Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 54 (2016) 175 – 180

www.elsevier.com/locate/procedia

6th CLF - 6th CIRP Conference on Learning Factories

Handling of Frequent Design Changes in an Automated Assembly Cell for Electronic Products

Alvaro Capellan [a,*], Olivier Roulet-Dubonnet [a]

[a]*SINTEF Raufoss Manufacturing AS, S.P. Andersens vei 5, 7031 Trondheim, Norway*

* Corresponding author. *E-mail address:* alvaro.capellan@sintef.no

Abstract

This paper presents a prototype flexible assembly cell used for the assembly of electronic products. The cell is the first prototype version of the coming assembly system for fire sensors at Autronica. It is developed specifically for testing different concepts to reduce development time for design changes and introduction of new variants. The cell consists of a robot, grippers, sensors, vision systems and fixturing systems which have been selected for in-line adaptivity and reconfiguration. The topics of developing generic vision programs and reducing programming time for vision and robot have been central. The aspects needed to be addressed during development are presented together with considered and chosen solutions. These solutions are also discussed and compared to other systems presented in recent publications on flexible assembly.

Keywords: Automation; Flexible assembly; Machine vision; Robotics; Electronic products

1. Introduction

An important challenge for learning factories is to maximize the factory's capability to adapt to changes resulting from continuous learning and improvement. In this context, flexible manufacturing is a key topic. Flexibility in a manufacturing system is defined as the capability to produce several products or variants, and to adapt to new, different, or changing requirements [1, 2]. Flexibility enables companies to reduce time and monetary investments under reconfiguration of a production line for a new product or variant.

For the manufacturing of electronic products, flexibility is of paramount importance. The electronic industry is under constant press to renew its product spectrum, due to continuous technological advances, consumer demand and fierce competition. In high-cost lands, companies producing electronic products are subject to high manufacturing cost pressure and are often specialized in higher quality and lower volume products. Manufacturing of such products is often challenging to automate. They are therefore in strong need for new flexible automated solutions, reflected in the increasing number of industrial actors focus their attention to the development of systems for automatic assembly of electronic products.

The cell presented in this paper has recently been reconfigured to host the prototype version of the future assembly system for fire sensors at Autronica. The main characteristic of this project is its focus on parallel design of the product, supply-chain and production systems approaching the different stages of product development in parallel. The geometrical design of the product, electronic components, and assembly methodology are developed in parallel and therefore constantly updated. This methodology, known as concurrent engineering, has the potential to save time during the development of the product, but adds new challenges to the actors in charge of the different stages [3]. Specifically for the cell, concurrent engineering introduces a new order of requirements for flexibility.

Flexible manufacturing is an active research area. G. Michalos [4] made a review of technologies in 2010 where he

doi:10.1016/j.procir.2016.05.066

mentioned more applicable technologies than the ones we can mention in this paper. We can however note a few promising technologies that have been of interest in our case:

- As mentioned by G. Michalos [4], cooperating robots is a solution to reduce number of fixtures and accessibility constraints. In this context, dual-arm robots are to be considered [5][6]. Our cell has two ceiling mounted robots, only one has been used so far, but the second one is currently being setup in order to reduce tool changes and reduce assembly time by parallelizing operations.
- Human machine cooperation is also to be considered. Although we are aiming for a complete automation of the assembly process without operators, some low volume variant might be assembled by hand and solutions for efficient cooperation with the robots have to be designed [7].
- Reducing or even removing fixtures is a promising idea also noted by B. Shirinzadeh [8].
- Instrumentation and the use of sensors to detect geometrical changes is a very promising concept noted by several authors [9, 10, 11]. Machine vision is named as a fundamental component of flexible manufacturing [12].
- Information and control systems are also important, as the recent Industry 4.0 paradigm emphasizes. At the cell level, the fundamental feature is to allow for external, high level control and monitoring through, for example the OPC-UA protocol and RFID integration [13], and provide data to, for example, statistical analysis.
- Integration with CAD data, has a high potential [14] although its realization currently demands a lot of custom development. Point cloud vision is in this area promising as demonstrated as, for example, O. Skotheim [15].

The presented cell attempts to build on these recommended technologies with a focus on design change, due to the constant geometrical changes that the components to be assembled are subject to during the product development phase.

This article presents the complete system, explaining the specific needs and reasons for the chosen hardware and software solutions. A large number of the solutions can be considered generic and easily transferred to other applications.

2. Assembly Cell description

Currently, the cell is used for assembling smoke detectors, formed by a set of plastic components and the accompanying electronic parts. Every component has a specific shape, and is assembled to other components using snapping as the main joining method.

The sequence of operations performed by the cell are the following:

1. Components lay randomly on a work surface.
2. A machine vision system identifies the components on the table, their positions and their orientations.
3. A robot arm picks the components and assembles them on a jig.

The cell is composed by a robot arm, a camera (fixed on top of it), fixturing and gripping devices. Figure 1 shows the main components of the cell.

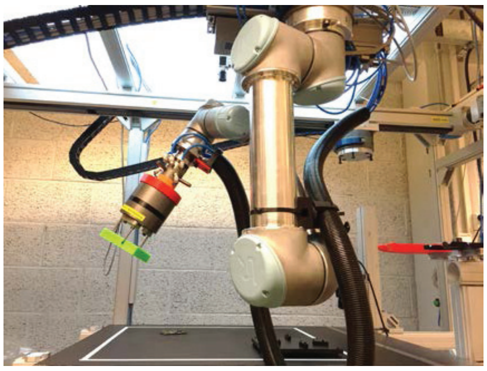
Fig.1 Assembly cell overview

2.1. Robot arm

An Universal Robots UR5 robot arm [16] is used for performing assembly operations. The robot is controlled using a python library that interfaces to the robot controller. The robot is equipped with a 6D force sensor and mounted on a linear axis.

Force feedback from the force sensor is planned to be used to ensure correct assembly by storing and comparing the assembly operation signatures, e.g. a missed or different click on a snap operation. Later on, it will be used to increase the flexibility of the process, for example by using force monitoring to stop a snap, instead of hard programming a move depth. It is also planned to be used together with intelligent algorithms for monitoring the operation forces and adjust operations. Force feedback is not implemented yet in the cell.

The linear axis adds an additional Degree Of Freedom (DOF) to the reach of the robot, increasing the size of the working area; it is not used directly in this application, but used regularly to reconfigure the cell for using the robots in other projects.

2.2. Machine vision system

One camera is installed on top of the cell and is used for capturing and analyzing the configuration of the different electronic components on the table. For analyzing the images captured by the camera, Tordivel's Scorpion vision software [17] is used. The software is programmed for detecting specific geometric features of the objects, and for communicating to the robot the poses required for picking and placing the components.

3D vision is used in the application for tackling with perspective issues due to 1) distance between camera and work surface, and 2) variations of height between components.

2.3. Fixturing devices

Modular fixtures are used for holding the electronic components in fixed positions. The fixtures are adjustable in order to handle design changes. They permit only one DOF to the component in one axis. This DOF allows the robot to place the components along that axis in the fixture.

Fixtures are attached to the metallic assembly table using magnets, making it easy to move fixtures or reconfigure the entire cell to the requirements of different projects.

2.4. Gripping devices

Currently, the robot picks and places components using three types of grippers:

a) Suction cups, vacuum.
b) Two-point servo-actuated parallel gripper.
c) Three-point rotary pneumatic gripper.

Each component to be assembled requires to be picked by a specific gripper. A vacuum gripper is used for components with a flat surface, whereas the parallel and rotary grippers are used for components that can be gripped by clamping two parallel surfaces, or a circular surface respectively. Specialized grippers will have to be added for special parts that will require to be assembled in future stages of the project.

Using grippers adds time to the overall duration of the assembly process, due to unproductive time used on tool changing operations. Reducing the number of grippers is a regular focus. Merged grippers, such as for example suction cups mounted on side of linear gripper will probably be used in the final industrial system.

2.5. Control program

The cell operations are controlled by a Python application. The program directs the assembly operations by communicating to the robot controller, the grippers, the sensors and the vision software; distributing data between devices. The communication to the devices is performed mainly through non-real-time Ethernet using TCP/IP or UDP.

The following sections describe selected characteristics of the cell and the aspects they attempt to solve or improve.

3. Image analysis program

In this project machine vision is used to localize components to be assembled. Setting up a robust vision program is a relatively time consuming operation. When a geometric change of the part happens, it is essential to minimize or better eliminate changes in the vision program.

Image analysis aims to identify specific components and their location by detecting specific geometrical features of the analyzed objects. The analysis is desired and implemented as generic as possible, but, as Lanitis et al. [18] noted, the specific nature of the components requires the implementation of specific detection algorithms.

In our cell, image analysis is performed according to the sequence presented in Figure 2. With the exception of specific algorithms included in the third and fifth steps, all the steps are generic and performed equally for the detection of every component.

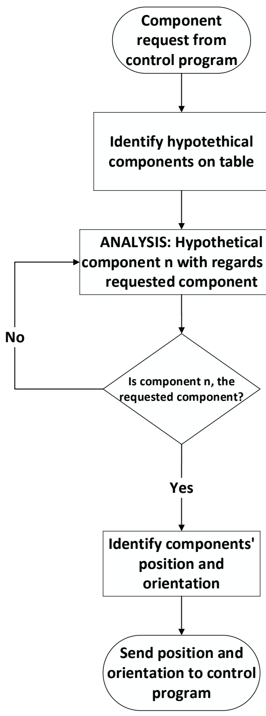
Component request from control program
Identify hypotethical components on table
ANALYSIS: Hypothetical component n with regards to requested component
No
Is component n, the requested component?
Yes
Identify components' position and orientation
Send position and orientation to control program

Fig. 2. Image analysis sequence

The different image analyzing functions are programmed using Scorpion toolboxes, acting as modules of code with generic functionalities. The main control program communicates to the image analysis program the desired inspection, and the results of the analysis are communicated back to the main control program. A more detailed description of the communication protocol is presented in Section 6.

The image analysis program starts when the main control program requests the position and orientation of a specific component. Hypothetical components, i.e. image regions that contain pixel values different from the values corresponding to an empty working surface, are detected by the image analysis program.

The analyzing toolbox iterates through all the hypothetical components, in search for specific geometrical features, unique for the component that has been requested by the main program. The iterative operation stops when the requested component is found. The program finally identifies the position and orientation of the object, which are then sent to the main control program.

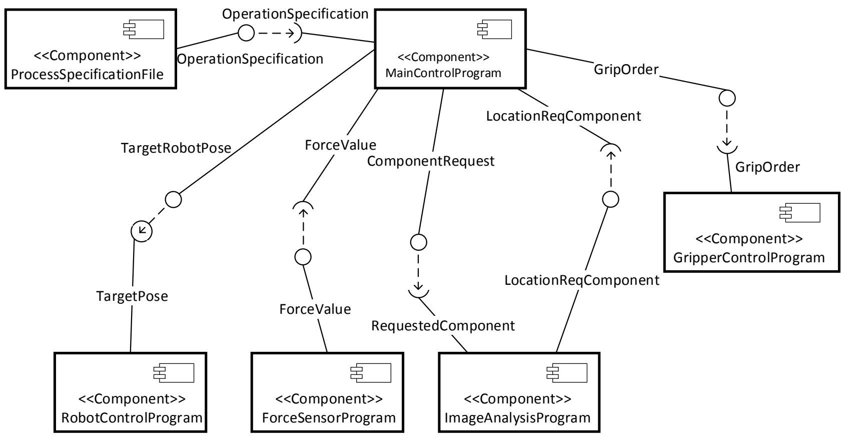
OperationSpecification
OperationSpecification
<<Component>> ProcessSpecificationFile
<<Component>> MainControlProgram
GripOrder
LocationReqComponent
TargetRobotPose
ForceValue
ComponentRequest
GripOrder
<<Component>> GripperControlProgram
LocationReqComponent
TargetPose
ForceValue
RequestedComponent
<<Component>> RobotControlProgram
<<Component>> ForceSensorProgram
<<Component>> ImageAnalysisProgram

Fig. 3 Component diagram of cell

4. Control Architecture

For the control system, a modular approach orchestrated by a main program has been chosen, see Figure 3. It is interesting to note that there are conflicting opinions and interests in this area. Until recently the robot manufacturers have built the robot controllers with the idea that the main logic would be implemented in the robot control. On the vision side, machine vision application developers often conceive the vision system as the place where the main control should reside, with robots and other sensors as slaves.

This cell was developed with the idea that an automated system is a collection of more or less intelligent devices that need good interfaces to communicate. Advanced logic is hard to implement in a system specialized for vision or robot movement, this kind of logic and orchestration is therefore implemented as an external software component in the cell.

Most of the control programs have been developed using the Python programming language. There are many parameters to consider for choosing the best environment solution for implementing the logic. We often implement the main logic using the Python programming with the arguments that Python 1) is one the highest level programming languages available, 2) does not require compiling, 3) has been designed from the ground up for integration, and 4) is now very often found as the available scripting languages for industrial software systems, for example the Scorpion vision system.

The number of components to mount in the project is currently around 10, but the number is increasing together with the number of variants, and it will probably be around 30 at production start. If every component is composed of a pick and a place path of 3 moves, this is about 180 moves to program. There are fundamentally two ways to write a control program for many operations:

a) A simple but long control program possibly split into smaller ones. This has the obvious flaw of being: 1) hard to maintain and; 2) costly to add a new variant since a new complete program for that part must be added.
b) A complex flexible program which handles all parts using both programming language features and sensors to adapt to different part geometries. The obvious pitfall here is of course a too high level of complexity.

Manufacturing flexibility is a topic that we, as researchers, are interested in developing further. We frequently collaborate with companies needing to reduce reconfiguration cost for variant handling or introduction of new products. Therefore, we have embraced the second option (b) and our work has focused on developing solutions to reducing maintenance and complexity.

5. Robot control programming

The robots from Universal Robots can be controlled with a few alternative methods. The method often demonstrated by the company to new customers is to use the graphical interface and jogging. Although quick and simple for a simple operation, this method does not allow to reach the precision necessary for the assembly of small electronic products.

The second official programming method is to write a complex program using the offered URScript language. There are two issues with this approach:

a) The robot controller is not designed to develop large programs; this can be solved since Universal Robot offers the possibility to write programs offline and send them to the robot as TCP IP.
b) The URScript is a simple programming language with limited support for common programming language features found in generic languages such as C#, Python and others.

We therefore chose to program the robot using an alternative method: the Python programming language and the python-urx [19] library written by one of the authors a few years ago. This library has already been used and referenced in a few publications from other teams [20, 21]. This approach allows the use of all the Python ecosystem libraries to program robots such as multithreading and homogeneous matrix through python-math3d [22]. Example code can be seen in Figure 4.

```
# connect and set robot using python-urx
rob = urx.Robot("192.168.1.100")
rob.set_tcp((0,0,0,0,0,0))
rob.set_payload(0.5, (0,0,0))
# return transformation matrix from base to tcp
t = rob.get_pose()
# matrix tranformation using python-math3d library
pose.orient.rotate_yb(90)
# move robot to new pose using python-urx
rob.movel(pose, acc=0.1, vel=0.2)
```

Fig. 4. Code example of a simple robot operation

6. Vision system communication

Since the core logic is written in an external software (Section 4), a solution to communicate with the vision system is required. Scorpion vision supports several communication modules such as RS232, OPC-DA (client), IO board, Profibus, modbus TCP or TCP/IP socket. Some of these are implemented in Scopion while others are available through the Scorpion python support.

For control with external software, TCP/IP is probably the most flexible and most used solution. Our team has developed several variants of custom protocols implemented at different customers or lab. This requires custom parsing of TCP stream on both sides and can be cumbersome to modify, thus XML messaging has also been used.

```
# initialize communication with scorpion vision
scrp = scorpion.Scorpion(ip, cfg.vision_port)
# take a picture
scrp.trigger()
# set scorpion values
v.set_value("robot_calib.robot_poses.pose1_x", 9.9)
# run a scorpion toobox
scrp.run_tool('RobotCalibration')
#run a tool and read following values when tool is
finished
x, y, c = scrp.run_tool("find_jig",
                        results=["find_jig.x",
                                 "find_jig.y",
                                 "find_jig.c"])
```

Fig. 5. Python commands to vision program

Based on preceding work, we propose to send JSON (JavaScript Object Notation) messages using the JSON-RPC protocol. This solution is currently implemented and tested in our cell. JSON is a lightweight data-interchange format, which is now often advised as a lighter and more human friendly alternative to XML. The implementation is open-source and can be found on the following repository: *https://github.com/SintefRaufossManufacturing/scorpion-json-rpc*
The current implementation allows exposing, configuring and running generic Scorpion toolboxes from python, thus the custom JSON/socket code does not need to be changed between projects. The vision program is implemented in Scorpion using the usual GUI Scorpion toolboxes. The toolboxes are run from the external cell control system using simple JSON-RPC calls, see Figure 5 for example code from Python. Any programming language supporting sockets and JSON can be used to control scorpion such as C#, C++, Go, etc depending on project requirements.

7. Pick and place process language

Pick and place operations are very repetitive, some operations may be complex but most are very similar. One can reduce programming time using methods and objects. For example:

```
op = MyOperation()
op.pick_pose = PP # specify pick pose for operation
op.tool = TL # specify tool for operation
op.exec()  #execute operation
```

Fig. 6. Schematic example of operation definition

This does the work, but this way of programming might not be the most efficient for defining such operations. In the future one may want to use a user interface, but as an intermediate step we propose a generic Pick and Place Process Language. This method is implemented and tested in our cell and enables the creation of GUI application to program the operations.

An overview of the major process definition languages can be found on internet [23]. These languages seem to focus on business processes and the application to robotic operations does not seem straightforward. In addition they are very verbose, this might in part be a consequence using of XML. Qiao et al. [24] present an attempt to specify a manufacturing language, which focuses on manufacturing plan control, and does not really fit robot move definition. The format is also not very human friendly. The software generation system presented by Spooner and Creak [25] seems to be alternative to PLC languages such as IEC 61131-3.

In our specific cell, JSON was considered since it is already used for vision communication, but JSON is rather limited in this scenario. Especially it does not support comments by design while we expect the description of a pick and place operation to be documented. A custom XML format is once again a possibility but XML is not easily readable for humans and very verbose.

We therefore have chosen to use YAML, *a human friendly data serialization standard for all programming languages*. YAML is by design easily editable and readable by humans, and is also easy to export and import to software systems. YAML also allow the definition of references. For example variant X1 of product X can reuse all operations of variant X2 and overwrite or add some.

An extract of a pick and place operation defined in YAML can be seen in Figure 7. It shows an operation where a part called 'component_1' whose position is assessed by a vision system is to be placed on a part called 'jig' with a given offset and approach direction.

The main control program reads the process.yaml configuration file at startup and executes operations as defined in the file. It is possible to have one YAML file per product or a large file describing several variants or products.

```
jig:
    localize:
        pose: vision # vision system provides pick coords
        offset: -0.007, -0.02, 0.33, 0, 1.57, 2.25
component_1:
    gripper:
        type: vacuum
    pick:
        pre:
            io: 6, 0# reset robot io before pick operation
        pose: vision
        offset: 0, 0, 0.007, 0, 0, +1.54
    place:
        pose: jig        # place on component named jig
        offset: 0, -0.037, 0.010, 0, 0, -2.3
        post:
            io: 6, 1# set robot io after place operation
```

Fig. 7. YAML process specification file example

8. Conclusion

In this paper, a prototype cell for flexible assembly of electronic components has been presented. Special attention has been given to the generic aspects of the systems and several solutions to increase flexibility and reduce programming time have been proposed.

Different aspects of the cell that contribute to system system flexibility from hardware to software are presented and discussed.

The cell is based on the use of a single camera that is used for locating the 3D coordinates of randomly placed components, which are pick and placed by a robot using a set of grippers for locating and assembling the components in modular fixtures.

Use of a process specification file contributing to faster operation specification is a central part of the system. The file, written in a human readable format, allows the user to specify in detail the different aspects to be considered for the assembly operation. Moreover, changes to the assembly operation can easily be implemented without investing large amounts of time for adapting system control code.

The cell is still at the prototype stadium but it has already provided results, such as triggering of design changes of the product and refinement or development the concepts described in this paper.

Acknowledgements

The authors would like to thank Norway's Research council (Norges Forskningsrådet) for the partial funding of the Parabel and Noca projects, where the work presented in the paper has taken place. In addition, we would like thank the industrial partners (Autronica, Inventas, Simpro/Noca) in these projects for their collaboration, guidance and feedback.

References

[1] Merriam-Webster Dictionary of the English Language (www.merriam-webster.com)
[2] De Toni A, Tonchia S. Manufacturing flexibility: a literature review. International journal of production research, 36(6); 1998, p.1587-1617.
[3] Sohlenius G. Concurrent Engineering. CIRP Annals-Manufacturing Technology; 1992
[4] Michalos G, Makris S, Papakostas N, Mourtzis D, Chryssolouris G. Automotive assembly technologies review: challenges and outlook for a flexible and adaptive approach. CIRP Journal of Manufacturing Science and Technology, Volume 2, Issue 2; 2010, p. 81-91
[5] Krüger J, Schreck G, Surdilovic D. Dual arm robot for flexible and cooperative assembly. CIRP Annals Manufacturing technology, Volume 60 (1); 2011, p. 5-8
[6] Tsarouchi P, Makris S, Michalos G, Stefos M, Fourtakas K, Kaltsoukalas K, Kontovrakis D, Chryssolouris G; Robotized assembly process using Dual arm robot; CATS 2014, 5th CIRP Conference on Assembly Technologies and Systems , Volume 23; 2014, p. 47-52
[7] Michalos G, Makris S, Spiliotopoulos J, Misios I, Tsarouchi P, Chryssolouris G; ROBO-PARTNER: Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future. Procedia CIRP, 23; 2014, p. 71-76.
[8] Shirinzadeh Bi. Flexible fixturing for workpiece positioning and constraining. Assembly Automation 22.2; 2002 p. 112-120
[9] Handelsman M. Advances in Industrial Robot Intelligence; 2016 http://www.industrialcontroldesignline.com/193001531;jsessionid=QXL0L5R3DTJPUQSNDLQSKHSCJUNN2JVN?printableArticle=true
[10] Solhaug A, Wetterwald LE, Dransfeld S, Knauserud O. Safe Assembly of Low Volume Complex Products. Proceedings of ISR/Robotik 2014, 41st International Symposium on Robotics; June 2014, p.1-8
[11] Jorg S, Langwald J, Stelter J, Hirzinger G, Natale C. Flexible robot-assembly using a multi-sensory approach. Proceedings of ICRA 2000, IEEE International Conference on Robotics and Automation; 2000, vol.4, p.3687-3694
[12] Pena-Cabrera M, Lopez-Juarez I, Rios-Cabrera R, Corona-Castuera J. Machine vision approach for robotic assembly. Assembly Automation 25.3; 2005 p. 204-216.
[13] Huang GQ, Zhang YF, Chen X, Newman, ST. RFID-enabled real-time wireless manufacturing for adaptive assembly planning and control. Journal of Intelligent Manufacturing, 19(6); 2008 p. 701-713
[14] Tsarouchi P, Michalos G, Makris S, Chryssolouris G; Vision System for Robotic Handling of Randomly Placed Objects; Procedia CIRP, 2nd CIRP Global Web Conference (CIRPe2013), Volume 9; 2013, p. 61-66
[15] Skotheim O, Lind M, Ystgaard P, Fjerdingen SA. A flexible 3D object localization system for industrial part handling. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2012 p.3326-3333
[16] http://www.universal-robots.com/products/ur5-robot/
[17] http://scorpion.tordivel.no/
[18] Lanitis A, Taylor CJ, Cootes TF. A Generic System For Classifying Variable Objects Using Flexible Template Matching. BMVC; 1993
[19] https://github.com/SintefRaufossManufacturing/python-urx
[20] Radlak K, Fojcik M. Integration of Robotic Arm Manipulator with ComputerVision in a Project-Based Learning Environment. Frontiers in Education Conference (FIE); 2015
[21] Elashry K, Ruairi G. An Approach to Automated Construction Using Adaptive Programing. Robotic Fabrication in Architecture, Art and Design 2014. Springer International Publishing; 2014. p.51-66.
[22] http://git.automatics.dyndns.dk/?p=pymath3d.git
[23] www.ebpml.org/status
[24] Qiao L, Kao S, Zhang Y. Manufacturing process modelling using process specification language. The International Journal of Advanced Manufacturing Technology; July 2011, Volume 55, Issue 5, p. 549-563
[25] Spooner N R, Creak G A. Process description language: an experiment in robust programming for manufacturing systems. Intelligent Systems in Design and Manufacturing, 227; Oct 1998