Performance Evaluation of Knowledge-based Kitting via Simulation

T. Kramer[1], Z. Kootbally[2], S. Balakirsky[3], C. Schlenoff[4], A. Pietromartire[4], S. Gupta[5]

Abstract— **The IEEE Robotics and Automation Society's (RAS) Ontologies for Robotics and Automation Working Group is dedicated to developing a methodology for knowledge representation and reasoning in robotics and automation. As part of this working group, the Industrial Robots sub-group is tasked with studying industrial applications of the knowledge representation. One of the first areas of interest for this subgroup is the area of kit building or kitting. This is a process that brings parts that will be used in assembly operations together in a kit and then moves the kit to the assembly area where the parts are used in the final assembly. It is anticipated that utilization of the knowledge representation will allow for the development of higher performing kitting systems. While our previous effort aimed at designing the basis for performance methods and metrics to determine higher performing kitting systems, this paper presents a system that evaluates the performance of kitting systems through simulation using specific metrics.**

I. INTRODUCTION

Applications for assembly robots have been primarily implemented in fixed and programmable automation. Fixed automation is a process using mechanized machinery to perform fixed and repetitive operations in order to produce a high volume of similar parts. In programmable automation, products are made in batch quantities ranging from several dozen to several thousand units at a time. Although today's state-of-the-art industrial robots are capable of sub-millimeter accuracy [1], they are often programmed by an operator using crude positional controls from a teach pendant. Moreover, drastic modifications in the design process are realized when parts need major changes or become to complicated in design, thus resulting in long set up time to accommodate the new product style.

Reprogramming these robots when the aforementioned conditions arise requires that the robot cell be taken off-line for a human-led teaching period. For small batch processors or other customers who must frequently change their line configuration, this frequent down time may be unacceptable. The robotic systems of tomorrow need to be capable, flexible, and agile. These systems need to perform their duties at least as well as human counterparts, be able to be quickly re-tasked to other operations, and be able to cope with a wide variety of unexpected environmental and operational changes. In order to be successful, these systems need to combine domain expertise, knowledge of their own skills and limitations, and both semantic and geometric environmental information. The challenge of expanding industrial use of robots is through flexible automation where minimized setup times can lead in to more output and generally better throughput. Flexible robots are required to be quickly reconfigured for design and process modifications. Furthermore, they need to be information driven so that they can reliably perform different sequences of actions on command. The cost for generating the information that drives the actions must be kept down and the information must be accurate.

This paper discusses a flexible kitting system. Kitting is the process in which several different, but related items are placed into a container and supplied together as a single unit (kit). Kitting itself may be viewed as a specialization of the general bin-picking problem. In industrial assembly of manufactured products, kitting is often performed prior to final assembly. Kitting, when applied properly, has been observed to show numerous benefits for the assembly line, such as cost savings [2] including saving manufacturing or assembly space [3], reducing assembly workers walking and searching times [4], and increasing line flexibility [5] and balance [6]. In industrial assembly of manufactured products, kitting is often performed prior to final assembly.

The increasing global competition demands the manufacturing industry to move towards state-of-the-art flexible systems. Therefore, it is important not only to know how to measure the complexity of an assembly operations – the tasks that consist a plan – but also to evaluate how well these operations are carried out by a manufacturing process. Recently, some theoretical measures have been explored to evaluate an assembly sequences under different contexts. For example, Sanderson and Homem de Mello [7] used the And/Or graph approach to represent feasible assembly/disassembly sequences and a function based on parts entropy measures as evaluation criteria for search in the And/Or graph space. Park *et al.* [8] described the evaluation criteria based on four evaluation functions to minimize the chances of assembly failures caused by spatial errors of parts and assembly equipment. Bonneville *et al.* [9] presented a genetic algorithm that uses the assembly trees as model for the plans to encode them in chromosomes. The liaisons graph and the geometric constraints of the model are then used to evaluate the assembly operations. Suárez and Lee [10] proposed a statistical measure of an assembly cost index to evaluate the

[1]T. Kramer is with the Department of Mechanical Engineering, Catholic University of America, Washington, DC, USA thomas.kramer@nist.gov

[2]Z. Kootbally is with the Department of Mechanical Engineering, University of Maryland, College Park, MD, USA zeid.kootbally@nist.gov

[3]S. Balakirsky is with the Georgia Tech Research Institute, Atlanta, GA stephen.balakirsky@gtri.gatech.edu

[4]C. Schlenoff and A. Pietromartire are with the Intelligent Systems Division, National Institute of Standards and Technology, Gaithersburg, MD, USA craig.schlenoff@nist.gov & pietromartire.anthony@nist.gov

[5]S. Gupta is with the Maryland Robotics Center, University of Maryland, College Park, MD, USA skgupta@umd.edu

assembly sequences in order to determine the optimal one.

This paper presents a system that evaluates the performance of kitting applications through simulation using specific metrics. The authors have developed a canonical robot control language (CRCL) [11] that attempts to be a lowest common denominator of robot programing languages. It is anticipated that kitting plans can be translated into CRCL command sets which may then be evaluated by standardized metric software. The CRCL command sets may then be translated into a specific robot platform's language.

The organization of the remainder of this paper is as follows: Section II presents an overview of the knowledge driven methodology that has been developed for this effort. Section III describes the Kitting Viewer tool that simulates the execution of a plan (CRCL command file) for changing a kitting workstation from an initial state to a goal state. Section IV reviews the test method and metrics that are used to compare the performance of different kitting planning systems. Section V describes a configurable scoring system that uses the five factors combined as specified by a scoring file to compute the score of the plan and Section VI concludes this paper and discusses future work.

II. KNOWLEDGE DRIVEN METHODOLOGY

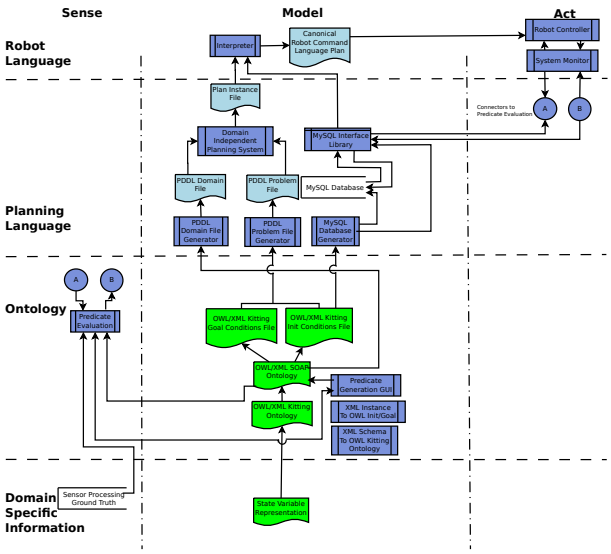

Fig. 1. Knowledge Driven Design extensions In this figure, green shaded boxes with curved bottoms represent hand generated files while light blue shaded boxes with curved bottoms represent automatically created boxes. Rectangular boxes represent processes and libraries.

The knowledge driven methodology presented in this section is not purposed to act as a stand-alone system architecture. Rather it is intended to be an extension to well developed hierarchical, deliberative architectures such as 4D/RCS [12]. The overall knowledge driven methodology of the system is depicted in Figure 1. The figure is organized vertically by the representation that is used for the knowledge and horizontally by the classical sense-model-act paradigm of intelligent systems. The remainder of this section gives a brief description of each level of the hierarchy to help the reader understand the basic concepts implemented within the system architecture. The reader may find a more detailed description of each component and each level of the architecture in other documented publications [13], [11].

On the vertical axis, knowledge begins with Domain Specific Information (DSI) that captures operational knowledge that is necessary to be successful in the particular domain in which the system is designed to operate. This includes information on items ranging from what actions and attributes are relevant, to what the necessary conditions are for an action to occur and what the likely results of the action are. The authors have chosen to encode this basic information in a formalism know as a state variable representation [14].

The information encoded in the DSI is then organized into a domain independent representation. A base ontology (*OWL/XML Kitting*) contains all of the basic information that was determined to be needed during the evaluation of the use cases and scenarios. The knowledge is represented in as compact of a form as possible with knowledge classes inheriting common attributes from parent classes. The *OWL/XML SOAP* ontology describes not only aspects of actions and predicates but also the individual actions and predicates that are necessary for the domain under study. The instance files describe the initial and goal states for the system through the *Kitting Init Conditions File* and the *Kitting Goal Conditions File*, respectively. The initial state file must contain a description of the environment that is complete enough for a planning system to be able to create a valid sequence of actions that will achieve the given goal state. The goal state file only needs to contain information that is relevant to the end goal of the system. For the case of building a kit, this may simply be that a complete kit is located in a bin designed to hold completed kits.

Aspects of this knowledge are automatically extracted and encoded in a form that is optimized for a planning system to utilize (the Planning Language). The planning language used in the knowledge driven system is expressed with the Planning Domain Definition Language (PDDL) [15] (version 3.0). The PDDL input format consists of two files that specify the domain and the problem. As shown in Figure 1, these files are automatically generated from the ontology. The domain file represents actions along with required preconditions and expected results. The problem file represents the initial state of the system and the desired goal. From these two files, a domain independent planning system [16] was used to produce a static *Plan Instance File*.

Once a plan has been formulated, the knowledge is transformed into a representation that is optimized for use by a robotic system (the Robot Language). The interpreter combines knowledge from the plan with knowledge from the MySQL database to form a sequence of sequential actions that the robot controller is able to execute. The authors devised a canonical robot command language (CRCL) in which such lists can be written. The purpose of the CRCL is to provide generic commands that implement the functionality of typical industrial robots without being specific either to the language of the planning system that makes a plan or to

the language used by a robot controller that executes a plan.

III. KITTING VIEWER

A software tool named the "Kitting Viewer" has been developed that simulates the execution of a plan (CRCL command file) for changing a kitting workstation from an initial state to a goal state. Usually, that is a plan for making one or more kits. The Kitting Viewer evaluates the plan as it runs. The Kitting Viewer source code is in C++ with OpenGL graphics.

A. Overview

The Kitting Viewer reads files describing the initial state, the goal state, and the plan for getting from the initial state to the goal state. Optionally, it also reads a scoring file. If no scoring file is specified by the user, a hard-coded default scoring file structure is used. The Kitting Viewer simulates execution of the plan, displays a view of the plan being executed, and produces and displays metrics about the plan. All of the metrics are numbers. All but one of the metrics are objective and require no human judgement. These metrics are presented in Section IV. The final metric is a subjective combination of the other metrics in which the other metrics are weighted and combined as specified by the scoring file. The scoring file may be edited as desired by the user. Scoring details are given in Section V.

The Kitting Viewer runs in two phases. In the first phase, each time the user gives a signal (presses the `g` key) the next command from the command file is executed. The Kitting Viewer may decide that a command cannot be executed, but if it decides a command can be executed, it assumes the command is executed properly. In the second phase, which starts after all commands have been executed, each time the user gives a signal the position of the next movable object in the goal file is checked.

Figure 2 shows the Kitting Viewer windows as they look after a test run has been completed. The display uses three windows, labeled Metrics & Settings, Kitting Viewer, and Kitting Command & Messages. The windows may be moved and re-sized independently, like other windows in a typical windowing system.

In Figure 2, the large blue object is a work table. The small blue object is the end effector changing station. The three empty small green boxes are parts trays that formerly held parts. The large green box on the left is a container for completed kits; it contains one kit. The large empty green box on the right formerly held an empty kit tray.

The Kitting Viewer window shows a 3D animated color view of the kitting workstation. The floor of the workstation is covered with a grid. The robot in the workstation is represented by a gantry robot spanning the entire width of the workstation. The gantry robot moves when any CRCL motion command is executed. The speed at which the picture of the robot is animated matches the actual commanded speed of the robot. Objects in the workstation move if the robot moves them. The view in the window may be translated, rotated, or zoomed at any time.

In the first phase of running the Kitting Viewer, the Kitting Command & Messages window shows the currently executing command or the most recently executed command, if no command is currently executing. In the second phase, the window shows messages describing success or failure in locating goal objects.

The Metrics & Settings window shows 12 (first phase) or 15 (second phase) metrics at the top. Below that it shows 13 robot settings and two Kitting Viewer settings. All but two of the robot settings correspond to items that may be set using CRCL commands. The extra two are the robot's maximum speed and maximum acceleration, which may not be reset. As commands are executed, metrics and settings are updated in the window.

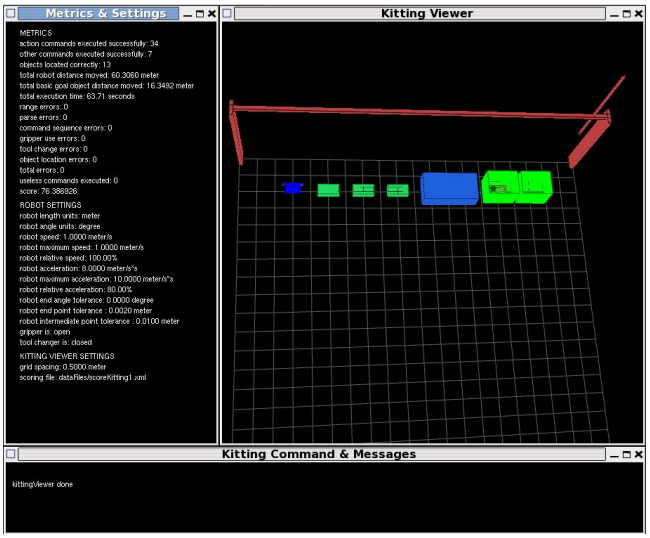

Fig. 2. Kitting Viewer Display

B. Errors

In order to fully evaluate a plan file, usually, if a plan error is detected, an error is recorded, but the Kitting Viewer continues to operate. A few errors are fatal.

When the command parser encounters a line that it cannot parse, it adds an UnreadableMsg to the list of commands it has parsed. The UnreadableMsg includes the text of the line on which the parse error occurred. When the UnreadableMsg is executed, the value of parse errors is increased by one and the UnreadableMsg is displayed in the Kitting Command & Messages window so the user can see the line that caused the problem.

C. Controlling The Kitting Viewer

Controlling the Kitting Viewer is accomplished by using the mouse and single keys on the keyboard. When the Kitting Viewer starts up, a set of one-line instructions is printed in the terminal window from which the Kitting Viewer was started. Those instructions have the same meaning as the longer explanations given below.

- `r` key – the `r` key toggles the behavior of the left mouse button between translating and rotating the picture. This is needed so two-button mice will work.
- Left mouse button – By default, the left mouse button is used to translate the picture.
- Middle mouse button – The middle mouse button is used to rotate the picture.
- Right mouse button – The right mouse button is used to zoom in or out.
- `h` key – If the `h` key is pressed, the view in the Kitting Viewer window returns to its original position.
- `g` key – If the `g` key is pressed when the plan is not completely executed and no command is executing, the next command in the plan is executed. If the `g` key is pressed when a command is in progress, it has no effect. If the `g` key is pressed when the plan is completely executed but not all goal objects have been checked, the position of the next goal object is checked.
- `t` key – If the `t` key is pressed, a combined image of all the windows will be saved in a file. The name of the file will be kittingViewer_N.ppm, where N starts at 0000 and increases by 1 each time the t key is pressed. The ppm (portable pixmap) format is a common graphics format that many graphics utilities can handle.
- `z` or `q` key – if the `z` or `q` key is pressed, the Kitting Viewer program exits, and the windows disappear.

D. Modeling

The project in which the Kitting Viewer was developed has modeled the state of a kitting workstation in both OWL and XML schema language. The XML model is kitting.xsd. An automatic code generator named the GenXMiller written by the authors at NIST is being used in the project. The GenXMiller will read an XML schema and produce C++ classes and a parser for XML data files corresponding to the XML schema. The GenXMiller was used to produce the C++ class model of a kitting workstation and the state file parser used in the Kitting Viewer. The initial and goal state files used by the Kitting Viewer are XML data files corresponding to the kitting.xsd XML schema.

The Kitting Viewer does a great deal of modeling while it runs. When the initial state file is read, a model of the initial state is built and saved as the current state of the workstation. A model of the goal state is also built and saved. As the Kitting Viewer runs and CRCL commands are executed, the Kitting Viewer determines the effect of executing each command on the current state and updates it. The second phase of Kitting Viewer operation compares the evolved current state with the goal state.

The logic of state changes is complex in some cases. The most interesting cases involve what to do when executing CloseGripper and OpenGripper commands. Details for CloseGripper are given below.

A key issue is that composite objects may go out of existence or come into existence while kits are being built. A kitting workstation builds kits. The kits do not exist in the initial state but they do exist in the goal state, so it is necessary to make them start to exist at some point in the process. The initial state includes parts trays with parts. When all the parts are removed from a parts tray with parts, it goes out of existence. Of course the parts tray remains, but it is no longer a component of a parts tray with parts. In the Kitting Viewer, a kit comes into existence when the first part is placed in a kit tray, and a parts tray with parts goes out of existence when the last part is removed from it.

For CloseGripper, if all of the following hold:

1. The robot is holding an end effector.
2. The end effector is a single cup vacuum gripper (that's the only kind of gripper the Kitting Viewer knows how to use).
3. Either:

   3A. There is a parts tray or kit tray with a topless boxy shape such that the gripper cup is within 0.1 mm of the bottom of the tray and is within 1 mm of the XY location of the origin of the tray. OR

   3B. There is a part with a boxy shape with top such that the gripper cup is within 0.1 mm of the top of the part and is within 1 mm of the XY location of the middle of the top of the part.
4. The gripper is able to pick up that type of part or tray.
5. The Z axis of the gripper is 0,0,-1 and the Z axis of the object is 0,0,1.
6. The gripper is open (implying the gripper is not holding anything).

Then the gripper will attach to an object. Call it B.

- If 3B above occurred, then B is a part
- If 3A above occurred with a parts tray in a parts tray with parts, then B is the parts tray with parts.
- If 3A above occurred with a parts tray not in a parts tray with parts, then B is the parts tray.
- If 3A above occurred with a kit tray in a kit, then B is the kit.
- If 3A above occurred with a kit tray not in a kit, then B is the kit tray.

When the gripper attaches to an object, the primary state changes (i.e. changes in states present in kitting.xsd) are that the gripper is closed, and the pose of the object is changed so that the object is located relative to the gripper. In addition, as described above, if the object is the last part in a parts tray with parts, the parts tray with parts will go out of existence. The Kitting Viewer has several other state variables for positions to make its work more efficient, and these are also updated.

IV. TEST METHODS AND METRICS

The Kitting Viewer embodies a test method for comparing the performance of different kitting planning systems. It may be used to compare different plans for building kits in which each plan to be compared has the same initial and goal states. The plans are an ordered sequence of actions for a robot to perform specified in terms of the CRCL.

The metrics currently implemented in the Kitting Viewer are described below. All but three are evaluated at the end

of each CRCL command with cumulative values. After all commands have been executed, the Kitting Viewer goes through the basic goal objects one at a time and evaluates the other three metrics (Objects Located Correctly, Total Basic Goal Object Distance Moved, and Object Location Errors) cumulatively. A basic goal object is a movable solid object in the goal file that cannot be decomposed into other solid objects. For example, a Part is a basic object, but a Kit is not (since it may be decomposed into a KitTray and Parts).

- Action Commands Executed Successfully – the number of action commands that have been executed successfully so far. An action command is any command that takes time to execute.
- Other Commands Executed Successfully – the number of commands that are not action commands that have been executed successfully so far – mostly setting commands. Executing these commands is assumed to take a negligible amount of time.
- Objects Located Correctly – the number of objects that were moved from their initial location to the correct goal location. This is not evaluated until all CRCL commands have been executed since the location of an object may change until the last command is done.
- Total Robot Distance Moved – the total distance that the tool tip has moved so far. This is calculated as the total of the distances between points in the move commands, taken in order (and starting at the place where the controlled point is located initially). The value is updated as each point is reached, not continuously.
- Total Basic Goal Object Distance Moved – the total net distance moved from initial location to final location by basic goal objects whose location has been checked so far.
- Total Execution Time – the total time taken so far by executing action commands. In the Kitting Viewer, this does not include any time that may elapse between when one command finishes execution and when the user tells the system to execute another command. The total execution time is meant to be very close to the actual amount of time that would be taken by the system without user intervention.
- Range Errors – the number of times a command tries to set a parameter to a value that is out of the allowed range of the parameter.
- Parse Errors – the number of lines in the CRCL command file that cause an error in the command file parser.
- Command Sequence Errors – the number of commands that are out of sequence. An InitCanon command is out of sequence if it is not the first command in the file. An EndCanon command is out of sequence if it is not the last command in the file. Other commands are out of sequence if they occur before InitCanon or after EndCanon.
- Gripper Use Errors – the number of OpenGripper and CloseGripper commands that cannot be executed because the robot is not holding a gripper.
- Tool Change Errors – the number of OpenToolChanger and CloseToolChanger commands that cannot be executed. For example, it is an error to attempt to execute either command anywhere but at a tool holder.
- Object Location Errors – the number of movable goal objects whose final position is not the one given in the goal file.
- Total Errors – the sum of the Range Errors, Parse Errors, Command Sequence Errors, Gripper Use Errors, Tool Change Errors, and Object Location Errors.
- Useless Commands Executed – the number of commands executed that do not change the state of the workstation. For example, executing a CloseGripper command when the gripper is already closed does not change the state of the workstation. Such commands have no effect, so they are useless. They are not counted as errors, but the scoring system may penalize executing them.

Because the CRCL commands include only primitive commands such as OpenGripper, CloseGripper, and Move and does not include task commands such as PickUpObject and MoveObject, the metrics do not include any measures of success or failure at picking up objects or moving them. For example, if the robot moves to a suitable position for picking up an object and closes the gripper, but the gripper is not suitable for picking up the object, the object is not picked up, but no error is recorded. In such cases, an error will almost certainly be found later in the position of the object that was not moved; in addition, the scoring system is likely to penalize the wasted motion.

In the Kitting Viewer, after all commands have been executed and the positions of the goal objects are being checked, the scoring system displays a running total score. How this is done is covered in Section V.

V. SCORING

The Kitting Viewer has a configurable scoring system that uses the following five factors combined as specified by a scoring file to compute the score. The factors and the score are recomputed after each movable goal object is checked.

- Right Stuff – The Right Stuff value is [(the number of objects in the goal file placed correctly so far) minus (the number of objects in the goal file placed incorrectly so far)] divided by [the number of objects in the goal file checked so far]. If that is less than zero, it is set to zero.
- Command Execution – The Command Execution value is the fraction of all commands in the command file that were executed correctly. That is [(the total number of commands executed successfully) divided by (the total number of commands executed successfully plus all errors except location errors)]. This factor does not change during the second phase of Kitting Viewer operation.
- Distance – The Distance value is [(two times the total distance moved from initial position to goal position by all basic goal objects that have been checked so

far) divided by (the total distance moved by the robot times the fraction of movable objects that have been checked)]. If the calculated Distance value is greater than 1, it is set to 1. The numerator is a crude measure of a short distance to move the robot in order to move the objects that have been moved so far to their goal positions.

- Time – The Time value is [(the teleport time) divided by (the total execution time multiplied by the fraction of movable objects that have been checked)]. If the calculated Time value is greater than 1, it is set to 1. The teleport time is [(two times the total distance moved from initial position to goal position by all basic goal objects that have been checked so far) divided by (the robot maximum speed)]. The teleport time is a crude measure of a fast time for moving the objects that have been moved so far to their goal positions.
- Useless Commands – The Useless Command factor is the value of the useless commands metric. This factor does not change during the second phase of Kitting Viewer operation.

A scoring file used in the Kitting Viewer is an XML data file conforming to the scoreKitting.xsd XML schema file. C++ classes and a parser for scoring files were generated automatically using the GenXMiller mentioned earlier.

The scoring file, as prepared by a user, designates each of the five factors as being either multiplicative or additive. For additive factors, a weight may be assigned in the file. For each factor, a valuation function may be assigned. A valuation function takes a raw factor with an arbitrary range and produces a number between 0 and 1. The final stage of producing a score combines numbers between 0 and 1. If a raw factor (useless commands, for example) is not necessarily between 0 and 1, a valuation function must be assigned to that factor. If a raw factor is always between 0 and 1 (right stuff, for example), a valuation function may be assigned, but is not required.

The scoring system is designed so that the score it produces is always between 0 and 100.

Each factor is designated as additive or multiplicative. A factor value V_i between 0 and 1 is found for each additive factor and a factor value U_i between 0 and 1 is found for each multiplicative factor. Each additive factor is assigned a non-negative weight W_i. An additive score S_a is produced by multiplying each additive value by its weight, adding the products together, and dividing by the sum of the weights. If there are no additive factors or their weights are all zero, $S_a = 1$.

$$S_a = \frac{(V_1 \times W_1) + (V_2 \times W_2) \cdots + (V_n \times W_n)}{W_1 + W_2 + \cdots + W_n}$$

The value of S_a will be between 0 and 1 since all the components of the equation are positive and the largest the numerator can be is the size of the denominator.

Then the total score S is found by finding the product of S_a, 100, and all the multiplicative factors.

$$S = (100 \times S_a \times U_1 \times U_2 \cdots \times U_m)$$

VI. CONCLUSIONS AND FUTURE WORK

Metrics for industrial kit building have been developed based on a knowledge model. A configurable method of computing a net score from the metrics has been developed. A Kitting Viewer has been developed that simulates the execution of a plan in the form of a robot command file intended to take a kitting workstation from an inital state to a goal state. The Kitting Viewer computes the metrics and a final score for the plan.

The Kitting Viewer is fully operational, but further improvements are planned. The first significant planned improvement is to allow identical parts in the goal file to be interchangeable. That is, for example, if parts A and B are identical, and the goal file has part A in position 1 and part B in position 2, it will be equally valid to put part B in position 1 and part A in position 2.

Other desirable improvements include: 1) using OWL instance files as input, 2) using grasp points (the model includes grasp points), 3) reading and displaying external shapes (the model includes references to external shape files), 4) implementing animation of gripper rotation, 5) using other types of grippers, and 6) implementing kinematics of commercial robots.

REFERENCES

[1] Control and M. Robotics Lab at the TS, "Measuring the absolute accuracy of an abb irb 1600 industrial robot," *http://www.youtube.com/watch?v=d3fCkS5xFlg*, 2011.

[2] O. Carlsson and B. Hensvold, "Kitting in a High Variation Assembly Line," Master's thesis, Luleå University of Technology, 2008.

[3] L. Medbo, "Assembly Work Execution and Materials Kit Functionality in Parallel Flow Assembly Systems," *International Journal of Production Economics Journal of Industrial Ergonomics*, vol. 31, pp. 263–281, 2003.

[4] G. Schwind, "How Storage Systems Keep Kits Moving," *Material Handling Engineering*, vol. 47, no. 12, pp. 43–45, 1992.

[5] Y. A. Bozer and L. F. McGinnis, "Kitting versus line stocking: A conceptual framework and descriptive model," *International Journal of Production Economics*, vol. 28, pp. 1–19, 1992.

[6] J. Jiao, M. M. Tseng, Q. Ma, and Y. Zou, "Generic Bill-of-Materials-and-Operations for High-Variety Production Management," *Concurrent Engineering: Research and Applications*, vol. 8, no. 4, pp. 297–321, December 2000.

[7] A. C. Sanderson and L. S. Homem de Mello, "Planning Assembly/disassembly Operations for Space Telerobotics," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, W. C. Chiou, Sr., Ed., vol. 851, January 1987, pp. 109–115.

[8] J. H. P. D. G. K. M. J. Chung, "A Framework for the Evaluation and Selection of Assembly Plans," in *Proceedings of 1991 the International Conference on Industrial Electronics, Control and Instrumentation*, vol. 2, October–November 1991, pp. 1215–1220.

[9] F. Bonneville, C. Perrard, and J. M. Henrioud, "A Genetic Algorithm to Generate and Evaluate Assembly Plans," in *Proceedings of the 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, vol. 2, October 1995, pp. 231–239.

[10] R. Suárez and S. Lee, "Towards a Standardized Cost Measure of Assembly Operations," in *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, vol. 1, April 1997, pp. 620–625.

[11] S. Balakirsky, T. Kramer, and A. Pietromartire, "Metrics and Test Methods for Industrial Kit Building," National Institute of Standards and Technology, Gaithersburg, MD, USA, NISTIR to appear, 2012.

[12] J. Albus, "4-D/RCS Reference Model Architecture for Unmanned Ground Vehicles," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2000, pp. 3260–3265.

[13] S. Balakirsky, C. Schlenoff, T. Kramer, and S. Gupta, "An Industrial Robotic Knowledge Representation for Kit Building Applications," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2012, pp. 1365–1370.

[14] D. Nau, M. Ghallab, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[15] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL–The Planning Domain Definition Language," Yale, Tech. Rep. CVC TR98-003/DCS TR-1165, 1998.

[16] A. J. Coles, A. Coles, M. Fox, and D. Long, "Forward-Chaining Partial-Order Planning," in *20th International Conference on Automated Planning and Scheduling, ICAPS 2010*. Toronto, Ontario, Canada: AAAI 2010, May 12–16 2010, pp. 42–49.