

# Interactive, Agent Based, Modeling and Simulation of Virtual Manufacturing Assemblies\*

Yi Yan, S. Ramaswamy

Software and Data Engineering Laboratory, School of Computer and Applied Sciences

Georgia Southwestern State University, Americus, GA 31709. Phone: 912-931-2100, email: srini@acm.org

**Abstract** - The application of agent-based approaches to developing interactive, virtual environments for manufacturing assembly processes is explored and demonstrated in this paper. The overall research focus is on the investigation and integration of appropriate behavior specification mechanisms with virtual reality to build interactive virtual environments for dynamic online simulation based design of manufacturing systems, with the intent of designing intelligent and robust manufacturing control software. Specifically the design of virtual environments using an agent-based design framework, taking into account risks and uncertainties, while embedding domain-specific intelligence about system failures and breakdowns, is investigated.

## 1. Introduction

Flexibility, reliability, autonomy and efficiency are important design characteristics of automated manufacturing systems [9]. XAnimate, a graphical, three dimensional simulation tool, which displays the numerical results of kinematic, dynamic equations and animates robot motion, is presented in [3]. It helps in design inspection and the observation of system performance. However, tools such as XAnimate do not provide any mechanisms for *interactive* dynamic simulation. Virtual reality is a combination of technologies whose interfaces with the human user can so dominate one's senses that the user intuitively interacts with the immersive and dynamic computed-generated environment [1]. The application of virtual reality for research issues in robot gripper grasping is presented in [11]. In [6], virtual prototyping is used to visualize a mechanical assembly with deformable components and test the system CAD model before it is physically fabricated. The simulation is interactive and dynamic, thereby allowing for the interactive simulation of real-world situations.

In virtual environments, real world objects are represented in detail as three-dimensional objects. The rapid growth of Internet software has fostered the development of Distributed Virtual Environments (DVEs). In such an environment, a group of people (agents) operate within the DVE and they have the ability to interact with each other and their surrounding environment. In military applications, virtual battlefields and armies of virtual soldiers are created to train

individual soldiers who are allowed to maneuver entities such as tanks and aircrafts [7]. Such an approach is highly cost-effective in terms of training and education. In recent years, virtual environments for housing, architecture and entertainment have also been developed. Researchers in biology, medicine and other engineering areas are using virtual simulation techniques to investigate and prototype research applications without investing substantial resources into building actual prototype systems. Virtual prototyping results can be used to improve system design and it can effectively reduce product development cycle time.

Advances in virtual reality have made it feasible to directly utilize virtual reality techniques for the modeling and realization of virtual manufacturing environments. The use of virtual reality in simulating manufacturing environments give designers the opportunity to play a pro-active role in identifying flaws and optimizing the design. Current techniques do not allow the designer to be progressively involved in the development process. Often, the designer is limited to the initial specification stages and her/his experience is not exploited completely. This approach offers the following advantages: (i) Evaluate the feasibility of the floor layout for maximum efficiency, (ii) Understand and refine the job flow in an automated manufacturing environment, (iii) Use interactive simulation techniques to speed the development of automated manufacturing systems, and, (iv) Evaluate the interaction between the various subsystems in the environment as well as design efficient human-interface components for the realization of the actual factory floor.

In this paper, a virtual environment prototyping model, as discussed later in this section, is used to develop the manufacturing application within a Distributed Interactive Virtual Environment, also known as DIVE, a virtual reality tool developed by the Swedish Institute of Computer Science [5]. DIVE is a loosely coupled heterogeneous distributed system running in an UNIX environment. Each virtual environment in DIVE is called a world and DIVE supports 3-D visual and audio worlds. Each "world" can have many different objects and has a global coordinate system. Each object is capable of exhibiting multiple functionality. For example, an object can send a signal when it touches another object in the virtual environment. Participants in the virtual

---

\* The research is part of the Virtual Environments in Education, Robotics and Automation (VERAM) project at the Software and Data Engineering laboratory at Georgia Southwestern State University and is supported in part by the National Science Foundation under grant #DMI-96100055 and the Georgia Board of Regents.

environment can interact with the virtual world by using either the keyboard or a mouse. Multi-users can join in the same virtual world over the Internet. In [8], and references therein, simple virtual simulations using DIVE are presented.

Any tool used for simulation based systems design must support scaleable hierarchy and minimal intrusion. By scaleable hierarchy, it is meant that the design should expand like a conical tree structure in depth [10]. A design will support minimal intrusion if it allows for the streamlined representation of the system communication structure (between the various sub-systems, or, sub-designs). This can be accomplished using object-oriented design techniques. Several modeling tools and methodologies, based on object-oriented techniques, have been developed for system specification and analysis. Current agent-based design methodologies extend the object-oriented design approach to intelligent agents operating in distributed environments. An agent, as considered in this research, is akin to a class of objects along with appropriate mechanisms for exhibiting intelligent behavior. Thus, objects can be used to implement agents.

From a practical point of view, such agent-based approaches for systems development will enable effective management and revision control during software evolution. By virtue of the design technique, such software systems will be easy to modify and maintain. The approach offers the following advantages. (i) Objects in the system can be defined, both in structure and behavior, in close correspondence to real-world objects. (ii) Object reuse aids in the rapid development of new software elements. (iii) Object inheritance enables the creation of new objects and relationships with minimal effort. (iv) Object encapsulation provides the appropriate distinction between object boundaries and is effective in identifying and controlling the propagation of errors. It also helps to establish well-defined communication mechanisms and to set up data and process security features.

Gathering the advantages offered by the above techniques, an agent based virtual environment development cycle has been defined, as shown in Figure 1. It consists of four distinct stages: (i) **Responsibility Analysis**: In this stage, the system requirements are used to derive various simple responsibilities (or goals) for the effective functioning of the system. (ii) **Agent Design**: In this stage the various system agents / objects (defined in relationship to real world objects) are defined based on the responsibilities identified in the previous step. (iii) **Evaluation**: The evaluation of agent responsibilities with respect to the objectives of the agent goals to better address the problem. This is termed the requirements tuning phase and it might be several iterations before a convenient and cohesive model is developed. (iv) **Refinement**: The refinement of overall system behaviors leads to the behavior tuning phase that results in a possible redesign for agents and the refinement of individual agent behaviors. It is to

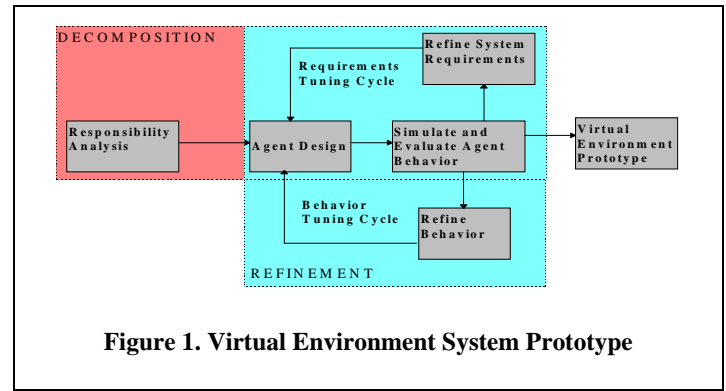


Figure 1. Virtual Environment System Prototype

be noted that the requirements and behavior tuning phases do not take place in isolation of one another. These are intricately linked and a designer may, in fact, iterate on these phases. Moreover, these phases may also be interleaved across these iterations. This interaction between these two phases is illustrated in Figure 1.

Virtual reality may be used for both spatial and logical reasoning issues in manufacturing environments. The application of virtual reality to spatial reasoning capabilities is well justified, especially in case of domains that exhibit the following characteristics: (i) hazards to human health, (ii) space / undersea exploration, and, (iii) understanding multi-dimensional problems such as atomic and molecular structures, etc. The application of virtual reality to logical reasoning issues in manufacturing is an evolving area of research and it is presently very controversial with respect to its net worth to any organization. It is the authors view that future manufacturing applications in spite of greater and wider automation, will keep the human element in the loop, wherein they will be required to have both critical decision making skills and keen physiological reflexes (aural and visual observation skills). In such cases, it becomes imperative that the right person be assigned to the right group of automated resources on the manufacturing floor to maximize efficiency and minimize safety related risks. Such a match often involves not only the technical skill of the human in question but will also involve a physiological assessment of the human so that the human can rightly "fit" into the working environment. An immersive virtual environment from this perspective, will be worthwhile in training new employees to interact with the virtual shop floor resources (current examples in this regard

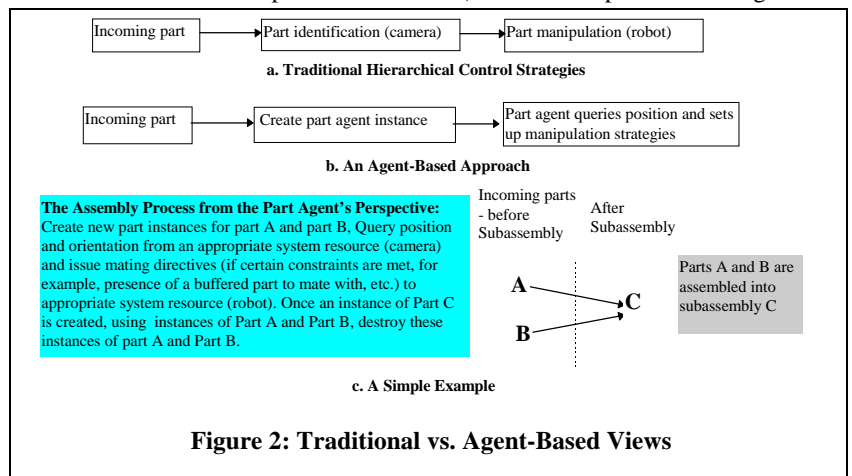


Figure 2: Traditional vs. Agent-Based Views

include flight simulators). It can be used to study employee reflexes to frequently occurring shop floor situations or exceptional situations (failures, etc.) that are rare in occurrence and thereby help in the placement (or job rotation) of these employees in appropriate environments. Such an exercise may also help to develop better human computer interfacing techniques based on advances in multi-media. Results from implementing such virtual environments can be used to judge non-logical issues such as fatigue

and efficiency of employees over time to shop floor situations.

This paper is organized as follows: Section 2 differentiates traditional approaches to agent-based design approaches and summarizes the example assembly process. Section 3 addresses the robot motion control and orientation adjustment algorithms used by the robot agent. Section 4 analyses different kinds of system failures and failure handling mechanisms implemented within the virtual environment. Section 5 concludes the paper and presents issues for future research.

## 2. Agents and the Example Assembly Process

### 2.1. Software Agents in Manufacturing Environments

It is well known that it has been difficult to find a widely acceptable definition for the word - "software agent". In [4], a brief introduction to its different meanings is provided. However, despite the inability of the agent research community to reach an amicable definition, one must be able to define some of the innate attributes that characterize an agent in *any* system. With such attributes in mind, instead of trying to define what a Software Agent (SA) is, a software agent is defined to be any system component (or part, or subsystem, etc.) that appropriately answers the question "aRe yoU A Software Agent? (R U A SA?)" where, R represents the ability to Represent, U the ability to Understand, and A the ability to Act. In detail, these attributes mean the following: (i)

**Represent:** Represent both behavioral and structural characteristics of the software agents comprising the system, (ii) **Understand:** Understand both local and system objectives, prioritize them according to knowledge about the other system agents, the environment and itself, (iii) **Act:** Perform either deliberative or reactive actions based on the agent's knowledge, with respect to events generated by itself, the environment and other system agents. The difference between the traditional and hierarchical control strategies and an agent-based control strategy is illustrated in Figure 2.

Software agent researchers have identified various issues in agent-design that need to be addressed. With respect to manufacturing systems, some of the issues that are worth investigating include: (i) Are agents in a manufacturing environment active, passive or both? For example, a shop floor may be controlled by an active shop floor agent, while a shop floor resource may be controlled by an active or passive agent, depending on its current use. (ii) Are the agents

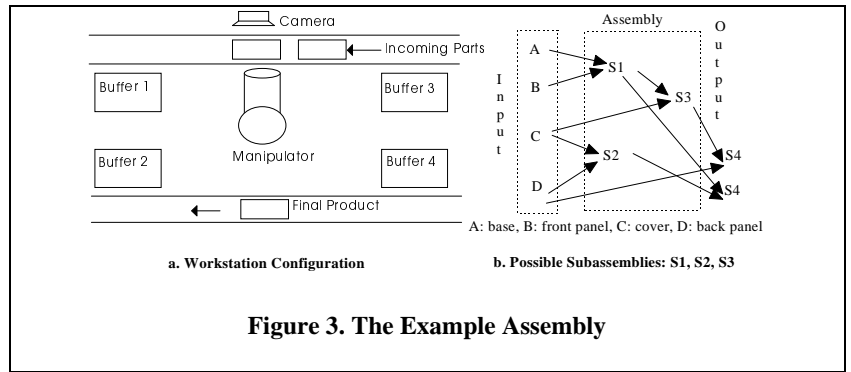


Figure 3. The Example Assembly

Part	Shape	Height(cm)	Length(cm)	Radius(cm)
Base	cylinder	10	-	50
Pole	cylinder	70	-	15
Big arm	beam	-	60	-
Small arm	beam	-	60	-
Small ball	ball	-	-	2.5
Gripper	bent beam	-	11.5	-

Table 1. Puma Components

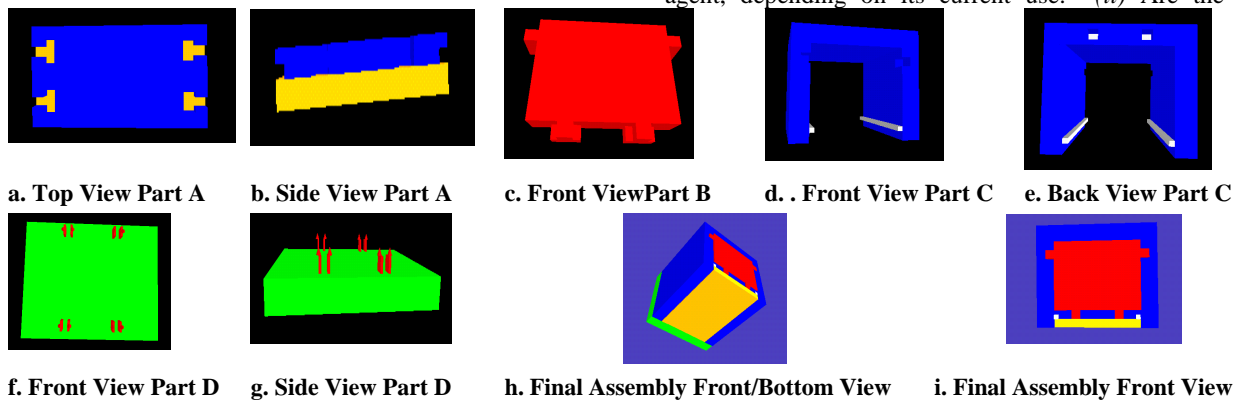


Figure 4. The Parts and the Assembly

powerful enough to have a private thread of control? Do they have access to “knowledge” contained in the process or processes that bring them into existence? If so, what are their access restrictions and how can they be designed so that they do not misuse such knowledge? For example, a robot agent may have built-in priorities about job scheduling. If a part agent gets access to the robot’s knowledge, can it exploit / manipulate this knowledge? (iii) Will the agents be cooperative or self-serving? In either case, how does one track and amend (if necessary) such behavior with respect to the system’s objectives? If cooperative, how is the extent of cooperation measured? If self serving, how does one devise measures to stem negative effects? (iv) What is the development process for these agents? How do we decide what amount of intelligence is necessary? (v) Finally, if the agent-based architecture is integrated with a distributed virtual reality framework on the Internet, will there be mobile agents which can “fly” across the network? If so, how will they be defined and distinguished from other “non-mobile” agents, what will be their role and what will be their limitations?

It is to be noted that in this paper, the interest is not in finding generally applicable analytical solutions to the above issues, but rather working on feasible implementations on a specific problem. The problem, a manufacturing assembly process for a PC manufacturing process, is discussed in the next subsection.

## 2.2. The Example Assembly Process

The prototyping environment illustrated in Figure 1 is used to model and simulate the assembly process shown in Figure 3a. Figure 3b details the assembly model for a PC assembly process. The assembly consists of four unique parts which arrive in random order. The four parts, as illustrated in Figure 4, include the following: the base or the mother board (part A), the front panel (part B), the cover (part C) and the back panel (part D). The motherboard (Figure 4a-Figure 4b) is a rectangular object with a pair of groves at each side of the board. It has two holes at its backside to lock the back panel and two T-shape dents at the front side to hold the front panel. The front panel (Figure 4c) has two T-shape feet and two cubic ears. The two T-shape feet will be inserted in the motherboard’s T-shape dents. The two ears of front panel are used to link with the cover. The  $\Pi$ -shape cover (Figure 4d-Figure 4e) is composed of three large pieces: the top panel and two vertical pieces. There is a horizontal lace at the inner side of each vertical piece. These laces will be coupled with the groves in both sides of the motherboard. At the front side of the cover, there is an indent at each vertical piece’s front side, which will embrace the two ears of front panel. There are two square holes at the rear side of the cover’s top. The cover has the same length as the motherboard. The back panel (Figure 4f - Figure 4g) has four auto-lock spikes, two at the top and other two at the bottom. The four spikes are pushed in the four holes at the backside of the motherboard and  $\Pi$ -shape cover when it is assembled with other parts. The four parts

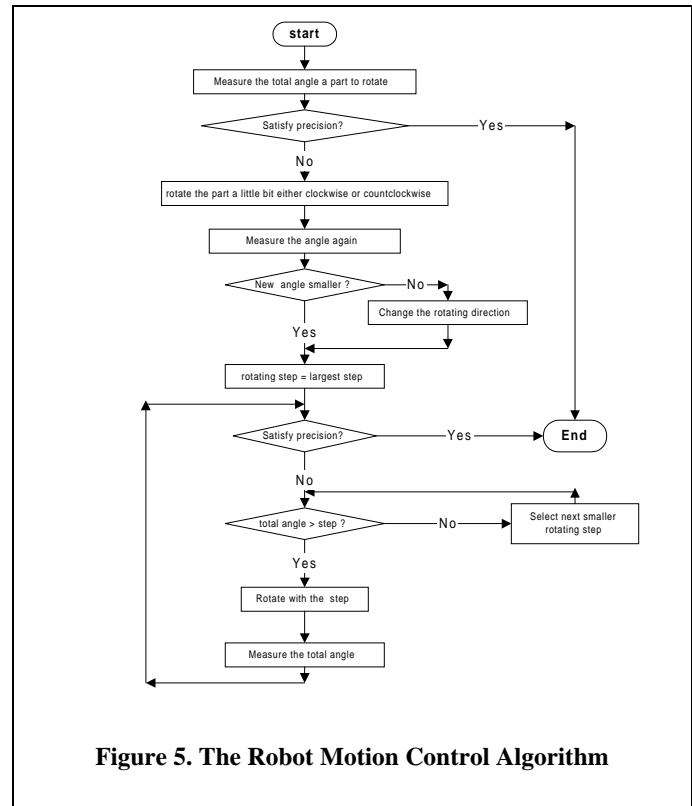


Figure 5. The Robot Motion Control Algorithm

will be interlocked when the box is assembled<sup>1</sup>. The final assembled object is also shown in Figure 4h and Figure 4i.

Since the part objects arrive in random order, the robot has to discern which part has arrived, then determine if the other parts necessary for the assembly have already been buffered. Incoming part object types are communicated to the robot agent by the part agent. If other necessary part objects are buffered, the robot agent then combines these objects in the desired order to form a subassembly object. Part objects contain information about the appropriate mating partners and the mating orientations which are communicated to the robot agent. However, if the part objects are not available, the robot buffers the current part object. Once individual part objects are assembled into a subassembly object, the part object instances are destroyed. Once the robot constructs a subassembly, it checks for the corresponding mating objects (either other subassemblies or parts) to complete the final product. If the corresponding objects are not present, the robot buffers the subassembly. Upon completion of the final product, the assembled product is deposited and sent down the output conveyor and the whole process then repeats itself again.

The shape characteristics of the parts determine the order of the assembly. The front panel will be inserted into the motherboard vertically. If the front panel arrives earlier than the motherboard, it will be buffered until the motherboard

<sup>1</sup> Parts are essentially called part objects / objects in the implementation of the virtual environment. Hereafter, these terms are used interchangeably. When discussing an implementation a part essentially refers to the part object.

comes in. If the motherboard comes earlier than the front panel, then the front panel can be put together with the motherboard directly to form subassembly S1. Similar to the subassembly of S1, S2 is assembled using the cover and the back panel. Moreover, when the cover comes in, if S1 has already been assembled, the cover is combined with S1 (this process has priority over forming subassembly S2, if the back panel is also buffered) by horizontal pushing to form subassembly S3. If the back panel comes earlier than the cover, the back panel will be buffered. When the back panel comes later than the cover and if S3 exists, then the back panel is assembled with S3 to form the PC box (S4). If both S1 and S2 have been assembled, they can be put together by horizontal pushing to form the PC box (S4). The possible orders of assembling process are sketched in Figure 3b. The details of the robot motion control and the part orientation adjustment algorithm is discussed in Section 3.

### 2.2.1. The PUMA 560

The Puma is a 6-DOF manipulator which consists of a large circular base, a tall vertical pole, a big arm, a small arm and an adroit gripper. The main dimension of each robot component is listed in Table 1. The vertical pole can rotate around its central axis. The big arm is hinged to the vertical pole by a horizontal cylinder. This horizontal cylinder can rotate while it is moving with the turning of the pole. The rotation of the hinge changes the big arm position. The small arm is connected to the big arm by a small horizontal cylinder hinge. This hinge can also rotate independently. The gripper is linked to the small arm through a small ball. The ball can rotate in any direction. The gripper can open and close to catch or release a part. Even if the big arm is not in the direction of the radius of the vertical pole, the small arm is designed to move in the vertical pole's radius direction at any time. Due to this structural characteristic, the moving of gripper can be decomposed in horizontal and vertical movements. The rotation of the pole triggers the horizontal movement of the gripper. The vertical movement of the gripper can be realized by rotating the big arm and small arm.

The small ball, which links the gripper and the small arm, is adroit. It can rotate in any direction with any angle. So the gripper can rotate to any degree around a vertical line. Due to space limitations, the small ball can only rotate the gripper 90° around horizontal line. It is very powerful for adjusting part orientations. A sensor attached to the small ball can detect the position of every object in the Puma's world.

### 2.2.2. Conveyor Belts, Buffers

The conveyor belts deliver incoming parts in random order and transfer out assembled products. The distance between the belts is 120 cm and they are

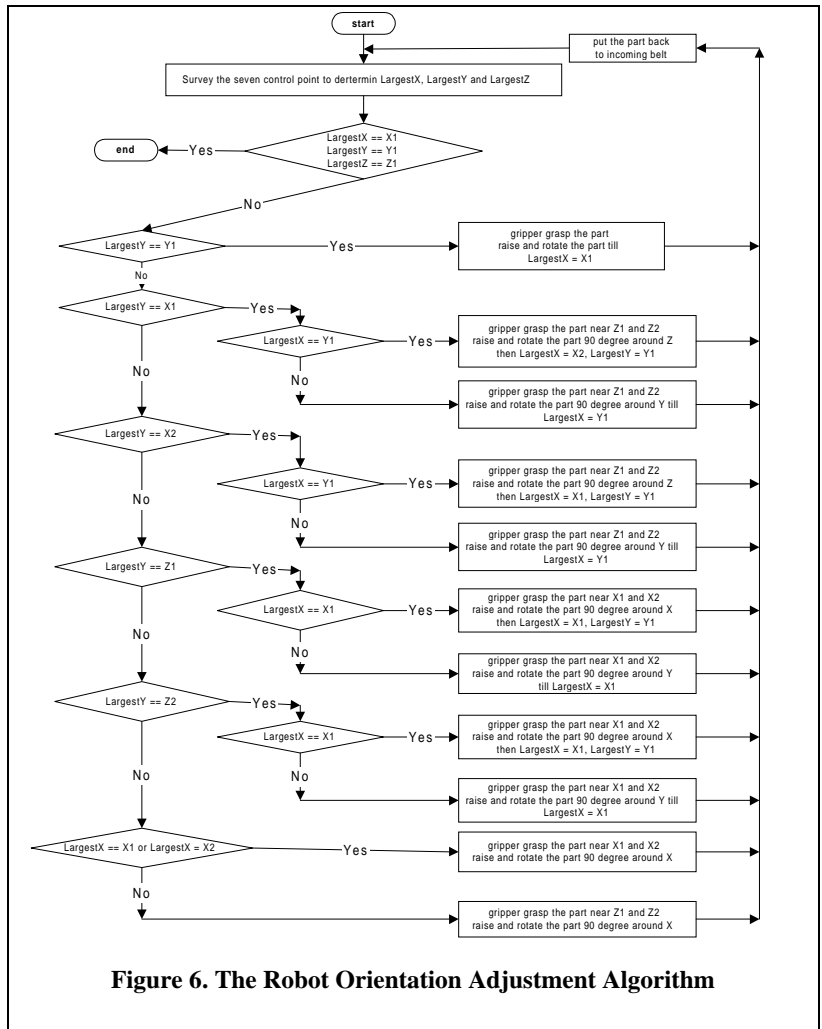


Figure 6. The Robot Orientation Adjustment Algorithm

parallel to each other. There are invisible obstacles, called stoppers, generally realized by an infra red or camera driven stopping mechanism, on both incoming and outgoing belts. When a part to be assembled is delivered, it will be stopped at the stopper. Stoppers can be added or removed according to the assembly requirements in the virtual environment. Four buffers are used to buffer intermediate parts and subassemblies during the assembly process. Failures due to irregular conveyor stopping may be easily implemented by user-induced errors as described in Section 4

### 3. Control Algorithms

When a part agent announces the arrival of an incoming part, the robot agent first inquires the part's position and then makes a decision to grasp the part. Parts in an assembly may be small and fragile; they can be easily destroyed by the

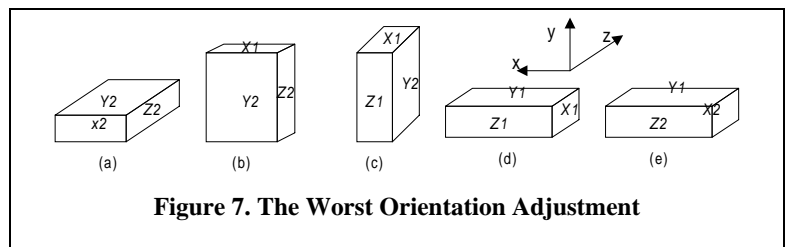


Figure 7. The Worst Orientation Adjustment

gripper if the robot cannot precisely control the movements of gripper. The robot motion control algorithm and the robot orientation adjustment algorithm are designed to work on fragile parts and improve system performance.

### 3.1. Robot Motion Control Algorithm

From the vertical pole to the small fingers of the gripper, all robot movements are rotations. Rotations are carried out in incremental steps. In each step, the robot rotates a small angle. The smaller the step, the smaller the total movement error. However, smaller rotating steps may cause serious performance problems. To efficiently perform the robot move operation, multiple rotating steps are used by this algorithm depending upon the distance to the part which is to be picked up / assembled.

The algorithm is a multi-step predict-and-move algorithm. It is divided into two phases (i) **Phase 1:** In phase 1, the correct

robot rotating direction is decided. If a robot needs to rotate by some angle, the angle is first measured. If the angle is less than the precision requirement, no rotation is needed. If the angle is larger than the precision requirement, the part is rotated by a very small step (less than the smallest rotating step). Then the total angle is measured again. If the new angle is smaller than previous measurement, the rotating step's direction is correct. Otherwise, the rotating direction needs to be changed. (ii) **Phase 2:** In this phase, the rotation is completed in the most efficient way. Initially, the largest rotating step is assumed and a check performed to see if the precision has been met before entering the loop. If the total angle is less than the current rotating step, set the next smaller step to be the rotating step. The recursive choice of rotating step repeats itself until a step smaller than the current total angle is found. Then the robot rotates the appropriate Puma component with that step in the pre-decided direction. After the rotation, the total angle is measured and compared with the precision requirement. If the precision criteria is not met, the process goes to selecting the rotating step.

Potential positioning errors can exist in each step of the robot movement. In this algorithm, the total angle is measured after each step of rotation. Due to this reason, errors in previous step movements do not have a cumulative effect to produce a larger error value. This makes the algorithm quite robust and efficient. This control algorithm is used to control the rotations of the pole,

big arm, small arm, ball and the gripper. In the implementation, there are up to five rotating steps in each procedure. The largest step is designed to be 200 times larger than the smallest step.

### 3.2. Orientation Adjustment Algorithm

Parts in the assembly are delivered by the part agent through the input conveyor not only in random order, but also in random orientations. For successful assembly, appropriate parts must be mated in the correct orientations. If a part's orientation is incorrect, it must be adjusted before it is assembled. In practice, different parts in various shapes / orientations may exist. The orientation adjustment algorithm is used to orient the parts in the right directions before mating. The burden of providing the information about part orientations and mating partners are on the part agent. The part agent embeds information about the current orientation,

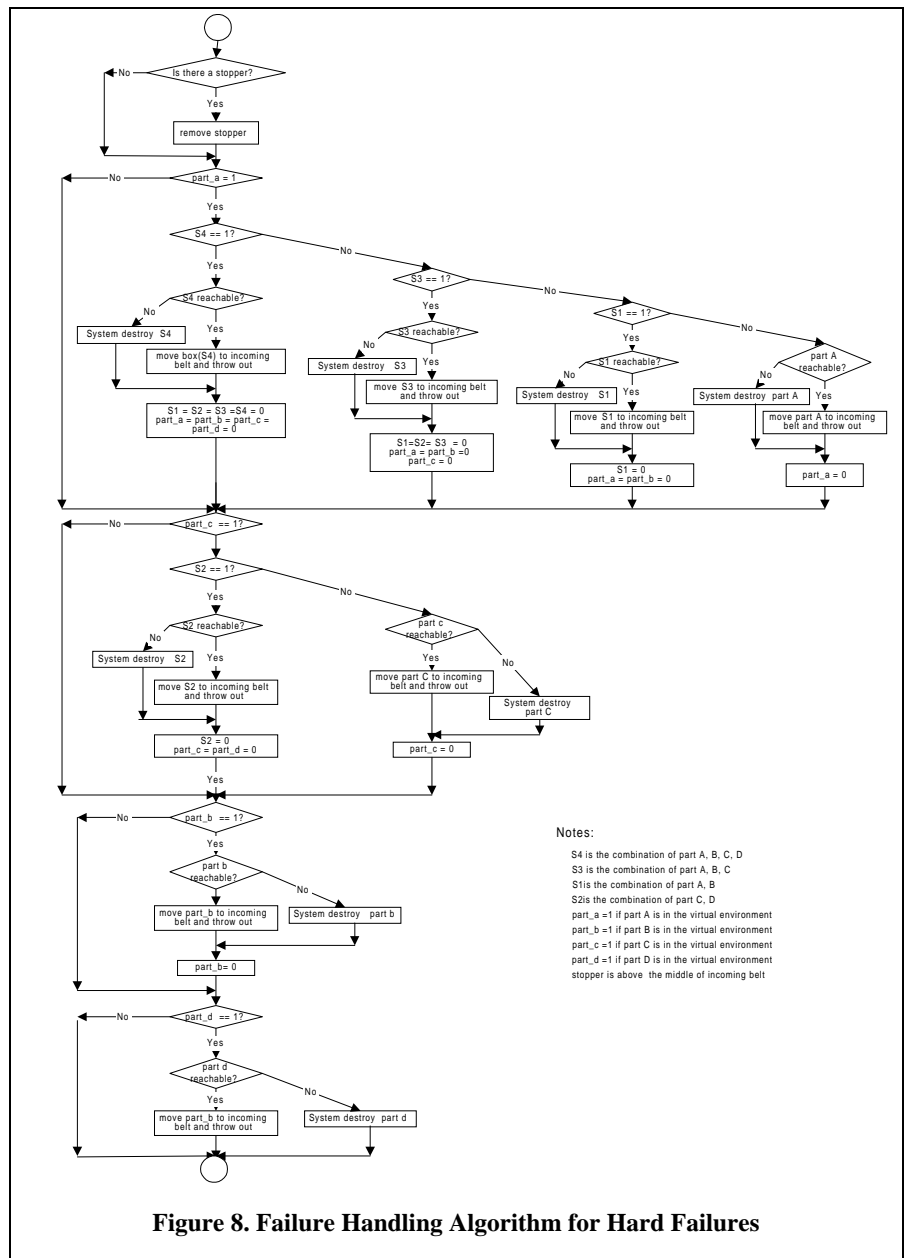
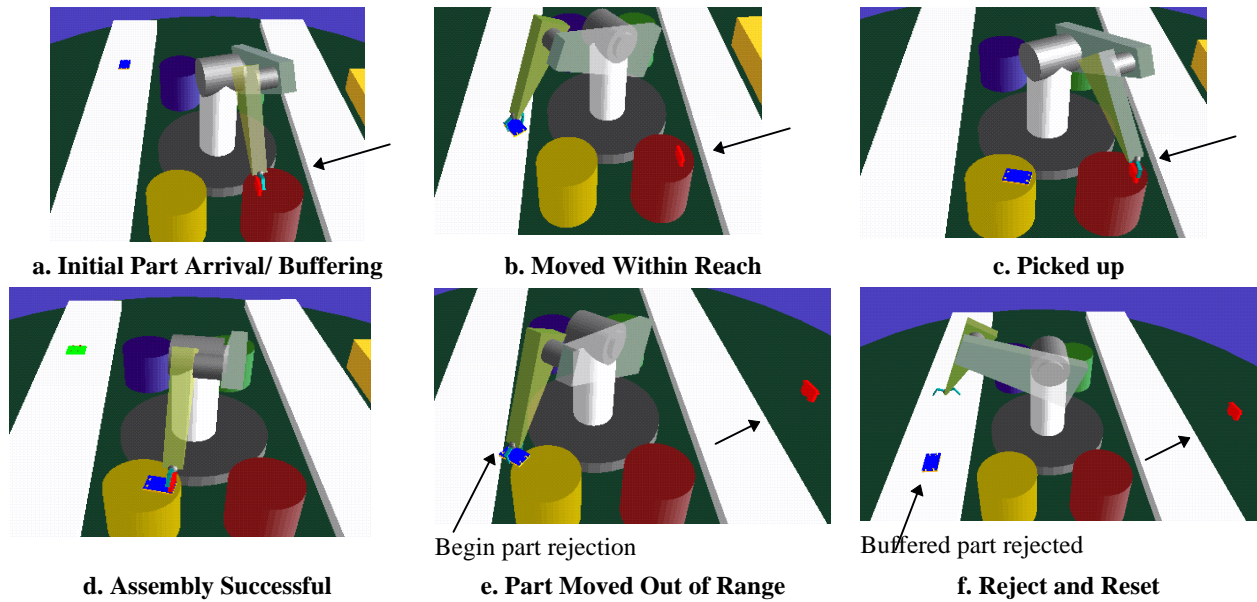


Figure 8. Failure Handling Algorithm for Hard Failures



**Figure 9. Handling User Induced Failures**

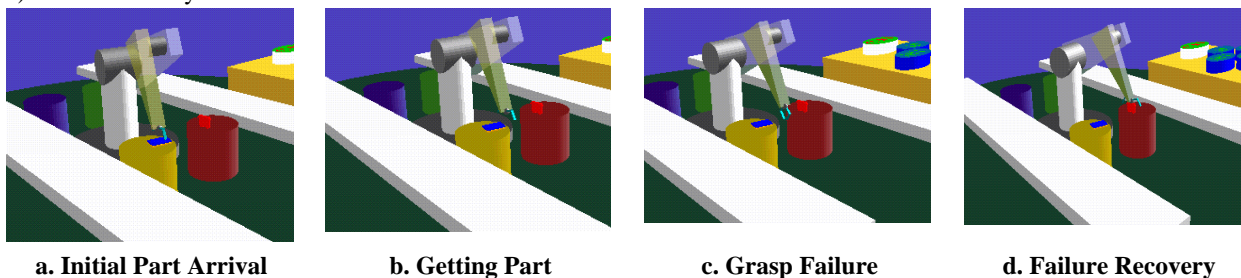
the correct orientation for mating and the compatibility of mating partners within the part specification as they arrive for assembly.

Real objects can be characterized by three dimensions. Since two points are required to completely define one dimension, six points are necessary to define an object's orientation. Moreover, every part has some regular geometric shape. In this algorithm, this shape is enclosed into a 6-face cube. It is further assumed that the part has only one orientation that is suitable for assembly. Therefore the algorithm is essentially to move the box until the part is in its assembly orientation. The centers of the six surfaces of the cube which encloses the part are defined as X1, X2, Y1, Y2, Z1 and Z2, with the center of the box defined by C. A part's proper incoming orientation for Puma system is shown in Figure 7e. If the part does not have this orientation, it must first be adjusted to this orientation.

When an object is delivered and it stops on the incoming belt, the object's position is first detected by a sensor and conveyed to the robot. Then the robot moves to the object with its gripper open. When the gripper is over the object, the gripper adjusts its orientation to a pair of control points (say, Y1 and Y2) which is very close to the X direction. Then the

orientation adjustment follows the algorithm in Figure 6. When an object is moved away from the incoming belt, the robot first raises the object to some height to avoid collision with other objects or the belt. In Figure 6, the largestX, largestY and largestZ denote the three control points which have the largest x, y and z value respectively. If an object comes in with a favorable orientation, i.e. largestX being X1, largestY being Y1 and largestZ being Z1, the orientation does not need to be adjusted. If the objects' Y1 is at the top surface, its orientation can be adjusted by rotating around the vertical axis until control point X1 has the largest x value. If one of control points in X1, X2, Z1 or Z2 is at the top surface, the robot first raises the object to some height, then turns the surface of Y1 to the top and puts the object back to the incoming belt. Then the orientation is adjusted similar to the case in which Y1 is at the top.

The worst orientation occurs when control point Y2 is at the top, as illustrated in Figure 7a. In this case, the LargestY is Y2 and LargestX is Z1. In this case, the object is grasped near Z1 and Z2, raised to some height, rotated 90° and put back, as illustrated in Figure 7b. For this orientation, the LargestY is X1 and the LargestX is Z1, so it is grasped near Z1 and Z2, raised, rotated 90° and put back as shown by Figure 7c. Now,



**Figure 10. Handling Grasping Failures**

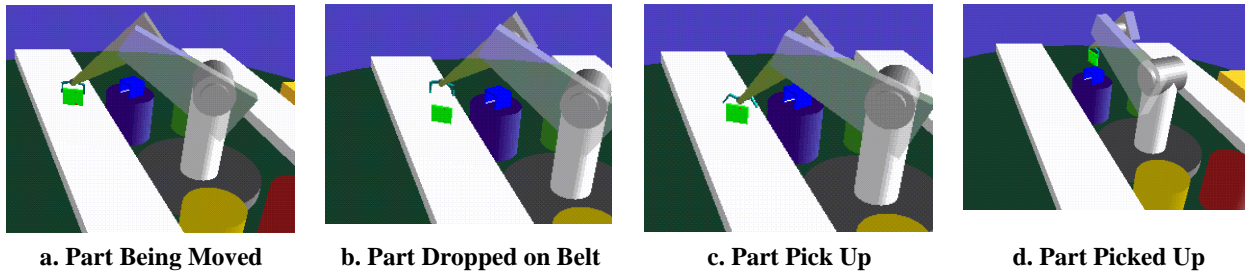


Figure 11. Move Failure - I

the LargestY is X1 and LargestX is Y1, it is grasped near Z1 and Z2, raised, rotated 90° and put back, as illustrated by Figure 7d. Finally, LargestY is Y1 and the orientation can be changed only by rotating the object around a vertical line. The final part orientation is shown in Figure 7e.

#### 4. Failure Handling

System failures happen randomly and are unpredictable. In a manufacturing assembly domain, like the problem investigated in this paper, it is crucial for the various system agents to exhibit some acceptable level of intelligence while handling failure situations. In this paper, potential system failures are classified as either external (caused by external factors) or internal failures (directly associated with the system). Internal failures may be further classified as hardware failures or software failures. Hardware failures can be minimized by good maintenance and component replacement policies. It is not the objective of this paper to discuss hardware failures or hardware failure recovery policies. Software failures are classified as either hard or soft failures. Hard failures are defined as failures in which the current operation has to be abandoned and restarted, while soft failures are defined to be failures in which the current operation can be performed to completion on successful recovery. To reduce software failures, fault-tolerant software techniques are necessary. Such techniques may allow the automatic recovery from system software failures. However, it is to be noted that different types of failures may dictate the use of different failure handling techniques.

Failures that need to be addressed in an interactive virtual simulation of manufacturing assembly processes may essentially be categorized into the following: (i) user induced failures, (ii) grasping failures, (iii) moving failures, and, (iv) grouping failures. Grouping failures may happen if a redundant part is delivered to the assembly, i.e., unnecessary, multiple instances of the same part object exists in a group/batch. It is assumed that the grouping of part objects

into a batch (a batch contains one each of the parts needed for the assembly) is fault free and thus, is not a possible scenario. However, such a failure may be handled very easily by disabling the stopper mechanism on the input conveyor and passing them to a reject area. The reject area is assumed to be at the end of the input conveyor belt.

While algorithms that deal with grasping and moving failures can be programmed into the software control mechanisms for the system, user induced failures may be effectively used in not only studying system responses, but also in studying user responses. Part objects that are out of the manipulator's reach are rejected by the robot agent. Once a part object that is out of range is detected, the robot continually searches for subassembly objects, followed by individual part objects, in that hierarchical order as illustrated by Figure 8. When any such objects are encountered it places them on the input conveyor and sends it to a reject area. Once all the objects in the system are rejected, the robot destroys the part that is out of reach automatically. In practice, this procedure will be accomplished through a operator request. In the sequel, the three different failure scenarios are discussed in more detail.

##### 4.1. Grasping Failures:

Grasping failures happen when the robot tries to reach a part object far away from the gripper. This failure is detected when the gripper closes completely without touching any object (i.e., the two fingers touch each other). Failure recovery is by repeating the sensing and grasping steps, i.e., when a grasping failure occurs, the robot initiates a re-detection of the target's position, re-plans its trajectory, and then moves to grasp the object. Figure 10 illustrates the handling of grasping failures.

In the implementation of the virtual environment, grasping failures are randomly generated. If a grasping failure is identified by the gripper, it generates a grasp failure signal. On this signal, the robot initiates a part relocation request which returns a revised target location. Given this information, the gripper's final position is recalculated, and robot moves

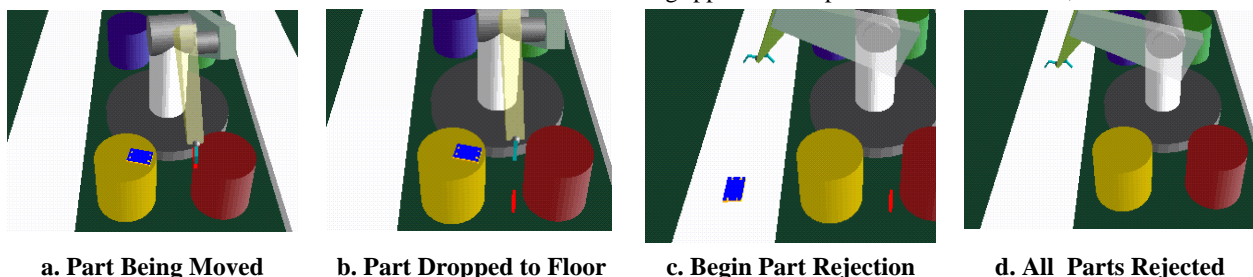


Figure 12. Move Failure - II

the gripper to its redefined position. In the simulation, while the orientation adjustment is being performed, grasp failures are not allowed to occur.

#### 4.2. User Induced Failures:

User induced failures are caused by a user's deliberate actions within the virtual environment. In the virtual environment, a user can move a part object when the object is either on the belt or on the buffer. For example, a user may move a part out of the robot's pickup range. Such failures are critical to understanding the robustness of the software algorithms used and the ability of the system agents to deal with such adverse situations. Although such failures may never occur in a real-world situation, these user induced failures may be used to study not only the system's predefined responses to such events, but in a multi-user training environment they may be effectively used to study operator responses in critical situations, thereby helping in skill matching and job placement.

User induced failures may produce two different kinds of failure situations: (i) **Move an object to another position within range:** When a user induced failure occurs and the object (part or subassembly) is moved to a new position within the robot's reachable space, the robot agent treats it like a grasping failure and it re-plans a new trajectory to pick it up. If it is a part object, then the object is picked up from its current position. If the object is a subassembly, then the part object to be mated with the subassembly is moved to this new position. The situation when a buffered part is moved is illustrated by the snapshots in Figure 9a - Figure 9d. (ii) **Move the object to an out of range position:** When a user induced failure occurs and the object is moved to a new position outside the robot's reach, the robot agent treats it like a moving failure (part dropped to the floor) and it rejects the current object and the rest of the objects in the assembly and the batch. This situation is illustrated by the snapshots in Figure 9e and Figure 9f.

#### 4.3. Moving Failures:

Moving failures may occur when a grasped object falls to the belt or to one of the buffers while being moved. Moving failures are also randomly generated in the simulation. When the grasped part object is moved through the failure location, it falls down. This random fall is either on the conveyor belts, the buffers or the ground.

In the virtual environment, only erect front and back panels (Parts B and D) can fall through the gaps between buffer and belt (i.e., the part widths are sufficient to allow the part to fall through the gap). Objects falling into these small gaps cannot be picked up by the robot, thereby leading to a hard failure. If the motherboard or the cover falls into these narrow gaps, they will stay either on a belt, the buffer, or partially on the belt and the buffer. Specifically, if the geometrical center of the part object falls in the gap, then the object will fall to the ground if the object can pass through the gap; otherwise, the object will not fall to the ground. If an object falls down, either between buffers 1 and 2, or between buffers 3 and 4, the robot can pick

up the object again. However, since such a fall may cause the object to be deformed, this is considered a hard failure. Thus, if a moving failure occurs and the object is not fallen to the ground, such a moving failure is considered a soft failure and the system recovers by failure handling. If the failure causes the object to fall to the ground, the failure is considered a hard failure and the robot just moves all the remaining objects out of the work space and requests operator assistance to destroy the fallen part. In the simulation, the fallen part is automatically destroyed after all the buffered objects are rejected.

Figure 11a - Figure 11d illustrate the case wherein the object is dropped on the conveyor belt. In this case the object is picked up and moved for assembly. Figure 12a - Figure 12d illustrate the case wherein the object is dropped on the floor between two buffers. In this case, the rejection algorithm, as discussed earlier, is activated to reject the remaining objects. The object that has fallen to the floor is thus rejected even though it could have been picked up and assembled.

### 5. Conclusions

This paper presents an agent based design of a virtual environment for a PC manufacturing assembly process. The multi-step predict-and-move algorithm has been devised to adaptively control robot movements. In this algorithm, first the correct direction for the move is decided by a small trial. Then the robot's movement is carried out with the largest feasible movement step. This provides an efficient and precise method to control robot movement. Due to the limitations of the Puma's gripper, a part's orientation cannot be changed in a single procedure. Therefore, the heuristic orientation adjustment algorithm has also been designed. User induced failures were interactively generated and accommodated online. Grasping and moving failures were randomly generated and handled effectively. The system agents, the robot and the part agents in particular, were designed to be capable of handling failure situations by communicating with one another and taking appropriate failure recovery specific to the problem domain.

The following issues are currently being addressed: (i) Part objects may be defined with different levels of deformity. In such a case, only certain part objects may deform during a fall to the ground and hence will cause a hard failure. While these failures will activate the rejection algorithm, non deformable part objects will not cause a hard failure and these will be accommodated as soft failures. (ii) Currently, the virtual environment is a single user environment. This is because of hardware limitations at the SODA laboratory. A multi-user virtual environment is necessary to address non-logical issues in the assembly process. However, since user actions are not the main objective of this paper, a single user virtual environment was sufficient to address the issues discussed in this paper.

### 6. References

1. Braham, R., and Comerford, R., "Sharing Virtual Worlds", *IEEE Spectrum*, pp.18-25, March 1997.

2. Greshon, N., Eick, S., "Visualization's New Track: Making Sense of Information", *IEEE Spectrum*, Nov 1995, pp. 38-56.
3. Marhefka, D., W., and Orin., D. E., "XAnimate: An Educational Tool for Robot Graphical Simulation", *IEEE Robotics & Automation Magazine*, pp.6-14, Vol.3, No.2, June 1996..
4. Huhns, M. N., Singh, M. P., "Agents on the Web: Agents are Everywhere!", [www address: http://www.computer.org/internet/9701/agents9701.htm](http://www.computer.org/internet/9701/agents9701.htm)
5. Andersson, C. Carlsson, O. Hagsand and Olov Stahl, "*The Distributed Interactive Virtual Environment Technical Reference Manual*", Swedish Institute of Computer Science, March 1994.
6. Fang, Y., and Liou, F.W., "Virtual Prototyping of Mechanical Assemblies with Deformable Components", *Journal of Manufacturing Systems*, pp.211-219, Vol.16, No.3, 1996.
7. Karr, R., Reece, D., and Franceschini, R., "Synthetic Soldiers", *IEEE Spectrum*, pp.39-45, March 1997.
8. Ormes, J., Ramaswamy, S., Cheng, T., "Modeling and Simulation of Manufacturing Systems in Interactive Virtual Environments", *6th IEEE International Conference on Emerging Technologies and Factory Automation, UCLA, Los Angeles, September 1997.*
9. Ramaswamy, S., and Valavanis, K. P., "Modeling, Analysis and Simulation of Failures in a Materials Handling System with Extended Petri Nets", *IEEE Transactions on Systems, Man, and Cybernetics*, pp.1358-1373, Vol.24, No.9, September 1994..
10. Robertson, G. G., et. al. "Information Visualization Using 3-D Interactive Animation", *Communications of the ACM*, Vol. 36, No. 4, April 1993, pp. 57-71.
11. Yun, M. H., Cannon, D., Freivalds, A., and Thomas,, G., "An Instrumented Glove for Grasp Specification in Virtual Reality Based Point-and-Direct Telerobotics", *IEEE Transactions on Systems, Man, and Cybernetics*, pp.835-846, Vol.27, No.5, Oct. 1997.