On-Line Manipulation Planning for Two Robot Arms in a Dynamic Environment

Tsai-Yen Li and Jean-Claude Latombe
Robotics Laboratory, Department of Computer Science, Stanford University
Stanford, CA 94305, USA

Abstract

In a dynamic and partially unpredictable environment, robot motion planning must be on-line. The planner receives a continuous flow of information about occurring events and generates new plans, while previously planned motions are being executed. This paper describes an on-line planner for two cooperating arms whose task is to grab parts on a conveyor belt and transfer them to their respective goals, while avoiding collision with obstacles. Parts arrive on the belt in random order, at any time. This scenario is typical of manufacturing cells serving machine-tools, assembling products, or packaging objects. The proposed approach breaks the overall planning problem into subproblems, each involving a low-dimensional configuration or configuration×time space, and orchestrates very fast primitives solving these subproblems. The resulting planner has been implemented and extensively tested both in a simulated environment and with a real dual-arm system. Its competitiveness has been evaluated against an oracle making (almost) the best decision at any one time. The planner compares extremely well.

Acknowledgments: This research was funded by ARPA grant N00014-92-J-1809. The authors thank Y. Koga and G. Pardo-Castellote for their constructive comments. They also acknowledge the use of the dual-arm system made available to them by Prof. R.H. Cannon in the Aerospace Robotics Laboratory.

1 Introduction

Off-line robot motion planning is a one-shot computation prior to executing any motion. All pertinent data must be available in advance. In contrast, on-line planning is an ongoing activity that receives a continuous flow of information about the robot environment. While planned motions are being executed, new plans are generated in response to incoming events.

Off-line planning is virtually useless in dynamic environments that involve events whose occurrences in time and space are not precisely known ahead of time. On the other hand, while on-line planning can potentially deal with such environments, it raises difficult temporal issues which have not been thoroughly addressed by previous research. Indeed, timing is highly critical since motions must be both planned and executed while their goals are still relevant. Opportunities to achieve a goal may exist only during short periods of time. If the on-line planner is too slow or does not focus on the right subproblem at the right time, it will fail to achieve goals that could have been attained otherwise. The efficiency of the planner measures against an instantaneous oracle making the best decision at every time. The greater the efficiency, the better the planner; but failing to achieve some goals is acceptable.

Here we investigate on-line motion planning for a specific, but practically interesting part-feeding scenario where two robot arms must grab parts as they arrive on a conveyor belt and transfer them to given goals without collision. This scenario is typical of robotic cells loading machines, assembling products, or packaging objects. Our planner breaks the overall planning problem into a series of subproblems and orchestrates very fast primitives solving these subproblems according to the incoming flow of information. Experiments with this planner in a simulated environment to evaluate its efficiency against quasi-optimal oracles have shown that it is quite competitive. We have also connected the planner to a real robot system and successfully run experiments with this integrated system.

2 Relation to Previous Work

Motion planning has attracted a great deal of interest over the last 15 years. Most of the research, however, has focused on off-line planning in static environments. A motion plan is then computed as a geometric *path*. A major concept produced by this research is the notion of the *configuration space*, or *C-space*, of a robot [15]. Various path planning algorithms based on this concept have been proposed [12]. A number of very fast planners have been implemented for robots with few degrees of freedom (usually, 3) [13, 2]. Reasonably efficient planners have also been developed for robots with many degrees of freedom (6 or more) [5, 2, 9]. But they still take too much time to be used on-line. Such path planners can be used to facilitate off-line robot programming. For example, the path planner in [5] computes collision-free paths of an 8-dof manipulator among pipes in a nuclear plant. In [7], a planner generates paths of a 5-dof rivet-

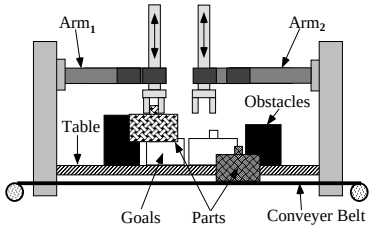

Figure 1: Two-arm robotic cell

ing machine to assemble portions of an airplane fuselage. In [3], planning is used to generate paths for the maintenance of aircraft engines.

Motion planning in the presence of obstacles moving along known trajectories is a step toward dealing with a dynamic environment [17, 6]. The C-space is extended by adding a dimension, time, yielding the *configuration×time space*, or *CT-space*, of the robot. Motion plans are generated in the form of robot's *trajectories*, i.e., geometric paths indexed by time.

Motion planning for several robots has been addressed in [8, 4, 2]. The *centralized approach* consists of treating the various robots as if they were one single robot, by considering the Cartesian product of their individual C-spaces [2]. This space is called the *composite C-space*. A drawback of this approach is that it often leads to exploring a large-dimensional space, which may be too time-consuming. An alternative is the *decoupled approach*, which consists of planning for one robot at a time. One technique plans the path of each robot separately and then tunes the robots' velocities along their respective paths so that no two robots ever collide [8]. However, the decoupled approach is not complete, i.e., may fail to find a motion of the robots even if one exists.

Manipulation planning extends motion planning by allowing robots to move objects. It consists of interweaving transit paths, where a robot moves alone, and transfer paths, where it moves objects, separated by grasp/ungrasp operations. These paths lie in different subspaces of the composite C-space defined as the Cartesian product of the C-spaces of all robots and movable objects. Manipulation planning has been studied for a single robot in [19, 1] and for multiple robots in [10, 11]. The regrasping issue with one robot has been specifically investigated in [18].

3 Scenario

Our scenario involves two robot arms, a conveyor belt, a working table, movable parts, and obstacles. The arms must grab parts as they arrive on the belt and transfer them to specified goals on the table where, for example, they will form an assembled product.

The robot system (Fig. 1-3) includes two identical SCARA-type arms, each having three links and four degrees of freedom. The first two links of each arm form a horizontal linkage with two revolute joints. The third link, which carries the gripper, translates up and down and rotates about its vertical axis. Each arm shares part of its workspace with the other arm. The arm that is closest to the beginning of the belt is called ARM1. The other arm is called ARM2.

Parts of different types arrive on the belt at any time, in random order, and with arbitrary orientations. A vision system detects and tracks them on the belt. The task of the arms is to grab as many parts as possible and transfer them to their goals, without collision. For each part X, the position and orientation where a gripper can grasp X is unique and given relative to X. The goal of X is also unique. When an arm releases a part at its goal, the part stays there until it is removed by an external mechanism. We assume that this mechanism never interferes with the arms.

Static obstacles are lying on the table. If an arm releases a part on the table, this part also becomes an obstacle. All obstacles lie below the horizontal volume swept out by the first two links of each arm. Similarly, an arm's gripper in its upmost position cannot collide with any obstacle. Hence, if an arm is not holding a part and its gripper is all the way up, it can only collide with the other arm. Such an arrangement is classical for SCARA-type arms, since otherwise motions would be too constrained to perform any useful task. However, when an arm holds a part, this part shares the same space as the obstacles. The belt is low enough so that when an arm holds a part with its gripper in its upmost position above the belt, the part is not hit by other arriving parts. This condition allows an arm to lift a part above the belt and stay there for a while, e.g., waiting for the next motion command.

A single-processor computer is dedicated to planning. The planner must make the best possible use of this resource to decide which arm motions to execute to transfer as many parts as possible to their goals.

Example: The above scenario is illustrated in Fig. 2 and 3, with a series of snapshots produced by our planner. Snapshots are indexed by time, with the run starting at time 0 and the sampling rate being 0.25sec per frame. The belt is on the left-hand side and moves downward. Each snapshot displays two configurations of the arms and moving parts; the one in darker grey is the current configuration, while the one in lighter grey is a configuration between the previous and the current snapshot. Parts of two types are fed during the run. We denote them by X_i and Y_j, where X and Y refer to the pentagonal and T-shaped parts, respectively, and i and j indicate the order of arrival. Each part disappears as soon as it is delivered to its goal.

In snapshots (2)-(4), ARM1 (the top arm) and ARM2 (the

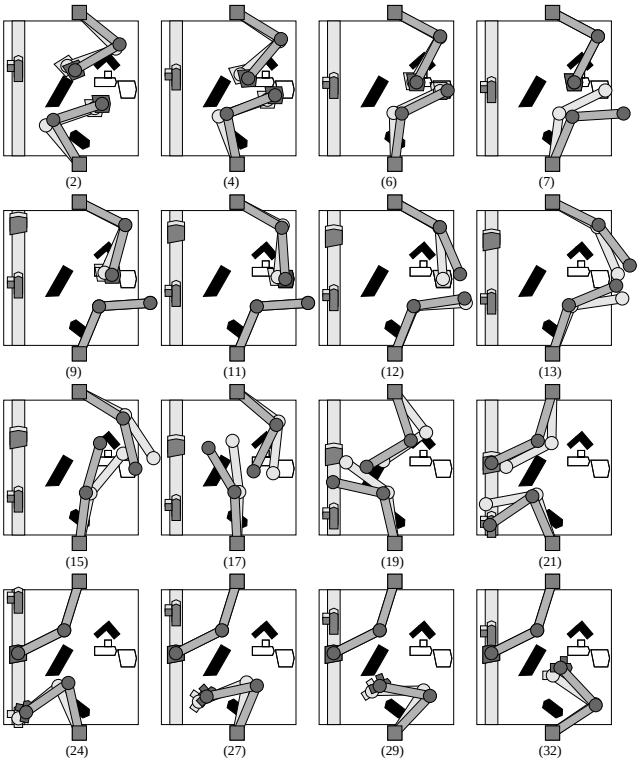

Figure 2: Example (part 1)

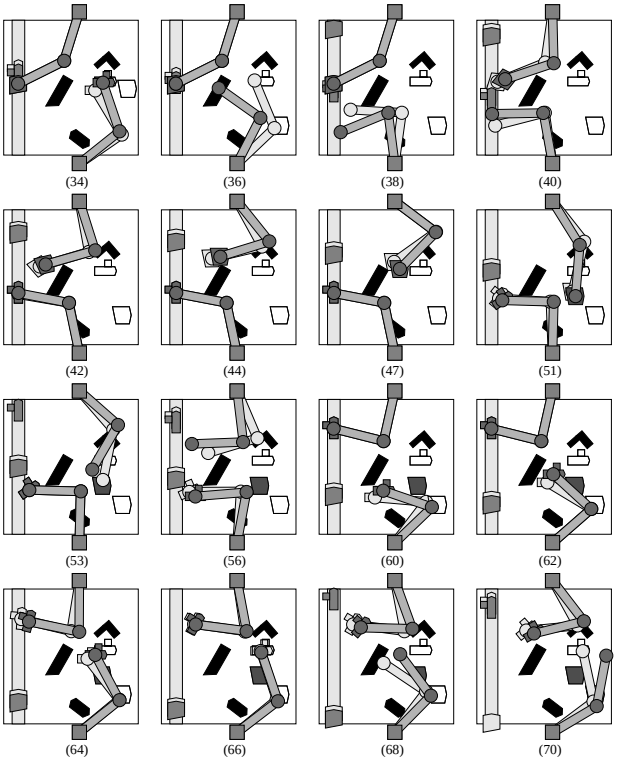

Figure 3: Example (part 2)

bottom arm) are simultaneously delivering X_2 and X_1 to their goals. In (6), X_1 reaches its goal and disappears. In (7)-(11) ARM1 performs the deliver motion of X_2, while ARM2 clears the way for this motion. There are two new parts, Y_1 and X_3, arriving on the belt. In (12), immediately after ARM1 has delivered X_1 to its goal, ARM2 starts executing a motion to grasp Y_1. Simultaneously, ARM1 performs a short motion to free the way for ARM2, as shown in snapshots (12)-(13). Snapshots (12)-(21) display the grasp motion of ARM2. Concurrently, in (15)-(21), ARM1 performs a motion to catch X_3. In (21), the arms grab X_3 and Y_1.

In (24)-(34) ARM2 delivers Y_1 to its goal, while ARM1 is staying still holding X_3 above the belt. In (36)-(38) ARM2 clears the way for ARM1, which starts executing the deliver motion of X_3. Note that the goal of X_i has changed between (34)-(36). This change is taken into account in ARM1's deliver motion, as shown in snapshots (40)-(51). In (40)-(42) ARM2 executes a grasp motion to catch Y_2 and starts moving Y_2 toward its goal in (51). In (53), since X_3's goal is not reachable by ARM1, ARM1 releases X_3 at an intermediate location reachable by ARM2. X_3 then becomes an additional obstacle which is taken into account by the deliver motion of ARM2 shown in (56)-(66). In (53)-(60) ARM1 performs a grasp motion to catch Y_3 and in (64)-(66) it starts moving Y_3, while ARM2 is delivering Y_2 to its goal. In (68)-(70), ARM2 clears the way for ARM1, which delivers Y_3 to its goal in (70) and beyond.

4 Overview

In the following, we denote the arms by A_1 and A_2, with A_1 standing for either ARM1 or ARM2, and A_2 standing for the other arm.

Planning primitives: In principle, if the arrival times of the parts were known in advance, the planning problem of our scenario could be solved off-line by searching through the high-dimensional composite C-space of the two arms and the parts. But this search would take prohibitive time. So, the problem needs to be simplified. Planning on-line makes simplifications even more necessary. One way to proceed is to break the composite C-space into several low-dimensional spaces.

How much simplification is suitable? We use the following rule-of-thumb: Planning a motion should take significantly less time than executing this motion. If planning is longer, the performance of the robot system degrades quickly. But if it only takes a small fraction of the time needed for execution, making it even faster has little effect on the system efficiency. This rule leads us to reduce the planning problem to a series of subproblems in spaces of dimension three. Indeed, there exist techniques that plan motions in such spaces much under the second on current workstations, while planning

in spaces of dimension four or higher takes one or several orders of magnitude longer [2].

Our planner mainly searches through two types of 3D spaces: the CT-space of an arm and the C-space of a part. We briefly discuss below the assumptions and heuristics which allow us to decompose the problem and limit planning to these spaces.

We allow the first two links of an arm to move only when its gripper is all the way up. Hence, an arm can only collide with the other arm. This allows us to represent the two arms in a 2D workspace as shown in Fig. 2-3. To reduce the problem further, we decouple arm planning so that we always plan for a single arm, say A_1, at a time while the other arm, A_2, can be idling or executing a previously planned trajectory. This simplified problem can be formulated as computing a trajectory in the 3D CT-space of A_1.

To shorten our presentation, we assume that translating the gripper to (un)grasp a part is instantaneous, and that orienting the gripper can always be coordinated with the motions of the other two links. We also consider that parts disappear immediately after they have been delivered to their goals. These assumptions can easily be removed and are not made in the implemented planner.

We represent a part X by its 2D projection into the horizontal plane. When X is being transferred by an arm, the part both translates and rotates in the plane, hence tracing a path in a 3D C-space. Obstacles map into this C-space as forbidden regions. Our planner computes X's path between its grasp and goal configurations in the subset of its C-space that is reachable by at least one arm. Through the arm's inverse kinematics, this path then entails the path of the arm holding X. If X leaves the space reachable by this arm, the planner will command the arm to ungrasp X at an intermediate location on the table where it can be regrasped by the other arm. If another part Y is currently being moved, this motion is temporarily ignored. When the paths of X and the arm holding it have been computed, they are coordinated with the ongoing motion of Y by tuning X's velocity appropriately.

In addition to the simplifications made above, our planner assumes that the arms can perfectly track the planned trajectories, with each joint being able to instantaneously change velocity. Unlike previous simplifications, this assumption yields discrepancies between the planning model and the real world. In Section 7 we will discuss how we overcome these discrepancies in order to run the planner with real robots.

Planning processes: A crucial issue in on-line planning is to react to events by focusing quickly on urgent subproblems and sizing opportunities to grab parts before they vanish. *Planning processes* are used by our planner to manage its activities over time.

At any time, all the parts currently on the belt or at intermediate locations on the table are listed in a queue Q. Assume that the planner starts with no parts in Q, and no arms moving. The arrival of a part on the belt triggers planning which consists of selecting a part X in Q (here, there is no choice, but usually there is one) and an arm A_1, and setting up a planning process whose task is (X, A_1), i.e., plan a motion of A_1 to grasp and deliver X to its goal. This process will be terminated upon the completion or failure of its task. Although its main task is to plan for A_1, this process may also generate a motion of A_2. E.g., if A_2 is currently immobile, the process may command A_2 to free the way for A_1. A new process is created whenever a process is killed or interrupts itself to allow for the execution of an already planned motion, and there exist a part $X \in Q$ and a non-moving arm A_i such that neither are currently assigned to a planning process.

The robot operations may be accomplished with different orderings of the parts and different assignments of arms to parts. Computing plans for all possible orderings/assignments and choosing the one that can grasp the largest number of parts in the shortest time would take prohibitive time to run. Instead, we assign tasks to processes according to the following heuristic rules:

1. The parts at intermediate locations on the table have higher priorities than those on the conveyer belt, since they may obstruct possible paths for these new parts.

2. The parts that are more advanced on the belt have higher priorities than those which are less advanced, since they will leave the arms' workspace earlier.

3. When both arms are not moving, ARM1 has higher priority than ARM2, since ARM2 is at the ending side of the belt and thus can be used as a backup for ARM1.

Concurrent planning: While a process is solving for a task (X, A_1), new parts may arrive on the belt and arm A_2 may be idling. The planner deals with this situation by breaking the task into two subtasks: the *grasp subtask* – plan a trajectory to grasp X – and the *deliver subtask* – plan a trajectory to deliver X to its goal.

A planning process interrupts and puts itself on hold between the two subtasks, allowing for the creation of a new process. More precisely, suppose that a process P_1 is created to plan for the task (X, A_1). P_1 first plans the motion for the grasp subtask and then puts itself on hold if a feasible motion has been found. While A_1 is executing this motion, the processor is free and can be used by other processes, say P_2, to plan for other tasks. Once A_1 has grasped X, P_1 resumes and solves for the deliver subtask. However, if P_2 is currently running, P_1 is put in the waiting state until P_2 is either interrupted or killed. While waiting for a deliver path, A_1 stays still with its gripper holding X all the way up above the belt. Processes thus take turns in using the processor.

Hence, a planning process P may traverse the following states during its lifetime:
- *Running*: P is running if it uses the computing resource to compute a plan.
- *On hold*: P is on hold while the arm assigned to it performs the grasp motion.
- *Waiting*: P is waiting if it needs to compute a delivery motion, but the computer resource is being used by another process.

If a process whose task is (X, A_1) fails to solve either the grasp or deliver subtask, it is immediately killed. However, in the second case, A_1 is already holding X. Then A_1 releases X on the belt as soon as there is enough distance between two incoming parts. There might still be a chance that the other arm can accomplish the task.

A planning process P_1 may be unable to plan the motion of an arm A_1 to deliver a part X because the goal lies outside the space reachable by A_1 or obstacles force X to leave A_1's reachable space. Then P_1 produces a motion of A_1 that moves X to an intermediate position where it can be regrasped by the other arm A_2. P_1 is killed and it will require another process P_2 (with arm A_2) to move X to its goal.

5 Planning Techniques

We now present a more detailed account of the activities carried out by the planner. The planner distinguishes between a variety of cases and applies a specific treatment to each one. Here, we consider a few representative cases; see [14] for an exhaustive description.

Representation of C-spaces and CT-spaces: Each arm A_i is modeled as a planar two-revolute-joint linkage, hence has a 2D C-space C_i. We parameterize a configuration q_i of A_i by the arm's joint angles θ_{i1} and θ_{i2}. We use the following metric over C_i: Let $\omega_{i,1}$ and $\omega_{i,2}$ be the maximal velocities of the first and second joints of arm A_i. Let $q_i = (\theta_{i1}, \theta_{i2})$ and $q'_i = (\theta'_{i1}, \theta'_{i2})$ be two configurations of A_i. The distance $D(q_i, q'_i)$ between these two configurations is:

$$D(q_i, q'_i) = \max\{\frac{|\theta_{i1} - \theta'_{i1}|}{\omega_{i,1}}, \frac{|\theta_{i2} - \theta'_{i2}|}{\omega_{i,2}}\}.$$

This definition is consistent with our assumption that arm joints achieve their planned velocities instantaneously: $D(q_i, q'_i)$ then measures the minimal time that A_i takes to travel between q_i and q'_i.

The CT-space CT_i of A_i is defined as $C_i \times [0, +\infty)$, with the third dimension being time. At every point $(\theta_{i1}, \theta_{i2}, t)$ in CT_i, the maximum velocities $\omega_{i,1}$ and $\omega_{i,2}$ define a cone of points reachable from $(\theta_{i1}, \theta_{i2}, t)$.

All C- and CT-spaces searched by our planner are represented as bitmaps. Cells containing "1"s designate the forbidden region where collision occurs. Cells containing "0"s form the free region in which paths and trajectories must lie. The cells in a CT-space bitmap projecting onto the same time interval form a *time slice*.

Planning a grasp motion: Let us consider the grasp subtask of the task (X, A_1). X may either be a part arriving on the belt, or a part previously ungrasped at an intermediate location on the table. Here we only consider the first case. Let t_c stand for the current time and q_1^s designate the configuration of A_1 when it starts executing the grasp motion. For any given time t, $q_1^g(t)$ denotes the configuration of A_1 at which it can grasp X. The map q_1^g is defined over the time interval $[t_g^{min}, t_g^{max}]$ during which X is on the belt within A_1's reach. It may yield two configurations, since the inverse kinematic equations of A_1 usually have two distinct solutions corresponding to two arm postures. The planner selects the configuration which is closest to q_1^s according to D.

The treatment applied by the planner depends on whether the other arm is currently moving, or not. Here we only consider the case where it is moving. This motion, which constrains the future motion of A_1, is mapped to a forbidden region in CT_1. Let us assume that A_2's motion is scheduled to end after time t_g^{max}; therefore, if A_1 can grasp X, its motion will terminate before the one of A_2. (The case where A_2's motion may terminate before A_1's is slightly more complicated, but is treated in a similar way.) The planner iteratively determines the time t_g when A_1 grasps X. For every selected t_g such that A_1 at $q_1^g(t_g)$ does not obstruct A_2's trajectory at any time $t \geq t_g$, it searches CT_1 for a trajectory joining the line $\{(q_1^s, t) | t > t_c\}$ to $(q_1^g(t_g), t_g)$, avoiding the forbidden region and satisfying the joint velocity constraints.

The planner performs this search backward, starting from the selected $(q_1^g(t_g), t_g)$. At every iteration of the search, it selects a pending node (q_1, t) of the current search tree such that $D(q_1^s, q_1)$ is minimum over all pending nodes. It computes nine potential successors of this node by successively setting the velocity of each joint to zero, its maximal value with positive sign, and its maximal value with negative sign, and integrating the corresponding motion over the duration of a time slice in CT_1's bitmap. If a potential successor q belongs to a "0" cell c of this bitmap and c has not been visited before, then q is included in the search tree as a new pending node and c is marked 'visited'. This best-first algorithm and the definition of D guarantee that the computed trajectory takes minimal time over all valid trajectories in the discretized search space. The search fails when no leaves in the search tree lie in time slices occurring after t_c.

The planner selects the successive values of t_g in increasing order between t_g^{min} and t_g^{max}, at the centers of the time slices in CT_1's bitmap. If the search fails

for one value of t_g and another value is considered, the new search discards every node's successor lying in a cell visited by a previous search. Indeed, at the bitmap resolution, this successor cannot be on a valid trajectory, otherwise the previous search would not have failed. So, each new value of t_g yields a small amount of additional computation.

Planning a deliver motion: We now consider the deliver subtask of (X, A_1). The planner first generates a path connecting the initial and goal configurations of X by conducting a best-first search in the bitmap representing X's C-space. This search is guided by a goal-oriented potential field similar to the NF2 function described in [12] and is restricted to the configurations of X where A_1 and/or A_2 can grasp X. If the search fails, A_1 puts X down on the belt or the table at its current location and the planning process is killed.

If a path is found for X, it entails a path for A_1 through the arm's inverse kinematics. The initial posture of A_1 is the one at the end of the previous grasp path. If in this posture one joint of A_1 reaches a limit, the planner includes an ungrasp operation in A_1's path, a subpath changing A_1's posture, and a regrasp operation, before resuming tracking X's path. If A_2 lies along the way of A_1's path, a motion of A_2 to clear the way is generated. This motion is computed in C_2, into which A_1's path maps as a forbidden region. The final configuration of A_2's is any configuration outside this region. The path of A_2, scheduled at maximal velocity, starts first. The motion of A_1, also at maximal velocity, starts as soon as it can no longer collide with A_2. To determine the starting time of A_1's motion, the planner maps A_2's trajectory to a forbidden region in CT_1; it then represent A_1's trajectory as a curve segment in CT_1 with its initial point at the time when A_2 is scheduled to terminate its motion; finally it translates this curve toward smaller values of time. The position of the curve just before it intersects the forbidden region due to A_2 gives A_1's starting time.

If X's path leaves A_1's workspace, the planner commands A_1 to put down X at an intermediate position where it can be regrasped by A_2.

Bitmap computation: The role of a bitmap representing a C- or CT-space is twofold. It provides a discretization of a continuous space prior to searching that space and allows for quasi-instantaneous collision checks.

Part's C-space: The C-space bitmap for a part X represents the forbidden region created by the obstacles. We model both X and the obstacles as unions of convex polygons, $\{X_i\}$ and $\{O_j\}$, respectively. Every pair (X_i, O_j) yields a subset of the forbidden region in X's C-space. Any cross-section of this subset at a constant orientation of X is itself a convex polygon that is computed in time linear in the number of vertices of X_i and O_j [15]. A polygon-fill transforms this polygon into a 2D bitmap. The 3D C-space bitmap of X is constructed by stacking fixed-orientation 2D slices.

Arm's C-space: The planner precomputes link-link bitmaps by treating one link of an arm as a fictitious robot free to translate and rotate in the plane and a link of the other arm as an obstacle. In each arm, we choose the reference point of the second link at the center of rotation of the second joint. Thus, if we fix the first joint angle θ_{11} of A_1, the reference point of the second link is also fixed. Given the configuration of A_2, we scan all possible values of θ_{11} in C_1's bitmap. Each value determines a cell in the link-link bitmap representing the interaction between the first link of A_1 and the second link of A_2. If a collision occurs, the whole column in C_1's bitmap is filled with "1"s. Otherwise, the position of the reference point defined by the current value of θ_{11} determines a column in the link-link bitmaps representing the interaction between A_1's second link and each of the two links of A_2. After shifting these two columns appropriately (to align their origins with the origin of the column of C_1's bitmap at the current θ_{11}) and removing the cells beyond the second-joint mechanical stops, we compute their boolean union and copy the result into the column of C_1's bitmap at the current value of θ_{11}.

6 Extensions

Changes in goals: The goal of a part can be changed at any time. If a part X arrived before its goal changed, but the deliver motion has not been computed yet, the new goal will be used by the planner when it solves for the deliver subtask. If, instead, the deliver motion has already been planned, it is executed without modification; when X reaches its previous goal, the planner plans a new motion to transfer it to its new goal.

New types of parts: New types of parts can be dynamically introduced. For the user, adding a new part means describing its geometry, defining its goal, and specifying the grasp position of a gripper. For the planner, it only requires computing new bitmaps. As long as the number of obstacles and parts of different types is not too large, this computation can be carried out on-line without significantly weakening the total system performance.

Changes in obstacles: The locations of the obstacles can be changed. Such changes must occur when no deliver motion is being executed. They require the planner to update the C-space bitmaps of the incoming parts. Obstacles can also be added or removed. Whenever an obstacle is added, new bitmaps describing the interaction of this obstacle with the various types of parts that may be fed are computed.

Anticipating catches: Rather than idling, if an arm is not moving, the planner generates a motion for that

arm to bring it to a predefined configuration where its gripper is close to the belt. Thus, when a new part arrives on the belt, the arm will be in a better position to catch it quickly.

7 Experimentation

Implementation: The planner has been implemented in C on a DEC Alpha workstation (Model Flamingo). It has been connected to both a graphic simulator and a real robotic system. The sequence of snapshots shown in Fig. 2-3 was produced by our planner connected to the simulator. The robot system in the simulator has the same general characteristics as the real system. The lengths of the first and second links of each arm are both 24in. The first joint spans a 135dg interval and the second a 285dg one. The maximal velocities of the joints are 15.2dg/sec. The belt moves at 4in/sec. The interval of time during which a part can be grasped is approximately 15sec.

An arm C-space bitmap has size 36×76, which corresponds to increments along each axis of about 3.75dg. The size of the bitmap representing a part C-space is on the order of $128 \times 128 \times 96$; the increments along the two position axes are approximately 0.57in long.

We performed various tests with our software and obtained the following average times for a representative sample of components:

- Computing a part-obstacle bitmap takes 41ms.
- Computing a C-space bitmap for a part takes 8.3ms.
- Updating a part C-space bitmap when an object is added onto the table takes 5.6ms.
- Constructing an arm C-space bitmap using the precomputed link-link bitmaps takes 0.4ms.
- Searching a part C-space bitmap (best-first search) is done at a rate of 65,000 nodes/sec.
- Searching an arm CT-space (best-first search) is done at a rate on 870,000 nodes/sec.

Evaluation: Ideally, the planner's efficiency should be evaluated relative to an instantaneous planner always making the best decision (we call such a planer an *oracle*). However, building such an oracle is not realistic, since it requires implementing an optimal off-line manipulation planner. Instead, we built a quasi-optimal oracle as follows: We let each arm move at maximal velocity along a simple path connecting two configurations, one where the gripper is above the belt, the other where it is above the table away from the belt. These motions are performed alternately, forward and backward, so that when one arm is above the belt the other arm is above the table. The trajectories are defined so that no collision occurs in the middle. Then we define a feeding sequence of parts so that when an arm is above the belt, a part is right there to be grasped and the goal of this part is exactly at the other end of the arm's trajectory. Finally, we distribute the obstacles so that no part collides with an obstacle when it is moved by an arm. By construction, the missing ratio of this oracle for the sequence of parts defined above is 0%.

Unc. (sec)	0.0	0.5	0.5	0.75	0.75	1.0	1.0
Slowing	No	No	Yes	No	Yes	No	Yes
Parts/Min	13.2	13.2	11.8	13.2	11.3	13.2	10.8
ρ_o	0%	20%	0%	25%	0%	28%	0%
ρ_p	13%	19%	6%	17%	2%	14%	0%

Table 1: Planner vs. quasi-optimal oracle

We ran the planner with the same obstacle distribution and the same sequence of parts. Each run was 8min long, during which on the order of 100 parts were being fed. Table 1 compares results obtained with the oracle and the planner. In column 1, we feed the parts with no uncertainty. The number of parts fed per minute is 13.2. The missing ratio ρ_o of the oracle is 0%, while the missing ratio ρ_p of the planner is 13%. In columns 2 and 3, we let parts arrive within a ± 0.5sec uncertainty interval. If the feeding rate is unchanged (column 2), ρ_o increases sharply to 20% (this is obtained by temporarily stopping the arms' motions whenever an arm reaches the belt prior to the arrival of the part); on the other hand, ρ_p increases slightly to 19%. Let us slow down the feeding rate just enough so that ρ_o becomes 0% (column 3); this requires stopping the arm motions given by the oracle, for a maximum of 1sec prior to any grasping operations. The belt now feeds 11.8 parts/min. ρ_p is also reduced to 6%. The subsequent columns show similar results when feeding uncertainty is ± 0.75sec and ± 1sec. When the feeding rate is slowed down just enough to make the ρ_o equal to 0%, ρ_p drops to 2% and 0%, respectively.

Connection to robotic system: We have integrated our planner with a dual-arm robotic system developed in the Aerospace Robotics Laboratory at Stanford [16]. We successfully experimented with this integrated system on examples similar to the one shown in Fig. 2-3.

Because the planner assumes that arm joints can change velocity instantaneously, the trajectories it computes cannot be executed accurately. Hence, the controller recomputes their time parameterization (without changing their geometry) using a realistic dynamic model of the arms. To guarantee that a recomputed trajectory remains collision-free, the planner is slightly more conservative than presented above. For example, consider a grasp trajectory of A_1 to be performed while A_2 is moving. The planner maps A_2's trajectory to a forbidden region in CT_1 and extends this region by its shadow along the negative time dimension. Thus the planned trajectory of A_1 can be arbitrarily translated toward the right

(greater values of time) without causing any collision. It is then sufficient for the controller, when it recomputes A_1's trajectory, to make sure that A_1 is never ahead of time relative to A_2. We deal with residual control errors by slightly growing the arm links before computing the link-link bitmaps. Similarly, localization errors by the vision system lead us to grow the objects before computing the part-obstacle and part-part bitmaps.

With the above modifications, motions are guaranteed to be collision-free, but an arm may not arrive in time to grab a part on the belt. This problem is handled by setting the maximal joint velocities in the planner smaller than the actual values. On our implementation, the velocity bounds given to the planner are constants that have been estimated through preliminary experiments.

Finally, we must consider grasping operations on the belt. When a gripper arrives within some distance of the part it is expected to grasp, the controller stops executing the planned trajectory. It then tracks the part using the last position given by the vision sensor and the measured velocity of the conveyor belt. When it is above the part, moving at the same velocity, the controller commands the grasp operation. While in this autonomous mode, the controller checks for collision between the arms. If one is going to happen, it stops both arms and reports the failure to the planner.

8 Conclusion

This paper describes an on-line manipulation planner for a dual-arm robot system whose task is to grab parts arriving on a conveyor belt and deliver them at specified goals. Parts arrive at any time, in random order. The planner uses information provided by a vision system to break the overall planning problem into a stream of rather simple subproblems and orchestrate fast planning primitives solving these subproblems. Experiments conducted with this planner show that it compares very well to quasi-optimal oracles. Since the planner also allows dynamic changes in obstacles, goals, and tasks, this result suggests that on-line planning may rapidly become more attractive than off-line planning. In fact, we believe that our on-line planner enables low-cost, flexible, and efficient part feeding. Other experiments not reported here (see [14]) show that an increase in computational speed does not yield a significant improvement in the planner's efficiency. As computers become faster, this result indicates that future research should be aimed at devoting more computation time generating motion plans that are quicker to execute.

References

[1] R. Alami, T. Siméon, and J.P. Laumond, A Geometrical Approach to Planning Manipulation Tasks: The Case of Discrete Placements and Grasps, *Robotics Research 5*, MIT Press, Cambridge, MA, 1990, 453-459.

[2] J. Barraquand and J.C. Latombe, Robot Motion Planning: A Distributed Representation Approach, *Int. J. of Rob. Res.*, 10(6), Dec. 1991, 628-649.

[3] H. Chang and T.Y. Li, Maintainability Study with Motion Planning, *Proc. IEEE Int. Conf. Rob. and Aut.*, Nagoya, Japan, 1995.

[4] M. Erdmann and T. Lozano-Pérez, On Multiple Moving Objects, *Algorithmica*, 2(4), 1987, 477-521.

[5] B. Faverjon and P. Tournassoud, A Practical Approach to Motion-Planning for Manipulators with Many Degrees of Freedom, *Robotics Research 5*, MIT Press, Cambridge, MA, 1990, 425-433.

[6] K. Fujimura, *Motion Planning in Dynamic Environments*, Springer-Verlag, New York, NY, 1991.

[7] L. Graux et al., Integration of a Path Generation Algorithm into Off-Line Programming of AIRBUS Panels, *Aerospace Automated Fastening Conf. and Exp.*, SAE Tech. Paper 922404, Oct. 1992.

[8] K. Gupta and S.W Zucker, Toward Efficient Trajectory Planning: Path Velocity Decomposition, *Int. J. of Rob. Res.*, 5, 1986, 72-89.

[9] L. Kavraki and J.C. Latombe, Randomized Preprocessing of Configuration Space for Fast Path Planning, *Proc. IEEE Int. Conf. Rob. and Aut.*, San Diego, CA, 1994, 2138-2145.

[10] Y. Koga et al., Multi-Arm Manipulation Planning, *9th Int. Symp. on Automation and Robotics in Construction*, Tokyo, 1992, 281-288.

[11] Y. Koga and J.C. Latombe, On Multi-Arm Manipulation Planning, *Proc. IEEE Int. Conf. Rob. and Aut.*, San Diego, CA, 1994, 945-952.

[12] J.C. Latombe, *Robot Motion Planning*, Kluwer, Boston, MA, 1991.

[13] J. Lengyel et al., Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware, *Proc. SIGGRAPH'90*, Dallas, TX, 1990.

[14] T.Y. Li and J.C. Latombe, *On-Line Manipulation Planning for Two Robot Arms in a Dynamic Environment*, Tech. Rep., Dept. of Comp. Sc., Stanford U., 1994.

[15] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," *IEEE Tr. on Comp.*, 32(2), 1983, 108-120.

[16] G. Pardo-Castellote et al., Experimental Integration of Planning in a Distributed Control System, *Proc. Int. Symp. on Experimental Rob.*, Kyoto, Oct. 1993, 217-222.

[17] J.H. Reif and M. Sharir, Motion Planning in the Presence of Moving Obstacles, *Proc. FOCS*, 1985, 144-154.

[18] P. Tournassoud, T. Lozano-Pérez, and E. Mazer, Regrasping, *Proc. IEEE Int. Conf. Rob. and Aut.*, Raleigh, NC, 1987, 1924-1928.

[19] G. Wilfong, Motion Planning in the Presence of Movable Obstacles, *Proc. ACM Symp. on Comp. Geometry*, 1988, 279-288.