A Simulated Sensor-based Approach for Kit Building Applications

Zeid Kootbally[1], Craig Schlenoff[2], Teddy Weisman[3], Stephen Balakirsky[4], Thomas Kramer[5], and Anthony Pietromartire[2]

[1] University of Maryland, College Park, MD 20740, USA,
zeid.kootbally@nist.gov,
WWW home page: www.nist.gov/el/isd/ks/kootbally.cfm
[2] Intelligent Systems Division, National Institute of Standards and Technology, Gaithersburg, MD, USA,
craig.schlenoff@nist.gov, anthony.pietromartire@nist.gov,
WWW home page: www.nist.gov/el/smartcyber.cfm
[3] Yale University, New Haven, CT 06520,
tjweisman@gmail.com
[4] Georgia Tech Research Institute, Atlanta, GA 30332, USA,
stephen.balakirsky@gtri.gatech.edu,
WWW home page: unmannedsystems.gtri.gatech.edu
[5] Department of Mechanical Engineering, Catholic University of America, Washington, DC, USA,
thomas.kramer@nist.gov,
WWW home page: http://www.nist.gov/el/isd/ks/kramer.cfm

Abstract. Kit building or kitting is a process in which individually separate but related items are grouped, packaged, and supplied together as one unit (kit). This paper describes advances in developing sensing/control and parts detection technologies enabling robust operation of kitting applications in simulation.
To pick and place parts and components during kitting, the kitting workcell relies on a simulated sensor system to retrieve the six-degree of freedom (6DOF) pose estimation of each of these objects. While the use of a sensor system allows objects' poses to be obtained, it also helps detecting failures during the execution of a kitting plan when some of these objects are missing or are not at the expected locations.
A kitting system is presented and the approach that is used to task a sensor system to retrieve 6DOF pose estimation of specific objects (objects of interest) is given.

1 Introduction

The effort presented in this paper is designed to support the IEEE Robotics and Automation Society's Ontologies for Robotics and Automation Working Group. 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. Industrial assembly of manufactured products is often performed by first bringing parts together in a kit and then moving the kit to the assembly area where the parts are used to assemble products. Agile and flexible kitting, when applied properly, has been observed to show numerous benefits for the assembly line, such as cost savings [6] including saving manufacturing or assembly space [17], reducing assembly workers' walking and searching times [23], and increasing line flexibility [5] and balance [14].

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. Although fixed automation provides maximum efficiency at a low unit cost, drastic modifications of the machines are realized when parts need major changes or become too complicated in design. In programmable automation, products are made in batch quantities ranging from several dozen to several thousand units at a time. However, each new batch requires long set up time to accommodate the new product style. The time and therefore the cost of developing applications for fixed and programmable automation is usually quite high. The challenge of expanding industrial use of robots is through agile and flexible automation where minimized setup times can lead to more output and generally better throughput.

The effort presented in this paper describes an approach based on a simulated sensor in an attempt to move towards an agile system. Tasking a system sensor to retrieve information on objects of interest should be performed in a timely manner before the robot carries out actions that involve these objects of interest. Objects in a kitting workcell are likely susceptible to be moved by external agents, parts trays may be depleted, and objects of different types can be unintentionally mixed with other types. Consequently, the system should be able to detect any of the aforementioned cases by tasking the sensor system to retrieve pose estimations of objects of interest.

Pose estimation is an important capability for grasping and manipulation. A wide variety of solutions have been proposed in order to extend the current structure of the systems to an agile system. Most of the efforts in the literature have focused primarily on solutions for robots whose mobility is restricted to the ground plane. Lysenkov *et al.* [16] presented new algorithms for segmentation, pose estimation and recognition of transparent objects. Their system showed that a robot is able to grasp 80% of known transparent objects with the proposed algorithm and this result is robust across non-specular backgrounds behind the objects. Dzitac and Mazid [10] proposed a flexible and inexpensive object detection and localization method for pick-and-place robots based on the Xtion and Kinect. The authors relied on depth sensors to provide the robots with flexible and powerful means of locating objects, such as boxes, without the need to hard code the exact coordinates of the box in the robot program. Rusu *et al.* [21] presented a novel 3D feature descriptor, the Viewpoint Feature Histogram (VFH), for object recognition and 6DOF pose identification for application where a priori segmentation is possible.

The organization of the remainder of this paper is as follows. Section 2 presents an overview of the knowledge driven methodology used in this effort. Section 3 describes the simulation environment for the domain of kitting. Section 4 details the methodology that has been developed to task a sensor system to retrieve information on objects of interest, and Section 5 concludes this paper and analyzes future work.

2 Knowledge Driven Methodology

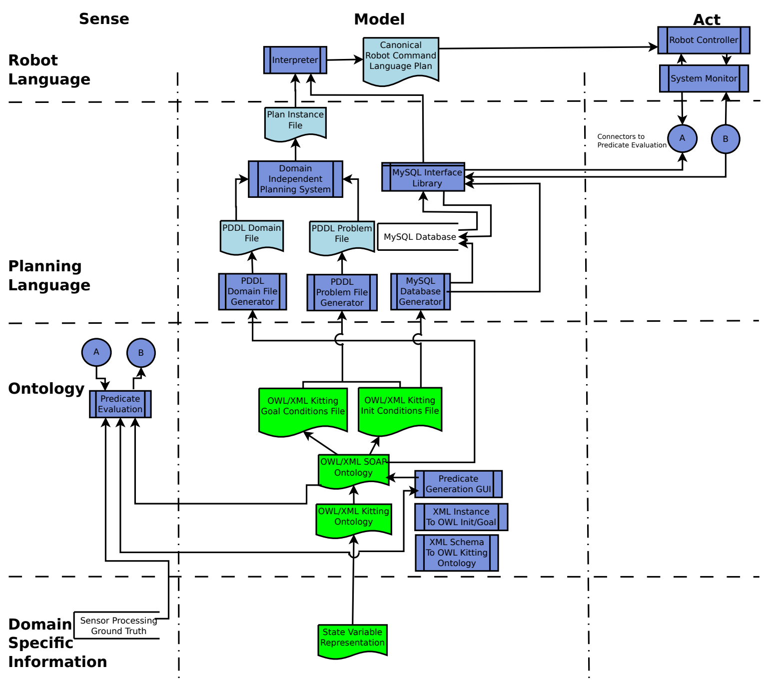


Fig. 1. Knowledge Driven Design extensions – In this figure, green shaded boxes with curved bottoms represent hand generated files while light blue shaded boxes with curved bottoms represent automatically created boxes. Rectangular boxes represent processes and libraries.

The knowledge driven methodology presented in this section is not intended 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 (Real-time Control Systems) [1]. 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 in order that the reader may better grasp the main effort described in this paper. The reader may find a more detailed description of each component and each level of the architecture in other publications [4].

2.1 Domain Specific Information

On the vertical axis, knowledge begins with Domain Specific Information (DSI). DSI includes sensors and sensor processing that are specifically tuned to operate in the target domain. Examples of sensor processing may include pose determination and object identification. It is important to note that the effort described in this paper assumes perfect sensor data that do not include noise. A detailed description of the simulated sensor is given in Section 3.

For the knowledge model, a scenario driven approach is taken where the DSI design begins with a domain expert creating one or more use cases and specific scenarios that describe the typical operation of the system. This includes information on items ranging from what actions and attributes are relevant, to what the necessary conditions (preconditions) are for an action to occur and what the likely results (effects) of the action are. The authors have chosen to encode this basic information in a formalism known as a state variable representation [19].

2.2 Ontology

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 a form as possible with knowledge classes inheriting common attributes from parent classes.

The OWL/XML SOAP (States, Ordering constructs, Actions, and Predicates) 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. Since both the OWL and XML implementations of the knowledge representation are file based, real time information proved to be problematic. In order to solve this problem, an automatically generated MySQL Database [9] has been introduced as part of the knowledge representation. A description of the MySQL Database is given in the following subsection.

2.3 Planning Language

Aspects of the knowledge previously described 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) [13] (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. From these two files, a domain independent planning system [8] was used to produce a static Plan Instance File.

While the knowledge representation presented in this paper provides the "slots" necessary for representing dynamic information, the static file structure makes the utilization of these slots awkward. It is desirable to be able to represent the dynamic information in a dynamic database. For this reason, the authors have developed a technique to automatically generate tables for storing, and access functions for obtaining the data from the ontology in a MySQL Database.

Reading data from and to the MySQL Database instead of the ontology file offers the community easy access to a live data structure. Furthermore, it is more practical to modify the information stored in a database than if it was stored in an ontology, which in some cases, requires the deletion and re-creation of the whole file. A literature review reveals many efforts and methodologies that have been designed to produce SQL databases from ontologies. Our effort builds upon the work of Astrova *et al.* [2]

In addition to generating and filling the database tables, the authors have created tools that automatically generate a set of C++ classes for reading and writing information to the kitting MySQL Database. The choice of C++ was a team preference and we believe that other object-oriented languages could have been used in this project.

2.4 Robot Language

Once a plan has been formulated, the knowledge is transformed into a representation that is optimized for use by a robotic system. 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.

3 Simulation Environment

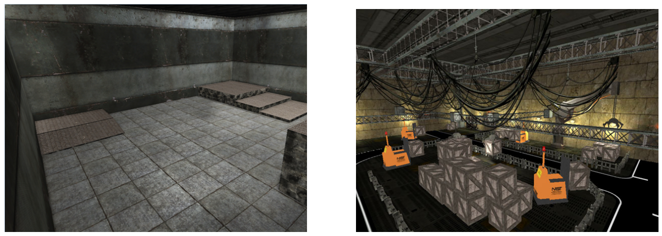
In order to experiment with robotic systems, a researcher requires a controllable robotic platform, a control system that interfaces to the robotic system and provides behaviors for the robot to carry out, and an environment to operate in. Our kitting application relies on an open source (the game engine is free, but license restrictions do apply), freely available framework capable of fulfilling all of these requirements. This framework is the Unified System for Automation and Robot Simulation (USARSim) [24]. It provides the robotic platform and environment.

3.1 The USARSim Framework

USARSim [7, 25] is a high-fidelity physics-based simulation system based on the Unreal Developers Kit (UDK) [12] from Epic Games. USARSim was originally developed under a National Science Foundation grant to study Robot, Agent, Person Teams in Urban Search and Rescue [15]. Since that time, it has been turned into a National Institute of Standards and Technology (NIST)-led, community-supported open source project that provides validated models of robots, sensors, and environments. Altogether, the Karma Physics engine [11] and high-quality 3D rendering facilities of the Unreal game engine allow the creation of realistic simulation environments that provide the embodiment of a robotic system. Furthermore, USARSim comes with tools to develop objects and environments and it is possible to control actors in the game through a TCP/IP socket API.

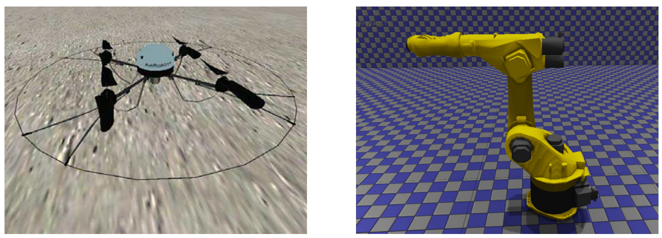
Through its usage of UDK, USARSim utilizes the physX physics engine [20] and high-quality 3D rendering facilities to create a realistic robotic system simulation environment (Figure 2). The current release of USARSim consists of various model environments, models of commercial and experimental robots, and sensor models. High fidelity at low cost is made possible by building the simulation on top of a game engine. By delegating simulation specific tasks to a high volume commercial platform (available for free to most users) which provides superior visual rendering and physical modeling, full user effort can be devoted to the robotics-specific tasks of modeling platforms, control systems, sensors, interface tools and environments. These tasks are in turn accelerated by the advanced editing and development tools integrated with the game engine. This leads to a virtuous spiral in which a wide range of platforms can be modeled with greater fidelity in a short period of time.

USARSim was initially developed with a focus on differential drive wheeled robots. However, USARSim's open source framework has encouraged wide community interest and support that now allows USARSim to offer multiple robots, including humanoid robots, aerial platforms (Figure 3(a)), robotic arms (Figure 3(b)), and commercial vehicles. All robots in USARSim have a chassis, and may contain multiple wheels, sensors, and actuators. The robots are configurable (e.g. specify types of sensors/end effectors) through a configuration file that is read at run-time. The properties of the robots can also be configured, such as the battery life and the frequency of data transmission.



(a) Test Room. (b) Factory,

Fig. 2. Sample of 3D environments in USARSim.



(a) Air Robot AR100B. (b) Kuka KR60,

Fig. 3. Sample of vehicles in USARSim.

3.2 The Simulated Sensor System

Poses of objects in the virtual environment are retrieved with the USARTruth tool. USARTruth is capable of reading out information on objects in USARSim by connecting as a client to TCP socket port 3989. The simulator USARTruth-Connection object listens for incoming connections on port 3989 and receives queries over a socket in the form of strings formatted into key-value pairs.

The USARTruth connection accepts two different keys, "class" and "name," which are both optional. When USARSim receives a new string over the connection, it sends a sequence of key-value formatted strings back over the socket, one for each Unreal Engine Actor object that matches the requested class and object names. An example of the strings returned by USARSim is given below along with a description for each key.

{Name P3AT_0} {Class P3AT} {Time 29.97} {Location 0.67,2.30,1.86} {Rotation 0.00,0.46,0.00} where:

- Name: The internal name of the object in USARSim.
- Class: The name of the most specific Unreal Engine class the object belongs to.
- Time: The number of seconds that have elapsed since the simulator start, as a floating-point value.
- Location: The comma-separated position of the object in global coordinates.
- Rotation: The comma-separated orientation of the object in global coordinates, in roll, pitch, yaw form.

4 System Operation

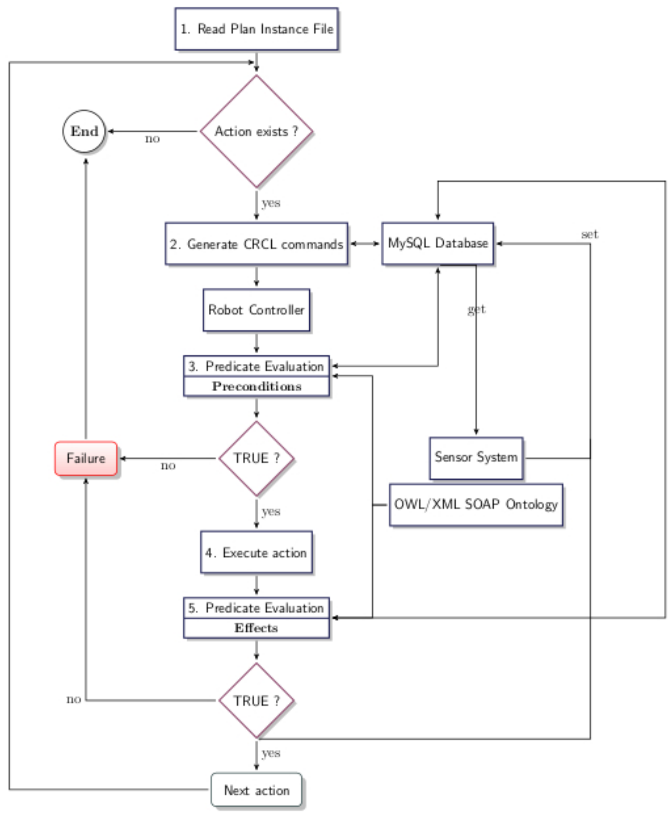


Fig. 4. Flowchart diagram for tasking the simulated sensor.

This section describes the use of a simulated sensor system to retrieve 6DOF poses of objects of interest in the kitting workcell. Figure 4 is a flowchart which represents the steps used for kitting, from parsing the Plan Instance File to the execution of each action from this file. Since the focus of this paper is on the use of the sensor system during kitting, the authors have limited the representation and description of Figure 4 around the sensor system and did not include the steps prior to the Plan Instance File generation. The reader may find this missing information in the description of Figure 1, in Section 2. The different steps depicted in Figure 4 are categorized into main components that are numbered. A description of each main component is given in the following subsections.

4.1 Read the Plan Instance File

As described in Section 2, the Plan Instance File is generated by the Domain Independant Planning System from the PDDL Domain File and the PDDL Problem File. An example of a plan is given in Figure 5 to build a kit that consists of one part of type D and one part of type E. At the beginning of the plan (line 1), the robot connects the proper end effector from the end effector changing station to the tip of the robot and puts the end effector back in the end effector changing station at the end of the plan (line 6). Lines 2 and 4 display the actions for picking up a part of type E and D, respectively. Lines 3 and 5 display the actions for putting these parts in the kit.

```
1 (attach-endeffector robot_1 part_gripper part_gripper_holder changing_station_1)
2 (take-part robot_1 part_e_1_ir part_e_tray_ir part_gripper)
3 (put-part robot_1 part_e_1_ir kit_a2b3c3d1e1 work_table_1 part_e_tray_ir)
4 (take-part robot_1 part_d_1_ir part_d_tray_ir part_gripper)
5 (put-part robot_1 part_d_1_ir kit_a2b3c3d1e1 work_table_1 part_d_tray_ir)
6 (remove-endeffector robot_1 part_gripper part_gripper_holder changing_station_1)
```

Fig. 5. Excerpt of the PDDL solution file for kitting.

4.2 Generate the CRCL Commands

Each action of the plan is sequentially interpreted and executed. The Interpreter takes as input a PDDL action from the Plan Instance File and outputs a sequence of CRCL commands. To facilitate late binding, the PDDL actions within the plan do not specify the exact locations of the parts and components that are involved. This kind of knowledge detail is maintained by sensor processing and is stored in the MySQL Database. As described in Section 2, the generation of the tables in the MySQL Database is followed by data insertion in these tables for all the objects in the environment. However, there is no guarantee that the poses of these objects are still accurate. They may have been altered in different ways. At this point, the sensor system is tasked to retrieve information about objects of interest. Objects of interests are the ones for which the poses are needed to execute some CRCL commands. Before tasking the sensor to retrieve (get message) the poses of objects of interest, the external shape of each object of interest must be retrieved from the MySQL Database.

An external shape is a shape defined in an external file. An external shape has a model format name (USARSim in this case) and a model file name which is the name of the file containing the model and may include a path. Using the information stored within the external shape of each object of interest, the system then parses the data coming in from USARTruth and updates the relative pose in the MySQL Database for each object returned (set message). Since USARTruth returns object locations in global coordinates, the relative pose for each object is updated without changing its transformation tree; that is, the object of reference for its physical location is unchanged.

The actual updated relative pose is computed according to

$$L' = LG^{-1}G' \tag{1}$$

where L' is the updated relative transformation, L is the old relative transformation (read from the MySQL Database), G is the old global transformation (computed from the transformation tree in the MySQL Database), and G' is the updated global transformation (retrieved from USARTruth).

Once the aforementioned process is performed, the Interpreter uses the latest data from the MySQL Database to generate a set of CRCL commands for the current action. The take-part action at line 2 in Figure 5 is interpreted as the sequence of CRCL commands displayed in Table 1, where the numerical data used in the `MoveTo` commands are computed with the latest information retrieved from the MySQL Database. The reader may find more information about the whole set of CRCL commands in [3].

```
initCannon()
Message (``take part part_e_1_ir'')
MoveTo({{-0.03, 1.62, -0.25}, {0, 0, 1}, {1, 0, 0}})
Dwell (0.05)
MoveTo({{-0.03, 1.62, 0.1325}, {0, 0, 1}, {1, 0, 0}})
CloseGripper ()
MoveTo({{-0.03, 1.62, -0.25}, {0, 0, 1}, {1, 0, 0}})
Dwell (0.05)
endCannon()
```

Table 1. A sequence of CRCL commands for a PDDL action.

Once a set of CRCL commands is generated, it is sent to the Robot Controller to be executed by the robot. The Predicate Evaluation process is then called before and after each set of CRCL commands is carried out by the robot.

4.3 Predicate Evaluation (Preconditions)

As mentioned in Section 2, a PDDL action consists of one precondition section and one effect section that are defined in the OWL/XML SOAP Ontology. Preconditions and effects consist of a set of predicates. For instance, the predicates in the precondition and effect for the action *take-part*(*robot*,*part*,*partstray*,*endeff*) is defined in Table 2.

The evaluation of the predicates in the precondition section assures that all the requirements are met in the environment before the robot carries out the action. As such, the output of the Predicate Evaluation process is a Boolean value.

The kitting system relies on the representation of spatial relations [22] that are stored in the OWL/XML SOAP Ontology to compute the truth-value of each predicate. Predicates cannot have more than two parameters due to the inherent definition of predicates.

precondition	*effect*
part-location-partstray(*part*,*partstray*)	¬part-location-partstray(*part*,*partstray*)
robot-empty(*robot*)	¬robot-empty(*robot*)
endeff-location-robot(*endeff*,*robot*)	part-location-robot(*part*,*robot*)
robot-with-endeff(*robot*,*endeff*)	robot-holds-part(*robot*,*part*)
endeff-type-part(*endeff*,*part*)	
partstray-not-empty(*partstray*)	

Table 2. The precondition and the effect for the action *take-part*.

In the case the predicate has two parameters, the external shape of each parameter is fetched in the MySQL Database and is passed to USARTruth which in turn queries USARSim to get the pose information of the object. Once the pose of each parameter is retrieved, the Predicate Evaluation process checks the spatial relation between these two objects. For instance, to evaluate the predicate part-location-partstray(*part*,*partstray*), the external shapes for *part* and *partstray* are retrieved from the MySQL Database and their poses are retrieved by USARTruth. Those poses are then used by a combination of spatial relations to compute the truth-value of this predicate.

In the case the predicate has only one parameter, robot-empty(*robot*) for instance, the external shape of this parameter is used by USARTruth to retrieve this parameter's pose from USARSim (pose of *robot* in this example). Since the computation of spatial relations always requires two objects, the external shape, thus the pose of each single other object in the environment is retrieved and used as the second object.

If all the predicates within the precondition section are true, the MySQL Database is updated with the current poses of these predicates' parameters. It is important that all the predicates are true in order to update the MySQL Database. This assures that a complete (stable) state of the environment is stored in the MySQL Database. If at least one predicate is evaluated to false, it is considered a failure and the kitting process is terminated.

4.4 Execute Action

When all the predicates within the precondition of an action have been evaluated to true, this action is executed by the robot. To confirm that the action was successfully accomplished, the Predicate Evaluation process evaluates the predicates within the effect section for this action. The effect section consists of predicates that are expected to be true after performing a PDDL action. The evaluation of the predicates within the effect section is performed to confirm that these expectations are attained.

4.5 Predicate Evaluation (Effects)

The methodology used to evaluate the predicates within the precondition section of an action is also used to evaluate the predicates within its effect section. As mentioned earlier, all the predicates within the effect section must be evaluated to true to define the action as successful. In the case of a successful action, the next action within the Plan Instance File is processed. Once all the actions within the Plan Instance File have been executed by the robot, the kitting process is complete.

5 Conclusions and Future Work

This paper describes the approach that uses a simulated sensor system to retrieve 6DOF pose estimations of objects of interest during kit building applications. The approach is mainly used during the predicate evaluation process. The use of a simulated sensor system allows the current kitting system to move towards an agile system where the current observations on parts and components are fed into the predicate evaluation process before and after action executions.

As mentioned in Section 2.1, pose information coming from USARTruth is assumed to be perfect. In a future effort, the authors will attempt to present a model for the different sources of noise relative to each sensor-based pose estimation step (similar to the one described in [18]), and use measurements of real sensor data to validate the model. Once validated, the new sensor system can then be used to validate the truth values of the predicates.

It is also intended to apply contingency plans once an action failure occurs. One of the contingency plans is to re-plan from a state of the environment that is stable. Information on this stable state is retrieved from the MySQL Database that was updated from the latest objects' poses. During the re-planning process, the new initial state becomes the stable state while the goal state stays unchanged.

The current kitting workcell involves objects that are originally placed on a non-movable surface and also involves an articulated arm with a non-movable base. To move towards an agile and flexible manufacturing system, the authors will need to address more challenging cases where parts can come into the work-cell via conveyor belts and where the robotic arm can be of gantry type. These new settings will need to be simulated and tested for kitting applications.

6 Acknowledgement

The authors would like to extend our gratitude to Grayson Moses, who worked at NIST as a Poolesville High School student volunteer and helped with the development of algorithms to task a sensor system to provide updated pose information.

References

1. J. Albus. 4-D/RCS Reference Model Architecture for Unmanned Ground Vehicles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3260–3265, 2000.

2. I. Astrova, N. Korda, and A. Kalja. Storing owl ontologies in sql relational databases. *World Academy of Science, Engineering and Technology*, 29:167–172, 2007.
3. S. Balakirsky, T. Kramer, Z. Kootbally, and A. Pietromartire. Metrics and Test Methods for Industrial Kit Building. NISTIR 7942, National Institute of Standards and Technology (NIST), 2012.
4. 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)*, pages 1365–1370, October 2012.
5. Y. A. Bozer and L. F. McGinnis. Kitting versus line stocking: A conceptual framework and descriptive model. *International Journal of Production Economics*, 28:1–19, 1992.
6. O. Carlsson and B. Hensvold. Kitting in a High Variation Assembly Line. Master's thesis, Luleå University of Technology, 2008.
7. S. Carpin, J. Wang, M. Lewis, A. Birk, and A. Jacoff. *Robocup 2005: Robot Soccer World Cup IX, LNAI*, volume 4020, chapter High Fidelity Tools for Rescue Robotics: Results and Perspectives, pages 301–311. Springer, 2006.
8. 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*, pages 42–49, Toronto, Ontario, Canada, May 12–16 2010. AAAI 2010.
9. Oracle Corporation. Mysql, November 2012.
10. P. Dzitac and A. Md. Mazid. A Depth Sensor to Control Pick-and-Place Robots for Fruit Packaging. In *12^{th} International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 949–954, 2012.
11. Epic Games. *MathEngine Karma ^{TM}User Guide*, March 2002.
12. Epic Games. Unreal Development Kit. *http://udk.com*, 2011.
13. M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl–the planning domain definition language. Technical Report CVC TR98-003/DCS TR-1165, Yale, 1998.
14. 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*, 8(4):297–321, December 2000.
15. M. Lewis, K. Sycara, and I. Nourbakhsh. Developing a Testbed for Studying Human-Robot Interaction in Urban Search and Rescue. In *Proceedings of the 10^{th} International Conference on Human Computer Interaction*, pages 22–27, 2003.
16. I. Lysenkov, V. Eruhimov, and G. Bradski. Recognition and Pose Estimation of Rigid Transparent Objects with a Kinect Sensor. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.
17. L. Medbo. Assembly work execution and materials kit functionality in parallel flow assembly systems. *International Journal of Industrial Ergonomics*, 31:263–281, 2003.
18. L. Meeden. Bridging the Gap between Robot Simulations and Reality with Improved Models of Sensor Noise. In J.R. Koza *et al.*, editor, *Proceedings of the 3^{rd} Annual Genetic Programming Conference*, pages 824–831, San Francisco, CA, 1998.
19. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
20. Nvidia. PhysX Description. *http://www.geforce.com/Hardware/Technologies/physx*, 2011.

21. R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu. Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram. In *Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, Oct 2010.
22. C. Schlenoff, A. Pietromartire, Z. Kootbally, S. Balakirsky, and S. Foufou. Ontology-based state representations for intention recognition in humanrobot collaborative environments. *Robotics and Autonomous Systems*, In Press, 2013.
23. G. F. Schwind. How Storage Systems Keep Kits Moving. *Material Handling Engineering*, 47(12):43–45, 1992.
24. USARSim. UsarSim Web. *http://www.usarsim.sourceforge.net*, 2011.
25. J. Wang, M. Lewis, and J. Gennari. A Game Engine Based Simulation of the NIST Urban Search and Rescue Arenas. In *Proceedings of the 2003 Winter Simulation Conference*, volume 1, pages 1039–1045, 2003.