Stacking Irregular-Sized Packages by a Robot Manipulator

Harry H. Cheng
Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California
Davis, CA 95616
Email: hhcheng@ucdavis.edu
World Wide Web: http://iel.ucdavis.edu
Rajan Penkar
United Parcel Service
55 Glenlake Parkway, NE
Atlanta, GA 30328

Abstract
To study the feasibility of stacking irregular-sized packages inside a trailer, a prototype autoloader system has been developed. It is believed that this autoloader is the first system specifically designed to stack rectangular packages with various dimensions inside a trailer. Operation principles and some design considerations of this autoloader system will be described in the article. This case study demonstrates how robotic technologies can be possibly applied to a complicated package-handling automation system.

1 Introduction

The current manual process for stacking packages with different dimensions and weights inside a trailer is time-consuming and physically demanding. The ultimate solution to improve the productivity with great volumetric and time efficiencies is to handle packages by a fully automatic system. To study the feasibility of such an automation system, a prototype autoloader system shown in Figure 1 has been developed. In our target system, a self-propelled carriage will support one or two UPSarm inside a trailer to stack packages wall by wall from the front to the rear of the trailer. The robot manipulator will load and stack packages weighing up to 70 lbs. The maximum package dimension is 48 inches; the maximum package mid-dimension is 26 inches; and the minimum individual package dimensions are 0.5 inch high, 6 inches wide, and 6 inches long. Our prototype system consists of a UPSarm, a nest, and a pre-orient table. The prototype UPSarm can automatically load and stack packages in a flat space; the size of the flat space is the same as the size of the flatbed of a trailer. The package-stacking process of this prototype autoloader system can be briefly described as follows. The dimension and weight of each testing package in our lab are measured beforehand and encoded in a bar code label attached to the package. The package selected by our stacking algorithm is first manually placed onto the pre-orient table where it is rotated into a proper orientation before being placed onto the stack (In the field application, packages will be moved onto the pre-orient table from a convey belt automatically.). The package is then moved to the nest where it is further repositioned if necessary. The nest is located below the shoulder of the UPSarm (not shown in Figure 1). The UPSarm swings back about -20 degrees

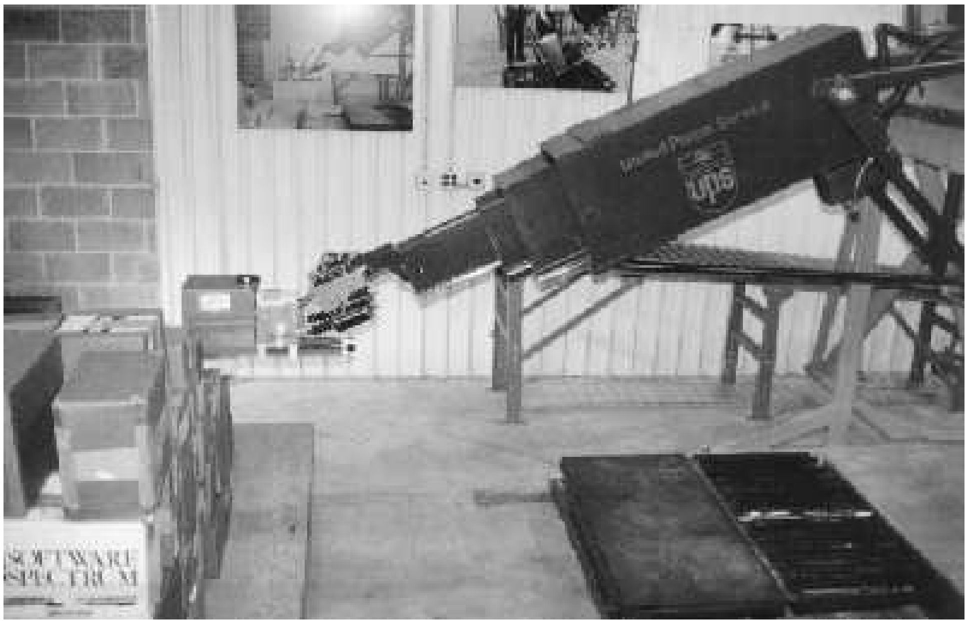

Figure 1: The prototype UPSarm.

from the vertical (corresponding to -20 degrees for the effective joint three of the arm) to pick up the package. Finally, the arm rotates and extends to place the package on the stack. The arm will return to the nest for the next package. The operation continues until all the testing packages are stacked. Handling of parcels with *non-uniformity* is what distinguishes our autoloader system from other standard palletizing processes. The major components, some design considerations, and experimental results of this autoloader system will be described in this article.

2 Mechanical Design

The prototype UPSarm loads and stacks packages weighing up to 70 lbs. Its load-carrying capacity of 70 lbs is higher than most robot manipulators used for conventional industrial applications such as welding and painting. The UPSarm will stack multiple walls of packages. Although the arm will operate inside a trailer with a constrained working space, a large workspace of the UPSarm is critical in order to stack multiple walls of packages simultaneously. Due to this application-specific nature of the robot manipulator, the design of the UPSarm is unique in its kinematic configuration. The geometrical configuration of the manipulator is shown in Figure 2. It is a ten degrees-of-freedom hybrid serial-and-parallel-driven redundant robot manipulator with a closed-loop kinematic chain [1-3].

The UPSarm can be considered as consisting of two subsystems: shoulder and wrist. These two subsystems are designed with closed and parallel kinematic chains primarily due to considerations of the rigidity and high payload-carrying capacity of the manipulator. Joints 1 to 7 are considered as shoulder subsystem of the UPSarm. The mechanical device used for the actuator joint 3 is essentially a slider-crank mechanism. The closed kinematic chain of the shoulder subsystem provides the UPSarm with a sufficient load-lifting capability. The three redundant prismatic joints 4, 5, and 6 of the UPSarm provide the arm with a long reachability, large workspace, and dextrous maneuverability, which are desired since the arm will operate inside a trailer with a constrained working space. The selection of the rectangular cross section for the forearm of the UPSarm is

Figure 2: The schematic configuration of the UPSarm.

based upon the stiffness-to-weight ratio. For the same weight, a rectangular cross section is stiffer than a square or circular cross section.

The wrist subsystem of the UPSarm shown in Figure 2 consists of three joints 8, 9, and 10. The wrist consists of a platform which is connected to the end-effector, two extensible links for actuators 9 and 10, one ball joint formed by one universal joint and actuator revolute joint 8, and a base connecting the wrist to the positioning structure of the robot. The platform is connected to two extensible links through two universal joints, and to the wrist base by another universal joint at point O_{8b}. The other ends of two extensible links are connected to the wrist base through two universal joints. By varying the link lengths (q_8, q_9) and the roll angle (q_8) of the ball joint, the platform can be manipulated with respect to the wrist base. In order for the wrist to operate as a three-degrees-of-freedom mechanism, each extensible link is designed with six degrees-of-freedom and the ball joint has three degrees-of-freedom. This wrist has three degrees-of-freedom with two in-parallel actuator drivers. This in-parallel actuation greatly enhances the rigidity and load-carrying capacity of the designed wrist, but it only provides a very small joint space for joint 8. The redundant joint 7 of the manipulator is thus introduced, which provides the UPSarm with more dexterity and workspace. Joint 7 is redundant and has essentially the same functionality as joint 8. The wrist subsystem of the UPSarm consists of two in-parallel actuators; it is quite different in design from the parallel mechanism, investigated by Hunt [4], Lee and Shah [5], and Waldron et al. [6], which is a fully parallel three degrees-of-freedom mechanism. An iterative numerical method is needed to solve the wrist kinematics of the UPSarm [3].

A major design criterion for the gripper is that it should not damage packages and their contents. Located at the end of the wrist of the UPSarm shown in Figure 1, the gripper is able to handle packages weighing up to 70 lbs by gripping above and below the package with one upper palm and two lower paddles. Two lower paddles extend to support the package below and retract after a package is released at the stacking position. The palm of the gripper can hold the package while the lower paddles are retracted. In order to hold the 0.5 inch minimum package size, a small gripping device has been built in. It handles small packages and retracts for large ones.

3 Motion Coordination

As described in the previous section, the UPSarm is a hybrid serial-and-parallel-driven redundant robot manipulator with a closed-loop linkage for joint 3. To facilitate the kinematic analysis of this hybrid serial-and-parallel-driven redundant robot manipulator, three coordinate spaces: the world Cartesian coordinate space, effective joint space, and actuator space are introduced. The manipulator end-effector can be represented in one of these three spaces. The 4x4 homogeneous transformation matrix $\mathbf{T}_w^t$ defines the tool center point (TCP) and orientation of the end-effector in the world Cartesian coordinate space by the 3x1 translational vector and 3x3 orientational submatrix, respectively. In our application, the world coordinate system $X_wY_wZ_wO_w$ shown in Figure 2 is fixed in a corner of a trailer. Relative location of moving and stationary objects such as the robot arm, grasped package, and stacked packages are reconciled with respect to this coordinate system.

The actuator space locates the end-effector by specifying actuator angles or translational distances in actuator axes which may be directly connected to motor shaft axes. Unlike an effective joint, a defined actuator joint always physically exists. The actuator value is a linear function of the encoder count of a motor encoder mounted on a motor shaft. For motion planning and real-time control, the parameters such as maximum speed, acceleration, deceleration, and torque limits of the manipulator can be specified in this actuator space. In this article, the actuator variables of the UPSarm are represented by $\mathbf{q} = [q_1, q_2, ..., q_{10}]^T$. Figure 2 shows the translational actuator variables of the UPSarm in its actuator space.

To facilitate the kinematic analysis, an effective joint space is introduced. An effective joint space is used to represent the manipulator in an effective serial kinematic chain. An effective joint may not be a physical joint and is a virtual joint such as the second effective joint of the UPSarm. The introduction of the effective joint space is mainly for simplifying design, analysis, and real-time control of hybrid serial-and-parallel-driven kinematic chains. In this article, the effective joint values of the UPSarm are represented by $\boldsymbol{\phi} = [\phi_1, \phi_2, ..., \phi_{10}]^T$. In the effective joint space, two parallel prismatic joints of the shoulder of the UPSarm is modeled as one translational and one revolute joints. Only the rotational part of joint 3 is included in the effective motion. The wrist shown in Figure 2 is represented as three effective revolute joints with three joint axes intersecting at a common point O_{8b}. The UPSarm forms a serial kinematic chain in its effective joint space.

The direct, forward, inverse, and indirect kinematic solutions described in [1] are used for manipulation of the UPSarm. These kinematic formulations are essential for real-time motion coordination of the hybrid serial-and-parallel-driven redundant UPSarm. In the context of this article, the trajectory planning refers to the process that calculates the required parameters, such as duration times of acceleration, slew motion, and deceleration of motion segments based on certain physical constraints. The trajectory generation is a process that computes the intermediate positions of the robot arm in a motion segment based upon the information obtained by the trajectory planner. The physical constraints of the robot manipulator can be specified in many different ways. For the practical application and implementation simplicity, we assume that each actuator has limited maximum velocity, acceleration, and deceleration, which are directly related to the available motor torque. These motion constraints are specified in the actuator space. Path planning will mainly calculate the necessary parameters based upon the information in the actuator space. Mechanical joint limits of each actuator are also physical constraints of the manipulator, which have to be checked in trajectory planning and trajectory generation phases of the robot manipulation. In a less rigorous terminology, the motion of a robot manipulator may be classified as point-to-point and continuous-path motions, which are implemented in our trajectory planner and trajectory generator for practical manipulation of the UPSarm. In a point-to-point motion, the target position can be specified in the aforementioned three coordinate spaces of the world Cartesian coordinate space, effective joint space, or actuator space. Our continuous-path includes straight line, circular curve, and spline curve. Besides, two adjacent point-to-point motion segments can be blended to form a continuous path.

A motion planning is a process that determines the next movement of a robot manipulator. It is in this stage that a point-to-point, continuous-path, or segment-blended continuous-path will be selected based upon a robot task specification. The robot motion planning for stacking packages is relatively simple. The initial motions of the UPSarm before reaching the stacked packages are point-to-point or segment-blended motions. The final stage of stacking a grasped package is a straight line motion. This straight line motion is critical because the package to be stacked may collide with those packages already stacked during this final movement. Therefore, for each selected package, an intersection detection must be performed to avoid a possible collision, which will be further discussed in the following sections.

The important step in the intersection detection of our autoloader system is to formulate the straight line motion of the UPSarm in the normalized distance parameter s-domain so that the spatial information can be separated from the time aspect under the s-formulation [2]. As a result, the intersection detection becomes a geometrical problem. The computations related to the time aspect such as acceleration, slew, and deceleration times can be avoided during the intersection detection phase. The computational complexity of the intersection detection algorithm is then independent of the speed, acceleration, deceleration, and motion profiles of a motion segment.

4 Stacking Algorithm

Each package will be processed by our stacking algorithm shown in Figure 3. The stacking algorithm is one of the key elements of our autoloader system. It informs the UPSarm where to place a package in a trailer. The current manual loading methods are the conceptual basis for the stacking algorithm. The algorithm mainly uses surface, corner, and volume information of the stacked packages and three dimensions (width, height, and depth) of the current package to decide its stacking position. A data structure which contains the coordinates of a corner and dimensions (width, height, and depth) of each rectangular package is used to keep track of the stacked boxes. The coordinates of these corners are expressed in the world coordinate system $X_wY_wZ_wO_w$ shown in Figure 2. Our stacking algorithm stacks two walls at a time. It starts with the first wall. When a package is not stackable in the first wall, it will be placed on the second wall, the first wall will be flushed and the algorithm continues to work on the second and third walls. For a given package, a search is

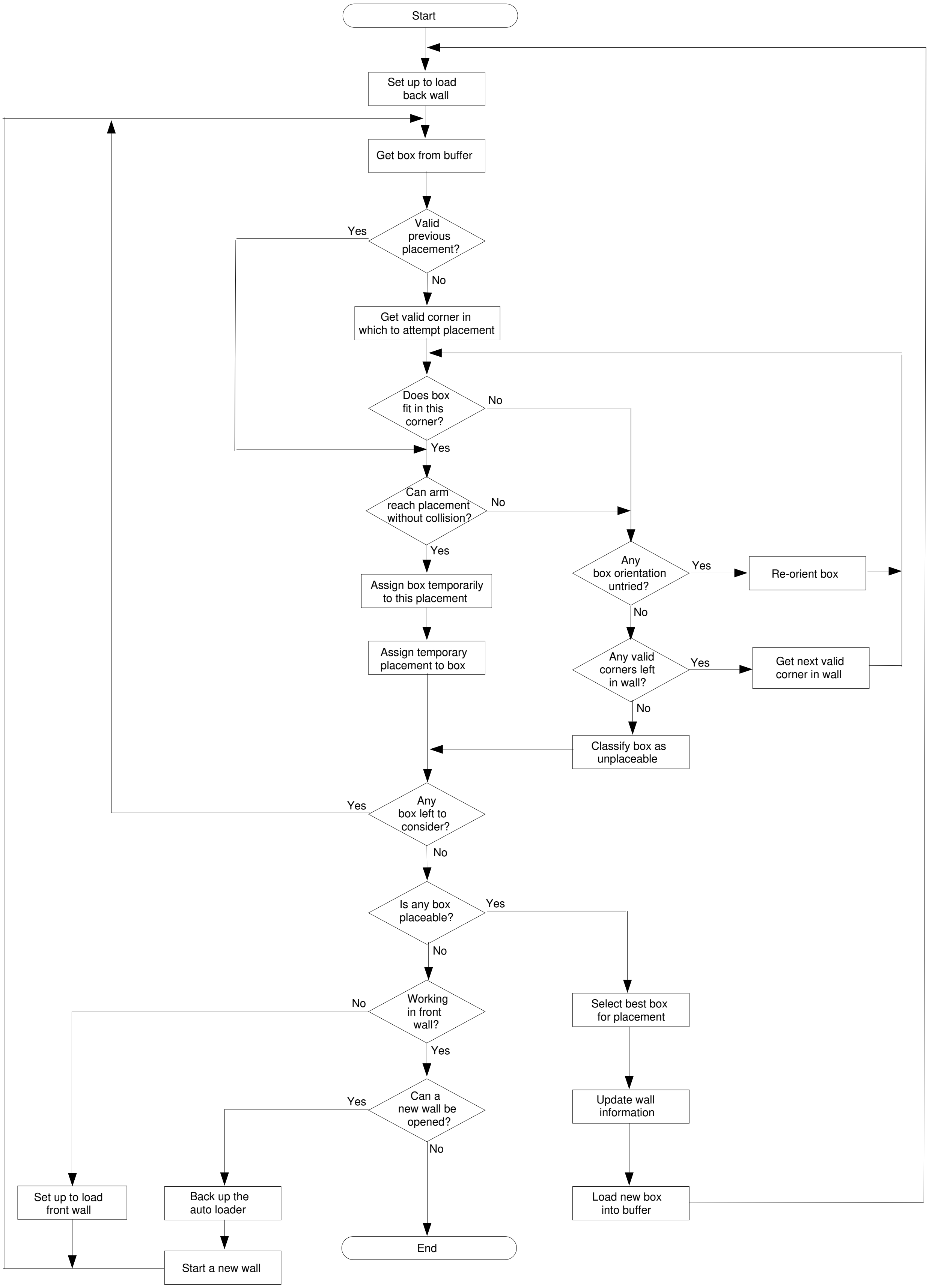

Figure 3: The stacking algorithm.

conducted until an available corner for placement is found. After such placements are found for all the packages in a buffer, they are merited against the following selection criteria: least blocked volume, lowest corner in the trailer, closest to the back of the trailer, closest to the side of the trailer, maximum corner surface, maximum box surface, least blocked surface, oldest box in the buffer. Going through the above check list, a package with a suitable stacking position can be singled out.

As mentioned before that the trajectory of the package gripped by the UPSarm during the final motion of the stacking process is a straight line with a fixed orientation in the world Cartesian space. If a package is loaded to the stacking position in horizontal direction during this final straight line motion, it is called a front-loading. If a package is loaded to the stacking position downward during the final motion, it is called a top-loading. Once the location of the stacking position of a package is selected by the stacking algorithm, the orientation of the end-effector of the UPSarm at the target stacking position can be decided according to its loading mode. Because only the rectangular packages are handled, using the volume information of the stacked packages, our stacking algorithm will guarantee that the package grasped by the gripper of the UPSarm will not collide with the stacked packages. But, the robot arm, especially bodies 5 and 6 with rectangular shapes shown in Figure 2, may collide with the stacked packages during the final stage of stacking.

Therefore, whenever a preliminary stacking position is decided by the stacking algorithm, the intersection detection between the stacked packages and the UPSarm along the final straight line must be performed to decide the final stacking position. When a stacking position for a package is calculated by the stacking algorithm, five linked lists corresponding to packages at its neighbors will be generated. The generation of these linked lists somewhat resembles the generation of an octree for representing the workspace of a robot cell. The packages at top, bottom, left, right, and front positions relative to the current stacking position are shown in Figure 4. The intersection detection is performed for each selected package, hence the efficiency of the intersection detection algorithm is crucial. Because we only check the intersection at sampled positions along the straight line before the package is actually being stacked, the separation of the time and spatial information for the straight line path of the UPSarm is strongly desired so that the unnecessary computations can be avoided. The inverse kinematic solution will be used to solve the effective joint values $\boldsymbol{\phi}$ along the final straight line path. The effective joint values $\boldsymbol{\phi}$ are then used to calculate the transformation matrix $\mathbf{T}_u^v$ for intersection detection, which will be discussed in the next section.

5 Collision Detection

To successfully load a package into its stacking position which is decided by our stacking algorithm, the robot arm during the stacking process must not collide with other stacked packages in a trailer. To this end, an efficient intersection detection algorithm is needed to detect possible collisions beforehand. It will check whether a planned path for the UPSarm is acceptable. In our application, the efficiency and reliability of the intersection detection algorithm are critical.

The collision detection of moving objects can be classified into three categories: multiple intersection detection, swept volume interference detection, and four-dimensional intersection detection [7]. The multiple intersection detection is intuitively simple. In this method, the positions and orientations of moving objects are sampled at a finite number of times and intersection detection between objects is performed at each time step [8, 9]. The interference refers to the state where two objects occupy the same area in the working space. In the swept volume interference detection, the volumes swept out by moving objects are generated first. The interference detection between these swept volumes is then performed [10-11]. The four-dimensional intersection detection will predict the exact collision time and location between the moving objects modeled as four-dimensional en-

Figure 4: Neighbors of a package at its stacking position.

tities (three spatial dimensions and time) [2, 11-14]. Collision detection using the configuration space has also been explored [15-16]. The configuration space approach is, however, not applicable to our application. In this approach, obstacles are first mapped into the configuration space before collision detection. In our application, obstacles dimensions (stacked packages) increase during the stacking process, this dynamic change of the manipulator workspace will be difficult to reflect in the configuration space. Besides, the UPSarm is a hybrid serial-and-parallel-driven redundant mobile robot manipulator with a closed-loop kinematic chain, it cannot be easily accommodated by the configuration space. The dimension change of the manipulator due to joints 4, 5 and 6 will further compound the problem. Fast algorithms have been developed for computations of the distance between moving objects [17-18]. Because of the application-specific nature, an efficient intersection detection algorithm thus has been developed for our prototype autoloader system.

5.1 Intersection of Two Rectangular Boxes

In our current robot application, the packages handled by the UPSarm are restricted to rectangular packages only. Links of the UPSarm that are prone to collision with stacked packages are in rectangular shapes as shown in Figure 2. Therefore, the rectangular solid modeling for objects is a natural choice for the collision detection in our application.

For the clarity of presentation, two rectangular solids under collision detection are identified as *object* and *obstacle* as shown in Figure 5. In Figure 5, there are three coordinate systems which are identified as the local object coordinate system $X_uY_uZ_uO_u$, local obstacle coordinate system $X_vY_vZ_vO_v$, and world coordinate system $X_wY_wZ_wO_w$. In general, the coordinates for vertices of the object and obstacle can be conveniently expressed in their corresponding local coordinate systems as is shown in Figure 5 for vertices U_1, U_7, V_1, and V_7. The information presented in the

Figure 5: Object, obstacle, and three coordinate systems.

local coordinate systems can be reconciled in the world coordinate system. Figure 5 also shows twelve edges $\mathbf{L}_k$ of the object and six surfaces of the obstacle, which are used for the convenience of presentation. In this article, the surface of S_i refers to a finite rectangular surface of the object or obstacle. The numbering system for surfaces of both object and obstacle shown in Figure 5 is the same.

It can be intuitively verified that there are only two possible intersection scenarios for the object and obstacle, which are referred to as type-A intersection and type-B intersection in Figure 6. If any edge of the rectangular solid object pierces any surface of the rectangular solid obstacle, it is called type-A intersection; and if any vertex of the obstacle lies inside the object, it is referred to as type-B intersection. Apparently, the roles of the object and obstacle in this definition can be switched without affecting the essence of the intersection detection algorithm.

5.2 Intersection Detection between the UPSarm and Stacked Packages

Since the computation time for intersection detection is proportional to the number of pairs of object and obstacle, we try to reduce the number of these pairs. When a stacking position for

Figure 6: Two types of object-obstacle intersection.

a package is calculated by the stacking algorithm, five linked lists corresponding to packages in the top, bottom, left, right, and front positions relative to the current stacking position shown in Figure 4 will be generated. At the same time, the straight line path of the final motion segment will be decided. The intersection detection between obstacles (the stacked packages stored in linked lists) and moving objects with variable dimensions (the UPSarm) along the final straight line path will be performed.

In this application, the world coordinate system $X_w Y_w Z_w O_w$ shown in Figure 2 is fixed in the corner of a trailer. In terms of the object and obstacle shown in Figure 5, we treat the stacked packages as obstacles. The obstacle coordinate system $X_v Y_v Z_v O_v$ in Figure 5 coincides with the world coordinate system $X_w Y_w Z_w O_w$ shown in Figure 2, which means that the transformation matrix $\mathbf{T}_w^v$ is an identity matrix. Only links from body 4 up to the end-effector of the UPSarm are possible to collide with the stacked packages. The variable bounding box which encloses these links are shown in Figure 7. If we treat the bounding box of the UPSarm as the object in the object-obstacle scenario addressed in the previous section, the body coordinate system $X_4 Y_4 Z_4 O_4$ is the local object coordinate system $X_u Y_u Z_u O_u$ in Figure 5. The transformation matrix $\mathbf{T}_u^v$ based on the D-H parameters of the UPSarm in its effective joint space becomes

$$\mathbf{T}_u^v = \mathbf{T}_u^w = \mathbf{T}_4^w = \mathbf{T}_b^w \mathbf{T}_1^b \mathbf{T}_2^1 \mathbf{T}_3^2 \mathbf{T}_4^3 \qquad (1)$$

The motion of the UPSarm is generated in the s-domain as described in [1-2]. For a point-to-point motion, the effective joint values $\boldsymbol{\phi}$ used in transformation matrices in the above equation can be obtained from the actuator values $\mathbf{q}$ by the direct kinematics. For a straight line motion, the effective joint values $\boldsymbol{\phi}$ in the above equation can be calculated by the inverse kinematic solution. Note that the z-coordinate $(d_8 + q_4 + q_5 + q_6)$ of vertex U_1 is a function of the time even expressed in the local coordinate system, which implies the *dimension change* of the moving bounding box during the intersection detection process.

The step size of 0.5 inch for intersection detections along the straight line is a good choice since it is the positional error tolerance of our system. The collision between the UPSarm and stacked packages most likely occurs when a small package is loaded to the position where there are many packages at its neighbors. The set of the small boxes 1 to 6 shown in Figure 7 inside the bounding box is a more accurate modeling of the concerned links of the UPSarm. The model uncertainty in the collision detection can be accounted for by increasing the sizes of the bounding box and

Figure 7: The variable bounding box and small boxes of the UPSarm.

small boxes. For the computational efficiency, the concerned links of the UPSarm is first treated as a variable bounding rectangular box. If it interferes with any stacked package, the small boxes inside the bounding box will be further tested for the final confirmation of a possible collision. If actuator value q_i for i equal to 4, 5, or 6 is zero, the variable small box i in Figure 7 will degenerate to a face. The intersection detection for this degenerated small box will be unnecessary. Since the algorithms used for collision detection of the small boxes and bounding box are the same, the small boxes inside the variable bounding box of the UPSarm are treated in the same manner. The packages at the neighbors of the current stacking package are maintained in linked lists so that the relative locations between the bounding box and stacked packages are known. Therefore, many impossible collision scenarios can be excluded from the intersection detection for the purpose of the computational efficiency, only the combinations shown in Table 1 need to be tested for the type-A intersection. For example, only edges $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3, \mathbf{L}_6$, and $\mathbf{L}_7$ of the bounding box may pierce the

Table 1: Location of packages and collision with possible edges of the bounding box of the UPSarm.

location	testing edges
left	$\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3, \mathbf{L}_6, \mathbf{L}_7$
right	$\mathbf{L}_1, \mathbf{L}_3, \mathbf{L}_4, \mathbf{L}_5, \mathbf{L}_8$
bottom	$\mathbf{L}_2, \mathbf{L}_3, \mathbf{L}_4, \mathbf{L}_7, \mathbf{L}_8$
top	$\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_4, \mathbf{L}_5, \mathbf{L}_6$
front	$\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3, \mathbf{L}_4, \mathbf{L}_5, \mathbf{L}_6, \mathbf{L}_7, \mathbf{L}_8$

Figure 8: Real-time distributed computing system.

stacked packages on the left-hand side of the current stacking position.

6 Experimentations

Before the system is integrated for the real-time operation to stack real boxes, the complete prototype autoloader system has been simulated graphically through SILMA Cimstation, a graphical robot simulation package [19, 20]. The motion coordination software based upon kinematics, trajectory planning, and trajectory generation are coded in the C programming language and first tested in a Unix operating system. The programs are then ported to pSOS^{+}, a real-time operating system environment used for real-time motion coordination of the UPSarm. The computer hardware for real-time motion control is shown in Figure 8, which is a VMEbus-based real-time distributed computing system. There are five VME boards in an industrial standard VME chassis. The Motorola MVME133 board is the target host running a MC68020 microprocessor with the clock speed 16.67 MHz and 1Mb dynamic RAM. This target board is connected to LAN. The C program is cross-compiled on a Sun SPARCstation and then down loaded to the target board via an ethernet connection. The trajectory planning task and trajectory generation task are two major tasks executed on the target board under the real-time operating system pSOS^{+}. It takes 7.98 ms to plan a point-to-point joint motion segment and 9.40 ms for a straight-line motion segment given positions of two end-points. It takes 0.6 ms and 4.72 ms to generate an intermediate position along the trajectory for joint and straight-line motions, respectively. The computation time for forward kinematics, inverse kinematics, and indirect kinematics each takes less than 3 ms. The direct kinematics takes about 6 ms, most of which spend on the Newton-Raphson numerical scheme for the wrist direct kinematics solution.

The system real-time clock with a preset time unit T_c of 1 ms enables the clock interrupt service routine to keep track of the system cycle time. A PID-based feedforward servo control algorithm is used in our control algorithm. The model-based feedforward torque is computed on the target board and sent to servo control boards through a VMEbus at every 32 ms, the trajectory generation rate T_t. Four Galil DMC530 VME servo control boards are responsible for servo control and low level communication of the control system. The motion command generated from the target board is also sent to these servo boards via the VMEbus at the trajectory generation rate. Each servo board controls three brushless DC motors. The PID closed-loop control algorithm executed on a MC68008 microprocessor controls motion of each brushless DC motor. The sampling time interval T_s of the servo control is 1 ms. Each servo board can also provide eight PLC-type (Programmable Logic Control) software controllable I/O ports, which have been used to control the emergency stop, motor brakers, and gripper.

The stacking algorithm is coded in C and verified graphically by a graphic software package PHIGS. The reliability of the intersection detection algorithm for intersection detection between the UPSarm and stacked packages in our autoloader system has also been verified graphically using a more sophisticated and computationally expensive collision detection method provided by SILMA Cimstation. In our prototype autoloader system, the stacking algorithm and collision detection module run in a Sun workstation with a graphic window which displays packages stacked by the UPSarm. The stacking position of the selected package by the stacking algorithm is sent to the robot arm as a target position of the end-effector through a RS-232 serial communication port of the target board. On a Sun SPARCstation 2 with 28.5 Mips, 4.2 Mflops, and clock rate 40 MHz, it takes at most 0.223 ms for detecting intersection between the bounding box in Figure 7 and a package at the front of the stack shown in Table 1. The intersection detection time 0.223 ms includes the CPU time for computation of the transformation matrix $\mathbf{T}_u^v$ in Eq(1). It takes at most 0.166 ms to detect intersection between the bounding box in Figure 7 and a package at other locations.

The performance of the stacking algorithm depends upon the buffer size, number of package orientations, loading mode, etc. The volume efficiency is defined as the ratio of the space stacked with packages over the total space of the trailer. With a buffer of 12 boxes, 6 orientations for a box, and two loading modes, our stacking algorithm can achieve about 70% volume efficiency. This average volume efficiency of the stacking algorithm is close to that of manual loading. Extensive studies of the stacking algorithm indicate that the top loading mode can be eliminated without a significant effect on the volume efficiency of the algorithm whereas the mechanical design of the manipulator can be simplified. Although the number of orientations is important to the volume efficiency, the maximum number of 6 orientations for a given box can be reduced in the algorithm. One of the most important factors in achieving a high volume efficiency is the number of boxes in the buffer. The buffer size should be sufficiently large. Otherwise, not only the volume efficiency, but also other performance of the stacking algorithm can be suffered. For example, if a large number of small boxes are followed by a large number of big boxes in an automatic package handling system, big boxes will be stacked on top of small boxes, which may result in instability. According to our stacking algorithm, large boxes tend to be placed at the bottom of the trailer. Because of the collision detection, the stacking algorithm tends to place small boxes in the front of a stacking wall.

7 Discussions

The primary purpose of the developed prototype autoloader system is to study the feasibility of stacking irregular-sized packages inside a trailer by a robot manipulator. The prototype system will provide guidance regarding the relative importance of various design features and components. For practical field applications of a robotic system in stacking irregular-sized packages, some of critical issues that remain to be solved are as follows:

(1) Although we have stacked packages with different dimensions in our prototype autoloader system, all testing packages are light-weight. The stacking algorithm needs to take the weight of a package into consideration so that heavy and large packages will be placed at the bottom.

(2) A vision sensing system such as stereo cameras and structure light need to be integrated into the system. When packages are heavy, the accumulated error from the distortion of packages, especially packages at the bottom, is significant. The stacking algorithm needs to maintain the actual positions of stacked packages from a vision sensing system.

(3) A fault-tolerant error recovery system is to be built into the automatic package handling system. It is possible that if any one of packages according to the stacking algorithm is misplaced, the stacking algorithm may not be able to recover from this error. The stacking algorithm may place a package in a corner without a supporting surface. Therefore, a force monitoring system may be needed for stacking irregular-sized packages. The package delivery is operated under a tight schedule. The loading facilities of package distribution hubs may need to be modified so that once a fatal error occurs in an autoloader system, package loading can be quickly resumed by manual loading.

(4) The speed of package loading by the autoloader system needs to be improved. The mechanical constraint is the primary factor in the limit of the speed of the system. The weight of the forearm of the UPSarm weighs more than 500 lb. Moving a mechanical manipulator with over 500 lb plus a 70 lb package at the end-effector with a reachability more than 3 meters at a high speed is a nontrivial task. The time-optimal control is desired in this case.

(5) A great volume efficiency is critical for the autoloader system. The tighter packages are loaded in a trailer, the less the number of trailers is needed. A great volume efficiency will result in great savings in dollars. The stacking algorithm therefore needs to be further improved. Experience indicates that the volume efficiency over 80% is achievable in the stacking algorithm. If the variations of dimensions of packages are limited, the performance of the stacking algorithm can be improved significantly.

(6) The autoloader system needs to be integrated with other delivery systems for information sharing. The dimension and weight as well as other delivery information such as destination address and priority of a package are invaluable for planning and scheduling of prompt delivery of the package.

(7) The ultimate decision of using the autoloader system for stacking irregular-sized packages in practical field operations depends on the cost and reliability of the system. Manufacturing and maintenance of sophisticated robotic automation systems by skilled engineers and technicians are expensive. Only reliable and cost-effective automation systems make practical field applications of robotic technologies possible.

8 Acknowledgements

We would like to thank A. Brandorff, C. C. Cai, L. Figarella, R. Lancraft, J. J. Lee, J. Lecko, S. Smith, N. Tan, and Z. Tumeh for numerous stimulating discussions and significant contributions to the development of the prototype autoloader system; and K. Nielsen, L. Newport, and J. Snyder for their guidance and support for the work described in this article.

9 References

1. Cheng, H. H., Real-Time Manipulation of a Hybrid Serial-and-Parallel-Driven Redundant Industrial Manipulator, *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol. 116, December, 1994, pp. 687-701.

2. Cheng, H. H., Real-Time Four-Dimensional Collision Detection for an Industrial Robot Manipulator, *Proc. of the 3rd National Conference on Applied Mechanisms and Robotics*, Cincinnati, Ohio, November 8-10, 1993, vol. 1, pp. AMR-93-016-1 to AMR-93-016-13; also *Journal of Applied Mechanisms and Robotics* (in press).

3. Cheng, H., Lee, J. J., and Penkar, R., Kinematic Analysis of a Hybrid Serial-and-Parallel-Driven Redundant Industrial Manipulator, *International Journal of Robotics & Automation* (in press).

4. K. H. Hunt, Structural Kinematics of In-parallel-actuated Robot Arm., *Trans. ASME, J. Mechanisms, Transmissions, and Automation in Design*, 105: 1983, pp. 705-712.

5. K. M. Lee and D. K. Shah, Kinematic Analysis of a Three Degrees of Freedom In-parallel Actuated Manipulator, *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 1, 1987, pp. 345-350.

6. K. J. Waldron, M. Raghavan, and B. Roth, Kinematics of a Hybrid Series-Parallel Manipulation System, *ASME Journal of Dynamic Systems, Measurement, and Control* Vol. 111, June, 1989, pp. 211-221.

7. Cameron, S., A Study of the Clash Detection Problem in Robotics, *Proc. IEEE Int. Conf. Robotics and Automation*, St. Louis, MO, March 1985, pp. 488-493.

8. Roach, J. W. and Boaz, M. N., Coordinating the Motions of Robot Arms in a Common Workspace, *IEEE Trans. on Robotics and Automation*, Vol. 3, No. 5, Oct. 1987, pp. 437-444.

9. Shaffer, C. A. and Herb, G. M., A Real-Time Robot Arm Collision Avoidance System, *IEEE Trans. on Robotics and Automation*, Vol. 8, No. 2, April 1992, pp. 149-160.

10. Boyse, J. W., Interference Detection among Solids and Surfaces, *Commun. ACM*, Vol. 22, No. 1, 1979, pp. 3-9.

11. Cameron, S., Collision Detection by Four-Dimensional Intersection Testing, *IEEE Trans. on Robotics and Automation*, Vol. 6, No. 3, June, 1990, pp. 291-302.

12. Esterling, D. M. and Rosendale, J. V., An Intersection Algorithm for Moving Parts, *NASA Symposium on Computer-Aided Geometric Modeling*, April 20-22, 1983, pp. 119-123.

13. Canny, J., Collision Detection for Moving Polyhedra, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 2, March 1986, pp. 200-209.

14. Kawabe S., Okano, A., and Shimada, K., Collision Detection among Moving Objects in Simulation, *Robotics Research, The Fourth International Symposium*, (edited by R. C. Bolles and B. Roth), 1988, pp. 489-496.

15. Lozano-Perez, T., Spatial Planning: A Configuration Space Approach, *IEEE Trans. Computers*, vol. C-32, no. 2, 1983, pp. 108-120.

16. Donald, B. R., A Search Algorithm for Motion Planning with Six Degrees of Freedom, *Artificial Intell. J.*, vol. 31, Mar. 1987, pp. 295-353.

17. Gilbert, E. G. and Hong, S. M., A New Algorithm for Detecting the Collision of Moving Objects, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1989, pp. 8-14.

18. Lin, M. C., and Canny, J. F., A Fast Algorithm for Incremental Distance Calculation, *Proc. IEEE Int. Conf. Robotics and Automation*, Cincinnati, Oh, 1991, pp. 1008-1014.

19. Craig, J. J., Issues in the Design of Off-Line Programming Systems, *Robotics Research, The Fourth International Symposium*, (edited by R. C. Bolles and B. Roth), 1988, pp. 379-389.

20. SILMA, Inc., *Cimstation User's Manual*, California, 1988.