% automation, but rather to make substantial improvements in time-to-product.

Each flat pattern comprises a set of tubular members. We then constructed a workplan to produce these kits from standard eight-foot stock parts and developed a collection of simple tools to speed this process:

- Optimize members within the stock components to avoid unnecessary waste
- Rough cut members to approximate size
- Print labels that identify component names and attach them to rough cut pieces
- Make precise cuts using robot plasma cutter and/or cutoff saw
- Stack members in kits for each flat subassembly.

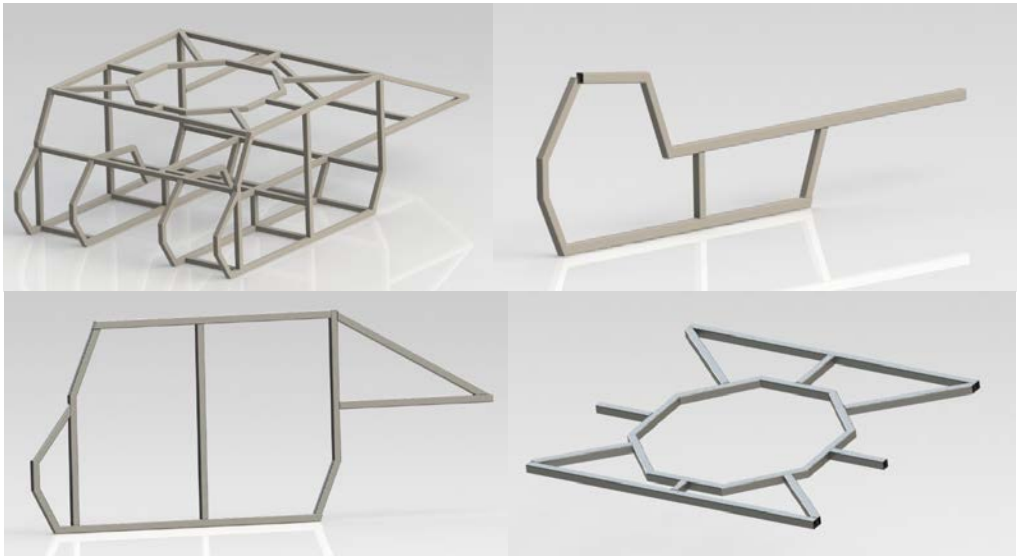


Figure CMU-4: Initial spaceframe design and three derived flat patterns

The procedure used augmented reality to guide human operators through setup for each flat and virtual-reality instructions to give the operator precise, step-by-step guidance. The next step automatically scanned the setup and developed machine instructions for welding. After robotically welding components on one side, the workstation system asked the operator to flip the subassembly so that the robot could weld the other side.

The system then directed the operator to assemble completed flats in a four-post fixture that facilitates 3D alignment. From this point, the human welder took over to complete the 3D

connecting joints. While some work will almost always remain for a human welder — because of accessibility constraints, the robot cannot reach certain positions — as these tools mature, the robot will be able to continue on with many of these operations.

After completing the spaceframe, we added several attachments, including a hood-support system (another welded structure), a composite hood, hinges, and various brackets. These parts were attached with nuts and bolts.

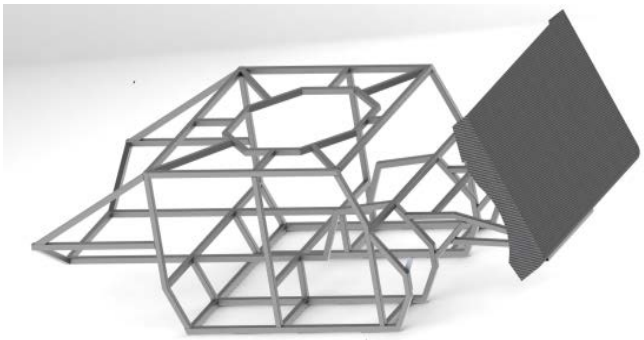
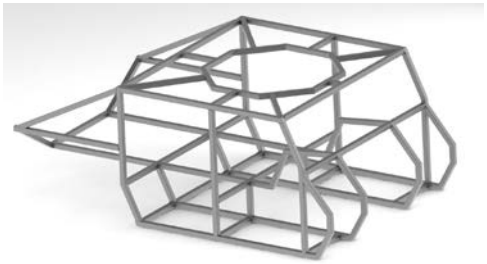


Figure CMU-5: Completed spaceframe assembly

This final assembly phase offered an opportunity to test our assembly planner and develop an assembly-sequence plan. The following table presents the parts list, which other systems, including our assembly sequencer, can use.

We conducted the test build during the final two contract months. This build cycle integrated multiple team tools, and we have documented the work in a video recording.

Part	Quantity	Description
	1	Welded HMMWV frame Process: MIG Welding, mixed human and robot
	1	Welded hood support structure Process: MIG Welding, mixed human and robot

	1	Composite HMMWV hood CO: University of Delaware

1.3 iFAB SMARTV / AVM Group Interactions

Throughout this project, SMARTV teams have collaborated with several other iFAB (and AVM) performers. Experience gained through these early interactions will simplify the challenges confronting future iFAB Foundry performers and are therefore critical to overall program success.

1.3.1 META / Vanderbilt University

Through this interaction we have investigated the impact of assembly tolerances. Beyond individual part specifications, other constraints may indicate parallelism, dimension, or other features defined over an entire assembly. To ensure satisfying overall design objectives, it is often necessary to identify an assembly's critical characteristics, features that allow a product to function properly.

1.3.2 iFAB / Boeing

Boeing has expressed interest in using our "motion planning for assembly" work to test the interaction with their human models. For example, can a human hand (holding a drill) fit into the available assembly space? This study will eventually result in an API to SMARTV technologies, so that our motion-planning can incorporate considerations of human operator geometry.

1.3.3 iFAB / Georgia Tech

Georgia Tech is building a process library (knowledge tree) that affects our research in two ways:

- Our part-to-process matcher uses the library as the source of process data. Since we must be able to query the library, we have helped, indirectly, to define their API.
- Our work is effectively building the iFAB Welding Node, so the process database should capture our process functionality.

1.3.4 iFAB / University of Delaware

We have used the University Delaware teams as a source of composite structures, such as the HMMWV hood. In addition, we have begun exploring with them how to formalize a bid-and-purchase relationship that models the marketplace that will operate under the iFAB umbrella.

1.3.5 iFAB / PARC

We have worked with PARC to clarify manufacturability constraints for waterjet cutting processes. This relationship will likely evolve as the PARC teams starts to cover a wider range of processes. Perhaps the GT Team might also indicate whether rapid constraint analysis is available from PARC for their processes.

2. iFAB SMARTV Welding Workstation

2.1 Task: CMU.1 & PME.1

Responsible organizations:

- Primary — Carnegie Mellon University, Pratt & Miller Engineering.
- Secondary — all subcontractors.

This project's main objective was to demonstrate an iFAB SMARTV Foundry node that concentrates on welding and assembly tasks, while adding some machining, water-jet cutting, FDM, and sheetmetal bending tasks. The larger Foundry will evolve and mature as time progresses, a characteristic also intrinsic to our Workstation's architecture: It is a dynamic "living" structure, distributed in design, that can easily accommodate new technologies and additional processes.

Our approach was to plan and integrate successive research results into a working system. The idea is simple: Analyze and exploit the relevant technologies to achieve a radically compressed delivery schedule. To exercise the Workstation, we chose to build an armor cage for the High Mobility Multipurpose Work Vehicle (HMMWV). Carnegie Mellon acted as the lead integrator, while Pratt & Miller served as the lead user. Pratt & Miller's experience manufacturing similar products with existing state-of-the-art technology has provided a baseline against which we have objectively assessed SMARTV advantages.

2.2 Hardware Setup

During the first few project months we designed, purchased, installed, and modified a robotic welding station, now dominated by an ABB IRB-2600 robot with appropriate welding configuration. Located in CMU's highbay workspace, the robot is shown here with the spaceframe that Pratt & Miller fabricated at their Michigan plant. Notice that the robot is mounted on a platform that elevates its arm relative to the work. This configuration dramatically enlarges the robot's working volume, so that it can reach more welding points.

Our next step was to develop a general-purpose fixture that can localize both the spaceframe and a general-purpose welding table. In fact the idea allows us to swap in and out the large spaceframe with the welding table. The welding table is designed with a large working surface that simplifies welding flat assemblies

. 

Figure CMU-6: Robotic welding workstation

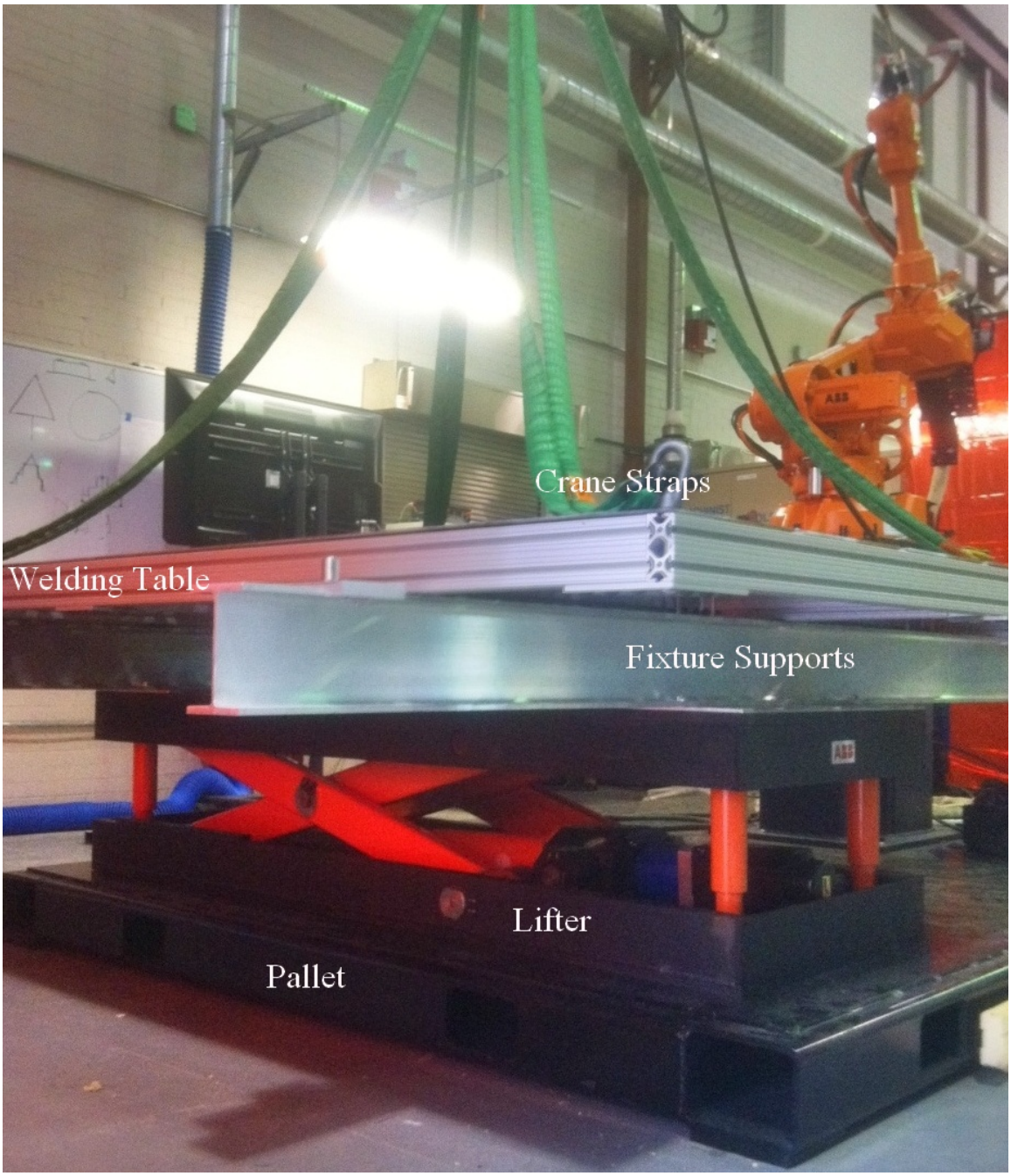


Figure CMU-7: iFAB robotic welding platform

Figure CMU-7 illustrates the welding platform's configuration. At the floor, the platform rests on a pallet (a), which mounts all robotic components and controllers. Atop the pallet sits a robotic lifter (b), effectively providing a seventh axis for the robot. In other words, the robot can literally track a part up and down as the lifter moves it. The lifter's purpose is, again, to increase the robot's working volume so that it can reach both the bottom and top of the work area (in this case, the HMMWV spaceframe). The fixture supports (c) allow a standard connection interface between various work environments. In our case, these work environments include both the full 3D spaceframe and a welding table (d) custom designed and built for this project. "Cup and pin"

fixtures attached to both table and spaceframe allow accurate placement of the two large, heavy structures, which we can move in and out using an overhead crane.

2.3 Robot Software

The next step in building the welding workstation was to modify the robot's control software. Internal to the robot we have implemented a "robot server" that runs in an infinite loop, reading commands from other external PCs and executing their intent. In this way, the commands can be constructed from offboard computers using complicated algorithms and a variety of programming languages.

```
While robot is running
        Get next command from Ethernet Port
        Case command
                Cmd1:  execute command 1;
                Cmd2:  execute command 2;
                …
        End Case;
End While;
```

Command Loop Running in Robot (written in ABB's RAPID programming language)

Another advantage of this design is that the program actually running on the robot is constant and well understood. Of course, it has become slightly more complex over time, as we slowly add features. For example, the welding commands require a large amount of data, so we have added logic to buffer this type of command and store it away while the robot prepares for the final welding move. Even with these improvements, the program's basic structure remains transparent.

2.4 Deliverables

- The Welding Workstation
- Overview video of the SMARTV Foundry toolchain
- Component videos showing system-capability demonstrations
- Source code (ABB's Rapid) for the algorithms (robot server)

3. Mixed-initiative Setups – Robots and Human Operators

3.1 Task: CMU.2

Responsible organizations:

- Primary — Carnegie Mellon University
- Secondary — University of Michigan and University of Maryland

A major bottleneck in programming robots for mechanical assemblies (e.g., small volume vehicle production) is setting up the environment for automation. Virtually every other manufacturing task, where various tools and parts must be configured before production can proceed, face exactly the same problem.

To address this challenge, we augmented the welding robot with technology to help the human operator rapidly set up various welding tasks. The key addition was a graphics projector, for which we selected a Microvision ShowWX+ laser projector. Several considerations guided our choice:

- The ShowWX+ incorporates the ultraminiature PicoP display engine in a very small unit, easily mounted on the robot arm without impeding (excessively) robot motion
- The projector employs three (RGB) scanning lasers convolved into a single beam. This configuration allows the projector to scan over its display area at 60 HZ and further requires no focus adjustment. Since the arm is constantly moving, this no-focus feature is essential for the welding-station configuration.

In addition to the ShowWX+ projector, we also installed a highly accurate laser-displacement sensor that can determine precisely where assembly member(s) have actually been placed. With this capability, the robot can compensate for human inaccuracy and the human operator can set up a welding task simply by placing members in approximately the planned positions.

We mounted both projector and laser-sensor in protective enclosures to avoid weld-spatter on these delicate instruments. Small doors in front of each unit can be opened and closed under robot control. These two protective enclosures have been “boxed” in Figure CMU-8 to show their position relative to the welding torch.

Figure CMU-8: Projector and sensor mounted on robot welding arm

3.2 Calibrating the Projector

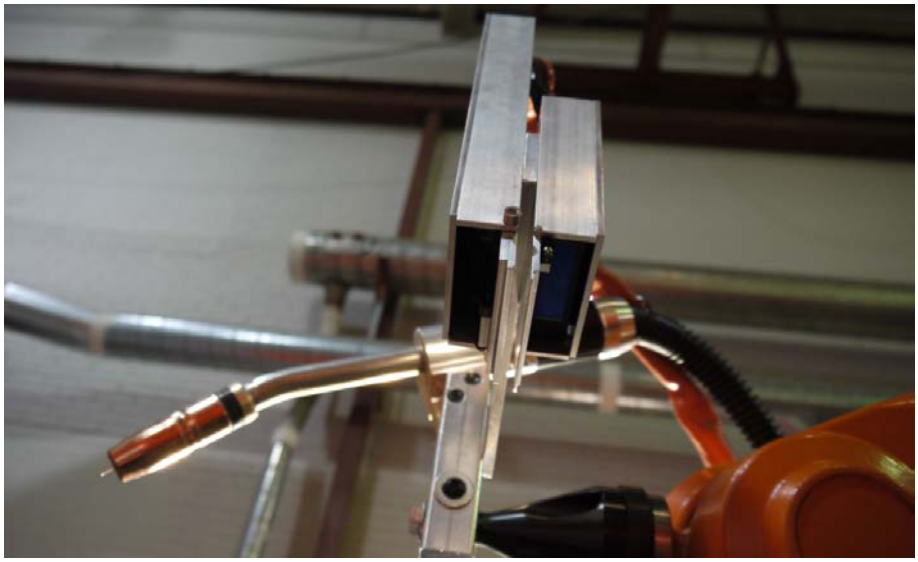


Figure CMU-9: Robot Tooling –Projector (larger box) and displacement sensor (small box) lasers, mounted at an angle to the welding torch and off to the side

For both projector and laser-sensor, we establish a "tool object" — the XYZ offset and orientation, as measured from the robot's wrist, of a point-observation "tool" — in a multistep process of our design:

- First, establish three "reference dots" on the table, which points lie at known positions in the robot's coordinate frame. One center point and two others establish the X and Y axis of the robot's coordinate frame. Thereafter, we can use the arm-mounted projector to calculate its pointing direction.
- Point the projector roughly downward and project crosshairs onto the table. Mark that position on the table, and then move the robot vertically, keeping the orientation and XY values constant. Since the projector is not likely pointed straight down at this point, the crosshairs appear to move on the table. Measure the XY crosshair movement on the table and from that data calculate the projector's vertical deviation angle. Then rotate the arm, by the negative of the calculated angle, around a vector calculated from the XY table coordinates.
- On completing the calibration procedure, we can project crosshairs, move the robot vertically, and see no apparent movement of the crosshairs. Generally, we are able to determine arm orientation to roughly one millimeter over a 1400mm distance, or 0.04 degrees.

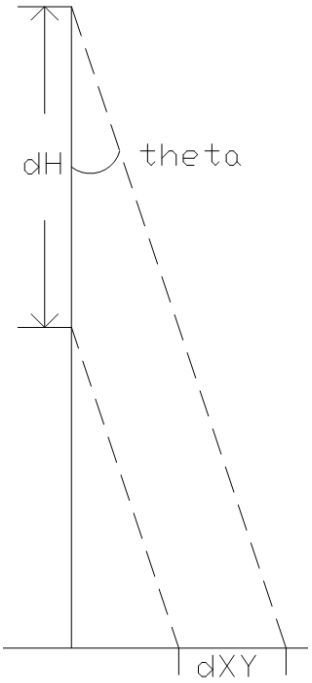


Figure CMU-10: Establishing the tool object

After establishing the projector's orientation, we need to find its XYZ offset. We do this by pointing it straight down, then jogging the robot so that the crosshairs line up with the reference dot on the table. We calculate XY offset based on reported robot position versus the reference dot position.

Calculating projector Z-offset is slightly more difficult. Our current method involves projecting a checkerboard pattern onto the table, and moving the arm from a low height to an elevated position in steps, and recording checkerboard corners at each height. We can observe the apparent movement of the checkerboard corners, and from those observations calculate distance from table to projector. The data is also used to calculate ray to pixel mapping for our augmented reality system.

Future work is automating this procedure, so that all measurement is done via a computer vision system, instead of a ruler. Running this calibration as a routine with no human input is quite feasible and will result in significant time savings.

Figure CMU-11: Calibrating the projector's Z-axis

3.3 Calibrating the Laser Sensor

To establish the laser-sensor's tool object, we follow the projector-calibration procedure up to the final step, establishing the tool's Z-offset, where we replace the checkerboard procedure and simply jog the robot straight down, slowly, until we reach the sensor's range. We then use that Z value to determine the sensor's zero position.

3.3.1 Laser Sensor

The laser-sensor system measures displacement, distance, and position with a laser unit (Micro-Epsilon 1402-200) mounted on the robot arm. The laser-sensor bounces a visible-light beam off the target surface (diffuse or specular), and returns the distance to the first point of contact. Sensor error, at a distance of 160mm, is less than 0.288 mm. To locate a point in space, we combine the measured distance with the robot's position and orientation at the time the distance measurement was taken. Timing is achieved via a synchronization signal and, sampling at roughly 30Hz, we can quickly generate a point cloud (Figure CMU-12).

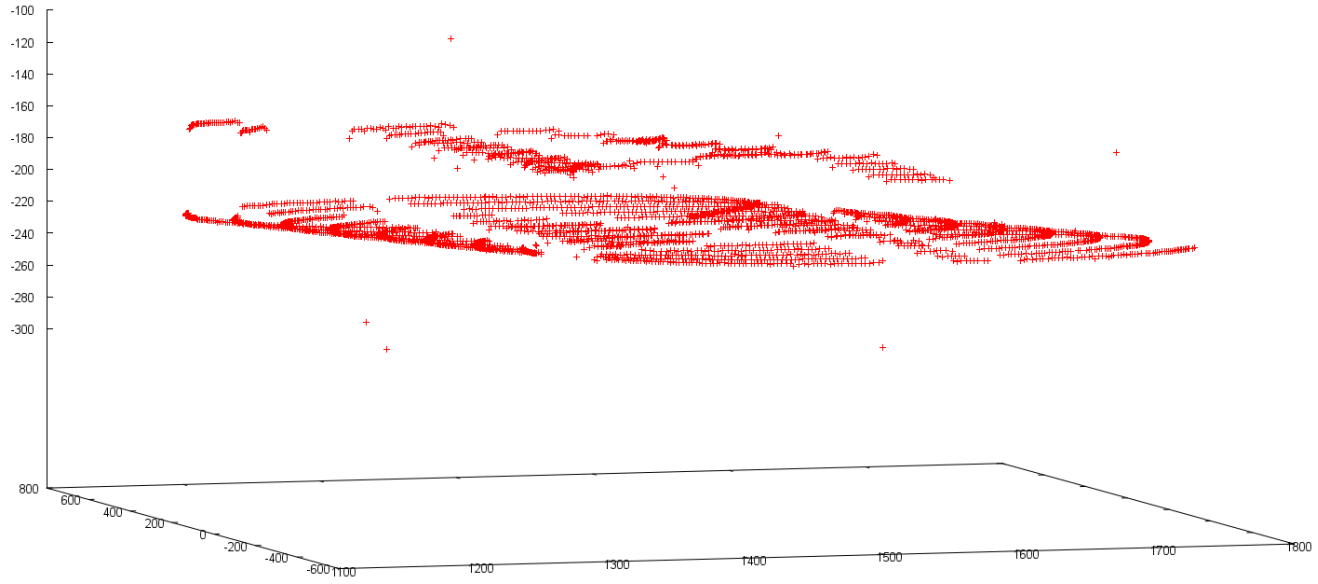


Figure CMU-12: Raw 3D data from a workpiece scan, obtained from a moving spiral inspection strategy and showing curved traces of table and workpieces resting upon it.

3.3.2 Laser-Sensor Accuracy

One of our primary concerns with the laser-sensor system was ensuring sufficient precision. The initial goal was to determine weld position to within one millimeter, which objective should require similar accuracy for individual point observations.

Several sources contribute measurement error (see Figure CMU-13), the major factors being:

- Robot position uncertainty (~0.35mm)
- Robot orientation uncertainty (~0.06 degrees)
- Displacement measurement uncertainty (~0.3mm).

The possible error region is roughly rectangular, with a worst-case dimension of roughly 1 x 1.5 x 1.5mm. Experimental results with gauge surfaces have indicated a region more on the order of 0.5x0.5x0.5mm.

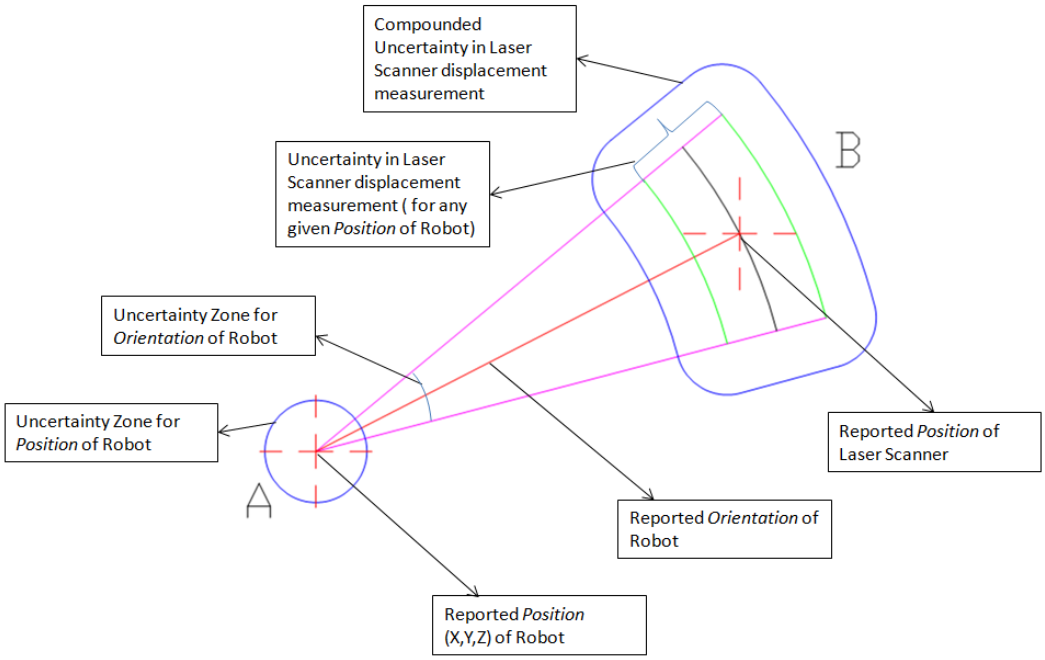


Figure CMU-13: Calculating the sensor's true accuracy involves stacking elements in the kinematic chain.

3.4 Identifying Weld-seam Positions

This effort is focused on measuring the positions of assembly members and weld seams in order to generate welding-machine instructions automatically.

Figure CMU-14: Hood-support subassembly

For example, in fabricating this five-piece subassembly for the HMMWV hood support system (Figure CMU-14), pre-welding inspection has two goals:

- Detect that the members are all correct, match the CAD model, and are positioned correctly relative to each other.
- Automatically determine weld-seam locations, which define the weld path and therefore the machine instructions.

To accomplish this goal we use the arm-mounted laser-sensor to "scribble" over the subassembly's bounding area and collect a series of points (distance from sensor to first point of contact). Analyzing this data sequence, we detect the edge crossings that mark arbitrary edge points on the weld members. In Figure CMU-15, this raw data is marked as small red circles.

3.5 Workpiece Localization

The robot system described so far guides the worker through component placement in a fast and instructive manner but does not guarantee complete accuracy, since placement is still depended on human motor skills. In Figure CMU-15, the projected blue region is the target location for placing the steel member. Empirically testing repeatability in placing a single member, we observed a standard deviation of 0.76 mm. This value translates to an uncertainty of around five millimeters in each X and Y offset from the nominal position. Since most subassemblies involve several members, this additive uncertainty reaches a value that cannot be ignored for the purpose of welding, which requires a tolerance of not more than one millimeter for acceptable execution.

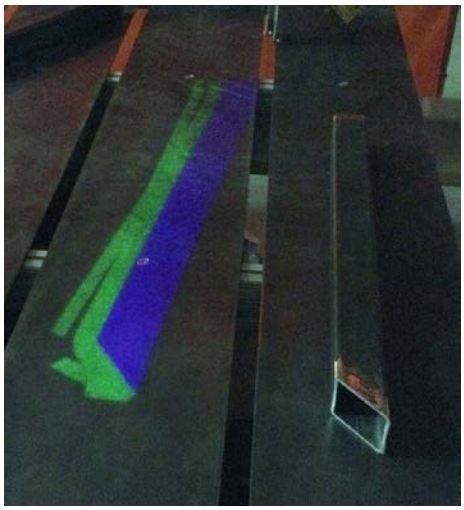


Figure CMU-15: Projected setup guidance

To accommodate worker-introduced placement variance, we developed a module that provides placement feedback. The system *observes* the component placement, *process*es the observed data, and *adapts* subsequent procedures to compensate for estimated changes. Based on misplacement or placement error (this decision is made by the worker after task placement) there are two diverse approaches to localize the task and thereby the weld seam – Global Inspection and Local Inspection.

3.5.1 Global Inspection

When misplacement is in the order of centimeters, the robot must inspect a large area around the target position. This procedure employs the laser sensor and a template-matching method, similar to those used in the field of computer vision, in three modules that inspect the layout, extract part edges, and optimize the template fit.

Inspection Plan

The inspection plan's objective is to hit as many edges as possible, spanning all members and covering all possible worker-introduced offsets. Several patterns, such as zigzags over the entire template, can be used for inspection. Since we want to avoid sudden direction/speed changes that would visibly jerk both robot and laser-sensor, we chose a spiral path, Figure CMU-16, to ensure smooth execution.

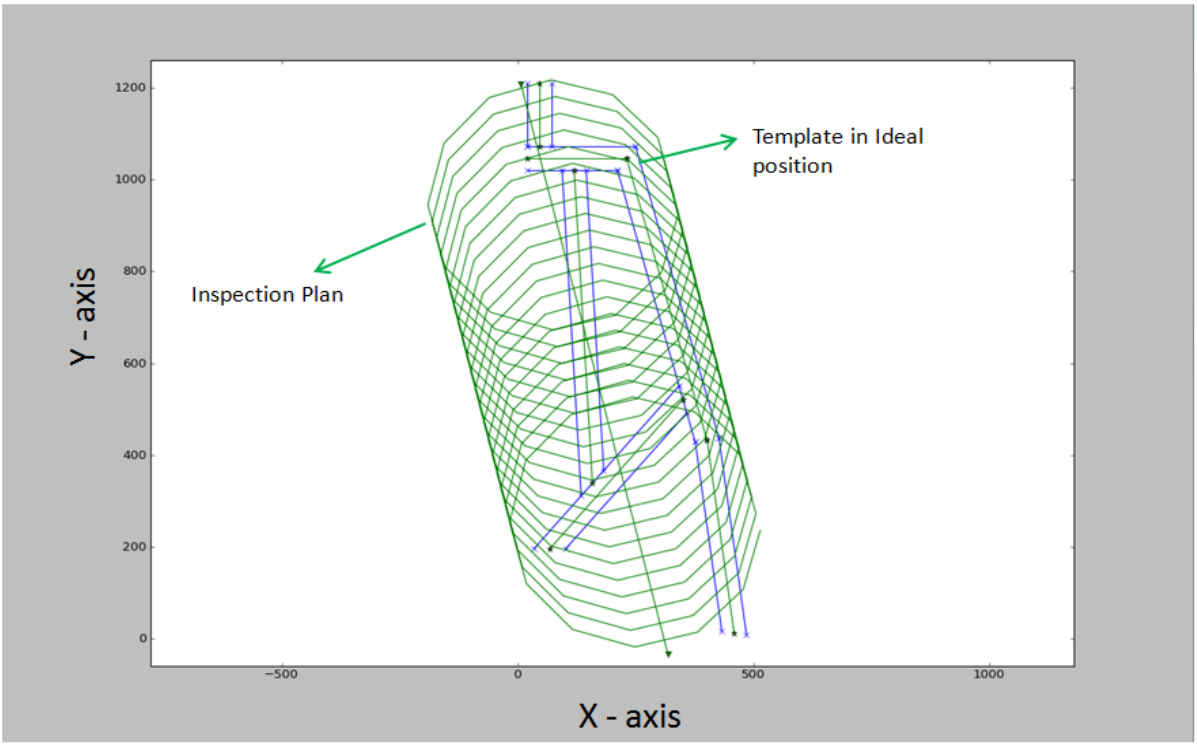


Figure CMU-16: Laser sensor's spiral inspection pattern

Extracting Edges

Laser-sensor data is synchronized with robot motion to obtain the Z-axis height during inspection, which measurements are then converted to 1-D data that represent Z values over time and ignore the X and Y dimensions. The objective is to extract the Z drop-offs at tubular member edges.

The extraction process uses two kernels:

- We first convolve the 1-D data with a step kernel of size 21 and use the actual height of the tubular frame (extracted from the CAD models) as the threshold to filter out laser-sensor noise. This convolution produces "candidates" that hold the actual edges we seek.
- A second step kernel of size 3 is convolved on the extracted edge candidates alone to improve edge localization. The threshold here again is the height of the tubular members, and the output of this filtration process represents the "edge-points" encountered during inspection.

Optimizing Template Fit

To obtain the actual position of the task on the workspace, we require a transformation that represents the offset and rotation of the template from its ideal position (representing the desired position of the task) to its actual position (representing the placement of the task by the worker).

The edge extraction process yields the data required to calculate this transformation, and the edges represent at least a sparse silhouette of the task, if not a continuous outline. The objective is then to rotate (Θ) and move the template (X, Y) from ideal position to cover the extracted edges as much as possible, which represents our objective function for optimization. The parameters to optimize are X, Y, and Θ representing the X-offset, Y-offset, and rotation of the template from its ideal position on the XY-plane.

- Objective Function

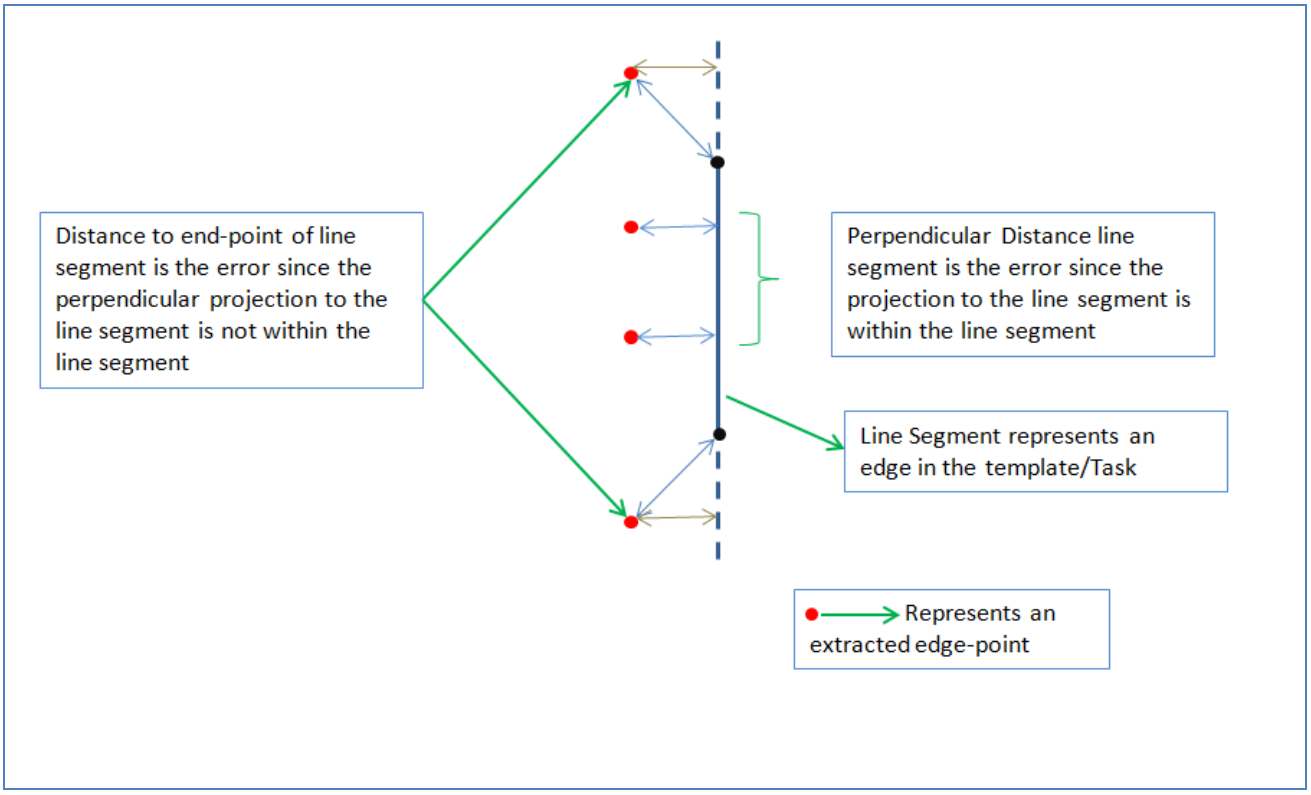


Figure CMU-17: Objective function for template matching

The objective function —error to minimize (through X, Y, Θ) — to get the best fit is designed as follows:

- Convert the task (from the STL data) to a 2D template representing the overhead view on the XY plane. The template in turn consists of individual line segments.
- Calculate the distances (not always perpendicular) between each of the member edge-points — obtained through edge extraction — and each template line segments (Figure CMU-17).
- For each edge point the Euclidean distance to the closest member represents its error.
- The total error for any position of the template is the sum of all edge-point errors.

To optimize these parameters, we use the Powell method, which finds the closest (local) minimum within the neighborhood, without calculating any gradients to do so. It simply takes a small step in the direction (bidirectional) of each of the parameters and proceeds along the one that gives the lowest error. Since we expect the actual position to be near (within a few centimeters of offset and limited rotation) the ideal position, we expect the global minimum to be near our starting point ([0,0,0] – X,Y, Θ).

- Parallel Optimization

To further improve our chances of finding the global minimum, while maintaining the same time to complete the optimization, we randomly pick eight different seed points for the algorithm. The seeds are generated within a neighborhood of 50 mm for the X,Y offsets and 45 degrees for rotation. Since our setup has eight separate processor cores, we can run all eight optimizations in parallel with different seed points. We pick the result that has the lowest error among all eight as the final result (Figure CMU-18).

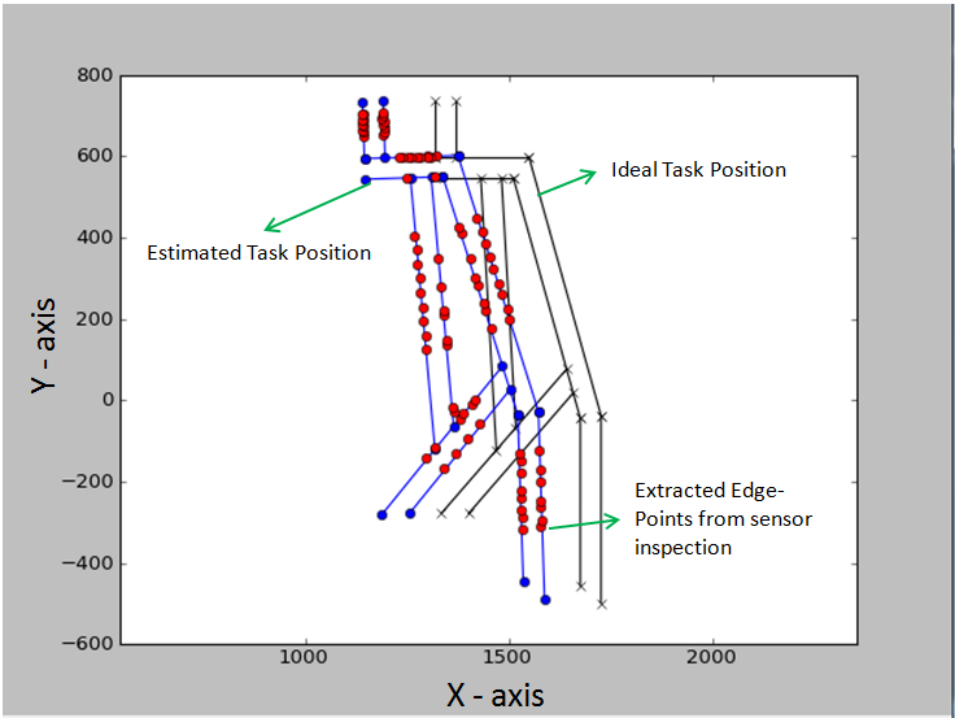


Figure CMU-18: Alignment of template and displacement-sensor data

The output of the optimization gives X,Y and Θ that represent the transformation to be applied to give the best estimate of the current (actual) position of the template. This data is converted to a 4x4 transformation matrix to be applied on the STL data and the weld positions within the iFAB files to give weld positions on the workspace.

In the experiments conducted so far we have seen that the template matching algorithm has been able to successfully identify the actual position of the task even during unreasonable placements of the members. This once again reinforces the requirement of a sane feedback system that can adapt to variations in placements.

One disadvantage is that there are a small number of cases that cannot be uniquely resolved in this manner. For example, when two straight pieces need to be welded, only one orientation and one translational parameter can be determined. This single counter-example suggests that some small additional data needs to be collected ***after*** the template matching to verify member positions. One good place to collect data is at the end of a member, which will generate good data and resolve this kind of data indeterminacy (locking in the 2nd translational axis) while further checking for layout errors.

- Extracting seams for welding:

  Once the template is matched to the raw data, the system can read weld-seams directly from the template description and then directly execute the welds by composing the template transform to the robot transform. This method has several advantages:

  - It can find multiple welds with a single scan, potentially saving considerable robot motion and time
  - The fit itself works better with more complicated patterns. The accuracy improves rather degrades with more data
  - It can be used to detect layout errors, so that costly mistakes can be detected before welding commences.

3.5.2 Local Inspection

When misplacement is smaller, in the order of a few millimeters, the actual weld seams are quite near their ideal locations, and a more narrowly focused inspection suffices. This procedure utilizes the displacement sensor to detect discontinuities along vertical faces of members forming the weld seam. The method yields unique offsets for each seam in the task, not a single configuration for the entire task, and each offset is independent of the others, unlike the situation where the global inspection applies.

Inspection Plan

Due to its limited scope, the plan for local inspection is far less intensive than that for the global case. Local inspection requires only a simple linear pass across the vertical member-faces at the junction that defines a weld seam (Figure CMU-19). Pass length is given as a parameter, and in experiments so far this value has been fixed at 3 mm. Though such an inspection path can be calculated for both end points of all welds, only one of each can be reached as the angular approach of the robot in its current configuration, with the laser scanner (oriented at 45 degrees along Z-axis) and mobile projector integrated to the end effector, is limited to within 0-180 degrees.

Seam Identification

Sensor data collected after each inspection path (for each seam) is expected to have an outline similar to that described by the member faces that define the weld seam. This data can be expected to be made up of two lines intersecting at the end point of the weld seam, each line belonging to the vertical face of each member defining the weld (Figure CMU-19). Assuming no variations in the seam length, the exact trace of the seam can be computed form the intersection point ([X,Y]) of the two lines and their angular bisector ([Θ]). These are the same parameters that are calculated in the template fit method described in the previous sections. These parameters constrain the weld seam under inspection.

This process is repeated for all inspectable welds. In instances where neither of the weld-seam endpoints can be reached, there is no information to localize it. In such cases, one solution is to observe the offsets of weld seams already inspected and, if they concur with one another, use those.

This approach boasts short processing times (2-3 seconds per weld) and can also accommodate distortions in the task (through its members) since computation is based on individual welds, independent of the shape of the task itself. Unlike the global inspection method, however, local inspection cannot flag abnormalities in task shape.

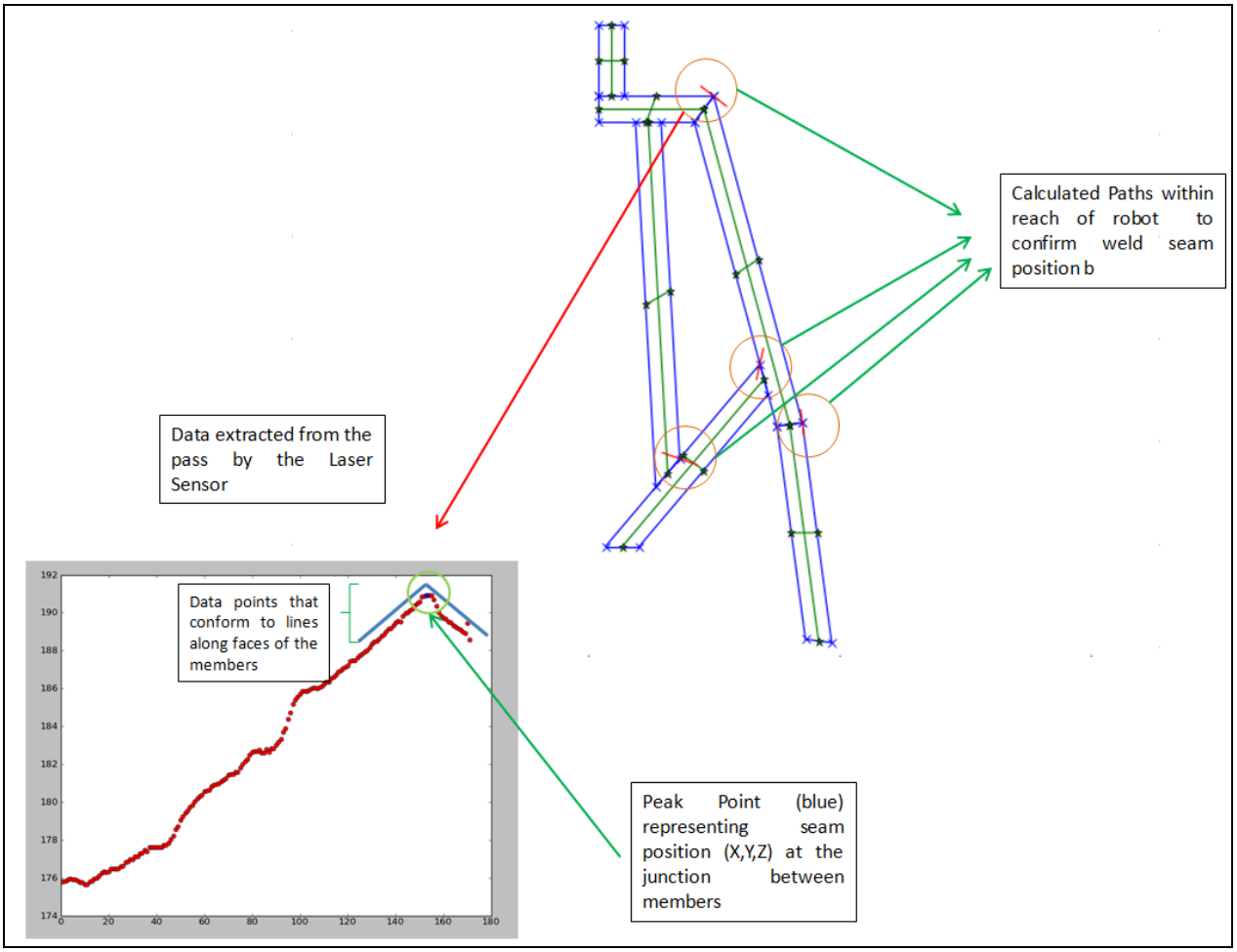


Figure CMU-19: Local inspection and seam identification

3.5.3 Fixturing

For positioning and holding workpieces on the welding table, the primary fixturing was small cylindrical magnets, chosen to provide easy manipulation and low profile while allowing the operator to slide a workpiece into target alignment. In operation, the robot projects a round position marker and crosshair lines, and the setup worker adjusts the magnets to lie centered under the mark. This novel technique exploits the projector's precision and human Vernier acuity — the ability to align pairs of lines and circles with high accuracy.

3.6 Discovering Workpiece Anomalies

Apart from capturing placement variance, our template matching algorithm can also be used to identify possible manufacturing errors/anomalies in individual members.

As one example, consider a cut angle that deviates significantly from design. The system should then avoid welding that member and discard it as defective. The template matching algorithm points out such members/cases (see Figure CMU-20).

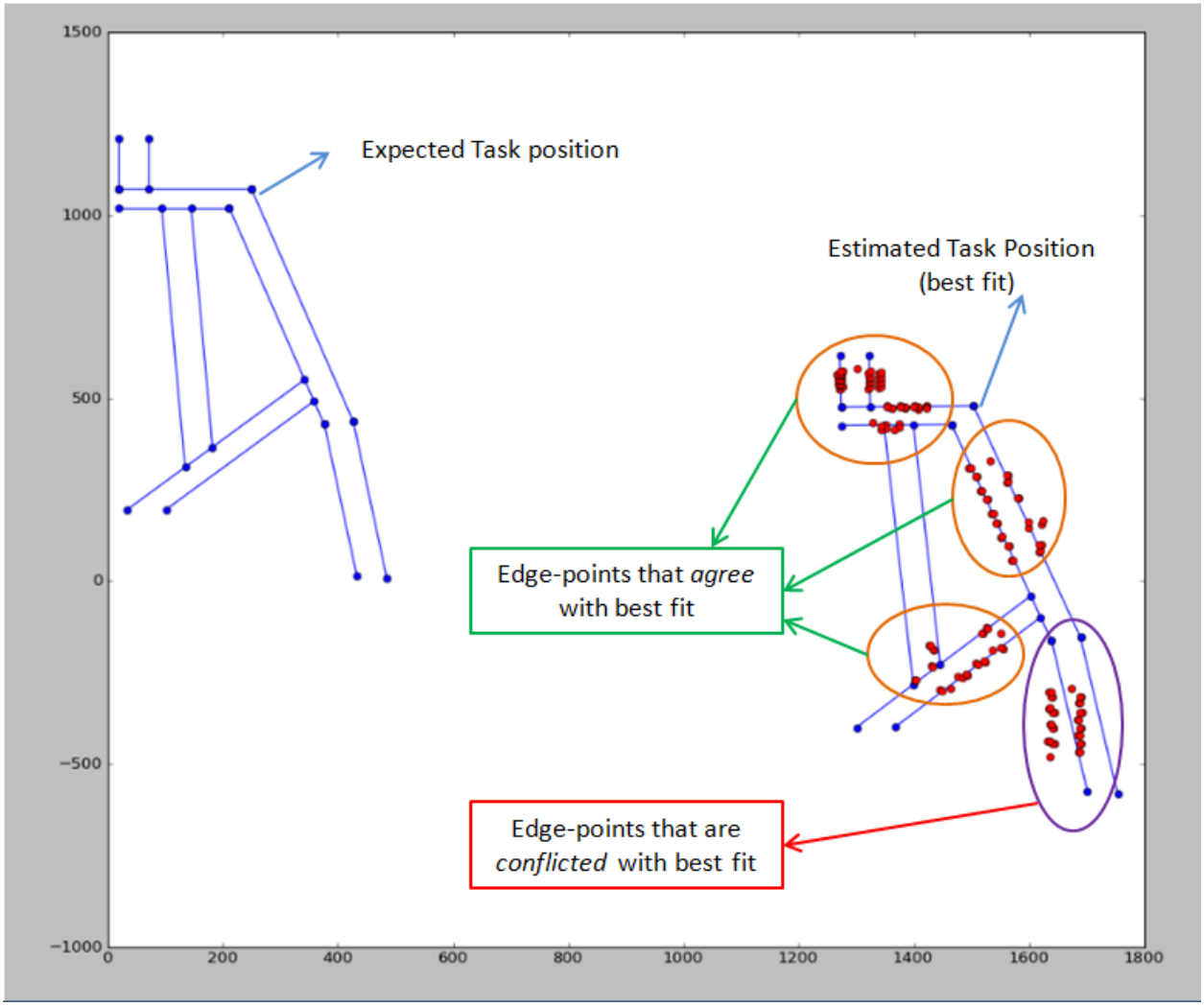


Figure CMU-20: Example data that does not match the template

3.7 Deliverables

- Videos, including http://www.ri.cmu.edu/videos/1/bourne_ifab_may2012/ifab_may_12_2012_quicktime_mpeg4.mp4, that demonstrate system capabilities
- Source code (Python) for the inspection, template-matching, and supervisory control algorithms

4. Setup-centered Process Planning

4.1 Task CMU.3

Responsible organizations:

- Primary — Carnegie Mellon University
- Secondary — University of Michigan

Continuing with the theme of compressing manufacturing time through automation, the Carnegie Mellon team developed *Task Composer*, an automated planner for computing efficient assembly plans that reduce manufacturing time by minimizing the number of human-performed setup tasks. Given a partial order representing the sequence in which the welds between parts in a subassembly should be achieved and a welding-node configuration describing the workspace geometry, Task Composer generates a minimum set of tasks, where each task represents a distinct parts layout, by determining the layout that maximizes the number of welds that can be achieved without having to move parts between welds. The resulting plan is automatically sent to the welding node software systems to inform the projection and inspection systems. Task Composer computes effective solutions within seconds and so replaces the time-consuming, human effort of planning setup actions. In addition to accelerating manufacturing time, Task Composer also enables quick plan generation, which helps in determining whether an artifact is easily manufactured with a given set of tools and fixtures and also facilitates good logistics decisions, such as selecting part and material suppliers that will best meet the time/cost constraints.

While, conceptually, Task Composer's constraint-based approach can be applied to three-dimensional assemblies, our current implementation focuses on the two-dimensional reasoning needed for subassembly flats. When invoked, the Composer starts with inputs for the partial order of subassembly connections (welds) and for the welding-node's configuration. Connection ordering comes from the sequence generated by the University of Michigan's analysis of tolerances based on their generated Liaison Graph. The coordinates corresponding to those connections are retrieved from the CAD design along with their constituent STL files for the parts. As a necessary step in generating the layout for a task, Task Composer has to move the parts out of their native STL coordinate system, which has an arbitrary origin, and place them into the welding-node coordinate system, i.e., the ideal position. The welding-node configuration contains the node layout, i.e., the robot's position, its minimal and maximal reach, and the worktable's size and position.

Figure CMU-21 depicts the control structure for Task Composer and a picture of an example welding node configuration where the smallest to largest values for the x and y dimensions go from left to right and from bottom to top respectively. The cycle for generating the tasks begins when the partial order of connections arrive. The first step is to check if there are any connections not currently assigned to a task. If so, try to add a new connection to the set of connections in the current task being generated. If no current task exists, then generate a new task first, make it the current task, and test the new connection. The selection of the connection to add is based on the partial order. The next connection to add is selected from the next set of connections not currently assigned to a task.

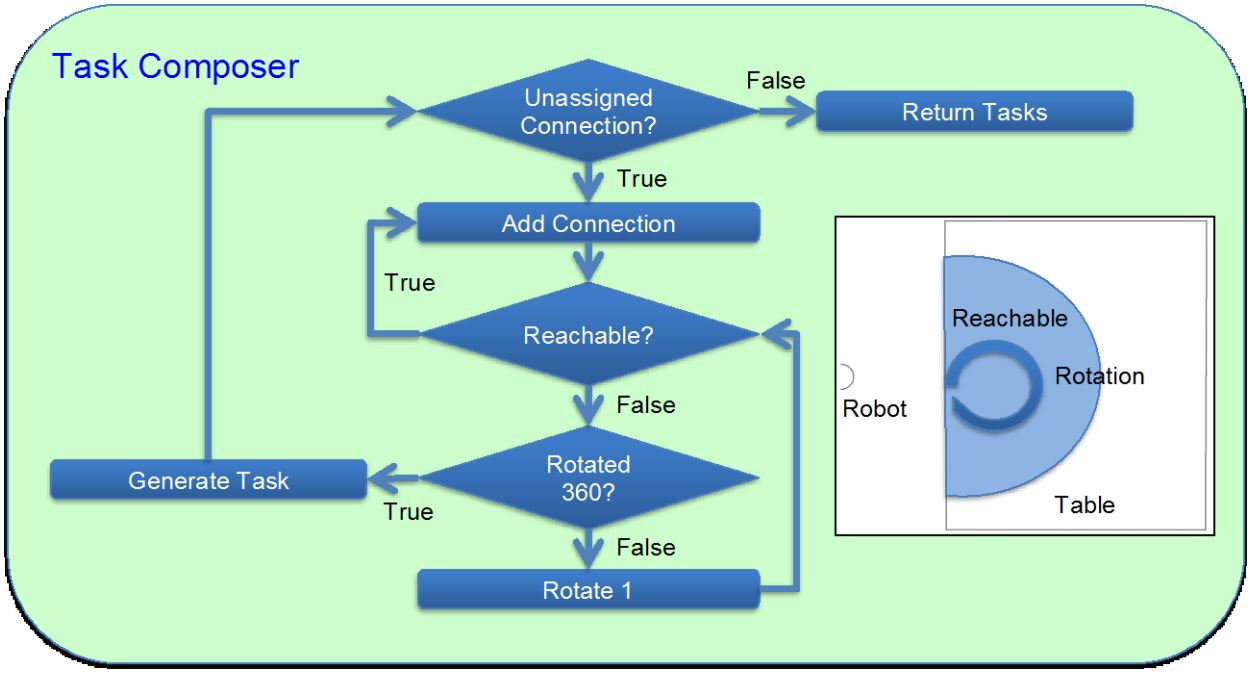


Figure CMU-21: Task Composer Architecture

In a total ordering of connections, the next connection to select is simply that following the one just added to a task. In sequences with partial orders, i.e., steps in which the order of some of the welds are not specified, search is required to generate separate plans for each possible ordering. But, before describing how that search is completed, we continue with showing how a single plan is generated. i.e., one with a total ordering specified.

A connection is tested to see if it can be added to a task by adding its coordinates to the coordinates for the other connections in the task. Then, a bounding box is computed for those points. The bounding box is then transformed to the origin of the ideal coordinates, which is at the robot. Then, the middle y value for the least x-value side of the bounding box is translated to the minimum weld distance along the x-axis. The points are checked to see if they are reachable by the robot using a simple radius check and assuming no collision. As we move to three-dimensional reasoning, collision checking becomes essential. In anticipation of this extension, we have developed a high-resolution motion planner and collision checker called *Motion Plan Generator*, which we describe in Section 4.4. If not all the connections can be reached, then the bounding-box is translated back to the origin and rotated one degree. The box is then moved back to the minimum weld distance point and the connection points are checked. This cycle continues until either all the points are reachable or the box has been rotated 360 degrees. If the new connection can be added, then the task remains open and a new connection is selected to be added. If the new connection cannot be added, then the task is closed and the new connection becomes the first connection to be tried in the new current task.

In tasks that have parts connecting to subassemblies in previous tasks, the subassemblies are treated as a single point, which is the center of the bounding box that encloses the subassemblies already completed. This aggregate point is the first one added to a new task and is interpreted like other connection points, except that its reachability test determines whether or not the point is on the table rather than if it is reachable by the welding robot. The actual dimensions of the

subassembly are also checked to see if they collide with the robot. Using the geometric center of the bounding box for the already welded connections assumes that point is the center of mass and if it is on the table, the subassembly will be supported. There are rare cases where that point is not the center of mass of the already assembled subassembly, in which case the task would not be achievable. To avoid this case, Task Composer would need to compute the actual center of mass for the already assembled parts.

The plan is completed when either all connections are assigned to a task or no more connections can be assigned, that is, no more tasks can be generated. In the latter case, those connections are assumed to be manual welds to be completed by humans. The plan is augmented to complete any welds that are lying on the table by adding copies of the computed tasks and modifying them by rotating them 180 degrees along the X axis. The plan is a full ordering of tasks in the order they were generated. The resulting plan is sent as a JSON-encoded string to the software systems in the welding nodes, which uses the tasks in the plan to inform the projection and the inspections systems. The ideal coordinates for the connections in the plan direct where and when the projection system should guide the placement of the part. The ideal coordinates also provide a baseline orientation for the parts for the inspection system.

4.2 Partial-order Search

When a partial ordering of connections is provided as input to Task Composer, the various alternative plans representing the different possible orderings of the connections are generated and the one with the minimum number of tasks is selected. In the case where the generation of the alternative plans is tractable within a user-specified time, an exhaustive search is performed to generate a plan for each possible ordering of the welds. For example, if the first weld is R and the possible second welds are X, Y, and Z, then plans are generated for (R,X), (R,Y), and (R,Z) using a breadth-first search and terminating when the first plan is completed. We can use a breadth first search because each step in the search represents the creation of a new task and our objective is to minimize the number of tasks.

In the case where the search cannot be achieved within the user-specified time, a best-first, greedy search is used. That is, rather than trying all possible plans in the partial ordering, only plans selected based on heuristics are generated. Currently, the two supported heuristics are *least-constraining* and *most-constraining*. In this context, constraining is determined by the size of the resulting assembly. The connection resulting in the smallest assembly is the least constraining. Conversely, the connection resulting in the largest assembly is the most constraining. Which heuristic results in the best plan depends on the situation. Using our example above of weld R having alternative successors X, Y, and Z, suppose that X is the least-constraining and Z is the most constraining. Then, the plans for (R X) and (R Z) are generated. By pursuing only the two orderings set by these heuristics in each case of their being a set of unordered connections, the number of possible plan generated is greatly reduced, yet, the best plan generated is usually still of high quality.

4.3 Task Composer Example

In this example, we demonstrate how Task Composer generates an assembly plan for the top octagon on the space frame. The non-green members in Figure CMU-22 are the octagon as it sits on top of the space frame. The subassembly for the octagon is submitted to the University of

Michigan's tolerance analyzer, which, for this example, generates a full-ordering for the connections in the octagon flat.

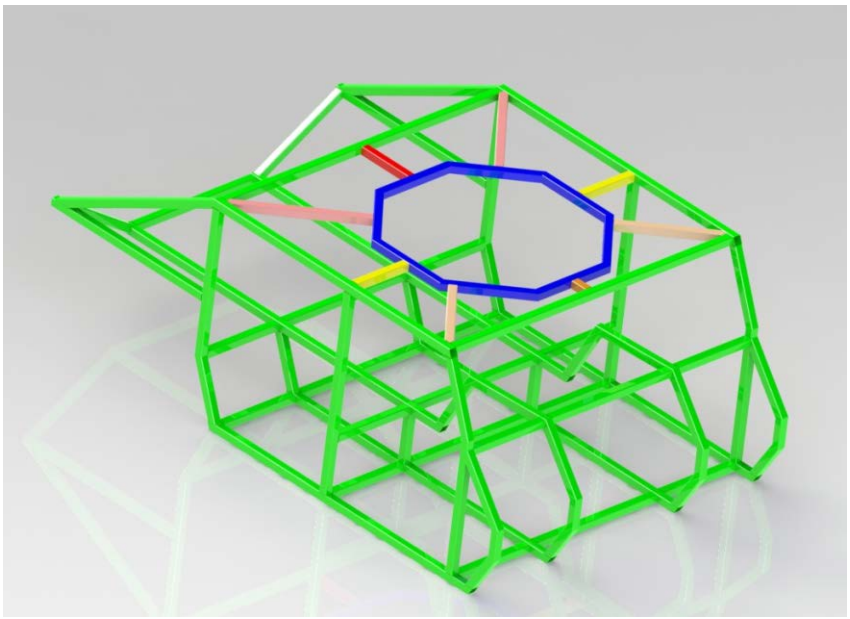


Figure CMU-22: Octagon on the Spaceframe

Given that sequence and the configuration of the welding node, the Task Composer generates the tasks for the provided sequence in Figure CMU-23 in less than a second.

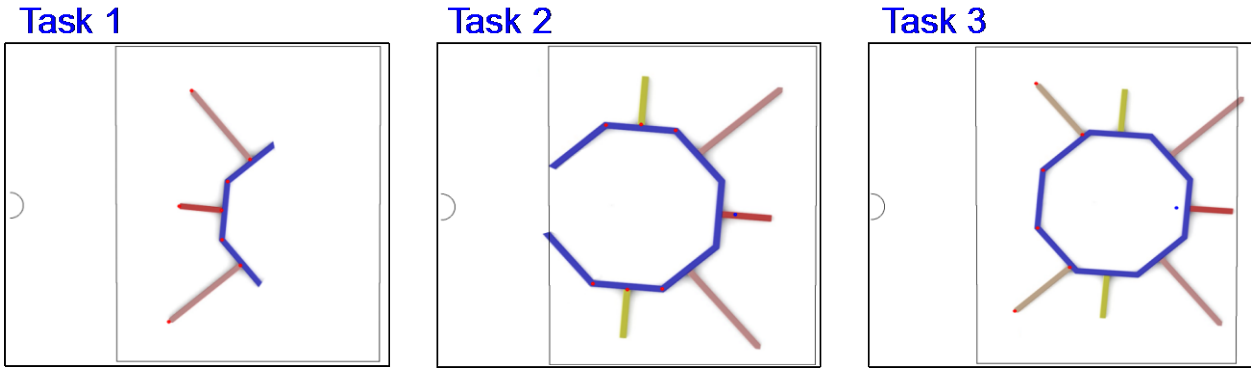


Figure CMU-23: Generated Tasks for the Octagon

The tasks are generated using the process described in Section 4.1. The red dots represent the connection and the blue dots represent the center of the subassemblies already welded. The resulting generated plan comprising the tasks represents a minimal set of human-performed layouts to support the automated welding of the octagon. Each task contains the ideal coordinates for each constituent connection. The plan is sent to the software systems to the welding node to inform those that facilitate the layouts for the tasks. Figure CMU-24 shows how, for each task, the provided connection points inform those systems. The projection system uses the tasks to determine what parts to place, in what order to place them, and where to place

them. The inspection system then verifies that the parts are where the tasks specify. Finally, the automated welds for the task are performed.

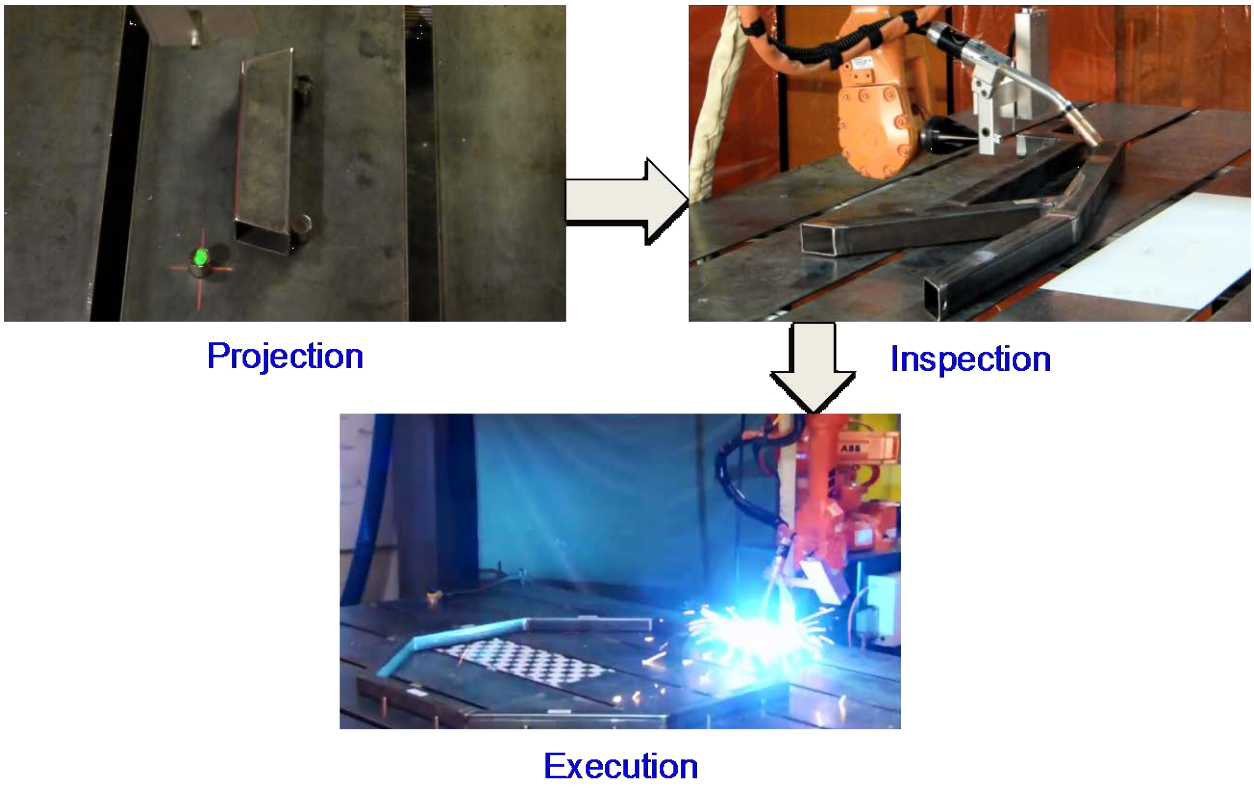


Figure CMU-24: Plan execution at the Welding Node

4.4 Motion Plan Generator

As described above in Section 4.1, we would like to extend Task Composer to apply to three-dimensional assemblies. As a necessary step towards this extension, Carnegie Mellon has developed a high-resolution path planner and collision checker called *Motion Plan Generator*. The Motion Plan Generator computes the precise motions plans required to perform the welds given the positions of the parts as determined by the Task Composer. The Motion Plan Generator is capable of generating collision-free motion paths using Rapidly-exploring Random Trees (RRTs). It generates the RRT by choosing random points in a scene and checking to see if they are reachable using both the path planner and the simulator provided in RobotStudio (as shown in Figure CMU-25).

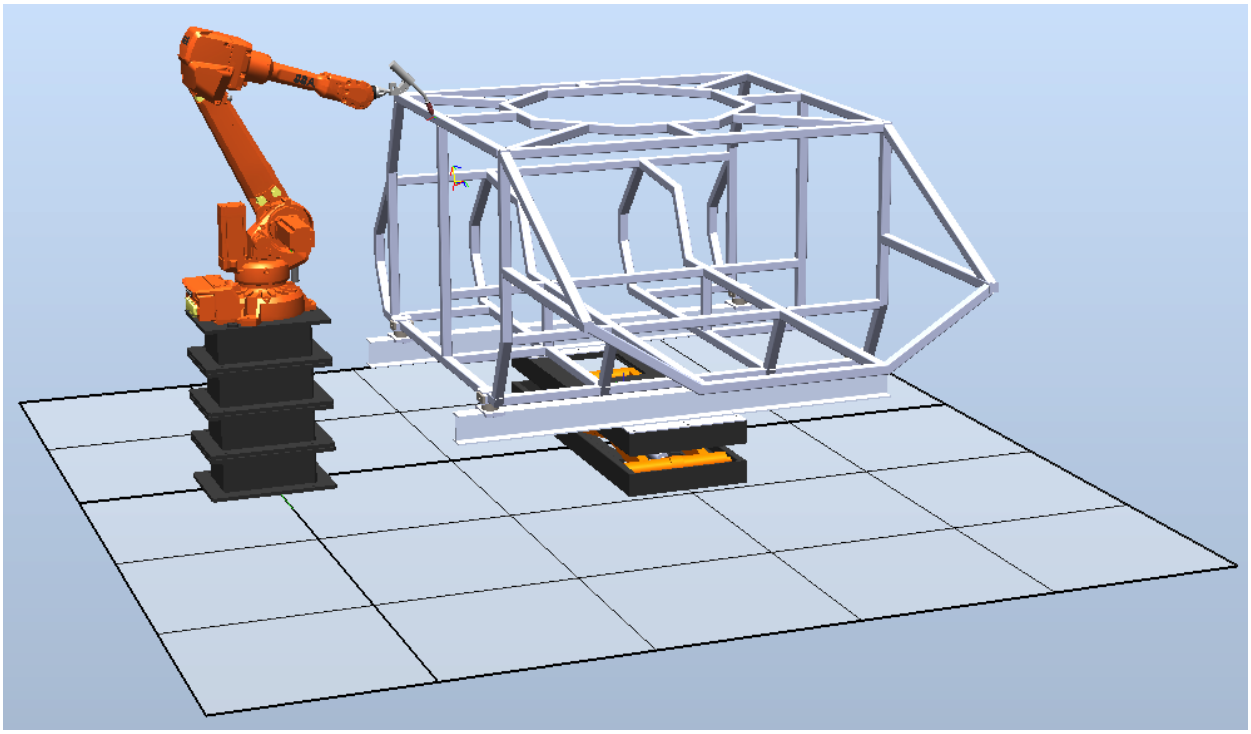


Figure CMU-25: Robot Studio Simulation Environment with Space Frame

The tree starts with a set of disbursed points within a set distance of the robot arm when it is in its neutral position. Each point is checked to see if there is a joint configuration that will enable the head to reach that point. Then, the tree is grown by generating new random points for each reachable point, with each new point being within a set distance of the reachable point. The new points are checked and the process continues with the new points being generated for the new reachable points. Each point in the tree has a series of joint configurations that allow the head to reach it. The size of the tree is determined by a parameter specifying the number of iterations to perform. When a motion path is needed for a new set of target points, the points proximal to the new points are selected and the search for the motion plan starts with the selected points' joint configurations, hence, reducing the time to find a solution.

Since we are using the RobotStudio simulator, the fidelity of this approach is very high. If a solution is found, it will be feasible in actual execution. This approach is not comprehensive in that there may be solutions that are not found, with the main determining factor being where the random points are chosen. Anecdotally, we have seen it find most expected solutions and even some that we would not have thought possible. If a path cannot be found for a weld, then the weld will be performed manually.

In addition to generating the motion plans for the actual welds in three dimensions, Motion Plan Generator would also be used to generate three-dimensional motion plans for the projection system, which uses the projector mounted on the robot arm to indicate where to attach the fixtures to the table and for the inspection system, which uses the laser (also mounted on the robot arm) to determine the exact position of parts on the table.

4.5 Deliverables

- Source code for the Task Composer and Motion Plan Generator systems

5. Training Instructions for Manual Assembly

5.1 Task: UMD.1

Responsible organizations:

- Primary — University of Maryland
- Secondary — Carnegie Mellon University, Pratt & Miller Engineering

Manufacturing customized vehicles involves complex, manual assembly operations, and human operators must be trained to perform these functions. In this situation, particularly at low production volumes and when utilizing shared setups and tools, generating assembly-training instructions can consume a significant portion of total production time.

To address this challenge, the UMD team pursued an assembly-sequence planning approach, employing motion-planning techniques to guide the creation of feasible assembly sequences for complex mechanical assemblies directly from 3D models. Then we developed a design framework that generates multimodal assembly instructions for human workers. The easy-to-follow instructions reduce learning time and eliminate the possibility of assembly errors. The system's ability to translate assembly plans into instructions enables a significant reduction in the time required to generate instructions and to modify them in response to design changes.

Most modern products get designed using 3D CAD systems. By exploiting the availability of 3D CAD data and advances in geometric reasoning, the UMD software system, the *Assembly Instruction Author* (Figure UMD-1), can generate assembly instructions in a matter of minutes. The system:

- Allows an assembly expert to demonstrate assembly operations in a virtual environment
- Automatically generates training instructions by extracting relevant details from these demonstrations and combining them with automatically-generated assembly plans

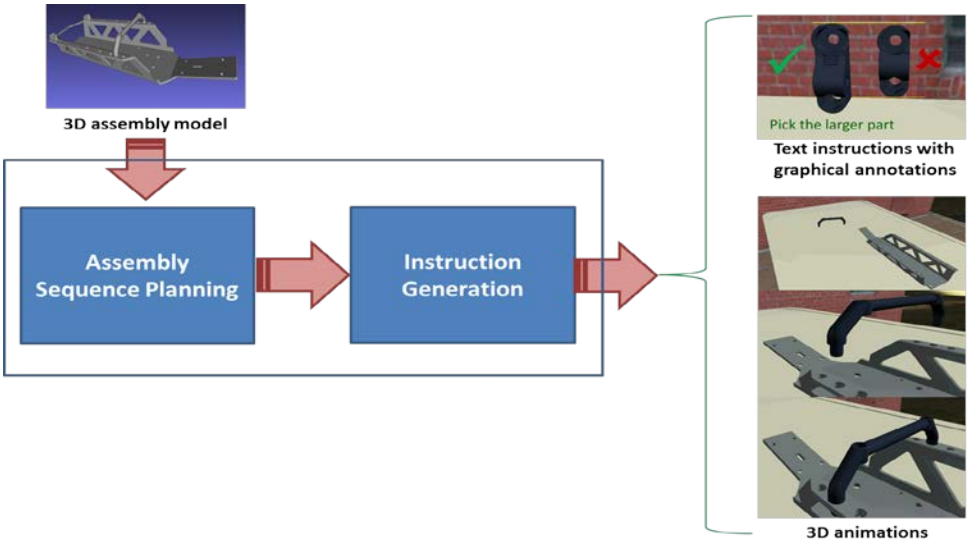


Figure UMD-1: *Assembly Instruction Author* system architecture

5.2 Assembly-sequence Planning

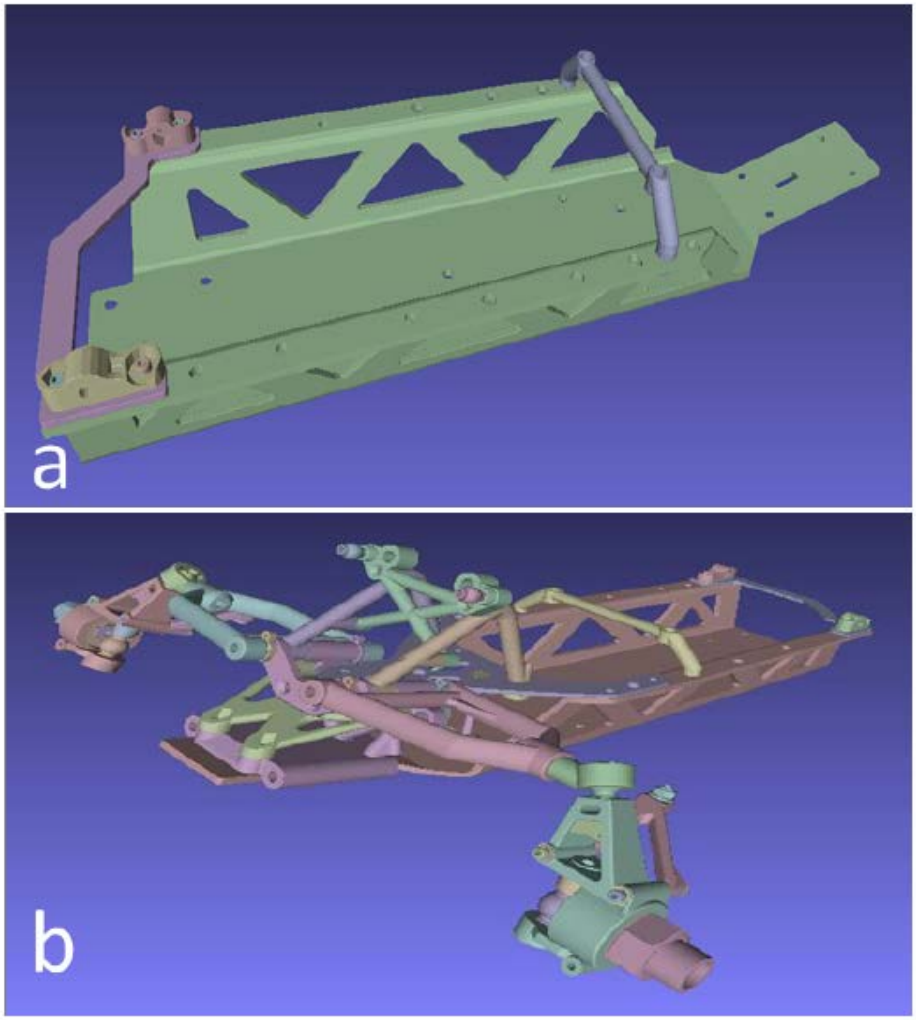


Figure UMD-2: (a) A simple chassis assembly of nine parts
(b) A complex chassis assembly of 71 parts

The UMD team developed an assembly-sequence planning framework that used motion planning techniques to conduct a thorough and automatic assembly analysis of the given design. Given an assembly model, the technique performed disassembly-sequence planning. Next, the disassembly information was used to generate feasible assembly sequences that can be used to assemble each part into the current subassembly. The UMD team tested the assembly sequence generation tools on the latest META deign package obtained from Vanderbilt University, shown in Figure UMD-2.

The input to the assembly sequence planning system is a 3D assembly model, which is a geometrical representation of a set of individual parts (that constitute the assembly) in their assembled configuration. The output of the system is a feasible assembly sequence in which each individual part can be assembled into the current subassembly, until the final assembly is achieved.

Next, the assumptions underlying the framework are explained briefly:

- All parts are rigid, all connections between parts are rigid, and once a liaison or connection is made, it remains in this way.
- The time taken to assemble each part depends upon the geometry of the part and the current target subassembly. Further, the abstraction of this assembly time related to each part is proportional to the disassembly time, which is computed by the motion planning module. The cost based on time information is useful to generate pre-order sets based on the complexity of the configuration space.
- Screws and nuts are included as members of the assembly; therefore, they belong to the assembly as parts. Any fastening method other than screw/nut must be removed from the 3D assembly model before our approach can be applied.
- The geometric assembly precedence constraints are completely based on the assembly model.
- The disassembly sequence is completely reversible.

Next, the problem formulation and various components of the assembly sequence planning framework—multiple random trees based motion planning, disassembly sequence planning, and generation of feasible sequences—are described briefly.

5.2.1 Problem Formulation

An assembly model is defined in a 3D scene. Let $\Omega = \{p_i ; i = 1, 2, .., n\}$ represent the set of parts that constitute the assembly, where n represents the total number of parts and p_i represents the i^{th} part. Further, let $O = \{o_i ; i = 1, 2, .., m\}$ represent the set of obstacles that constitute the crowded scene, where m represents the total number of obstacles and o_i represents the i^{th} obstacle; each obstacle o_i is the geometrical representation of a rigid body in a fixed position that belongs to the scene. In order to evaluate assembly feasibility, we need to find the assembly relationships between individual parts of an assembly. This information can be extracted from its 3D assembly model. For example, we automatically extract the total number of individual parts n and all the implicit geometrical information based on polygonal triangulation. Initially Ω and O have the same assembly elements and the same information. The assembly model must satisfy some consistency requirements in order to have a feasible 3D workspace. For our purpose, a feasible 3D workspace is a workspace in which all the assembly parts respect their shape and volume as a rigid body in every pose; no assembly part intersections and initial collision between multiple parts are allowed. In addition, the information about position and orientation of each part with respect to relative and absolute frames is extracted and used to compute the transformation between the two reference frames.
Let (x_i , y_i , z_i) and $(\alpha_i , \beta_i , \Upsilon_i)$ represent the position and orientation of part p_i, respectively. Now, the configuration $q = (x_i , y_i , z_i , \alpha_i , \beta_i , \Upsilon_i)$ defines the posture of the part in the scene.
Considering that the assembly parts are moved and disassembled one by one, an assembly admits a disassembly sequence if an escape path for disassembling each assembly part p_i can be found. Given the initial assembled configuration q_{init}, the problem consists of finding a feasible escape path, avoiding collisions, from q_{init} to a disassembled configuration q_{goal}.

5.2.2 Multiple Random Trees based Motion Planning

Rapidly-exploring random trees (RRT) based motion planning is a state-of-the-art search technique that provides feasible solutions for part navigation in crowded scenes, a problem representative of searching non-convex, high-dimensional spaces. In this technique, the tree is constructed in such a way that any sample in the search space is added by connecting it to the closest sample already present in the tree. RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly-chosen point to the tree. The pseudo-code to generate RRTs is shown in Figure UMD-3. The sequence of configurations between the initial and goal configurations are considered to be legal if it neither intersects any obstacle nor ends up in a self-intersection. The set of obstacles and the part assembly have to be dynamically updated in order to correctly describe the 3D environment and validate the computed path of the part in the environment.

$\mathrm{RRT}(q_{init}, K, \Delta t)$
$\tau.init(q_{init});$
for $(k = 1\ to\ K)\{$
    $q_{rand} \leftarrow RANDOM\_STAGE();$
    $q_{near} \leftarrow NEAREST\_NEIGHBOR(q_{rand}, \tau);$
    $u \leftarrow SELECT\_INPUT(q_{rand}, q_{near});$
    $q_{new} \leftarrow NEW\_STATE(q_{near}, u, \Delta t);$
    $\tau.add\_vertex(q_{new});$
    $\tau.add\_edge(q_{near}, q_{new}, u);$
$\}$
Return τ

Figure UMD-3: Pseudocode to generate RRT

However, RRT cannot be directly applied to complex assemblies that are composed of large number of parts, which often lead to scenes with high obstacle densities. The large number of obstacles generates a highly constrained environment with very narrow passages, resulting in the RRT's failure to generate a valid state. In order to address this problem, our approach uses multiple RRTs that dynamically modify the environment description to generate a valid escape path. It dynamically modifies the number of trees for each assembly part based on the environment constraints associated with the part. A highly constrained environment may require a large number of trees to find the escape path for an assembly part. Multiple RRTs can provide greater robustness across narrow passages and crowded environments. In traditional multiple RRT algorithms, the number of trees is kept fixed. However, in our approach, the number of trees is changed on-the-fly; that is, a new tree is added only if it is required by merging the information about the "old" tree with the "new" tree instantaneously.
The flowchart and pseudo-code to generate multiple random trees is shown in Figure UMD-3 and Figure UMD-4, respectively. The algorithm implicitly computes the Voronoi regions of each configuration node in a tree, and then selects a node for expansion with a probability proportional to the size of its Voronoi region. This thereby speeds up the exploration of the search space.

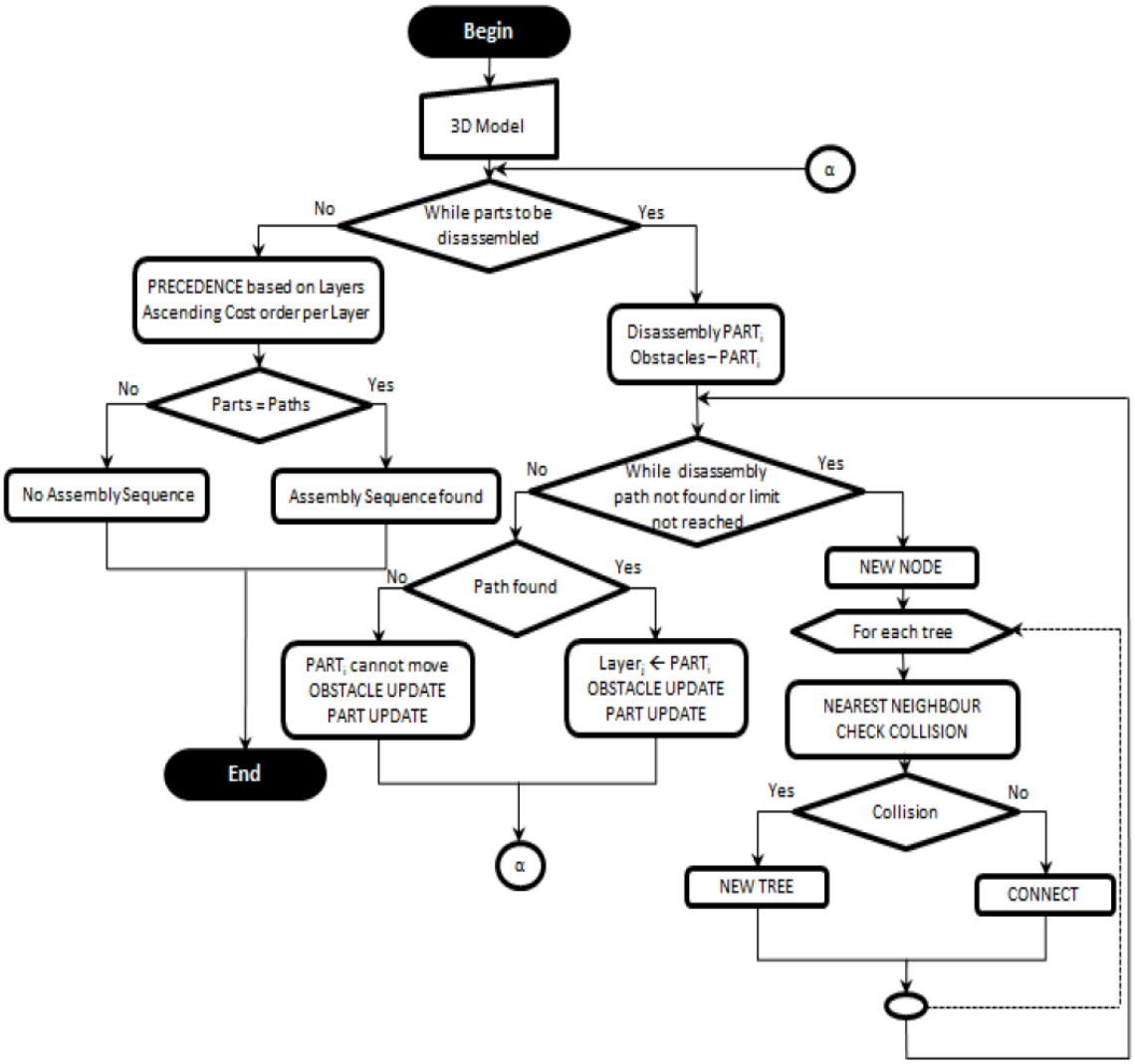


Figure UMD-4: Flow chart of multiple random trees used for assembly sequence generation

```
O ← OBSTACLE(Ω);
While (|Ω|! = 0){
    τ.init(q_init);
    O ← UPDATE(O − p_j);
     While (path not found) {
        q_rand ← RANDOM_STAGE();
        for each TREE {
            q_near ← NEAREST_NEIGHBOR(q_rand, τ);
            u ← SELECT_INPUT(q_rand, q_near);
            CHECK − COLLISION();
            If (collision == 0) {
                q_new ← NEW_STATE(q_near, u, Δt);
                τ.add_vertex(q_new);
                τ.add_edge(q_near, q_new, u);
            }
            else NEW TREE;
        }
    }
    If (path found){
        L_i ← NEW_PART(Cost_j, p_j);
        O ← OBSTACLE_UPDATE(O + p_j);
        Ω ← PART_UPDATE(Ω − p_j);
    }
}
C_ik ← PRECEDENCE(L_i, O, Ω);
If (|Ω| == |PATHS|)
    ASSEMBLY_SEQUENCE(INV(L, C, Ω));
else No FEASIBLE ASSEMBLY_SEQUENCE;
```

Figure UMD-5: Pseudocode for generating assembly sequences using multiple random trees

5.2.3 Disassembly Sequence Planning

The multiple random trees based motion planning uses the geometric information extracted from the assembly model to explore the global freedom of each part that allows it to be completely separated from the assembly. A part is said to be locally free if it can move an arbitrarily small distance from its original position in the assembly. Clearly this condition is necessary, but not sufficient to disassemble the part.

However, the accessibility that allows the part to be removed to any arbitrary destination outside of the bounding box represents global freedom. Satisfying this condition enables the system to determine which part movements are forbidden and which movements are feasible, and thereby generate feasible escape paths so that the parts can be disassembled. For this purpose, a hierarchical exploration structure is considered. The hierarchical exploration structure is composed of layers, where each layer *i* represent the precedence constraint of the layer *i*-1 for disassembly. Specifically, the system considers each part and checks if it is physically blocked by another part before it can be fully removed out of the assembly. Parts that can be removed in

this manner during the first attempt fill the first layer. This process is repeated to fill the second layer and so on, until all the parts are disassembled. As a result, the process is bounded by a maximum of n^2 iterations. The resulting hierarchically layered structure comprising part groups that can be removed at each layer gives rise to assembly precedence relations. Generation of such disassembly layers for the simple chassis subassembly of nine parts is shown in Figure UMD-6.

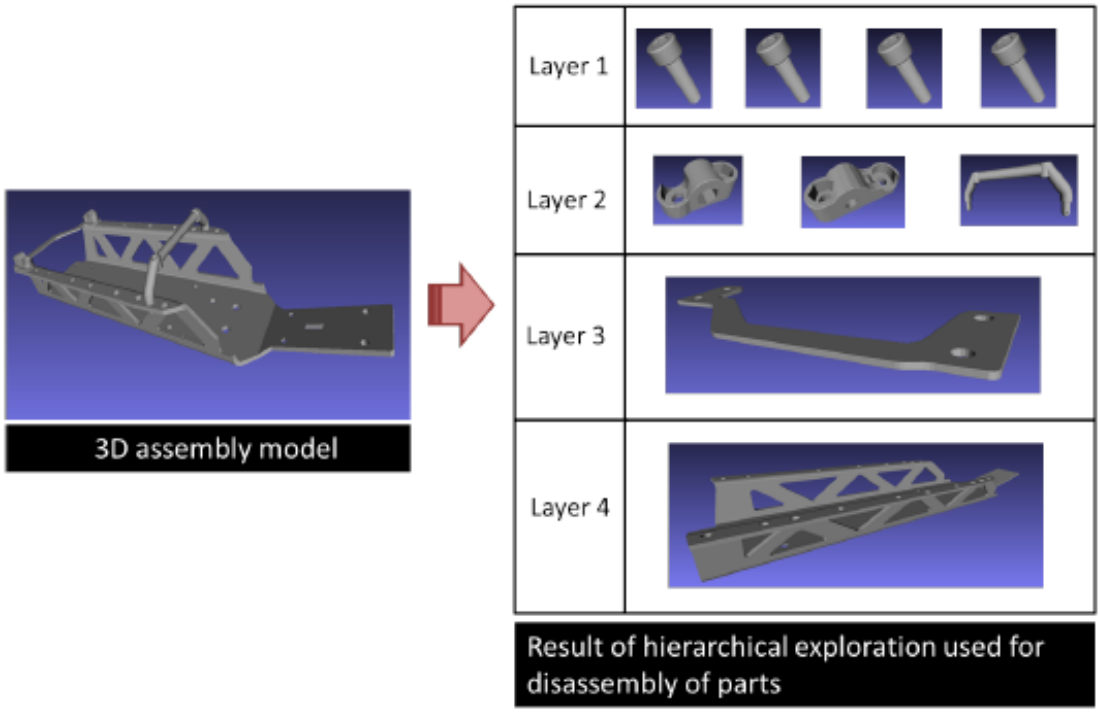
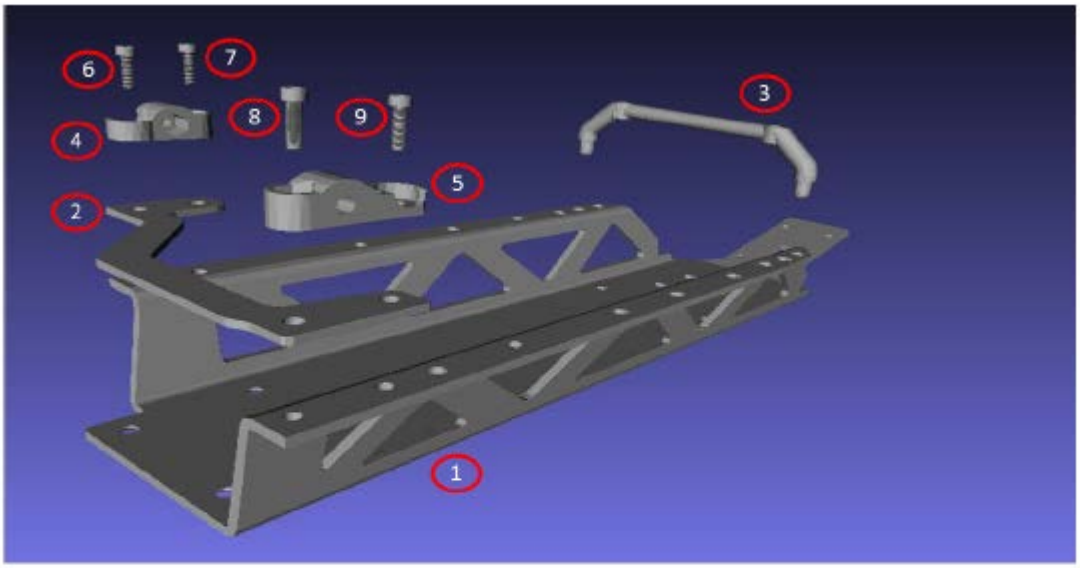


Figure UMD-6: Generating disassembly layers for a simple, nine-part chassis assembly

5.2.4 Generating Feasible Sequences

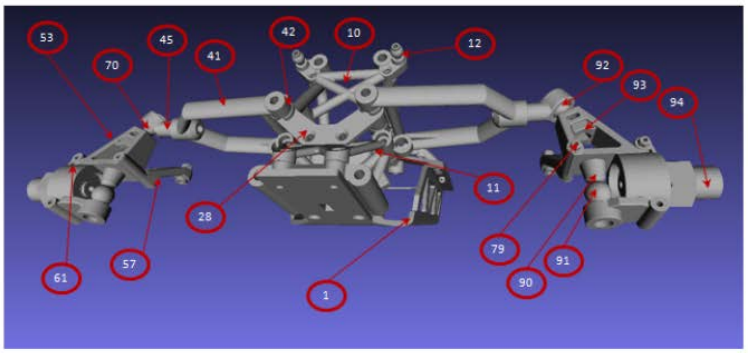
Removing a part from a set of parts is the same as attaching it onto the same set of parts under most circumstances. Therefore, the results from disassembly sequence planning can be used for the purpose of generating feasible assembly sequences. Once that the disassembly sequence is generated, a simple exploration in the opposite direction (from right to left) of the disassembly sequence can generate a feasible assembly sequence. For assembly analysis, most methods assume one actor that allows only one assembly operation at a time. However, the hierarchical exploration structure contains all the information necessary to determine whether parts can be assembled one by one or a subset of parts can be assembled at the same time. This decision making capability can be used to determine the number of operators involved in the assembly process. A feasible assembly sequence generated for the simple chassis assembly by using our approach is shown in Figure UMD-7.



Linear Assembly Sequence								
1	2	3	4	5	6	7	8	9
Main_ chassis	Rear_ brace	cenroll barset	4_radio	8_radio	SHCS 4_1	SHCS 4_2	SHCS 8_1	SHCS 8_2

Figure UMD-7: Generating a feasible assembly sequence for simple chassis assembly

Till now, we used the example of a simple chassis assembly of nine parts to illustrate the working of our assembly sequence planning approach. Here, we report results from the tests conducted on a complex assembly of 71 parts, shown in Figure UMD-2(b). In the initial information extraction phase, the system correctly determined the number of parts as 71, and extracted all the geometrical information based on polygonal triangulation. As mentioned earlier, the assembly model was checked for consistency requirements that allow a feasible 3D workspace. During this process, we found a case in which a modeling error caused two part models to intersect with each other. This problem was rectified before the assembly model was input to our algorithm. Next, the information about position and orientation of each part with respect to relative and absolute frames were extracted and used to compute the transformation between the two reference frames. The result of disassembly sequence planning for the complex assembly is shown in Figure UMD-8. Note that the 71 parts form groups that fill into a total of 11 layers. Figure UMD-9 shows a feasible sequence generated by the system for the complex assembly.



Layer	Part Number
1	56, 58, 61, 62, 65, 66, 68, 69, 71, 75, 79, 81, 84, 85, 89, 90, 91, 93, 94, 12, 13, 3, 4, 5
2	59, 63, 64, 73, 82, 86, 88, 96, 10, 28, 2
3	57, 72, 80, 95, 8, 9, 22, 23 11, 18, 19
4	60, 83, 7
5	53, 76, 6
6	70, 92, 99
7	51, 45, 100, 101
8	50, 44, 102, 103
9	47, 41, 104
10	48, 42, 49, 43
11	1

Figure UMD-8: Generation of disassembly layers for complex chassis of 71 parts

Assembly Sequence for 71 parts →																		
1	43	49	42	48	104	41	47	103	102	44	50	101	100	45	51	99	92	70	6
76	53	7	83	60	19	18	11	23	22	9	8	95	80	72	57	2	28	10	96
88	86	82	73	64	63	59	5	4	3	13	12	94	93	91	90	89	85	84	81
79	75	71	69	68	66	65	62	61	58	56									

Figure UMD-9: A feasible assembly sequence generated for a complex assembly

The UMD team identified several META model bugs (Figure UMD-10) with respect to implementing the motion planning algorithm used in assembly sequence generation:

- Two parts didn't show up (Front_Shock_Arm x 2).
- Three parts weren't recognized as solid bodies, so these parts were not exported correctly (Front_Arm_Lower_Left: Prt0005_1, Front_Hub_Right_Asm: 85408_1, and Front_Hub_Left_Asm: Prt0012).
- Part intersections and axis misalignments caused problems while implementing the motion planning algorithm.

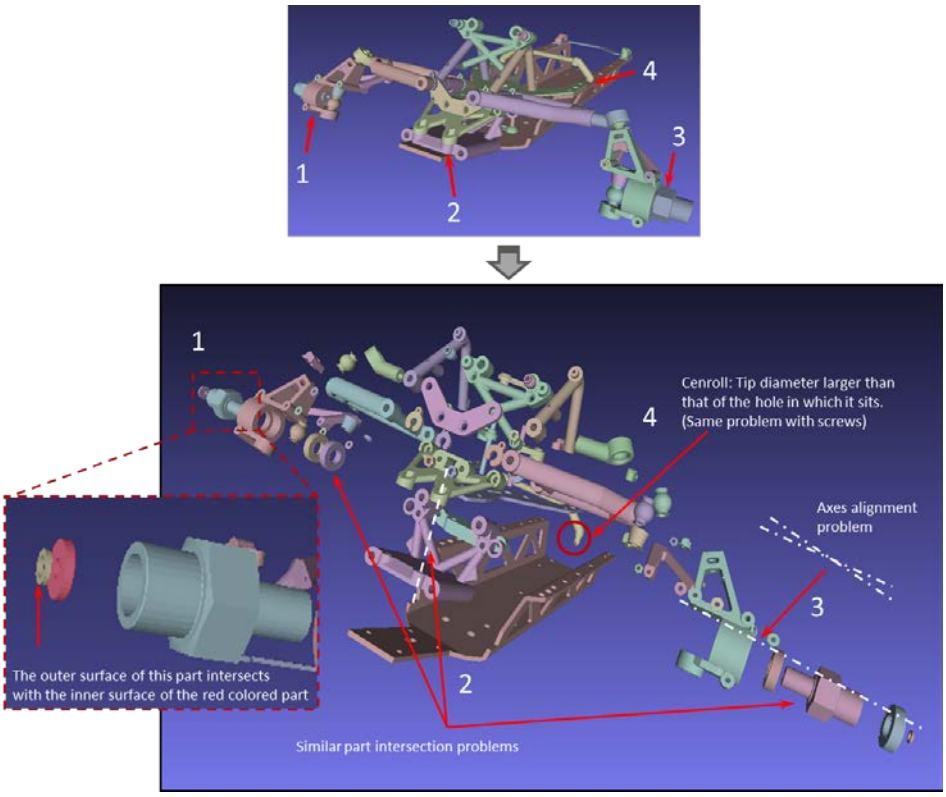


Figure UMD-10: Overview of bugs in META models.

Consequently, the UMD team fixed the bugs that were identified in the META models. Two example cases are shown below:

- Front_Arm_Set intersects Front_Bulkhead_Set (Figure UMD-11)
- Front_Bulkhead_Set_2 intersects Front_Bulkhead_Set_4 (Figure UMD-12).

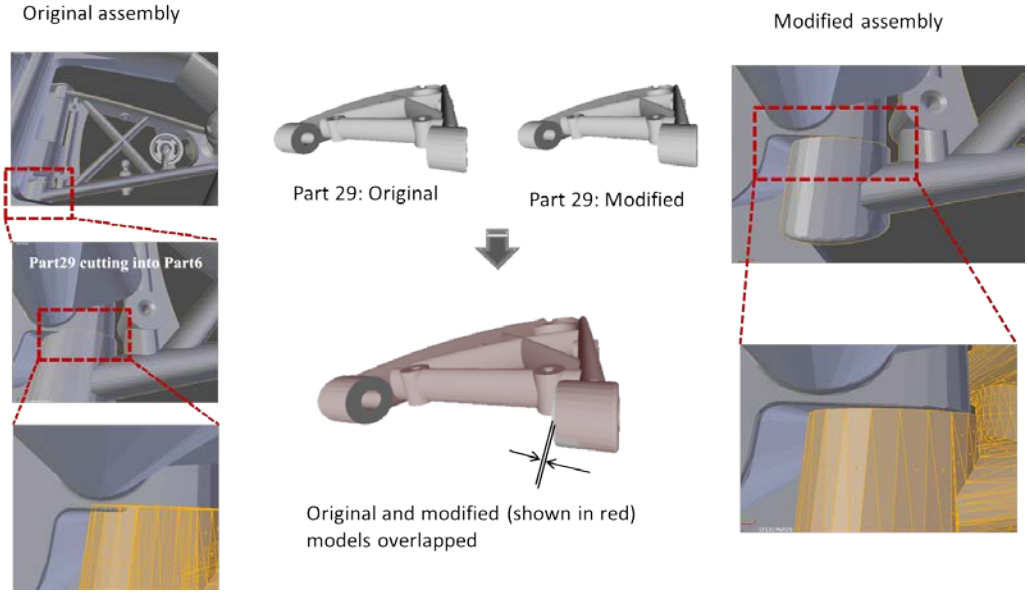


Figure UMD-11: Front-arm set – Problem issue and consequent modification

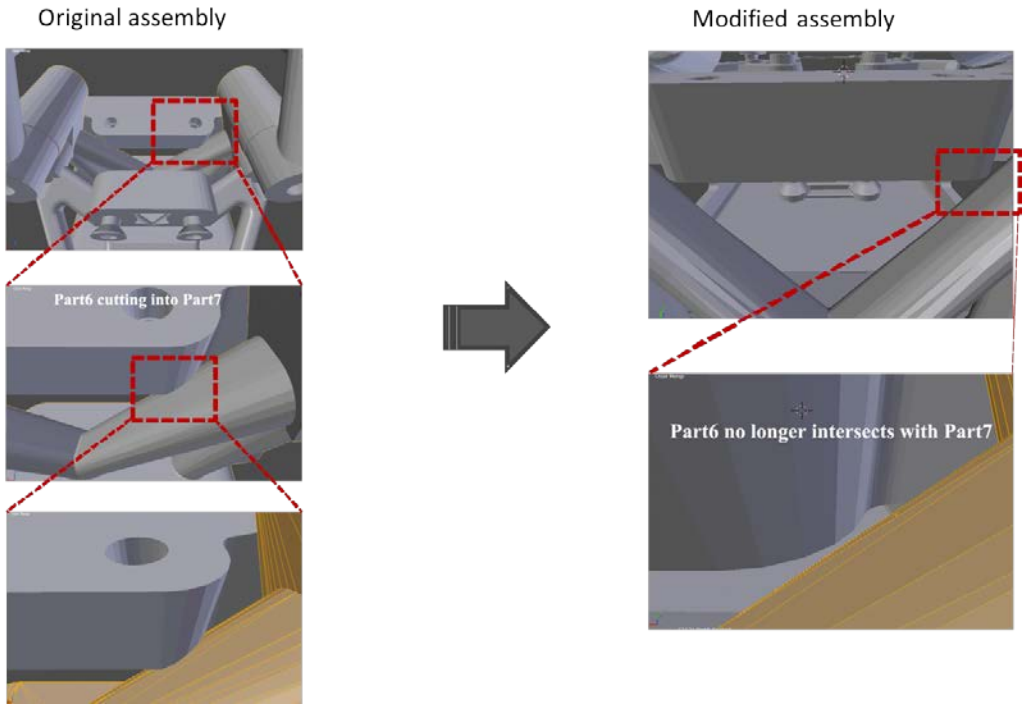


Figure UMD-12: Front_Bulkhead_Set_2 – Problem issue and consequent modification

5.3 Generating Human Instructions

The UMD team developed a system to generate text instructions, including graphical annotations and 3D animations, for human operators at each assembly step. An assembly sequence was considered as input to the system. These instructions were displayed on a large monitor situated at an appropriate viewing distance from the human carrying out assembly operations. By utilizing a speech-recognition module, the system allows the human to use a hands-free interface to sequence the flow of instructions easily. Figures UMD-13(a) and (b) show snapshots from a live footage of a human viewing an animation of placing a central roll bar onto the main chassis and the human subsequently performing the assembly operation, respectively.

Figure UMD-13: (a) Human operator views an assembly instruction;
(b) Implementing the instruction

The input to the instruction generation system is a linearly ordered assembly sequence represented in the extensible markup language (e.g., plan.xml). The contents of each step in the

plan.xml file will be used to generate multimodal instructions in the form of text, images, and animations.

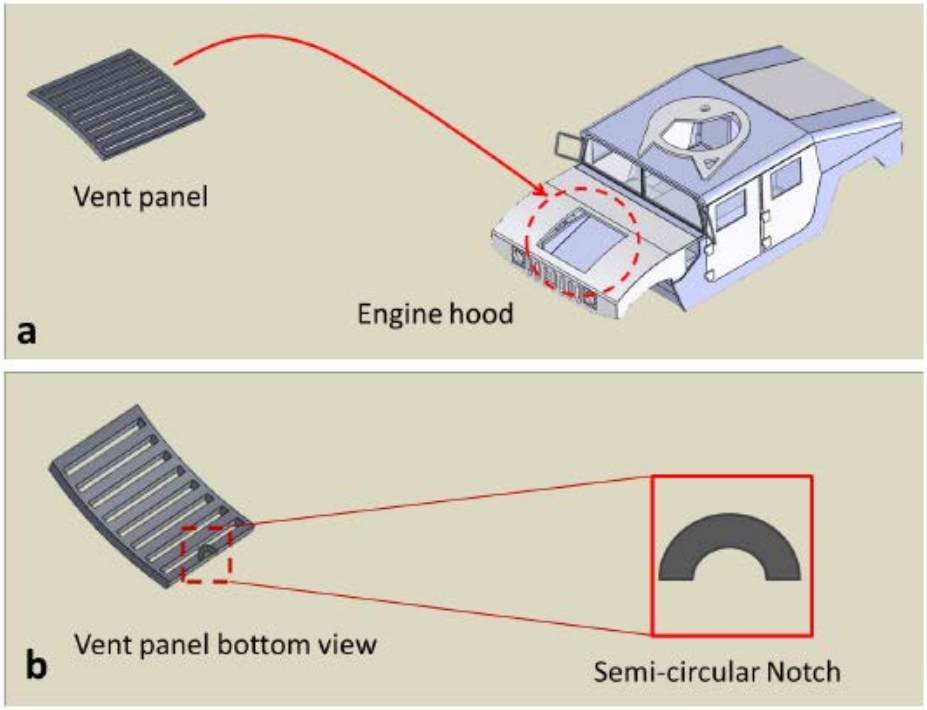


Figure UMD-14: (a) Attaching a vent panel to engine hood
(b) Bottom view, showing semicircular notch on one edge

Next, the various components that constitute the design of the instruction generation system — the language and grammar for text instructions, the process of how the animations will be generated automatically, automated part identification, and instruction presentation — are described briefly.

5.3.1 Text Instructions

The language for text instructions will consist of simple verbs such as Identify, Attach, Position, Rotate, Push, Pull, Lift, Lower, Check, Pick, Place, Use, etc. Examples of grammatical constructs for the text instructions include:

7. **Lift** PART? **by** HEIGHT?
8. **Use** HOW MANY? PART? **of** type TYPE?, **capacity** CAPACITY?, **length** LENGTH?
9. **Position** PART? **on** LOCATION?
10. **Position** PART? **so that** FEATURE-A? **aligns with** FEATURE-B?
11. **Attach** PART? **at** LOCATION?
12. **Attach** PART-A? **to** PART-B? **so that** FEATURE-A? **mates with** FEATURE-B?

To illustrate text instructions, consider the process of attaching an asymmetric vent panel to the engine hood as shown in Figure UMD-14(a). To the naked eye, the panel appears symmetric,

leading to confusion in correctly orienting it before placing on the hood. In Figure UMD-14(b), notice a semicircular notch on one bottom-side panel edge. This feature can be used during instruction presentation to guide the human worker in mating the correct panel face with the engine hood before proceeding with attachment. This framework then generates the following text instructions:

1. **Press** YELLOW BUTTON **to reset** ROBOT2
2. **Apply** FLUX **on edges of** VENT-PANEL-BOTTOM (special instruction)
3. **Pick up** VENT-PANEL MANUALLY
4. **Position** VENT-PANEL **so that** VENT-PANELNOTECHED-EDGE **aligns with** HOOD-FRONT-EDGE (View Animation)
5. **Attach** VENT-PANEL **to** HOOD **so that** VENT-PANELBOTTOM **mates with** HOOD-TOP
6. **Press** ORANGE BUTTON **to start** ROBOT2
7. **Press** BLACK BUTTON **to switch off** ROBOT2 **when** ROBOT2 **halts**

5.3.2 Automatically Generating Animations

The information extracted from plan.xml includes details about the initial scene, labels of the parts and/or subassemblies involved in the assembly operation, and their initial/final postures. This data from each step of the plan is used to invoke a simulation of the assembly operation in a virtual visualization environment, which was developed based on Tundra software. An automated motion planning module interacts with the visualization environment and computes a collision-free motion of a part from its initial posture (e.g., the hood lying in a shelf) to its final posture (e.g., placing of the hood onto the engine compartment of the space frame). Visualization of this computed motion in the visualization environment results in animations that will be appropriately labeled and saved as video clips in a local computer directory. We use multiple random trees, a variation of rapidly-exploring random trees, for the purpose of motion planning. In Section 4.2, we reported the details of how we use multiple random trees based motion planning to automatically generate feasible assembly sequences directly from 3D models of complex assemblies.

5.3.3 Part-identification Instructions

Usually, when a set of parts is presented to a human worker, he/she can easily distinguish among most of them. However, a few may look similar to each other, leading to confusion about which to pick. We have developed a part identification tool to determine automatically the presence of such similar looking parts and present them in a way that allows a worker to identify and pick the correct part. For this purpose, a similarity metric between two parts is constructed based on the following attributes:

- Part volume and surface area
- Basic shape statistics, such as surface types and their corresponding areas
- Gross shape complexity
- Detailed shape complexity that includes surface area and curvature information.

We consider two assembly examples to illustrate our part identification approach: (a) A simple chassis assembly of five parts, shown in Figure UMD-2(a) and a complex chassis assembly of 71 parts, Figure UMD-2(b). Table UMD-1 shows the dissimilarity matrix for the five parts of the simple chassis assembly, computed using the above technique. Dissimilarity values were in the range [0, 1], where a zero value meant that the two parts were fully similar to each other and a value of one meant that they were fully dissimilar to each other.

Part Number	1	2	3	4	5
1	0.000	0.764	0.753	0.667	0.593
2	0.764	0.000	0.312	0.746	0.678
3	0.753	0.312	0.000	0.737	0.656
4	0.667	0.746	0.737	0.000	0.739
5	0.593	0.678	0.656	0.739	0.000

Table UMD-1: Dissimilarity matrix for a set of five parts that constitute the simple chassis assembly (0 – Part is fully similar; 1 – Part is fully dissimilar)

Now, corresponding to each part $p^{(i)}$, the mean dissimilarity $d^{(i)}_{mean}$, with standard deviation values $d^{(i)}_{std}$ is computed over the remaining parts. Any part $p^{(j)}$ whose dissimilarity value lies below ($d^{(i)}_{mean}$ - $d^{(i)}_{std}$) is considered as similar to $p^{\{i\}}$. Figure UMD-15 shows the mean and standard deviation values for all the five parts. Note that part $p^{(2)}$ (Radio box 4) is similar to $p^{(3)}$ (Radio box 8). Consider that the human must pick up and assemble $p^{(3)}$ into the current subassembly before picking up $p^{(2)}$. Now, an animation is created, in which the two similar parts detected in the previous phase are lifted vertically and positioned adjacent to each other, with appropriate annotations that enable the human to recognize the correct part for pick up before proceeding for assembly. Snapshots from such a 3D animation generated by integrating the part identification module into the motion planning system are shown in Figure UMD-16. Part identification results for the complex chassis assembly are shown in Figure UMD-17. The part shown in the dashed square is the only one similar to part $p^{(45)}$.

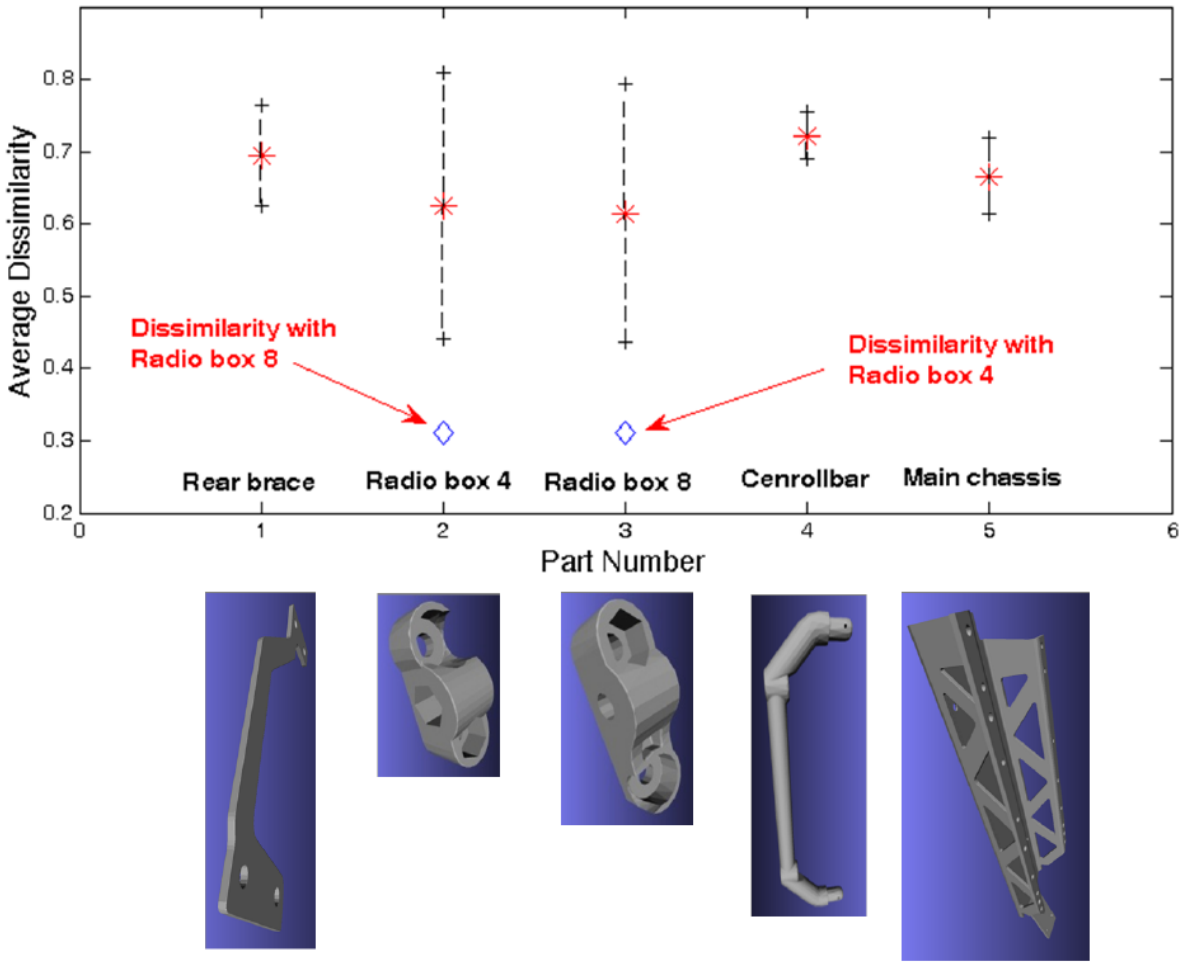


Figure UMD-15: Simple chassis assembly: Mean dissimilarity values and standard deviations with respect to. each part.

Figure UMD-16: Snapshots from the 3D animation used to instruct the human to identify — and pick — the correct part.

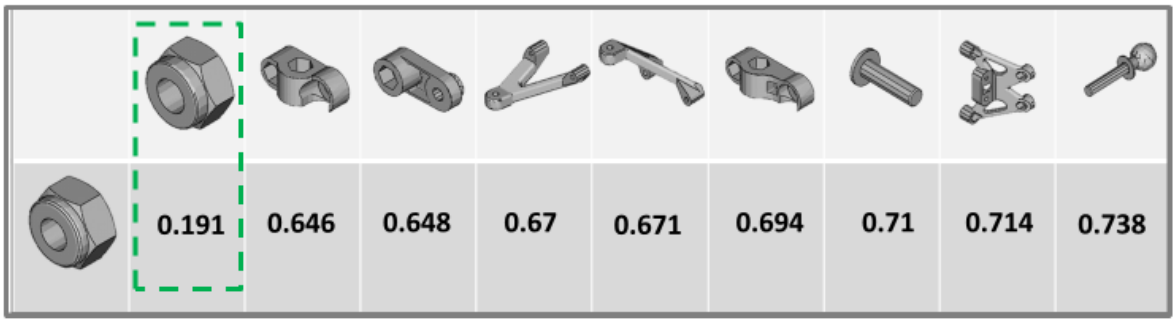


Figure UMD-17: Complex chassis assembly – Individual dissimilarity values of nine example parts with respect to. $p^{(45)}$. Only one part with a dissimilarity value of 0.191 is detected as similar.

To present instructions to the human worker, a webpage coded with .php scripts is generated by using the data extracted from plan.xml, filling the appropriate language constructs with this data, and querying the corresponding videos stored in the local folders.

We report the instruction generation results for the chassis assembly of Figure UMD-13(a). We consider the following assembly sequence as input to the instruction generation system:

1. Position MAIN CHASSIS at POSTURE 1 on ASSEMBLY TABLE
2. Position CENTER ROLL BAR at POSTURE 2
3. Position REAR BRACE at POSTURE 3
4. Position RADIO BOX 8 at POSTURE 4
5. Position RADIO BOX 4 at POSTURE 5

We assume that the assembly location (Posture 1) is selected by the user. Postures 2 to 5 are computed by combining information about the posture 1 and the relative/absolute reference frames extracted from the assembly model. Figure UMD-18 shows snapshots of the instructions — text, graphical annotations, and 3D animations — generated by the system, for each assembly step. Note that a part identification instruction precedes every assembly operation in which a new part must be picked up and attached to the current subassembly. This progression results in a set of ten instructions.

Figure UMD-18: Generated chassis-assembly instructions

5.4 Visualization in Tundra

The UMD team employed the Tundra environment to generate an augmented reality animation. We built and introduced into Tundra 3D models of assembly-critical resources, including welding robot, welding table, part shelves, sheetmetal-bending machine, waterjet-cutting machine, and laser-cutting machine. Figure UMD-19 shows a snapshot of one perspective on this environment. We also visited the CMU project site to study the actual welding workstation. Lessons gained through observation facilitated matching UMD's virtual augmented reality with CMU's real configuration. Figure UMD-20 shows a snapshot of the virtual augmented reality setup developed by UMD.

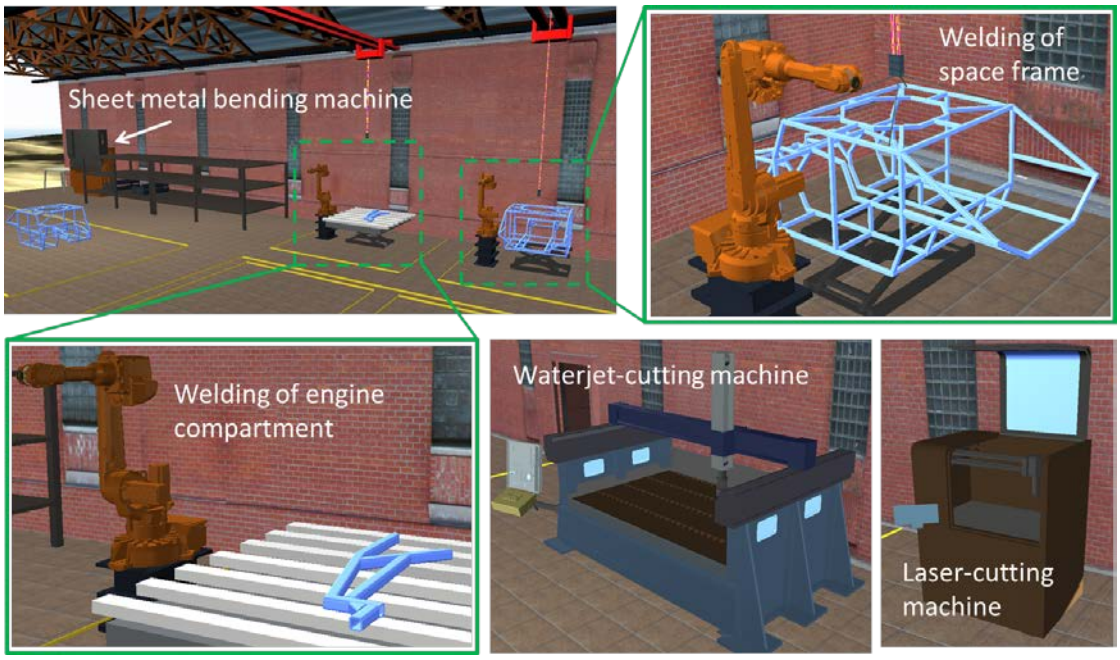


Figure UMD-19: Visualization of different manufacturing and assembly setups

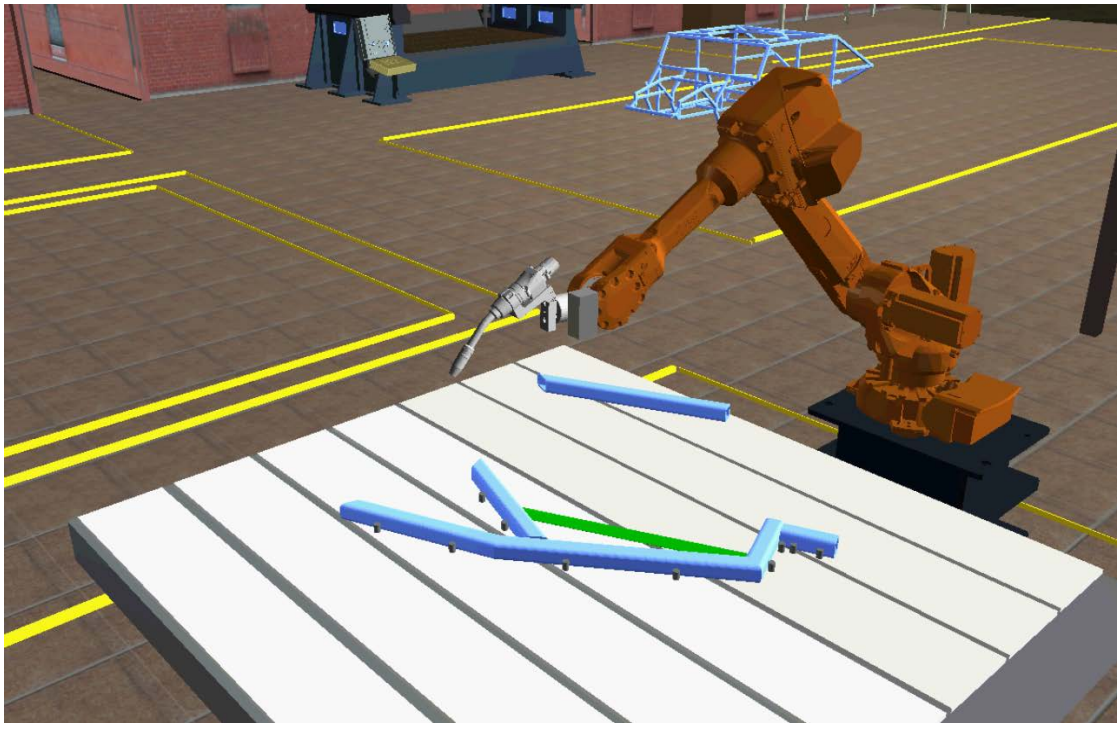


Figure UMD-20: UMD's virtual augmented reality, visualized in Tundra.

The UMD team also collaborated with Boeing to develop an API for the motion-planning system. We described the representation and specified the required inputs to their API. Boeing received STL files of parts, hand models, assembly workbench, and position/orientation

information for various entities in the scene. We implemented the motion-planning API and sent it to Boeing for testing.

5.5 Deliverables

- Videos showing a demonstration of system capabilities
- Modified META models and 3D models as built and imported into Tundra
- Source code for the algorithms

6. Automated Process Planning for Cutting Operations

6.1 Task UMD.2

Responsible organizations:

- Primary — University of Maryland
- Secondary — Carnegie Mellon University

Producing automated vehicles involves many different types of cutting operations, often performed on highly automated machine tools (e.g., waterjet, laser, and mill). Utilizing such resources effectively requires the capability to generate automatically instructions for running the tools with minimal human intervention.

To meet this need, the UMD team has developed an automated, reconfiguration-friendly process-planning system that utilizes shared tools/setups and effectively communicates with the production-planning system to exploit availability constraints on the relevant equipment. The system maximizes overall throughput and helps reduce the time needed to turn CAD models into physical parts.

We also developed an associated web interface that allows a user or a software agent to interact with the planner and obtain details such as manufacturability, delivery time, and cost. In particular, the user uploads an AutoCAD DXF part model and then specifies additional requirements, such as DXF file units, material type, thickness value, edge quality, etc.. Based on this input, the planner checks whether a certified operation plan for the part exists. If it finds no plan, it conducts a detailed manufacturability analysis on the part. If the planner then determines that the part cannot be manufactured, the corresponding reason is displayed and the user is invited to upload a new part. If the planner finds manufacture feasible, it then displays estimated delivery time, cost, and other relevant information.

6.2 Process Planning for Waterjet Cutting

The UMD team installed the planner interface software on a web server and tested its functionality with two sample DXF files: one of a good part and one including a manufacturing error. Figure UMD-21 shows a working demonstration of inputs to and outputs from the process-planning system for a waterjet cutting machine.

Our system architecture also incorporated additional details for analyzing manufacturability and for generating cutting instructions. In particular, we divided the system architecture into two separate modules, one for each modality. For the automated manufacturability analysis of 2D cutting operations, the communication between the human/web service and the manufacturing analysis service was achieved using the following functions: *capability query*, *capability*, *quote request*, *quote*, *part order*, and *order confirmation.*

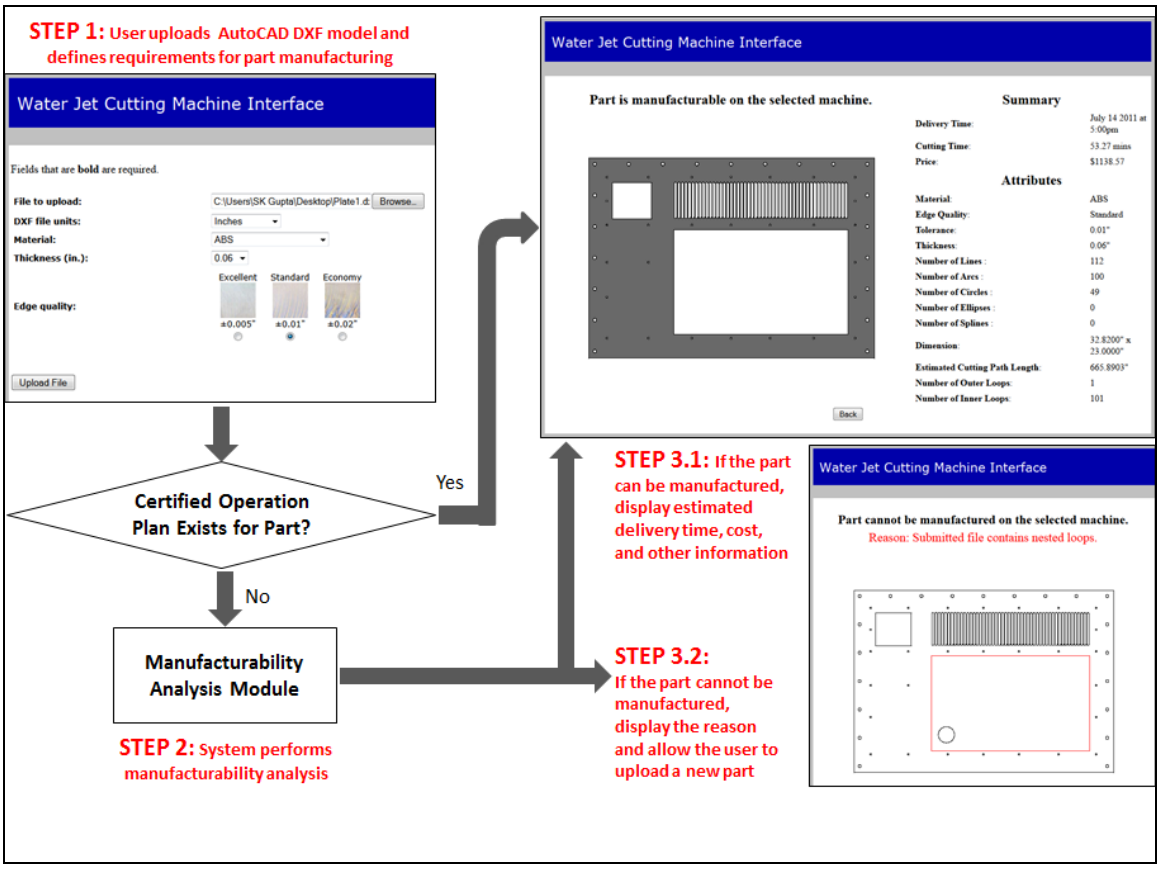


Figure UMD-21: Process planning for a waterjet cutting machine

The manufacturing analysis service interfaced with the process knowledge, the physical machine, and certified plans in order to arrive at manufacturability decisions. The system architecture for generating cutting instructions is shown in Figure UMD-22.

For one sample part, the planner determined that, out of four machines, the first two were incapable of manufacturing the part. Machines 3 and 4, however, could do the job. Using the part's DXF file, the system then converted polylines into arcs, automatically added tabs and leads, and generated an ORD file comprising detailed cutting instructions. We provided the Lockheed Martin team the necessary material and steps on how to operate the waterjet cutting service so that they could incorporate it into their agent framework.

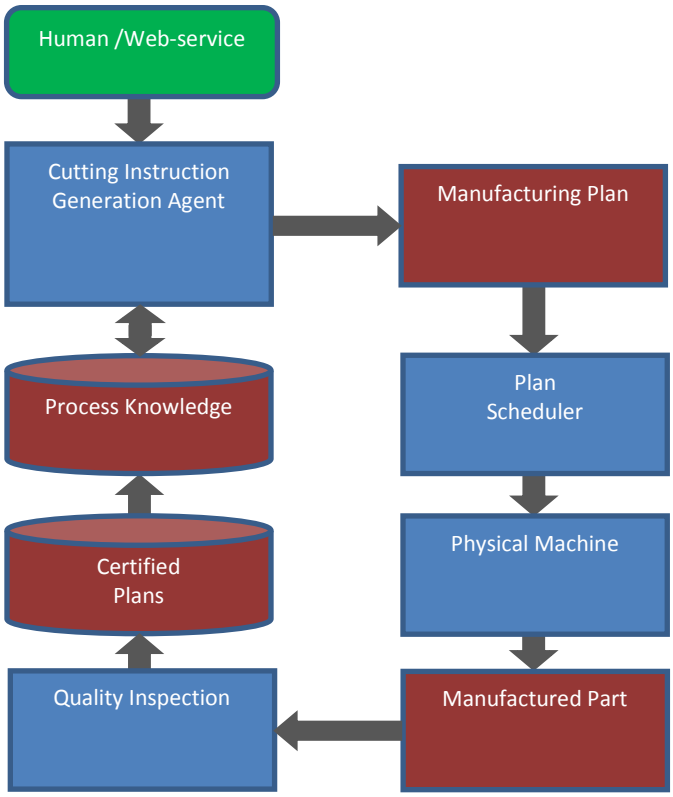


Figure UMD-22: System architecture for generating cutting instructions

Bracket is one of the parts used in vehicle hood assembly. The CMU team provided us with the Bracket design and other specifications, including material type, thickness, edge quality, and dimension units. We then used the waterjet cutting machine web service to conduct manufacturability analysis for the part (Figure UMD-23). The system determined that the OMAX 2626 machine could manufacture the Bracket and automatically added lead-ins, lead-outs, and tabs, along with cutting instructions. Cardinal Scientific's shop fabricated two copies and shipped them to CMU.

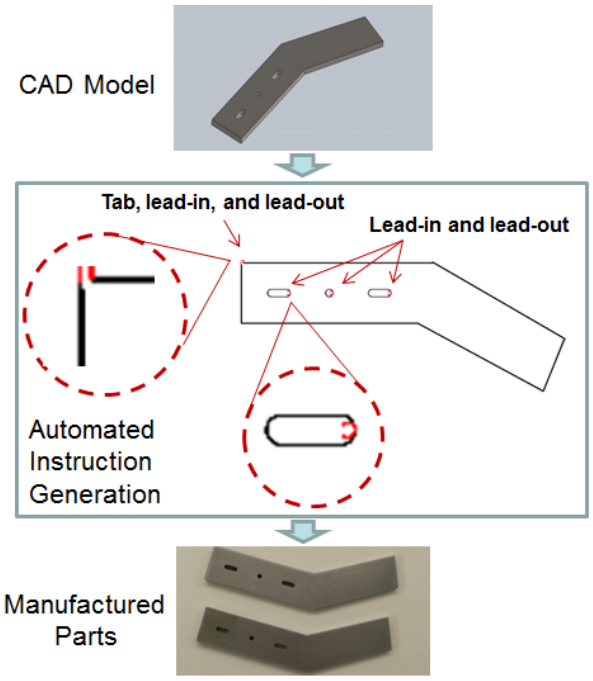


Figure UMD-23: Manufacturability analysis for the Bracket

6.3 Process Planning for Laser Cutting

The UMD team also developed a laser cutting web service and demonstrated "CAD to finished part in minutes" capability by automatically analyzing manufacturability, generating instructions, and fabricating parts. We next developed a client application that can be used by any external computer agent to connect automatically to their waterjet and laser-cutting web services, obtain machine capability, upload part design, obtain quote, and subsequently place an order to manufacture the part.

The web service developed for the 2D cutting processes required the outline and thickness information to be uploaded in order to perform manufacturability for a part design. However, the META package consisted of 3D part models and didn't explicitly provide the part outline and thickness information. Therefore, the UMD team designed a preliminary algorithm to extract this data from 3D part models automatically.

6.4 Deliverables

- Videos demonstrating the planning system capabilities
- Source code for the algorithms
- Sample part models to test the algorithms

7. Manufacturability Design Rules from Process Models

7.1 Task UMD.3

Responsible organizations:

- Primary — University of Maryland
- Secondary — University of Michigan

Manufacturing machine specifications are often given in terms of physical parameters of the machines. For example, a milling machine may be defined terms of length of the x, y, and z axis travel, spindle horsepower, maximum available spindle RPM, resolution of table travel, and maximum weight supported by the table. Translating this data into the attributes of parts that can be realized on the machine requires significant manufacturing expertise. So often, manufacturability-based design rules are not explicitly available. This situation requires an expert manufacturing engineer to perform the manufacturability analysis.

To address this challenge, the UMD team's developed a systematic methodology to extract design rules from process models by exploiting constraints on tool travel, tool size, and uncertainty in tool location. This approach produces explicit design rules and emphasizes those that are general in nature as well as those that are META-program compliant. We also developed a tool that evaluates a part's manufacturability under 2D cutting operations and a web-based service that interacts with the tool.

7.2 Rules Based on Tool-travel Constraints

The UMD team identified manufacturability rules derived from how far a tool can travel inside a machine tool. By analyzing tool-travel constraints in X and Y dimensions and accounting for the tool size, we generated a size envelope in which the part must fit to be manufacturable on a given machine. This rule allows the part to be rotated to fit in the envelope.

First we computed length and width of the bounding box for the part as a function of the rotation angle θ. The manufacturability rules states that:

> If there exists an angle θ such that $L(\theta) \leq L_{max}$ and $W(\theta) \leq W_{max}$
> Then the part does not violate tool travel size constraint.
> Where, $L(\theta)$ is the length of bounding box if the part is rotated by angle θ;
> $W(\theta)$ is the width of bounding box if the part is rotated by angle θ;
> L_{max} is the length of the size envelope;
> W_{max} is the width of the size envelope.

Figure UMD-24 shows a result in which, given a part design, the system automatically computed and displayed the orientation that allowed the raw stock to fit within the machine before it can be cut to produce the desired part.

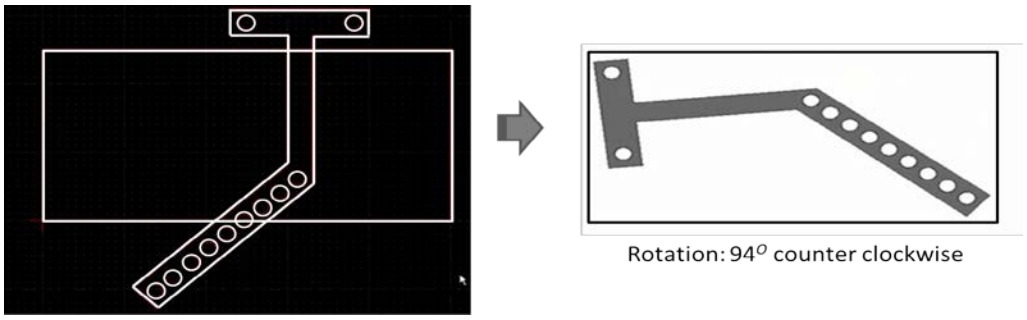


Figure UMD-24: The system computes the angle by which the part must be rotated to fit within the machine's work envelope

7.3 Rules Based on Tool Size

We also developed rules based on tool size, which rules impose constraints on minimum concave corner radius.

7.4 Rules Based on Uncertainty in Tool Location

Based on tool-location uncertainty, we developed rules that constrain inter-feature distances to ensure that features could be clearly resolved.

7.5 Determining 2D Cutting Feasibility

Finally, we designed an algorithm that accepts 3D step files from the META package and determines whether or not the part can be manufactured in a given 2D cutting process. The algorithm works as follows:

1. Construct a triangular mesh representation of each part model
2. Compute the surface normal for each triangle
3. Group triangles into bins according to their surface normal. For example, all triangles with a normal aligned along the same direction fall into the same bin.
4. Check the resulting histogram for three conditions:
   d. Existence of two bins whose surface normals lie in opposite directions with respect to each other
   e. Surface normals of all other bins are orthogonal to that of the above two bins
   f. Separation between the triangles in the two bins doesn't exceed the maximum thickness allowed by the 2D cutting process.

   If all three conditions are satisfied, the given 2D cutting process can manufacture the part.

7.6 Deliverables:

- Videos demonstrating the system's capabilities
- Source code for the algorithms
- Sample part models to test the algorithms

8. Generating Manufacturability Feedback

8.1 Task UMD.4

Responsible organizations:

- Primary — University of Maryland
- Secondary — University of Michigan

In traditional factories with in-house production capabilities, manufacturing engineers review a design and provide manufacturability feedback to the design team. This model will not work with the manufacturing style envisioned for the iFAB Foundry. To bridge the gap, the UMD team undertook to create manufacturability-analysis software tools that design teams can themselves use. With such tools, a design team can select a candidate iFAB Foundry and download its associated design rules. The manufacturability-analysis system can then evaluate a candidate part design against foundry-specific rules and offer the design team appropriate feedback.

The team developed a two-component system: a tool that determines the manufacturability of a given part with respect to 2D cutting operations and a web-based service to interact with the tool. Operation begins by downloading the target-machine capabilities — comprising such details as available materials, allowed thickness values, and edge quality types— as an XML file. Next, the part design, in the form of a 2D DXF file, and part-attributes such as material type, thickness, type of units, and edge quality are uploaded as tool input.

For a specific waterjet cutting machine, the system evaluates part manufacturability with respect to multiple criteria:

- Geometric validity of part model
- Desired edge quality of the part
- Machine size
- Part material
- Available stock sizes for the selected material
- Part thickness
- Feature sizes on inner and outer part profiles
- Nested, intersecting, or open loops in the part design

If the system determines that the part is not manufacturable, it then generates detailed annotations on the part, identifying the problem source(s). Figure UMD-25 shows example parts we used to test the system's ability to analyze manufacturability automatically.

Figure UMD-25: Parts used to test the manufacturability-analysis tool

Figure UMD-26 shows the manufacturability analysis results on the Rear Brace, one of the five chassis-assembly parts provided by the META team.

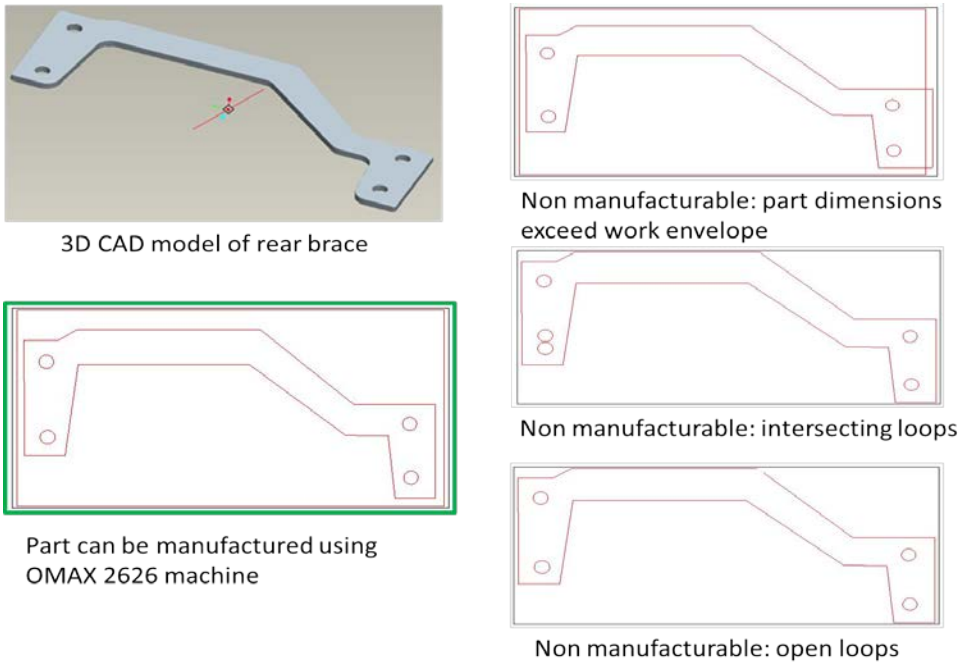


Figure UMD-26: Results of manufacturability analysis of different part designs for Rear Brace

Next, the UMD team implemented the preliminary algorithm described in previous task, which could accept 3D part models as input, to carry out manufacturability analysis of the parts in the META package. An example is shown in Figure UMD-27.

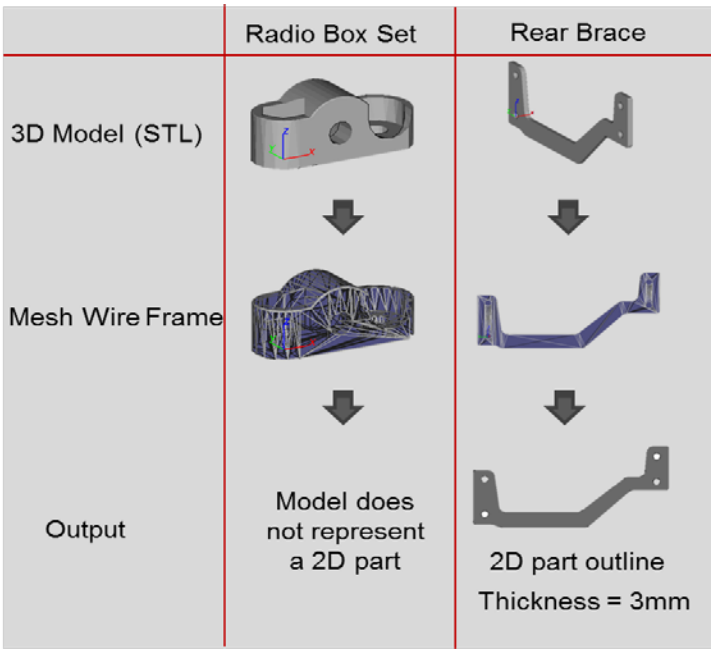


Figure UMD-27: Manufacturability analysis of two parts in the META package

8.2 Deliverables

- Videos demonstrating the system's capabilities
- Source code for the algorithms
- Part models to test the algorithms

9. Mixed-initiative Setups – Analyzing assembly constraints

9.1 Task: UMI.1

Responsible organizations:

- Primary — University of Michigan

One bottleneck in manufacturing complex assemblies lies in predicting whether the final assembled product will be capable of meeting desired tolerances. Accomplishing this goal requires optimizing assembly sequence, reference datum flow chain, and part-tolerance allocations, all of which in turn drive selection of the manufacturing process.

The UMI team addressed this challenge with the scheme illustrated in Figure UMI-1. CAD data for a new product design first goes through a chain of preprocessing tools that identify the assembly structure, including part tolerances as well as other relevant information, and translate it into an XML format that structures the Engineering Bill of Materials (eBOM). Various assembly sequences are then analyzed and ranked, checking for over-constraint joints and identifying key characteristic (KC)[1] dimension delivery chains. Best ranked sequences are then further analyzed via Monte Carlo simulation in task UMI.2 to estimate the assembly variability.

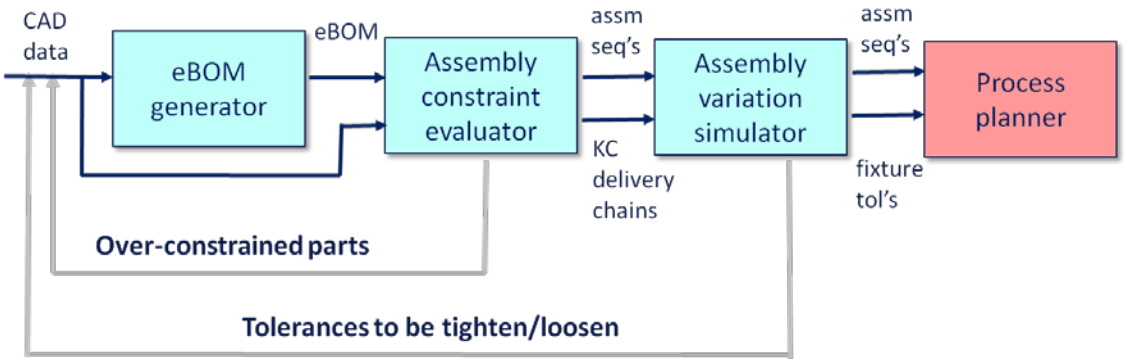


Figure UMI-1: Overview of Assembly Variation Tool

9.2 eBOM Generator

The eBOM generator module reads CAD data from a SolidWorks Assembly files and generates the eBOM XML code, using the schema illustrated in Figure UMI-2. The main schema building block is the "subassembly," which contains data fields for:

- Name/ID/Bounding-box
- List of parts

[1] Key Characteristics represent a particular type of assembly tolerance that determines whether final the product will function as designed. Such features might include dimensional constraints, such as hinge-block colinearity, where all hinge axes must lie close to a theoretical straight line.

- List of other subassemblies (lower in the assembly tree)
- Mating constraints between parts/subassemblies
- List of assembly-level dimensions and tolerances

The other noteworthy schema building block schema is the “Part,” containing data fields for:

- Name/ID/Bounding-box
- Material
- List of CAD features and feature data
- List of part-level dimensions and tolerances

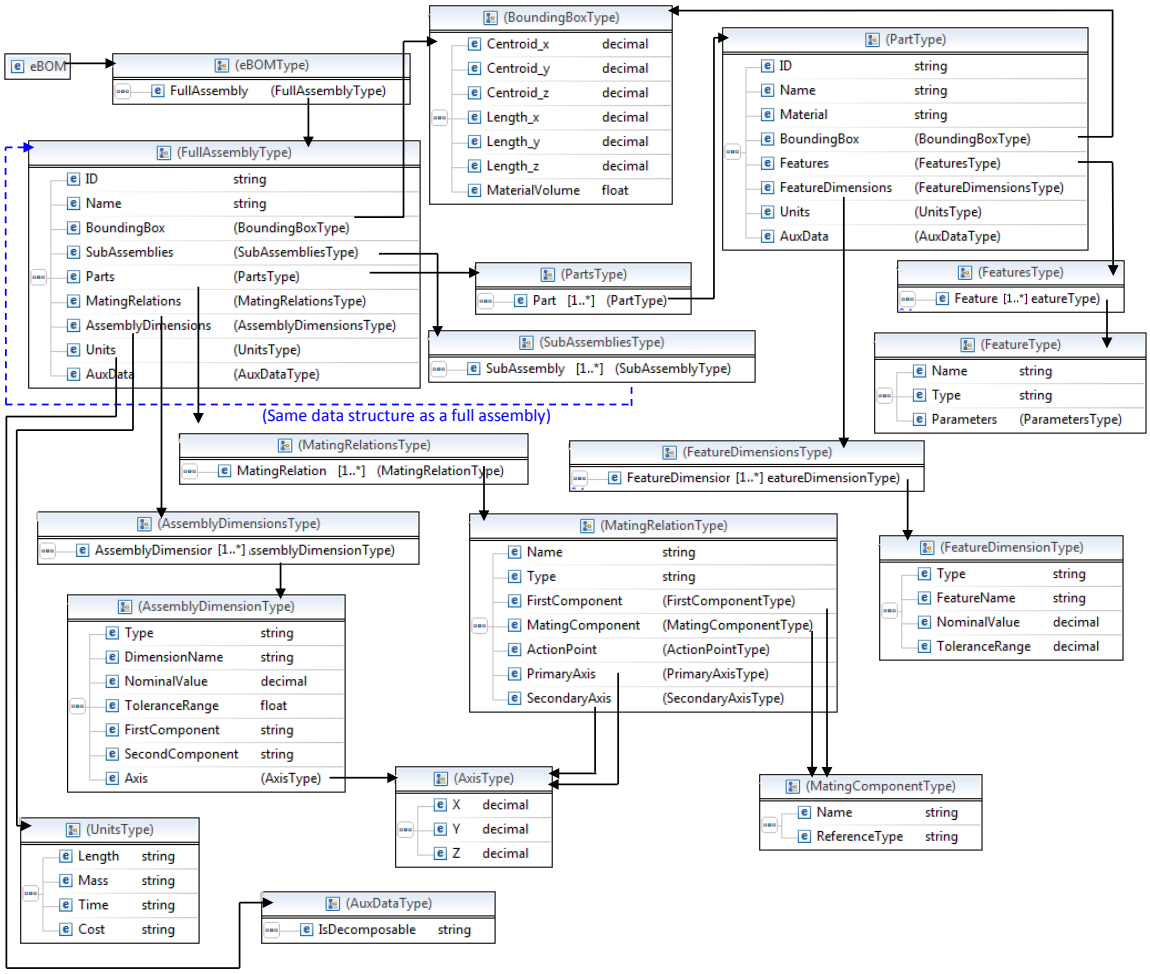


Figure UMI-2: eBOM Schema

9.3 Assembly Constraint Evaluator

Each eBOM assembly/subassembly is processed by the Assembly Constraint Evaluator (ACE) for early evaluations of dimensional quality. Since an assembly sequence can significantly affect the final product’s dimensional fidelity, the ACE module attempts to generate a ranked list of feasible assembly sequences. Sequence generation begins by first constructing a Liaison Graph representing a subassembly (Figure UMI-3) and comprising:

- Nodes representing the components
- Edges representing mating constraints between components
- Special marker links representing key characteristic (KC) dimensions between different components. KC dimensions on the same component are not affected by the assembly sequence and are thus not included in the ACE analysis

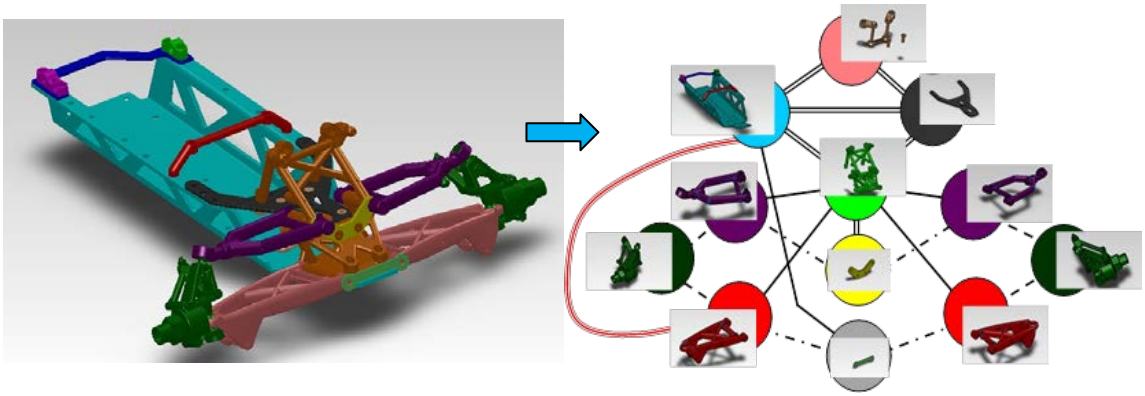


Figure UMI-3: Liaison Graph representing a META Challenge example

Sequence generation proceeds via a partial enumeration scheme using binary partitioning of the Liaison Graph. Upon making a partition, a kinematic analysis is conducted in order to check for:

- Adjustability of KCs
- Over-constraints between components.

Binary cuts are given heuristic penalty scores where over-constraints exist, and/or where KCs lack adjustability. Partitions of the Liaison Graph that still retain more than one component are also partitioned via binary cuts, and the partitioning continues recursively until the graph is fully decomposed into individual nodes. Back-tracing a sequence of binary cuts of the graph re-constitutes an assembly sequence, and summing the penalty scores of each cut provides a total score for that particular assembly sequence. Assembly sequences are then ordered on penalty scores, where top-ranked sequences have the least over-constrained steps (hopefully none), and maximum KC adjustability (expected to enable tight assembled-product tolerance without requiring excessively tight tolerances during manufacturing)

Generating the Liaison Graph from eBOM information is mostly an automated task but often, however, requires a human user's sanity check, due to ambiguity inherent in interpreting certain types of CAD mating constraints. For example, common pin-in-hole joints may have differing degree-of-freedom constraints, depending on the designer's intent and assigned tolerances.

In addition to detecting over-constraints and KC adjustability, Liaison Graph partitioning also identifies KC delivery chains, lists of the components and joints that affect each KC. Each of the top-ranked (via the heuristic penalty score) assembly sequences is then passed along with their associated KC delivery chains to the Assembly Variation Simulation tool (task UMI.2), which predicts expected variations in each assembly and the probability of satisfying required tolerances.

9.4 Tolerance-aware Assembly Sequences

With this algorithm, we seek an assembly sequence that maintains key behavioral tolerances — Key Characteristics — required in the final product.

9.4.1 Constructing the Liaison Graph

In order to construct the initial Liaison Graph, each *Component* (part or subassembly) in the subject assembly (selected from the eBOM list) is used to instantiate a liaison node. Next, mating constraints in the eBOM are translated into edges in the Liaison Graph. Each component ID in the eBOM corresponds to a unique node in the Liaison Graph. Kinematic constraints (points of action and axes) are associated with graph edges according to type of mating constraint in the eBOM. Finally, all assembly-level dimensions that have tolerance requirements in the eBOM are translated as special marker links (indicating KCs) in the Liaison Graph. Connectivity of the special marker links (termed KC-edges) is set by the terminals of the associated dimensions in the eBOM. The KC-edges also store information about the type (linear or angular) and vector direction of the KCs.

9.4.2 Generation/Exploration of Assembly Sequences

Decomposing an assembly can be modeled as a series of steps that recursively partition the Liaison Graph, and backtracing the decomposition steps reconstitutes the assembly sequence. The current implementation considers only binary decompositions and each partitions the graph into exactly two subgraphs, corresponding to exactly two *pieces* (single parts or pre-assembled part collections) placed together in a single assembly step.

Generating candidate graph partitions is done by selectively removing edges from the graph (corresponding to disassembling a joint or mating constraint). Given a graph with n edges, there are 2^n possible combinations of edge removal, though not all of them result in a binary partition. The current implementation uses exhaustive enumeration, which we tested to solve example problems with up to 18-20 edges in reasonable computational time. For larger problems, implementing a randomized search such as genetic algorithm is a conventional solution. Checking that an edge removal combination results in a binary partition of the graph is done by a standard one-to-all graph connectivity search.

A penalty score is given for each candidate partition of a Liaison Graph. The penalty score is higher if the partition corresponds to an over-constrained assembly step, which is checked by kinematic analysis of the joints that correspond to the removed edges in the graph, while considering the *pieces* in the two subgraphs as two rigid bodies. The penalty score is also slightly higher if the partition *breaks* a KC-edge without joint adjustability along the direction vector of the KC (which indicates that tolerance stack-up will happen in the assembly).

A list of candidate partitions is ranked by their penalty score, and a number of the top-ranked partitions are explored further by considering binary decomposition of the subgraphs. The process continues until all subgraphs cannot be partitioned further (graph contains only one node). Since not all partitions in a given step are explored further, the scheme is only a partial enumeration (does not explore all possible assembly sequences), but the overall runtime is $O(2^n)$, whereas a full enumeration scheme would be (2^{n^2}).

9.4.3 Analyzing Graph Decomposition Sequence

The overall penalty score for an assembly sequence is computed as the sum of penalty scores of each binary decomposition step of the Liaison Graphs that constitute the sequence. In addition to ranking sequences by the overall penalty score, the *delivery chains* of each KC are identified. The delivery chain for a KC, which is later used in the Monte-Carlo simulation in task UMI.2, is a list of components and joints whose dimensional variation affects the overall dimensional variation of the KC.

Identification of the KC delivery chain is taken into consideration during each binary portioning step of the Liaison Graph. For each KC-edge, the following is done:

- If the terminal nodes of the KC are both in only one partition (only of the two subgraphs), then the KC-edge is copied to that partition only.
- If the partition breaks the KC-edge (terminal nodes on separate subgraphs) and the kinematic analysis indicates adjustability along the direction vector of the KC, the joint is added to the list of the KC delivery chain, but the KC-edge is not copied to the subgraphs. This corresponds to adjustability during assembly that allows the variation in the KC to be just as good as the joint/fixture accuracy, irrespective of the component tolerances.
- If the partition breaks the KC-edge (terminal nodes on separate subgraphs) and the kinematic analysis indicates no adjustability along the direction vector of the KC, the joint is added to the list of the KC delivery chain and the KC-edge is copied to both subgraphs. Terminal nodes of the KC-edge are changed in the subgraphs to the nodes where the edge cuts was done to form the partition.
- At the end of the decomposition, when all subgraphs contain only one node, the nodes that have a KC-edge are added to the list of the KC delivery chain. This corresponds to the parts whose dimensional variation affects the KC.

9.5 Deliverables

- Source code (C++) for the eBOM generator
- Source code (C++) for the assembly sequence generator and analysis of over-constraints, and KC delivery chains
- Implementation into web service (http://tobiko.engin.umich.edu), and PHP source code

10. Mixed-initiative Setups – Simulating assembly variations

10.1 Task: UMI.2

Responsible organizations:

- Primary — University of Michigan

A product design for iFAB manufacture will include subassemblies and parts to be welded/joined at multiple stations. Dimensional variation in the final product accumulates as assembly proceeds. Commonly known as variation stack-up, the amount of this assembly variation depends on individual part variations as well as on the assembly sequence itself. If the stack-up exceeds Key Characteristic assembly tolerances as specified in the CAD model by the designer, the product may not function as intended. This task developed, tested, and validated software tools that can automatically simulate assembly variation.

10.2 Assembly Variation Simulator

Given the CAD data, eBOM, and a list of assembly sequences from the Assembly Constraint Evaluator as input, the Assembly Variation Simulator computes a (smaller) list of assembly sequences ranked according to the probability that assembly variations exceed KCs, and for each sequence, the sensitivities of part tolerances to the assembly variations. These outputs are fed back to the designer and also to the assembly process planner for further analysis of assembly motions.

Based on the observation that AVM vehicles are composed primarily of highly rigid parts, the Simulator regards all parts as having no springback when released from a fixture and so computes assembly variations merely thorough the kinematic chains of mating features and fixtures.

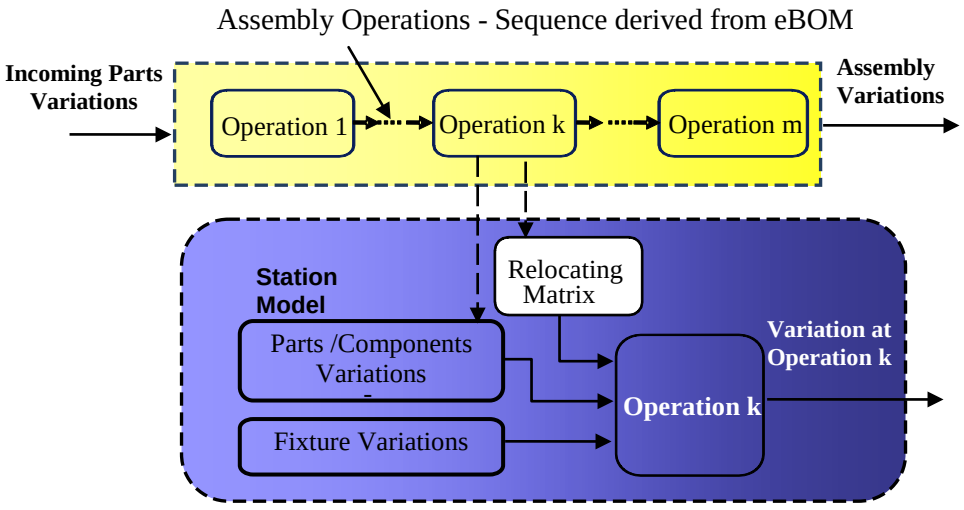
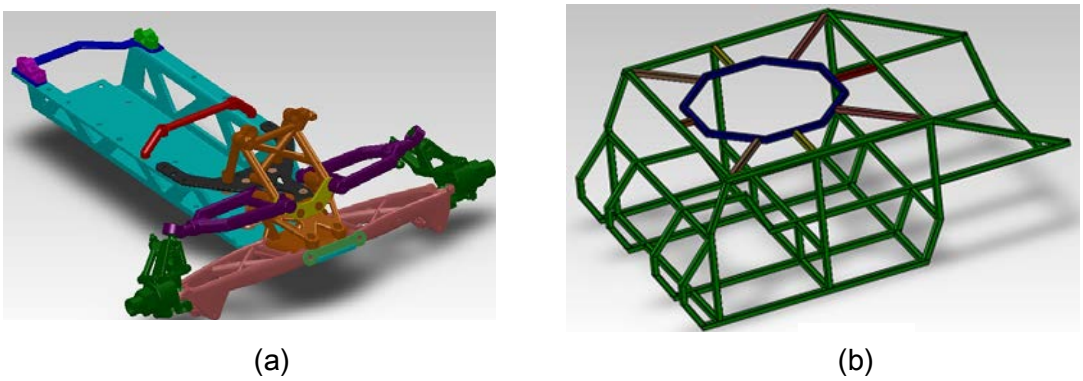


Figure UMI-4: Variation Simulation

Assembly variability is analyzed via Monte Carlo simulation of instances of the dimension projections of components and joints in the KC delivery chain (obtained via tools in UMI.1) along the direction of the assembly level KCs. Individual simulation is done at the operation level and recursively run through the entire assembly procedure until the assembly is completed, as illustrated in Figure UMI-4

10.3 Sample Results

Test studies of eBOM generator, Constraint Evaluator (ACE), and Variation Simulator (AVS) included the Vehicle Chassis structure (Figure UMI-5a) and META-Challenge HMMWV spaceframe (Figure UMI-5b). Examples of toolchain throughput for these two cases are shown in Figures UMI-6 and UMI-7



(a) (b)

Figure UMI-5: CAD Model of: (a) Vehicle Chassis, and (b) META Challenge spaceframe

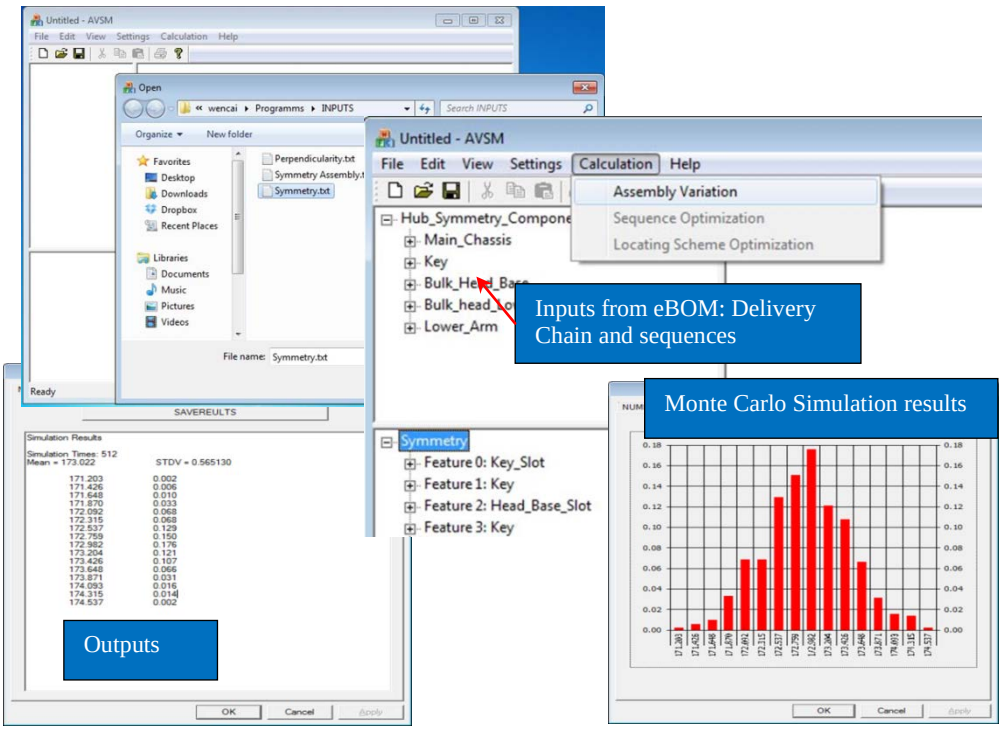


Figure UMI-6: Chassis assembly simulation (KC: symmetry of left and right hubs)

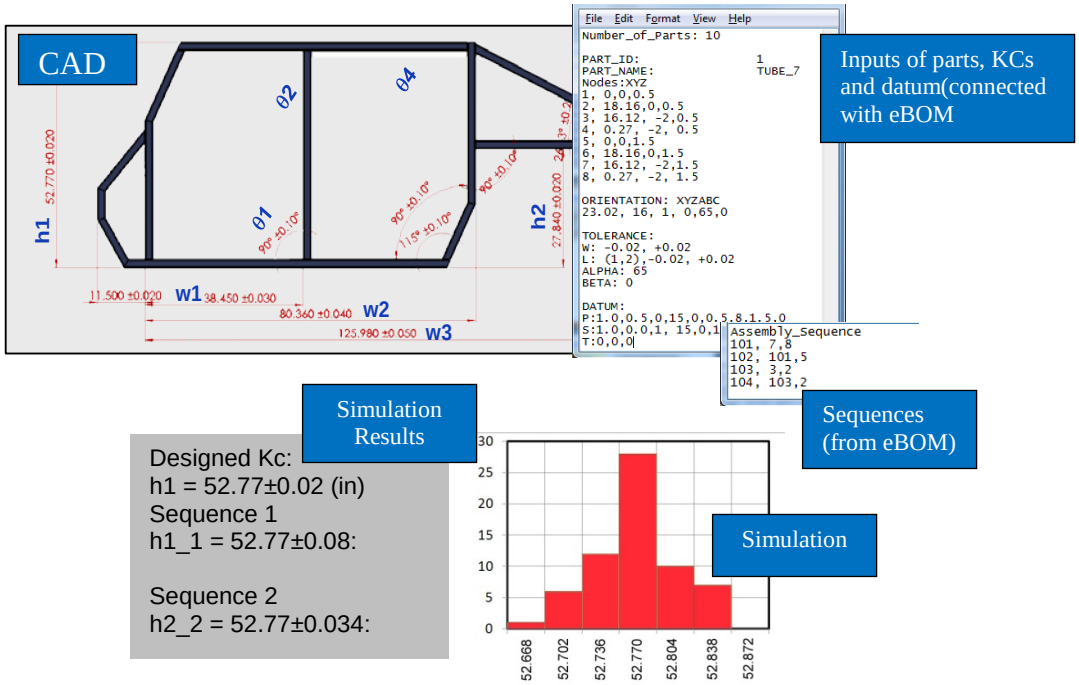


Figure UMI-7: Spaceframe fabrication example (KC: h1- Height of the Side Frame)

10.4 Deliverables

- Source code (C^{++}) for the assembly variation simulation
- Implementation into web service (http://tobiko.engin.umich.edu) and PHP source code

11. SMARTV Information Infrastructure and Process Matching

11.1 Task: LM.1

Responsible organizations:

- Primary — Lockheed Martin Space Systems Company

To achieve the AVM-target fivefold speedup over conventional manufacturing technologies, SMARTV requires an information architecture that can provide the "glue" between analysis systems, improving schedule efficiency and eliminating data-transfer errors. The challenge is to develop a system that can:

- Provide a customized interface to each AVM stakeholder
- Maintain persistent, readily accessible storage of submitted designs and analysis results
- Connect the diverse SMARTV tool suite into one automated, end-to-end system.

Another key to compressing manufacturing schedules lies in identifying the optimal process — with respect to cost, time, reliability, etc. — for a given part or assembly. Solving this process-matching problem will enable us to replace human heuristics and intuitive process-selection with a rigorous method that formally explores all options automatically.

A third leg on the SMARTV "information stool" is reducing a major contributor to schedule slip: rework resulting from infeasible designs. The challenge of analyzing manufacturability is to provide designers feedback on:

- Part features that cannot be manufactured
- Expected yield of an assembly process, given the selected-part tolerances.

11.2 The SMARTV Information System and Tool Suite

The Lockheed Martin team addressed these challenges in a two-part approach, developing:

- A SMARTV information system, user interface, information framework, and integrated software tools
- A suite of manufacturing-analysis tools that complement those developed other teams.

The SMARTV system targets several different types of users, including META, FANG, and iFAB. For META and FANG, the information system provides a method for submitting a design manufacturing query, automatically analyzing the situation, and returning feedback on cost, time, manufacturability, etc. The system allows an iFAB Foundry Manager to select the relevant machines in his Foundry, so that a given META/FANG design can be analyzed with respect to the particular Foundry configuration.

The information system consists of a web-based, front-end interface, an information storage/transfer system, and interface routines connecting the system to various analytic tools. The web interface provides portals tailored to specific needs of each user type. A multiagent framework and database implementation provide a scalable system for handling an arbitrary

number of users and tools. Interface routines seamlessly connect a design manufacturing query to the tools.

In addition to the SMARTV information system, the Lockheed Martin team developed a suite of manufacturing analysis tools. To perform process matching, we integrated the aPriori cost-estimation package into the system. We enabled authoring of manufacturing design rules so that a part can be automatically checked against each rule for violations, and we implemented several rules as demonstration. The tolerance stack-up analysis tool returns an assembly's yield, given its component part tolerances.

11.3 Key Infrastructure Components

11.3.1 Web Interface Design

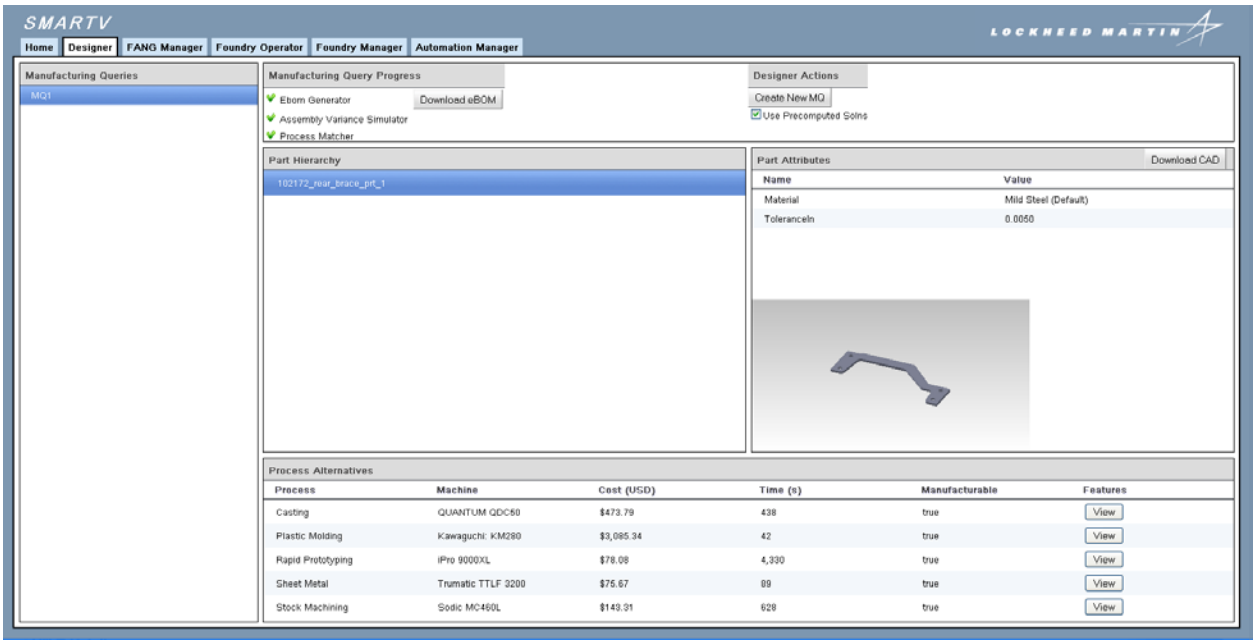


Figure LM-1: Designer interface to the SMARTV information system

Figure LM-1 depicts the SMARTV web interface, showing tabs for each user type: Designer, FANG Manager, Foundry Operator, Foundry Manager, and Automation Manager. We developed the interface using the Google Web Toolkit (GWT).

Designer Tab

The designer tab allows a META designer to submit a design for a manufacturing query and provides the interface for viewing both the submitted design and feedback from the manufacturing analysis tools. The designer defines the manufacturing query input by supplying a ZIP file containing the part and assembly CAD files as well as an XML file of metadata, such as the part material and tolerances.

The left column shows a list of the manufacturing queries submitted. The top panel allows the designer to create a new manufacturing query and to view the progress of the analysis tools. The Part Hierarchy panel shows the assembly structure of the submitted design and a single listed

part in the case of a manufacturing query for one part. The Part Attributes panel shows the attributes of the selected part as well as an image of the part. The Process Alternatives panel reports the outcome of the analysis tools used for process matching: cost and time estimation using aPriori and manufacturability analysis using the design-rule checker. Both tools are described below.

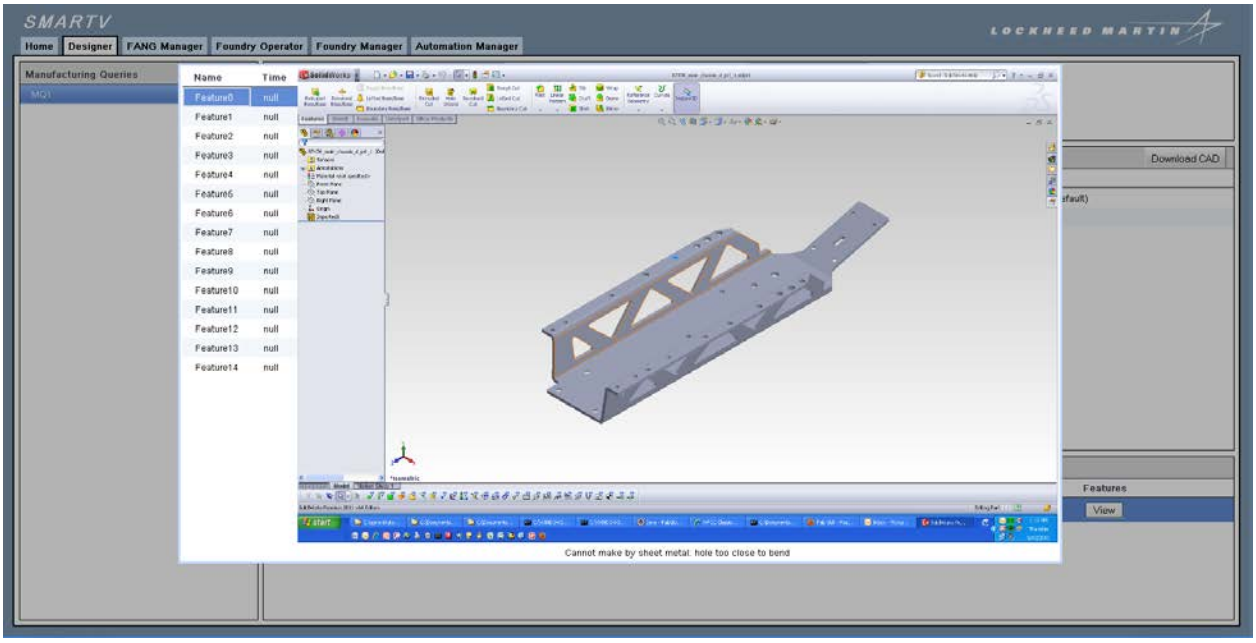


Figure LM-2: Manufacturability-analysis feedback to Designer

Figure LM-2 shows an example of manufacturability analysis feedback. In this example, the META designer has submitted the chassis part shown. The SMARTV system calls the design-rule checker (described below), which in turn automatically generates images highlighting features that violate a design rule. In this example, the rule checker finds several holes too close to a sheetmetal bend. Here the Designer observes one problematic hole.

FANG Manager Tab

The FANG Manager tab extends the Designer tab functionality by allowing a FANG Manager to analyze a design with respect to a given Foundry configuration. The tab provides two additional panels: one listing possible Foundry configurations available and another detailing the machines/processes within the selected Foundry.

Foundry Operator Tab

The Foundry Operator tab provides instructions that the UMD team's *Assembly Instruction Author* generates. Here a Foundry Operator selects the manufacturing query he is assigned and retrieves the sequence of videos that illustrate the assembly process.

Foundry Manager Tab

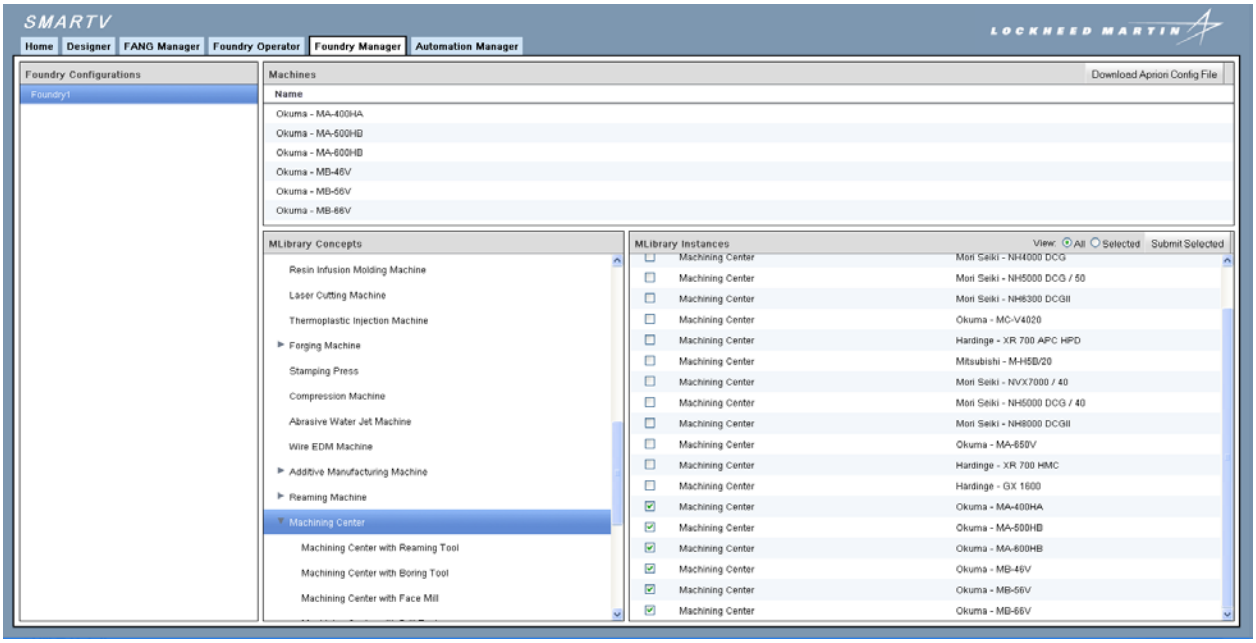


Figure LM-3: Foundry Manager interface

The Foundry Manager tab, pictured above, allows the Foundry owner to identify, from the MSysML process library, those machines specific to his particular Foundry. Upon submitting a selection of machines, a SMARTV routine creates an aPriori process library input file that can be imported into aPriori so that the process matching phase can evaluate a part with respect to the processes the Foundry Manager has selected.

Automation Manager Tab

The Automation Manager tab allows a user to retrieve the instruction generation solution, edit the solution videos, and resubmit the solution so that the Foundry operator can retrieve the solution. The instruction generation process is not completely automated. After the motion planner determines the collision-free trajectories and sequences for each part, the solutions post to the Automation Manager tab. The Manager subsequently imports the solutions into his video editor and defines the camera sequences and motions so that the instruction video is clear. He then resubmits the revised assembly-instruction videos.

11.3.2 Interface Implementation

All user interface-related classes are contained within the **com.lmco.atc.ifab.gwt** package, which is split into **client** and **server** partrs. Everything in the client package runs in the user's browser. GWT automatically translates this code into JavaScript, so web applications can run without plugins. Everything in the server package runs on the host and acts as an interface to the SMARTV services, including the JADE agents and MongoDB database.

com.lmco.atc.ifab.gwt.client.activity – Contains the logic for each panel in the interface. Acts as an intermediary between the data coming from the server and the actual UI widgets.

com.lmco.atc.ifab.gwt.client.event – Contains the events that are fired so various parts of the UI can interact asynchronously.

com.lmco.atc.ifab.gwt.client.mapper – Contains classes that map areas of the screen to activities as a function of place.

com.lmco.atc.ifab.gwt.client.model – Contains the proxy interfaces to the server-side data objects. These proxies allow separation between the server objects which do not have to be translatable to JavaScript. The proxy mechanism is carried out with GWT's RequestFactory.

com.lmco.atc.ifab.gwt.client.place – Contains classes that represent different states of the UI. For SMARTV they represent the various tabs.

com.lmco.atc.ifab.gwt.client.rpc – Contains classes for GWT's Remote Procedural Call (RPC) mechanism. This allows for the client to call methods on the server.

com.lmco.atc.ifab.gwt.client.ui – Contains some UI utility widgets.

com.lmco.atc.ifab.gwt.client.view – Contains the primary UI widgets displayed to the user. These are driven by activities. They receive data from the activities and push user generated events, e.g. clicks, back to the activities.

com.lmco.atc.ifab.gwt.client.util – Contains some boiler plate utility functions for UI widgets.

com.lmco.atc.ifab.gwt.server.dataservice – Contains the server-side classes to support the RequestFactory. Provides an interface to MongoDB.

com.lmco.atc.ifab.gwt.server.locator – Contains more server-side classes to support the RequestFactory.

com.lmco.atc.ifab.gwt.server.rpc – Contains the server-side classes to support GWT's RPC mechanism.

com.lmco.atc.ifab.gwt.server.servlet – Contains server-side classes for HTTP servlets for uploading and downloading files.

com.lmco.atc.ifab.gwt.server.uiagent – Contains an interface to the JADE framework. Interactions are performed through the UI Agent.

11.4 SMARTV Information Framework

The Lockheed Martin team employed two main technologies to realize the SMARTV Information Framework:

- The Java Agent DEvelopment (JADE) environment processes multiple manufacturing queries, parts, and analysis tools simultaneously in a distributed computing environment
- The Mongo Database provides scalable, persistent storage of manufacturing queries and Foundry specifications.

Figure LM-4 shows the overall Information Framework architecture, including user interfaces, agent based back end, and a suite of tools to process a design.

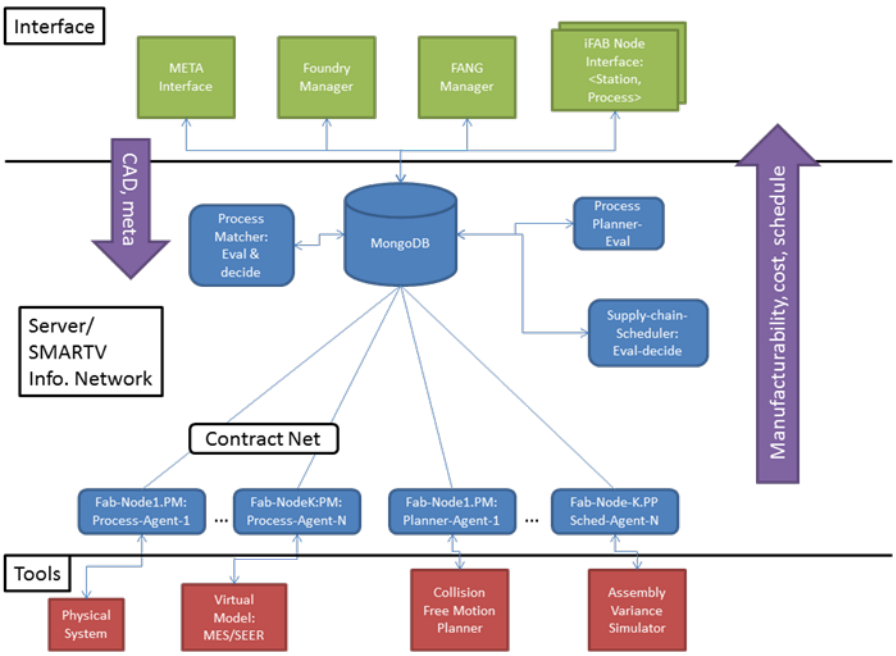


Figure LM-4: SMARTV Information Framework architecture. System input comprises CAD models of parts and assemblies as well as metadata for each part and assembly operation. Output includes manufacturability feedback and tradespace values, such as total time and cost.

11.4.1 JADE Multiagent System

A multiagent software system connects several tools in the SMARTV tool chain providing automation and organization of the process matching phase. We developed this agent based system using the Java Agent DEvelopment (JADE) Framework as it is extensible to a network of machines and platform independent. As each META designer signs in to the web-based GUI, a master agent assigns a manufacturing query agent to that designer to coordinate the manufacturability and cost/time estimating process. That agent then interfaces with the manufacturing analysis tools: eBOM generator, Assembly Variance Simulator, and the Process Matcher and reports the results via the web interface.

In addition to naturally handling the asynchronous processing of many manufacturing queries and many parts within those queries, the multi-agent based method allows for a contract net protocol to be used for distributed, automated process matching. In this scheme, each part is assigned to a broker agent who then transmits a call for proposals to all process agents. Each process agent then uses its process model (defined in aPriori as described below) to determine the cost and time to manufacture the part given its geometry and attributes (tolerance, surface finish, etc.) and the attributes of the process. Each process agent then submits the cost and time as its proposal and the part broker selects the most competitive bid based on the weighted objective function defined by the META designer.

11.4.2 Mongo Database

In order to maintain persistent storage of the information coming into the iFAB system, we are implementing a database. The database technology is MongoDB, a concurrent, scalable, widely used database solution. The MongoDB can store arbitrary file types thus supporting the storage

of both metadata in a format such as XML as well as the solid model files. Information transfer in the agent based system will be data mediated in order to maintain a constant, persistent state of the process matching, process planning, and scheduling phases. An agent will communicate with another other directly via a built in serial asynchronous protocol to notify the agent that data exists in the database to be retrieved. Thus the primary method for data transfer will be via the database.

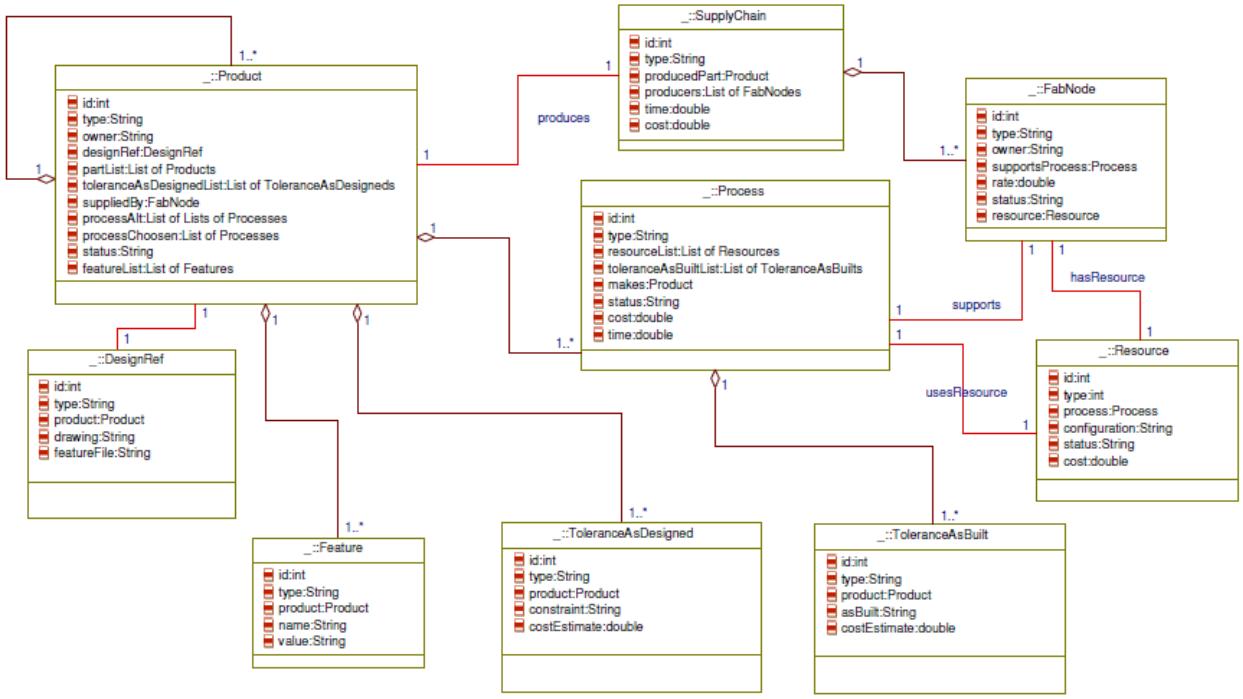


Figure LM-5: SMARTV database schema

Figure LM-5 shows the database schema, describing the skeleton of information that will be filled as the manufacturing simulation progresses. When a META design is submitted, each part and subassembly is stored as a product, where each subassembly contains a partList comprising parts and subassemblies it contains. Each product has a DesignRef object containing the user submitted files (CAD models and metadata) and a list of Feature objects each describing a user specified attribute in a predefined language. Each product also has a list of ToleranceAsDesigned objects, each containing a tolerance constraint on a given dimension or geometric relation. A product is produced by a process which has its own ToleranceAsBuilt defining the tolerances it can achieve. The process is defined by a Resource object and is associated with a specific FabNode. The SupplyChain object comprises a list of FabNodes that make up the Foundry and the SupplyChain produces a final product.

11.4.3 Information Framework Implementation

Database operations are performed using Morphia, a type-safe library for mapping Java objects to and from MongoDB. This mechanism allows storage of arbitrary Java objects, rather than Mongo's native format, JSON. Additionally, queries can be made on the Java attributes. All database related classes are in the **com.lmco.atc.ifab.data** package.

com.lmco.atc.ifab.data.DAO – Holds all Data Access Object (DAO) classes. The most important class in this package is MorphiaDAO<T>, the parent class used to query the database. Methods include: save(T), getById(String), get(), delete(T), deleteById(String), and count(). This class can be extended to provide data object-specific methods.

com.lmco.atc.ifab.data.domain – Holds all data objects. The most important class in this package is DatastoreObject, the parent class for all objects to be stored in the database. It has fields for id, name, and version as well as annotations to support the Morphia mapping. This class is extended for each database entity.

com.lmco.atc.ifab.data.scripts – The class ClearDBScript is a convenience script for clearing the contents of the database (not recoverable).

11.5 Tool Integration

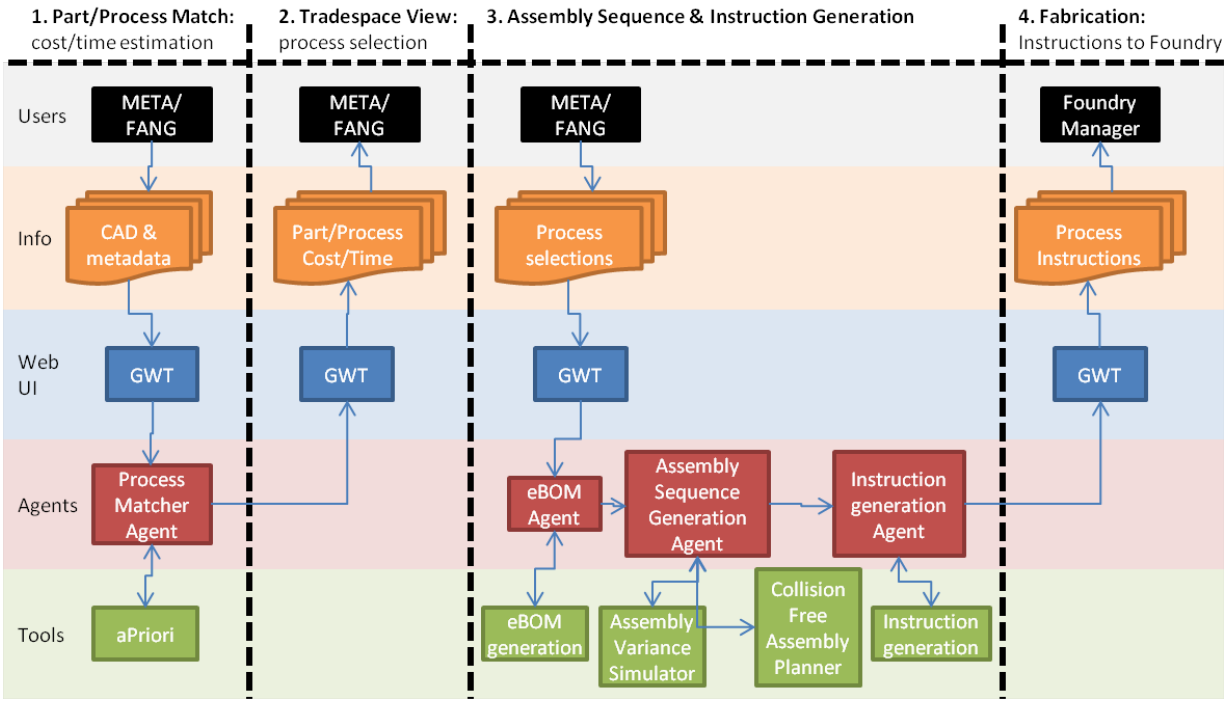


Figure LM-6: Typical flow from META design to Foundry Manager instructions

The SMARTV system seamlessly integrates the manufacturing analysis tools so that META/FANG design can be processed automatically providing rapid feedback. Further, once a design has been chosen, it integrates the instruction generation tool to provide instructions to the Foundry for assembly.

Figure LM-6 depicts a typical flow from META design, through part/assembly operation and process matching and assembly sequence generation, to generation of instructions for the Foundry Manager. For each of the four successive stages shown, the web interface mediates information flow between the users and the underlying automation system. Agents, developed using the JADE framework, parse the user information and interact with a automated tool to process the information for the next step.

For each META manufacturing query, the agent system maintains a query database object that grows as the query passes through the system and more solutions (e.g. process match, assembly sequence, manufacturing instructions) are generated. This database object uses the eBOM (CAD assembly hierarchy) schema to maintain information about parts and assemblies. As the query flows through the system, part objects are assigned processes and the top level assembly is assigned the assembly sequence and the manufacturing instructions.

In stage 1, the META user inputs the CAD files and metadata (in an XML format) to the system as a manufacturing query. The process matcher agent, interacts with the aPriori cost/time estimation software to compute the cost/time to produce each part using each process in the library. In our approach to process matching, we provide either the META user or an optimization routine the cost, time and manufacturability information for manufacturing a given part with a given process. Using the objective function the META user or optimization routine selects the lowest cost process. In stage 2, this agent returns this information to the META designer so that he can choose based on his trade study parameters. The META user then returns his process match selections to the system via the GWT interface. Stage 3 comprises a three step process: an eBOM generator creates an eBOM.XML given the CAD files, an assembly sequence generator determines a valid assembly sequence, and an instruction generation routine uses the assembly sequence to produce detail instructions for the Foundry Manager. The assembly sequence generation agent interacts with two tools: an assembly variance simulator that determines the assembly sequence with highest probability of success and an assembly planner that finds a collision free assembly sequence and part trajectories. In the final stage, the instructions pass to the Foundry Manager via the GWT interface.

The assembly sequence generation agent directly interacts with the assembly variance simulator routines developed by our University of Michigan collaborators. The assembly variance simulator takes as input an eBOM generated from the eBOM generator. Using a Monte Carlo simulation, the assembly variance simulator determines the probability that a key dimension will be within the desired tolerance hence predicting the assembly yield.

The assembly sequence generation agent also interacts with the collision free assembly planner developed by our University of Michigan collaborators. It submits the assembly files in the manufacturing query to the motion planner routine and retrieves the assembly sequence solution and stores it in the manufacturing query database entry.

Converting the assembly sequence to assembly instructions currently requires human intervention. The Automation Manager tab described above allows a user to retrieve the assembly sequence solution, import the solution into a video editor and define the camera angles and motions, and resubmit the instructions to the Foundry Manager/operator via the web interface.

11.6 Process Matching

11.6.1 Candidate Methods

The process matching problem involves determining the optimal process (with respect to the metrics: cost, time, and manufacturability) for each part and each assembly operation. To determine the process for each, we need a model with some level of fidelity that computes the metrics. The highest fidelity model would simulate the entire process to determine the required

tools and operation sequence and hence cost and time; if no such solution exists, the part/assembly operation is not manufacturable with that process. A lower fidelity model would estimate the metrics using a rule checker and a knowledge base to implement an expert system. The results would be less accurate and the manufacturability check would be an approximation – checking rules are satisfied while not being able to predict manufacturability for all parts/assembly operations with complete accuracy.

For the high fidelity approach, we reviewed several cost estimation software packages including SEER-MFG from Galorath, aPriori, and a package from Micro Estimating Services and found the aPriori solution to be superior.

A higher fidelity method for determining cost, time, and manufacturability would be to simulate the process using a commercial CAM package. To date, the state-of-the-art in Computer Aided Process Planning that links CAD and CAM to automatically generate a process plan from a part can handle a limited range of parts. Such a software package requires an automated feature recognition algorithm and the available algorithms are limited to a certain class of parts (e.g. prismatic, 2.5D milled components). Hence, such an approach would require a human in the loop to take each part and process it in a CAM package to estimate the time and tools needed and hence cost. Moreover, this approach requires ownership of CAM package for each manufacturing process (e.g. CNC, sheet-metal, FDM, etc.) As this is beyond the budget of this project, we chose not to pursue this approach.

11.6.2 Implementation

In order to perform process matching, the META designer must have access to the cost, time, and manufacturability of each part for each process. This enables the designer to perform a trade study in order to determine the process that can manufacture the part at the cost and time that is acceptable. We surveyed several cost estimating software packages and found that the system developed by aPriori to be the most comprehensive. The figure below shows an example of a part analyzed in aPriori using the SLS process. First, the program is able to estimate cost, time, and manufacturability for a part directly from its geometry and specifications without requiring the user to specify the operations. For example, for machining with a three-axis mill, the program determines cost by examining factors such as size, tolerance, roughness of the features, bounding box dimensions, weight, material type, and machine properties (e.g rapid traverse rate, tool replacement time, and power & spindle speed). While this estimation is not based on the full CAM solution for the tool paths which provides higher accuracy, this solution provides an adequate estimation automatically and in a reasonable amount of time. The estimating process is further automated by processing all parts in bulk for each process.

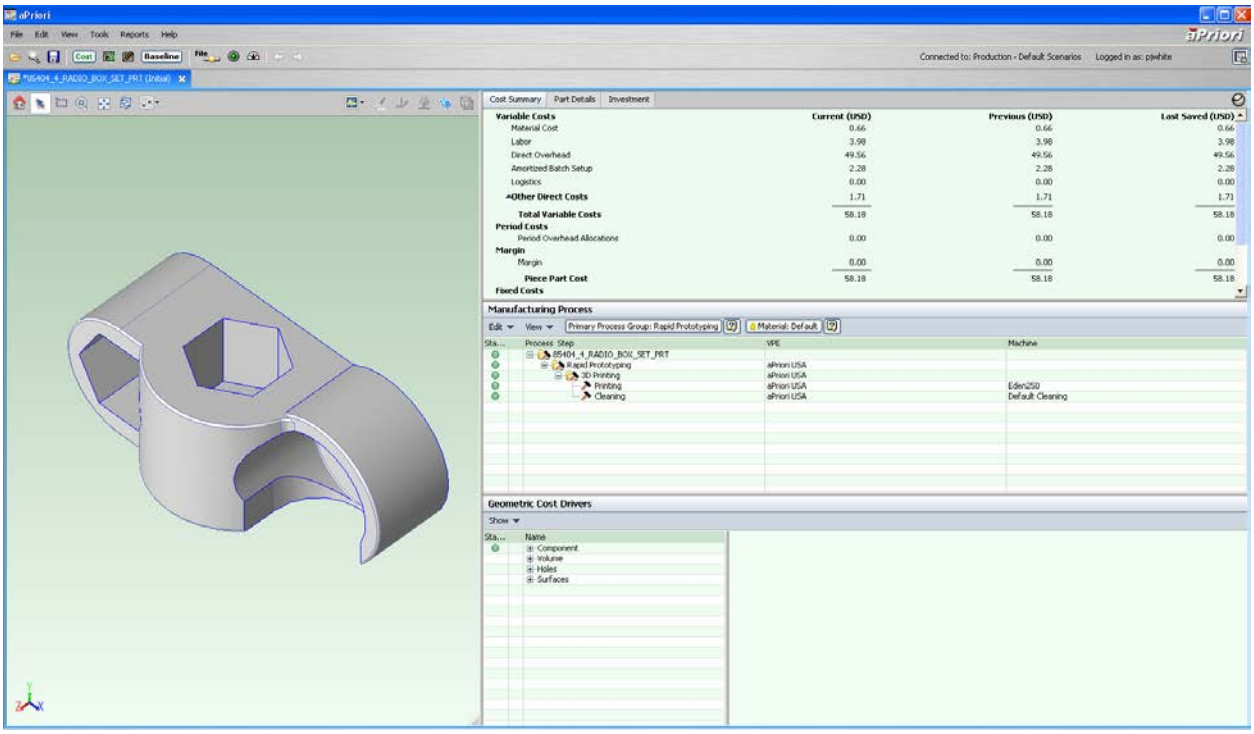


Figure LM-7: aPriori cost estimating software showing the cost estimation for a baja part using the SLS manufacturing process

The aPriori cost estimation tool comes with a range of processes (e.g. machining, molding, casting, forging, sheet metal, and welding). For each of these processes, it provides a library with several instances of machines each with the process attributes used to compute cost and time. This library is extensible: below we show integration of the MSysML database by importing several machines with their properties into the aPriori process database.

The 2012 R1 aPriori software version, released in April 2012, provides a new command line interface to the program enabling fully automated cost/time estimation. The command line interface connects to the bulk costing tool that can process many parts in batch. The user specifies the parts for estimation by directing aPriori to a directory of CAD parts. Using an Excel file, the user specifies metadata for estimation including material, process, and machine. Executing the command line with these inputs defined in an input text file, calls the aPriori bulk process tool and estimates cost/time for each part. aPriori stores the solutions in an output Excel file.

To integrate this tool with the agent system, providing full automation, we use interface routines to produce the necessary input files and retrieve the solutions. The process matching agent produces an input text file and calls the bulk costing tool from the command line. It then waits for the estimation process to complete, indicated by the cessation of the aPriori process and subsequently parses the output Excel file to retrieve the cost and time for each part. It repeats this procedure for each process in the process library. For each part, the agent adds a process feedback element containing the cost and time to produce that part for each process.

The bulk costing module allows the user to define the material, process, and process machine for each part. However, it does not allow the user to specify tolerances, flatnesses, etc. In the

future, aPriori will provide an API enabling the user to input these specifications to estimate cost based on these parameters as well. In addition, the bulk costing module does not allow for cost/time estimation for assemblies. Again, the API will provide a means for interacting programmatically with the nominal aPriori system to perform these cost/time estimations.

11.6.3 Automation

To enable processing of many manufacturing queries with respect to cost and time, we implemented several Java methods that interface with aPriori. These routines create input file for aPriori, pass this input file with the part directory to aPriori, execute aPriori, and retrieve the cost and time from the aPriori output file. aPriori uses Excel input and output files. The routines uses the Java Excel API to create Excel input files and extract the solutions from the Excel output files.

11.6.4 Routines

The SMARTV SharePoint/deliverables repository contains the Java routines used to interact with aPriori.

TestAprioriAutomation.java

The main routine in this file contains several example implementations where a part file is passed to the AprioriAutomation processMatch routine and an array of AprioriCostEstimation's are returned. Calling the toString method for an AprioriCostEstimation will return a string summarizing the cost estimation result.

AprioriAutomation.java

- costPart method

  This method takes as input a part file and two string arrays: headerRowSpecial and partRow defining the process to use in the cost estimation. It creates an aPriori input Excel file and a properties text file using methods in AprioriUtilities. It executes aPriori using the Java Runtime class and retrieves the results Excel file, importing it using the AprioriUtilities readAprioriResultsOnePart method. Finally, it prints the estimation results to the console and returns the cost estimation object.

- processMatch method

  This method takes as input a part file and calls the costPart routine for each of six processes using default settings. It returns the results in an array of cost estimation objects.

- processMatchForDemo08 method

  This method is similar to processMatch however the processes used in the cost estimation are from the MSysML database.

- getPreComputedSolution

  This method loads previously computed aPriori solutions rather than executing aPriori real time.

AprioriUtilities.java

- defaultX string arrays

  These strings hold the header and part/process information used to define the cost estimation input for aPriori.

- writeAprioriInputFile

  This method writes an input file for aPriori used by the command line interface to set the input and output files.

- writeAprioriInputXls

  This method writes the Excel input file for aPriori that defines the part and process information for estimation.

- runApriori

  This method executes aPriori and outputs the command line output to the console.

- readAprioriResults

  This method extracts the estimated cost and time from the Excel output file from aPriori for a bulk run with multiple parts.

- readAprioriResultsOnePart

  This method extracts the estimated cost and time from the Excel output file from aPriori for a bulk run with a single part.

AprioriCostEstimation.java

This class contains input and output variables for an aPriori cost estimation. The toString method returns a string summarizing the cost estimation.

ReadExcel.java and WriteExcel.java

These files contain methods to read (readRange) and write cells (write) from/to an Excel file. There are several supporting methods and a main function used for testing.

11.7 Design-rule Checking – Manufacturability Analysis

To identify parts that cannot be manufactured with a given process, the Lockheed Martin team has developed a design-rule checker that automatically evaluates part geometries. This tool allows designers to achieve a target manufacturability by reworking a product at design time, thus saving costly rework.

In order to extend the range of manufacturability rules checked and to automate the analysis, we developed a VBA add-on to the SolidWorks CAD package. The software includes a master routine that takes as input parts and analyzes each with respect to each process in the database. Currently, we support five rules for machining and sheetmetal bending. Using the SolidWorks API, we analyze the geometry by looking for features or feature relations that are not manufacturable by a given process. These rules include checks for:

- Flat-bottom drill holes
- Drilled holes with their center off the part face

- Drilled holes where drill axis and faces axis are not parallel
- Pockets without sufficiently large corner radii
- Holes too close to a bend in a sheetmetal part.

These rules can be extended to include feedback from subject-matter experts and handbooks. The SMARTV SharePoint/deliverables repository contains the SolidWorks VBA routines used for the rules.

11.8 Tolerance Stack-up Analysis

Using Monte Carlo simulation of an assembly's component parts, the Lockheed Martin team has developed a tool that can return expected assembly yield. Coordinated with manufacturability analysis, this tool enables designers to minimize rework and optimize both manufacturability and yield.

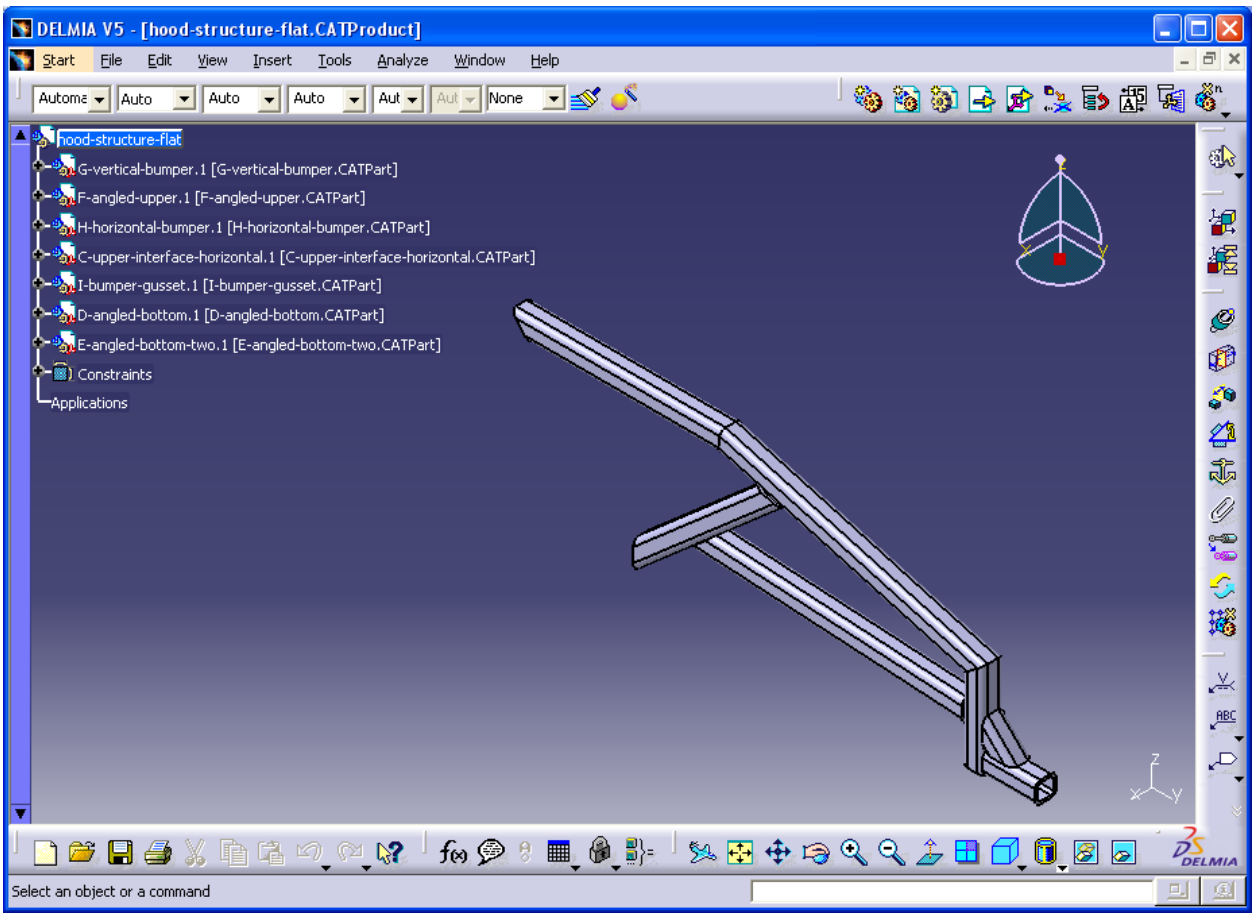


Figure LM-8: Example welding assembly analyzed by the VBA Monte Carlo simulator.

The Lockheed Martin team also used SolidWorks as a basis for analyzing welding-assembly variance. We developed a simulator that interfaces with the DELMIA CAD analysis package via an VBA routine. The routine communicates with a JADE agent to integrate with the manufacturability network. The simulator takes as input an assembly of parts each with tolerance defined on the parameters of both the parts and the assembly mates. Using the

tolerance to define the variance on these parameters, the software executes a Monte Carlo simulation of many instances of the assembly process. Given the adjustability in the assembly parameters, the software determines the probability that the assembly can be put together. We examined a welding case (Figure LM-8) where the lengths and cut angles of the square channel stock were allowed to vary in each Monte Carlo run.

11.9 Process Library / MSysML Integration

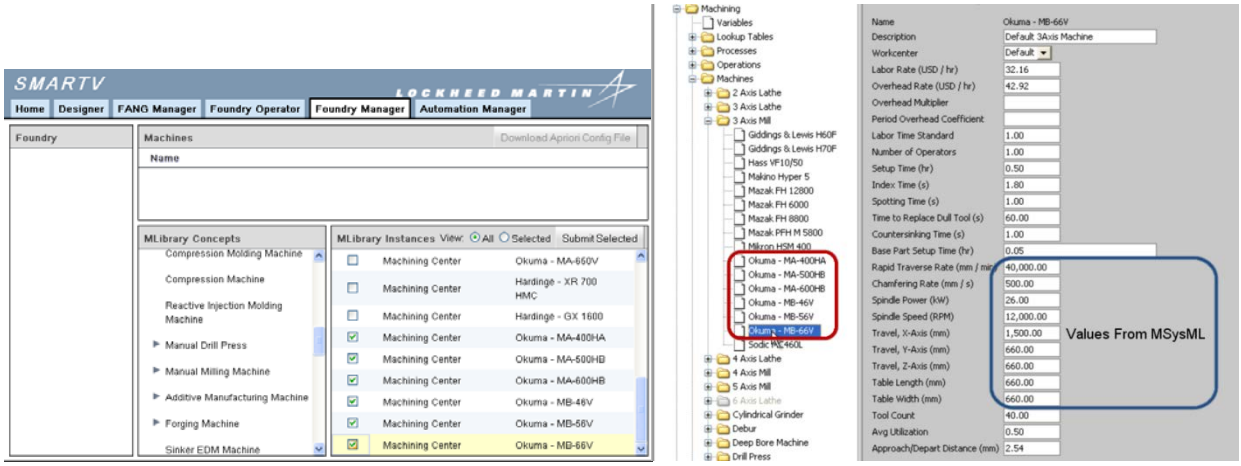


Figure LM-9: (left) Foundry Manager tab (left) allows the Manager to select machines from MSysML database. (right) SMARTV routines convert the MSysML format to the aPriori format, then imports the machines into aPriori with values defined from MSysML.

We also integrated the InterCAX-developed MSysML process library with the aPriori software. aPriori comes with a limited library of processes, each with attributes necessary for the cost/time model to compute an estimation. The process library database enables us to expand the aPriori library significantly. The key enabler for integration was determining a mapping from the InterCAX process schema to the aPriori schema. In addition, where InterCAX processes lack attributes necessary for aPriori's cost/time model, we worked with InterCAX to add these properties.

The Foundry Manager tab allows a Manager to create a Foundry configuration using a library of machines, processes, tools, etc., provided by InterCAX. The Manager specifies the processes available in his Foundry and the process matcher will perform cost/time estimation based on these processes only to determine an overall bid for that particular Foundry. In this way, the META designer can not only choose from different processes for each part but also from a list of available foundries.

A SMARTV routine automatically converts the selected machines into an format suitable for import to aPriori. When performing a cost estimation, the META/FANG designer simply selects the desired Foundry configuration and aPriori uses the processes in that Foundry to compute the cost and time for each part.

11.10 Results: Process Matching

11.10.1 Spaceframe Examples

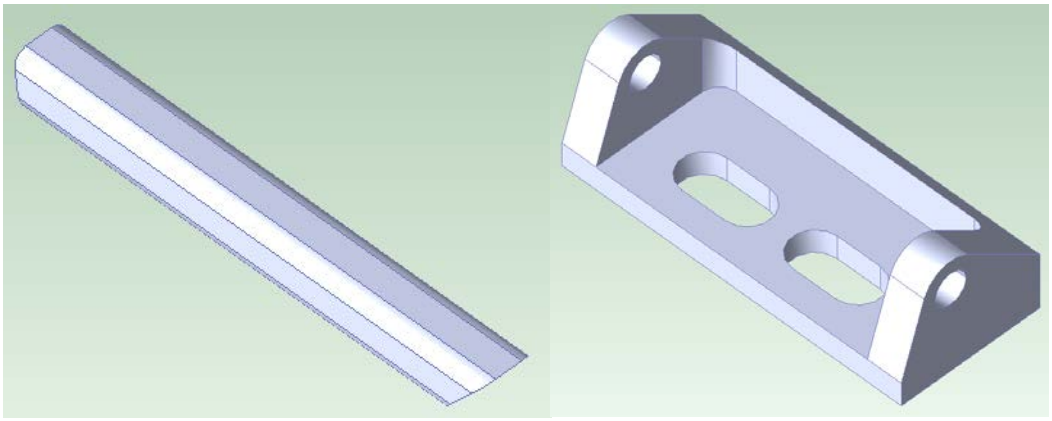


Figure LM-10: Square-channel tube (left) and hinge block (right) parts estimated in aPriori

Figure LM-10 shows two parts we analyzed in aPriori. For each part, we estimated manufacturing cost and time using the default process in aPriori. Tables LM-1 and LM-2 report the results. In each case, the intuitively expected process has the lowest cost. The first table shows that the lowest cost process for manufacturing the square channel tube is the Bar and Tube Fab process. Likewise, Stock Machining is the lowest cost, as expected, for the hinge block part. The costs are per unit and each cost estimation is for a production quantity of one unit. For the hinge block, the cost per unit significantly decreases for quantities greater than one, as the setup cost is amortized over the batch.

Process	Cost [USD]	Time [s]
Bar and Tube Fab	16.46	86
Casting	239.71	238
Forging	201.48	250
Stock Machining	316.44	8709

Table LM-1 Cost and Time estimation for square channel tube part showing Bar and Tube Fab as the lowest cost process

In many cases, aPriori can detect non-manufacturable features, based on a set of manufacturability rules. The original hinge block part file provided a chance for aPriori to detect a non-manfucturable feature. The original hinge block part did not have corner radiuses in the pocket. aPriori detected that these feature could not be manufactured with the side mill available to the three-axis CNC machine. Hence this part failed the Stock Machining cost-estimation process and the part was returned to the designer to modify for manufacturability.

Process	Cost [USD]	Time [s]
Casting	242.36	763
Forging	195.15	503
Stock Machining	127.27	1784

Table LM-2: Cost and Time estimation for hinge block part showing Stock Machining as the lowest cost process

11.10.2 Baja Example

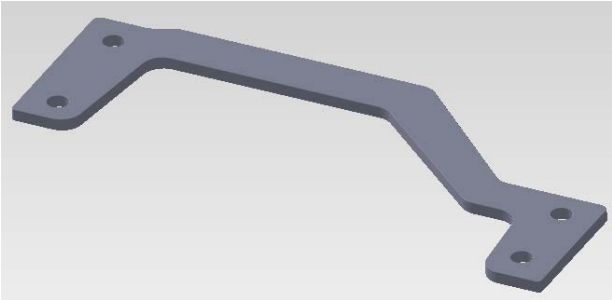


Figure LM-11: Example Baja Part

Process	Cost [USD]	Time [s]
Casting	474	438
Forging	1694	222
Plastic Molding	3256	42
Rapid Prototyping	78	4330
Sheet Metal	76	89
Stock Machining	143	628

Table LM-3: Estimated cost/time for Baja part

In this demonstration, we used aPriori to conduct process matching for the Baja part (Figure LM-11). We use aPriori to compute the estimated cost and time to produce the above part using one of six different processes. As expected, the sheetmetal process (which in this case involves only laser cutting) has the lowest cost and time.

11.11 Results: Design-rule Checking – Manufacturability Analysis

11.11.1 Baja Chassis Example

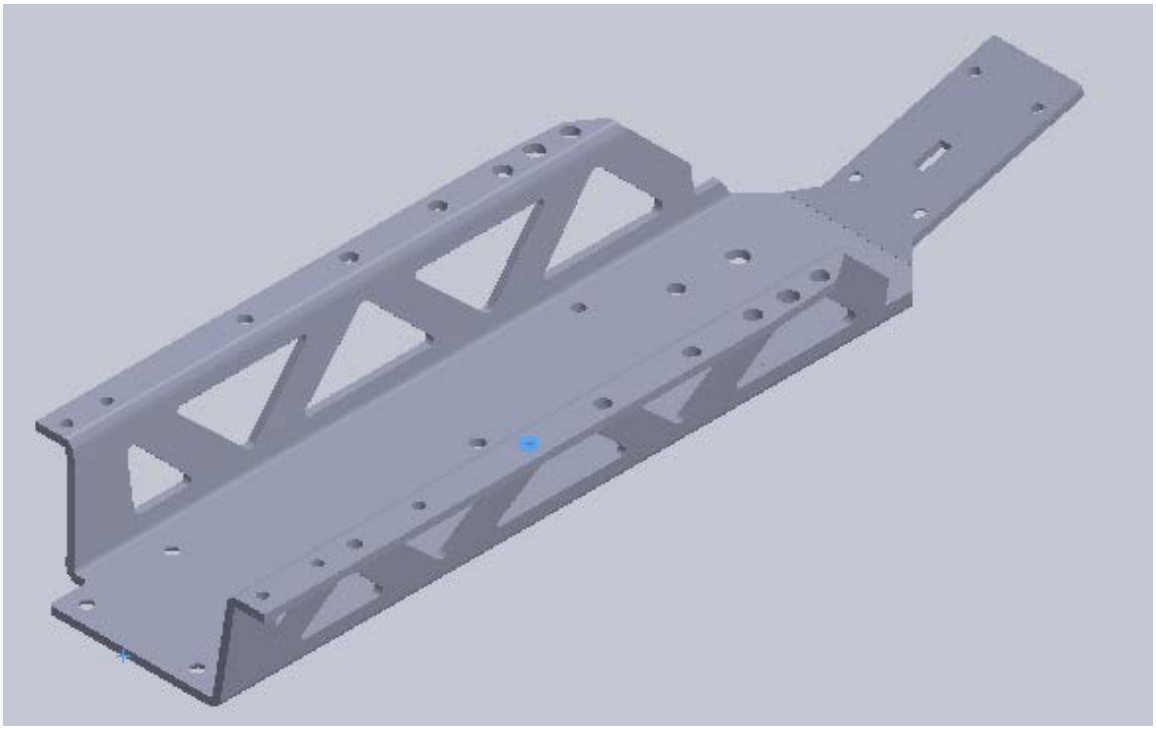


Figure LM-12: The VBA/SolidWorks Design-rule Checker finds 20 holes that are too close to a bend and creates screenshots as feedback to the designer.

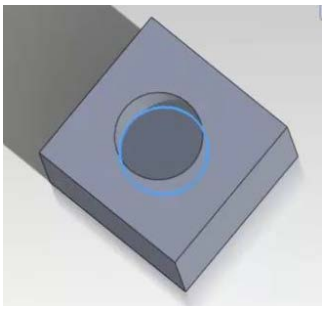
In this demonstration, we used algorithms written in VBA and integrated with SolidWorks to check geometric design-rule violations. The Checker found 20 holes too close to a sheetmetal bend that would thus deform during bending. The VBA routine finds violating holes and creates a screenshot of the hole in violation to return to the designer as feedback.

11.11.2 Primitive Examples

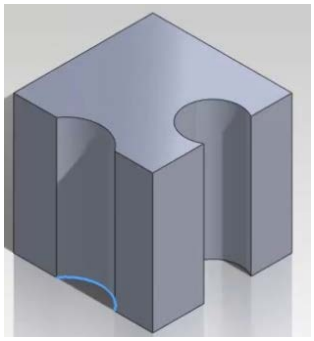
The following examples illustrate the basic functionality of each of the design rules. The parts are trivial in order to focus on the design-rule violation. The SMARTV SharePoint/deliverables repository contains the VBA routine for each design rule.

Flat Bottomed Drilled Hole

The flatBottomedHole.swp routine searches the part for holes that do not have a conic bottomed and hence cannot be manufactured using a drill bit. The figure above shows an example where the routine highlights the perimeter of a flat hole bottom.

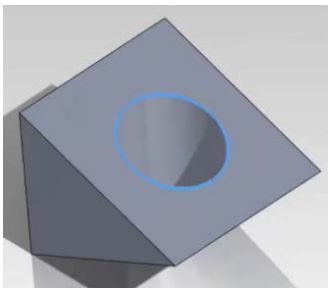


Drilled Hole Center Off Edge



The holeOnEdge.swp routine checks the geometry for holes to be drilled where the drill axis does not go through stock material. The figure above shows an example where a portion of the hole perimeter is outlined whereas the other hole in the part has its axis go through material and hence does not violate the design rule.

Drilled Hole Axis Not Normal to Plane

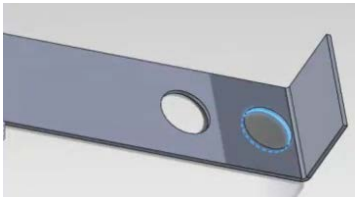


The holeOnSlant.swp routine checks for holes where one of the two hole surface profiles are non circular, indicating the hole axis is not normal to the surface to be drilled. The figure above shows an example case where a hole surface profile is highlighted indicating the routine detected a violation.

Milled Pocket without Corner Radius

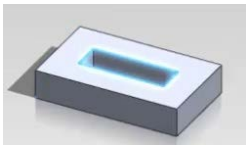
The sheetMetalHoleTooCloseToBend.swp routine checks each hole center distance to each bend to see if the following condition is true and hence the design rule is violated:

*(centerToEdgeDistance - radius) < (2 * thickness + bendRadius)*



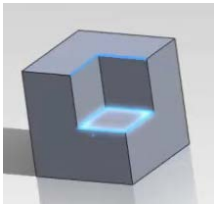
The figure shows an example where the routine finds that the highlighted hole is too close to the sheetmetal bend whereas the other hole is sufficiently far away.

Sheetmetal Hole Too Close to Bend



The nonRadiusedPocket.swp routine checks for a closed contour with any sharp corners (i.e. without a radius). The figure shows an example where the routine detects that the highlighted interior contour has sharp corners and hence cannot be milled with a conventional end mill.

Milled Notch without Corner Radius



The threeNonRadiusedCorners.swp routine checks for three intersection straight edges indicating that there is no radius and hence the feature cannot be milled with an end mill with any orientation of the part. The figure above shows an example with a notch that cannot be milled is highlighted.

11.12 Results: Tradeoff Study – Cost and Time vs. Yield

This demonstration shows how SMARTV effectively links two tools to provide trade space feedback. First, it accesses aPriori to compute the cost and time to produce a part given the required tolerance. Then it calls a tolerance stack up analysis tool (either LM's DELMIA based tool or UMI's AVS) and computes the yield for the assembly given the tolerance of the parts.

The specific example uses the seven-bar subassembly of Figure LM-19. Each bar can be manufactured from tube stock in one of three ways: band saw, band saw and machining, and band saw and grinding. Each has increasingly tighter tolerance and hence an increase in cost and time. For each process, we use the tolerance stack up analysis tool to determine the yield of assemblies that have a critical dimension fall within an acceptable range.

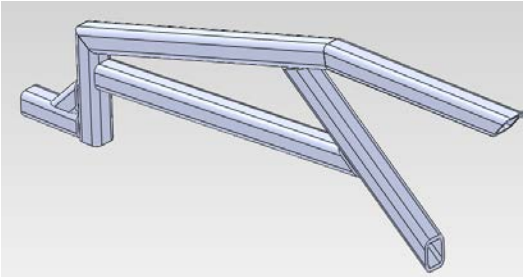


Figure LM-19: Seven-bar subassembly

As expected, the yield improves as the minimum tolerance decreases and cost and time increase as minimum tolerance decreases due to added processes (machining or grinding).

Process	Minimum Tolerance [in]	Cost [USD]	Time [s]	Yield [%]
Band Saw	0.3	146	416	20%
Band Saw and Machining	0.03	364	1279	80%
Band Saw and Grinding	0.003	388	2040	100%

Table LM-4: Assembly yield for seven-bar subassembly

11.13 Results: Process Library / MSysML Integration

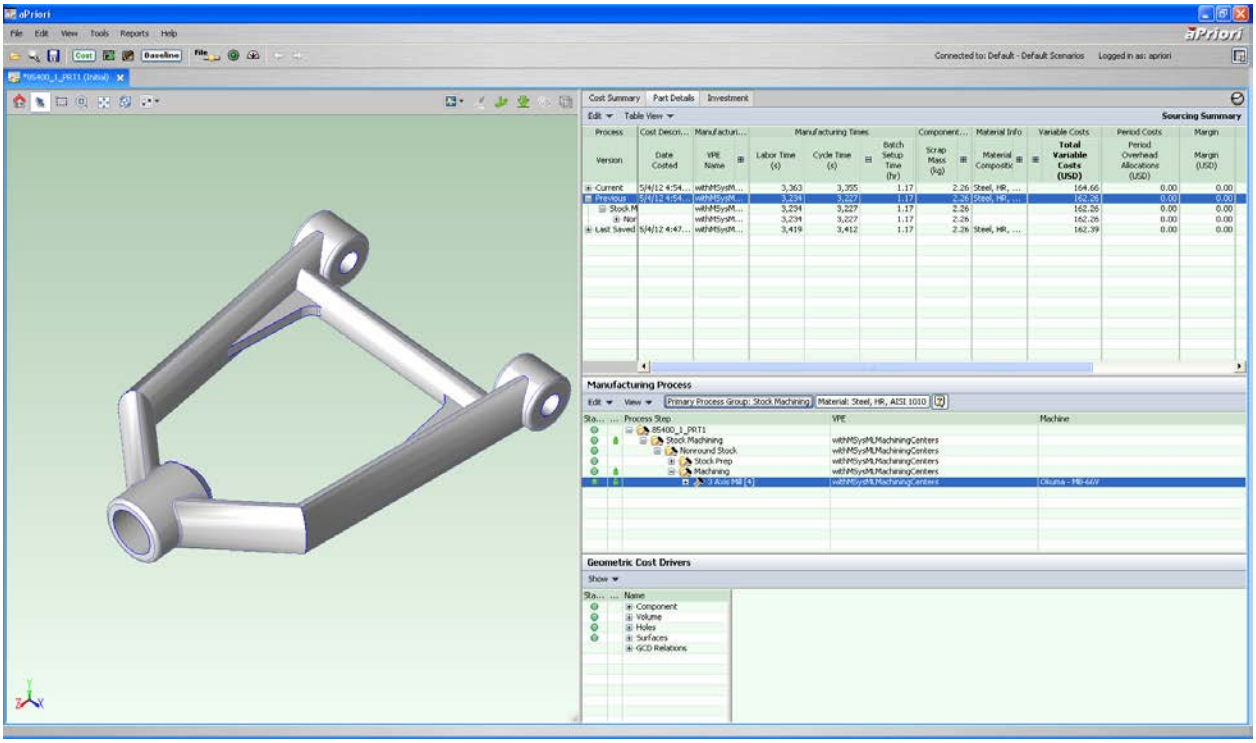


Figure LM-20: Part for milling fabrication

This demonstration shows integration with the MSysML database. In this example, the Foundry Manager selects six CNC machines to add to his Foundry and later use to cost a part to choose the best machine.

Machine (3 Axis Mill CNC)	Cost [USD]	Cycle Time [s]	Tolerance [mm]	Material
Okuma - MA-400HA	163.39	3294	0.051	Steel, HR, AISI 1010
Okuma - MA-500HB	161.63	3200	0.051	Steel, HR, AISI 1011
3354	164.63		0.25	Steel, HR, AISI 1012
3300	163.53		0.08	Steel, HR, AISI 1013
Okuma - MB-56V	162.26	3227	0.3	Steel, HR, AISI 1014
Okuma - MB-66V	164.66	3355	0.08	Steel, HR, AISI 1015

Table LM-5: Cost and time estimates for each three-axis milling machine from the MSysML to produce the Figure LM-20 part

Machine (3 Axis Mill CNC)	spindle - power rating [W]	spindle - max speed [rpm]	worktable - rapid traverse rate [mm/min]	billing rate [USD/hr]	observed tolerance - milling [mm]	axis travel - x axis [mm]	axis travel - y axis [mm]	axis travel - z axis [mm]	worktable depth [mm]	worktable width [mm]
Okuma - MA-400HA	26000	15000	60000	32.16	0.051	560	610	625	625	560
Okuma - MA-500HB	29100	25000	60000	32.16	0.051	700	900	780	780	700
Okuma - MA-600HB	28400	12000	60000	32.16	0.25	1000	900	1000	1000	1000
Okuma - MB-46V	18500	15000	32000	32.16	0.08	560	460	460	460	460
Okuma - MB-56V	11000	25000	32000	32.16	0.3	1050	560	460	560	460
Okuma - MB-66V	26000	12000	40000	32.16	0.08	1500	660	660	660	660

Table LM-6: The aPriori cost model uses the values here to estimate the cost and time for each machine

MSysML Name	aPriori Name
spindle - power rating	power
spindle - max speed [rpm]	spindleSpeed
worktable - rapid traverse rate	rapidTraverse
billing rate	workCenterLaborRate
axis travel - x axis	travelXAxis
axis travel - y axis	travelYAxis
axis travel - z axis	travelZAxis
worktable depth	tableLength
worktable width	tableWidth
name	name
observed tolerance – milling	N/A for now

Table LM-7: Mapping from the MSysML namespace to the aPriori namespace

11.14 Results: Instruction Generation Integration

This demonstration shows integration with the motion planner video playback.

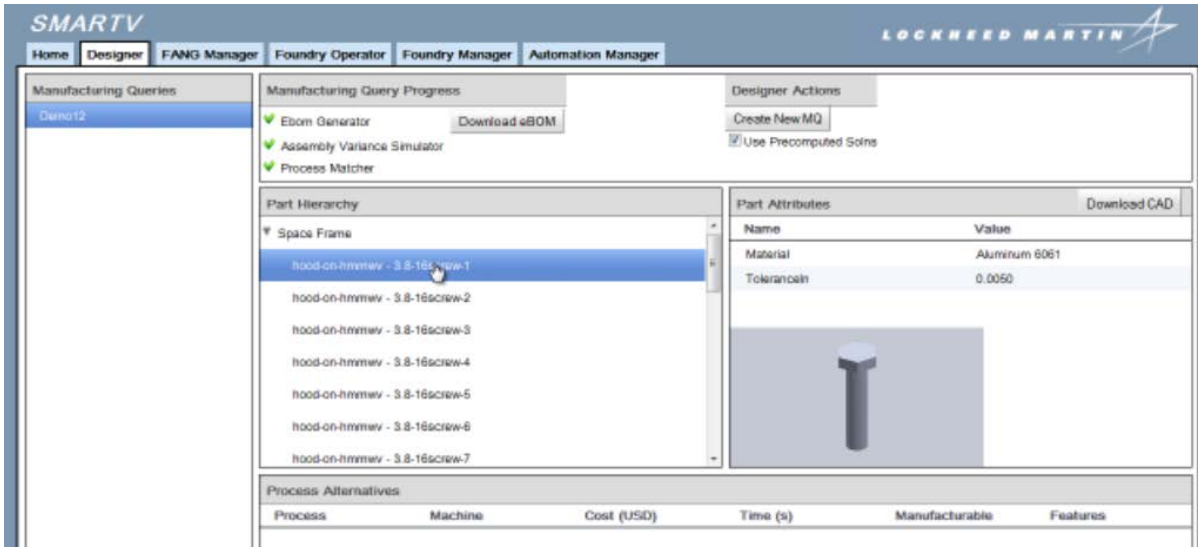


Figure LM-21: Submitting the Baja assembly

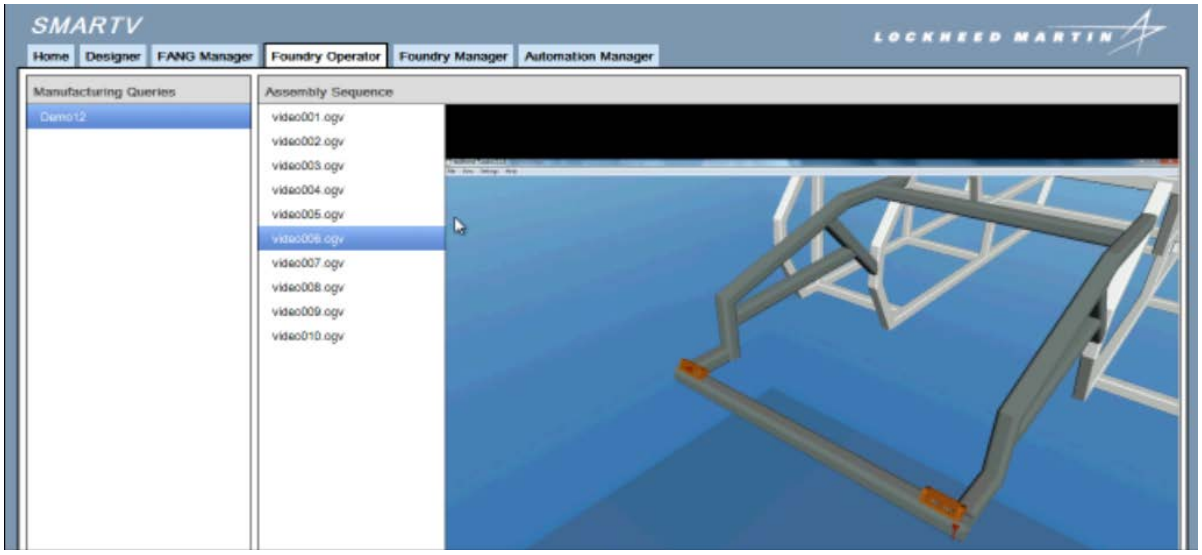


Figure LM-22: Foundry Operator can view video playback of the motion planner