Improve the Agility of Kit Building Applications via Sensor Tasking in Simulation

Zeid Kootbally[1], Craig Schlenoff[2], Teddy Weisman[3], Stephen Balakirsky[4], 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

Abstract. 6 degree of freedom (6DOF) pose estimation of objects are required for robotic grasping and manipulation during kit building. Kit building or kitting is a process in which individually separate but related items are grouped, packaged, and supplied together as one unit (kit). It is anticipated that utilization of the knowledge representation will allow for the development of higher performing kitting systems.
In this paper, kitting is performed by a robotic arm that manipulates parts/components (objects of interest) by following an ordered sequence of steps that constitute a plan. To prevent availability failures of objects of interest, it is necessary to know their pose estimation prior executing each step of the plan. This is performed by querying a simulated sensor system thus providing an agile system where the system can cope with objects of interest in the case they are moved within robotic work cells by external agents.
This paper describes advances in developing key software, sensing/control, and parts-handling technologies enabling robust operation of kitting applications in simulation. In particular, an interface is presented and a description of the methodology tasking a sensor system to retrieve 6DOF pose estimation of objects of interest is given.

1 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. 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 in to more output and generally better throughput.

Agility mainly represents the idea of "speed and change in business environment" and consists of two main factors: 1) Responding to change in proper ways and due time and 2) Exploiting changes and taking advantage of them as opportunities [21]. Agile manufacturing demands a manufacturing system that is able to produce effectively a large variety of products and to be reconfigurable to accommodate changes in the product mix and product designs [11]. Reconfigurability of manufacturing systems, product variety, velocity and flexibility in production are critical aspects to achieving and maintaining competitive advantage. The concept of agility has an impact on the design of assemblies and therefore, methodologies for the design of agile manufacturing are needed.

Some of the key words and terms linked to "agile manufacturing" are [13]:

- Fast – A very high speed of response to new opportunities.
- Adaptable – The ability to change the current approach and change direction with ease.
- Reconfiguration – The ability to quickly reconfigure a firm structures, facilities, organization, and technology to meet unexpected and short lived market opportunities.
- Transformation of knowledge – The ability to explicitly transform raw ideas into a range of competence which is then used in both products and services.

The effort presented in this paper describes a methodology and tools in the area of industrial assembly of manufactured products. This effort is designed to support the IEEE Robotics and Automation Society's Ontologies for Robotics and Automation Working Group. 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. 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. Agile and flexible kitting, when applied properly, has been observed to show numerous benefits for the assembly line, such as cost savings [5] including saving manufacturing or assembly space [15], reducing assembly workers walking and searching times [20], and increasing line flexibility [4] and balance [12].

The effort described in this paper tends to move towards the ideas of a fast and adaptable system. Tasking a system sensor to retrieve information on objects of interest should be performed in a timely manner before performing any

actions of the plan that involve these objects of interest. Parts and components in a kitting work cell are likely susceptible to be moved during the course of time by external agents and thus, the system should be able to cope with these unexpected changes.

The organization of the remainder of this paper is as follows. Section 3 presents an overview of the knowledge driven methodology that has been developed for this effort with a particular focus on the areas where the sensor system interacts with. Section 2 describes the simulation environment for the domain of kitting . Section 4 details the tools and the new methodology developed to task a sensor system to retrieve information on objects of interest, and Section 5 presents conclusions and future work.

2 Simulation Environment

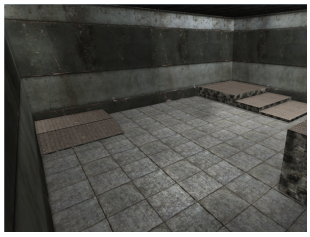
In order to experiment with robotic systems, a researcher requires a controllable robotic platform, a control system that interfaces to the robotic system and provides behaviors for the robot to carry out, and an environment to operate in. Our kitting application relies on an open source (the game engine is free, but license restrictions do apply), freely available framework capable of fulfilling all of these requirements. This framework is composed of the Unified System for Automation and Robot Simulation (USARSim) [22] framework that provides the robotic platform and environment, and the Operating System (ROS)[5][9] framework that provides the control system.

2.1 The USARSim Framework

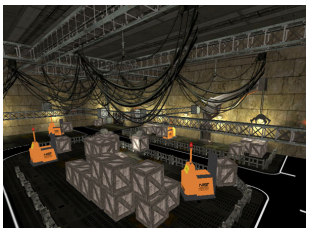
USARSim [6, 23] is a high-fidelity physics-based simulation system based on the Unreal Developers Kit (UDK) [8] from Epic Games. USARSim was originally developed under a National Science Foundation grant to study Robot, Agent, Person Teams in Urban Search and Rescue [14]. Since that time, it has been 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.

Through its usage of UDK, USARSim utilizes the physX physics engine [17] and high-quality 3D rendering facilities to create a realistic simulation environment (Figure 1) that provides the embodiment of, and the environment for a robotic system. 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

[5] Certain commercial software and tools are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools identified are necessarily the best available for the purpose.

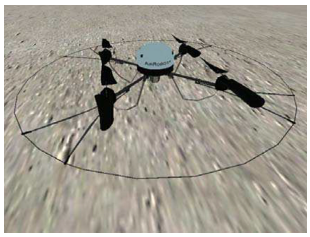


(a) Test Room.



(b) Factory,

Fig. 1. Sample of 3D environments in USARSim.



(a) Air Robot AR100B.

(b) Kuka KR60,

Fig. 2. Sample of vehicles in USARSim.

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 2(a)), robotic arms (Figure 2(b)), and commercial vehicles. In USARSim, robots are based on physical computer aided design (CAD) models of the real robots and are implemented by specialization of specific existing classes. This structure allows for easier development of new platforms that model custom designs.

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.

2.2 The ROS Framework

ROS [9] is an open source framework designed to provide an abstraction layer to complex robotic hardware and software configurations. It provides libraries and tools to help software developers create robot applications and has found wide use in both industry and academia. Examples of ROS applications include Willow Garage's Personal Robots Program [24] and the Stanford University STAIR project [18]. Developers of ROS code are encouraged to contribute their code back to the community and to provide documentation and maintenance of their algorithms.

ROS possesses a large range of tools and services that both users and developers alike can benefit from. The philosophical goals of ROS include an advanced set of criteria and can be summarized as: peer-to-peer, tools-based, multi-lingual, thin, and free and open-source [19]. Furthermore, debugging at all levels of the software is made possible with the full source code of ROS being publicly available. Thus, the main developers of a project can benefit from the community and vice-versa.

In the effort described in this paper, the authors have used ROS as the main system controller. Once a CRCL plan file is generated by the interpreter from the PDDL plan file (Robot Language level in Figure 3), the CRCL plan file is parsed by the ROS controller and each robot command in this file is executed by the simulated robotic arm.

3 6DOF Pose Estimation within the Knowledge Driven Methodology

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 [1]. The overall knowledge driven methodology of the system is depicted in Figure 3. 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 to captivate 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 documented publications [3].

3.1 Domain Specific Information

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

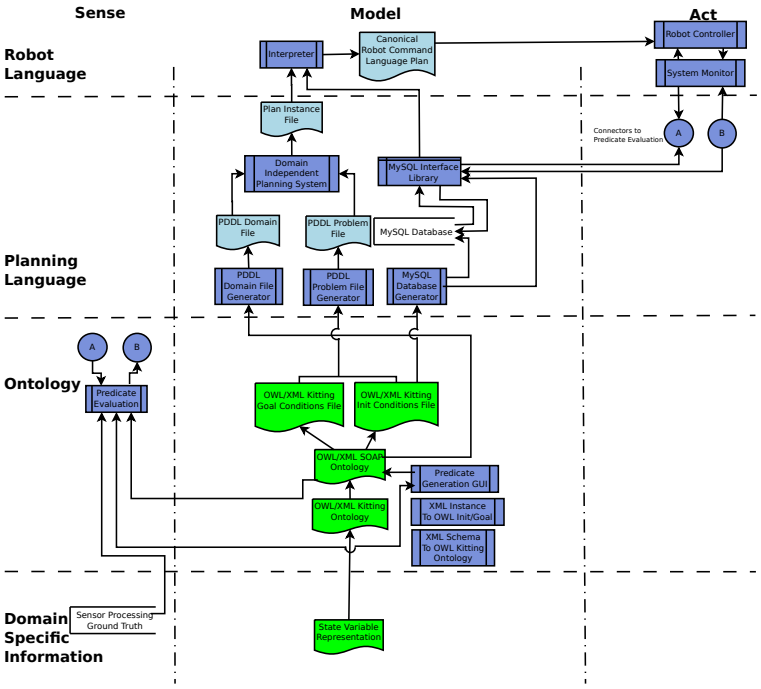


Fig. 3. Knowledge Driven Design extensions – In this figure, green shaded boxes with curved bottoms represent hand generated files while light blue shaded boxes with curved bottoms represent automatically created boxes. Rectangular boxes represent processes and libraries.

are. The authors have chosen to encode this basic information in a formalism known as a state variable representation [16].

3.2 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 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 has been introduced as part of the knowledge representation. More information on the generated MySQL database is given in the description of the Planning Language level.

3.3 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) [10] (version 3.0). The PDDL input format consists of two files that specify the domain and the problem. As shown in Figure 3, these files are automatically generated from the ontology. From these two files, a domain independent planning system [7] 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 for automatically generating 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.

The interaction of the MySQL database with the Connectors to Predicate Evaluation (in the Act level) is detailed in Section 4.

3.4 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 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.

4 Example of Operation

4.1 TODO

- Describe the interaction between the predicate evaluation and the update of the MySQL database from sensor data.
- Describe in more details the work that Teddy did to query the sensor

USARSim can be directly queried to return the 6DOF pose of a specific object, or of every object of a certain type. The simulator USARTruthConnection 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. The return strings include the following keys:

- 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.

The virtual sensor system uses USARTruth to simulate real-world sensors by opening a new USARTruth connection and sending one request for each different Unreal Engine class name found in the MySQL database. For simulation purposes, the Unreal Engine class name is stored as an object's "external shape."

The system then parses the data coming in from USARTruth and updates the relative pose in the MySQL database for each object returned. Since USARTruth returns object locations in global coordinates, the relative pose for each object is updated without changing its transformation tree; that is, the RefObject for its PhysicalLocation 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 database), and G' is the updated global transformation (retrieved from USARTruth).

5 Conclusions and Future Work

References

1. James 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, 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.
4. 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.
5. O. Carlsson and B. Hensvold. Kitting in a High Variation Assembly Line. Master's thesis, Luleå University of Technology, 2008.
6. 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.
7. 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.
8. Epic Games. Unreal Development Kit. *http://udk.com*, 2011.
9. Willow Garage. ROS Wiki. *http://www.ros.org/wiki*, 2011.
10. 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.
11. A. Gunasekaran. Agile Manufacturing: A Framework for Research and Development. *International Journal of Production Economics*, 62:87–105, 1999.
12. 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.
13. P. Kidd. Two Definitions of Agility. http://www.cheshirehenbury.com/agility/twodefinitions.html, 2000.
14. 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.
15. L. Medbo. Assembly work execution and materials kit functionality in parallel flow assembly systems. *International Journal of Industrial Ergonomics*, 31:263–281, 2003.

16. D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
17. Nvidia. PhysX Description. *http://www.geforce.com/Hardware/Technologies/physx*, 2011.
18. M. Quigley, E. Berger, and A. Y. Ng. *AAAI 2007 Robotics Workshop*, volume 85, chapter STAIR: Hardware and Software Architecture, pages 6–23. 2007.
19. M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Y. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.
20. G. F. Schwind. How Storage Systems Keep Kits Moving. *Material Handling Engineering*, 47(12):43–45, 1992.
21. H. Sharifi and Z. Zhang. A Methodology for Achieving Agility in Manufacturing Organizations: An Introduction. *International Journal of Production Economics*, 62:7–22, 1999.
22. USARSim. UsarSim Web. *http://www.usarsim.sourceforge.net*, 2011.
23. 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.
24. K. Wyobek, E. Berger, H. V. der Loos, and K. Salisbury. Towards a Personal Robotics Development Platform: Rationale and Design of an Intrinsically Safe Personal Robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2165–2170, Pasadena, CA, 2008.