On-line Collision Avoidance for Multiple Robots Using B-Splines

Eric Paulos*
paulos@cs.berkeley.edu

Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

Abstract

Real world assembly sequences consists of multiple assembly steps, many of which can be performed in parallel. In practice this parallelism is often not exploited because of the complexity involved in avoiding collisions between all of the robots. In this paper we describe a simplified method of achieving smooth collision free paths for multiple robots within a single assembly workcell. Our method is simple because we exploit properties of B-splines to reduce the problem to path planning without moving robots. We develop a path planner to compute an initial linear path. Using that path as input, a trajectory generation tool creates a collision free path of any desired continuity. We exploit three properties of B-splines: (i) continuity for smooth paths, (ii) convex-hull for collision avoidance, and (iii) locality for dynamic course alteration without loss of continuity. In addition, our system runs in real-time, easily accommodating multiple robots. Finally, we describe a user level visualization tool for this system.

1 Introduction

Typical assembly tasks exhibit locality. An action in a local region is followed by a transport to a new local region where the next operation is performed. We exploit this *locality of assembly operations* (See §5.1) and observe that we can categorize every robot into one of three possible states: (1) idle, (2) transport, or (3) task. In addition we claim that this is a correct state model for almost any robot workspace, particularly in an assembly workcell. The two states: idle, robot not involved in any motion or task, and transport, robot in motion between a start and goal position, are rather clearly defined concepts. The task state includes local motions that require full control of the robot, not simply satisfiability of start and goal positions. Typical tasks are end-effector compliance and damping, search strategies, tactile sensing, line scanning, probing, and any other approach that relies on some form of specialized feedback control. The advantage of our technique is that we can ignore motions of other robots, reducing the problem to path planning without moving robots.

In this paper we investigate the development of four tools, a workcell free-space path generation tool, a general trajectory generation tool, a workcell management agent, and a user workcell visualization tool. We demonstrate how these tools combine into one powerful utility for managing the interactions of multiple robots. The application platform that this system was actually designed for is an assembly workcell robot. However, it can be applied to any multi-robot system that can maintain relative position information.

1.1 Previous and Related Work

The importance of planning and generating collision free paths for multiple robots has led several others to propose methods for solving this problem. One approach is to add time to the dimension of the configuration space. In general a k-dimensional changing environment can be represented as $(k+1)$-dimensional space-time. Collision free path planning then occurs in this higher-dimensional space-time. Reif and Sharir [17] prove the existence of collision free paths in this, monatomic time, higher-dimensional space. Prioritizing the order in which to plan the individual robot paths is explored by Erdmann and Lozano-Pérez [7] and later by Buckley [4]. Aggarwal and Fujimura [1] describe a specialized use of the cell-decomposition approach in space-time to handle ro-

*Financial support provided by National Science Foundation Presidential Young Investigator Award #IRI-8958577 and National Science Foundation Grand #IRI-9114446.

tations of planar robots. When objects are moving non-deterministicly, Tsubouchi and Arimoto [21] show that it is sometimes possible to track and forecast the motions of these other moving obstacles and plan paths based on those results. The development of path planning for multiple robots that are tethered is studied by Parsons and Canny [15] and later by Hert and Lumelsky [11].

The use of splines in trajectory generation has been proposed by Lin *et al.* [12], and more recently by Yi and Kim [22] who demonstrate methods to optimize trajectories using cubic splines. Simon and Isik [19] present the use of trigonometric splines to compute minimal jerk trajectories on a 6R robot. Choi *et al.* [5] examine how the tension parameters of Catmull–Rom splines can be used for building trajectories for infinitely flexible snake-like robots. However, the application of B-splines for multiple robot path planning and trajectory generation is rather new. Thompson and Patel [20] characterize a technique that uses multiple knots for interpolation of joint angles on a single 6R robot. They gain interpolation at the cost of locality. Our approach differs significantly in that we exploit the B-spline properties of locality, continuity, and convex-hull for collision avoidance, and dynamic course alteration in a multi-robot system.

1.2 Overview

In our system a workcell management agent orchestrates the high-level actions within the workcell (See Figure 1). One duty of the workcell management agent is to grant robots permission to traverse the workcell. When a robot desires to travel to a new area of the workcell, the workcell management agent grants the request and uses the path generation tool to provide the robot with a 1-dimensional C^0 path to follow to the goal. The robot in turn hands this path off to the trajectory generation tool to be converted to a 1-dimensional C^{k-2} curve, where k is the order of the B-spline (see §3.1). We also compute the derivatives of the curve so that they can be used for passing raw motion commands down to the low-level robot controller.

The B-spline trajectory will also be used to perform collision avoidance for the robot. This is done by broadcasting its current B-spline interval to all of the other robots. From that information the other robots can determine the k control vertices with non-zero B-spline basis functions that form the convex hull of the moving robot's location. The changing convex-hull of a robot as it traverses various intervals of the B-spline path is shown in Figure 2. The other robots enter this convex obstacle into their obstacles list. If any of the other robots need to perform a motion, the first level path planner will handle the moving robot correctly. This can be observed since even though the robot is in motion, it appears as a fixed obstacle. When the robot moves to the next interval, it broadcasts this so that the other robots can change their view of the workcell obstacles, removing one vertex and adding one vertex to the moving robot's convex-hull obstacle. This technique is applicable to any number of moving robots. The only condition is that the robots go through a mutually agreed upon arbitrator for disputes. This is the job of the workcell management agent.

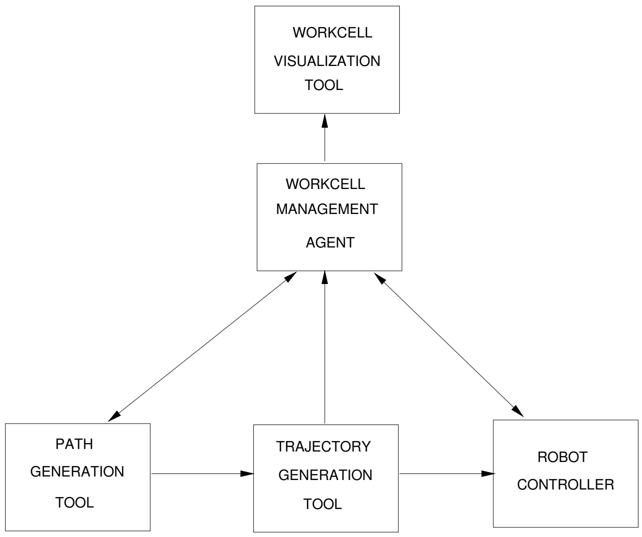

Figure 1: The interactions of the different tools involved in the collision avoidance framework

2 Path Generation Tool

Each robot maintains its own version of the workcell state. The state is represented by the robot's current configuration, the boundaries of the fixed obstacles, and the convex regions containing other robots. From the vantage of each robot, idle robots appear as fixed obstacles. Task robots appear as fixed obstacles defined by the particular convex region allocated for that operation. All of the remaining robots are transport robots and appear as fixed obstacles defined by the k, where k is the order of the B-spline (See §3.1), control vertices of the current B-spline interval.

Each robot performs its own Minkowski sum operations on the various robots and fixed obstacles in the workspace, reducing itself to a point. As a re-

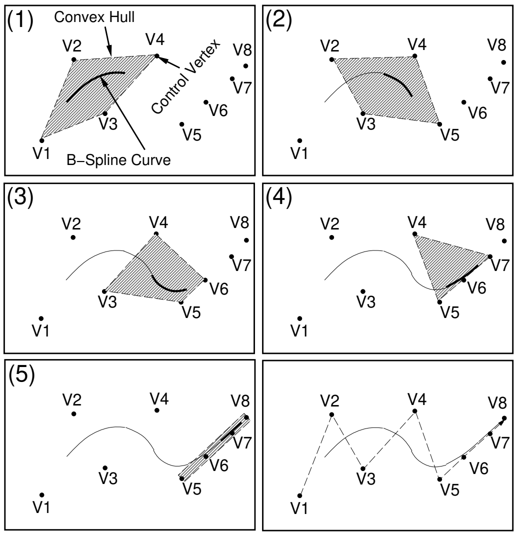

Figure 2: A B-spline robot path with continuously updated ($k = 4$) control vertices forming the convex hull of each C^2 curve segment

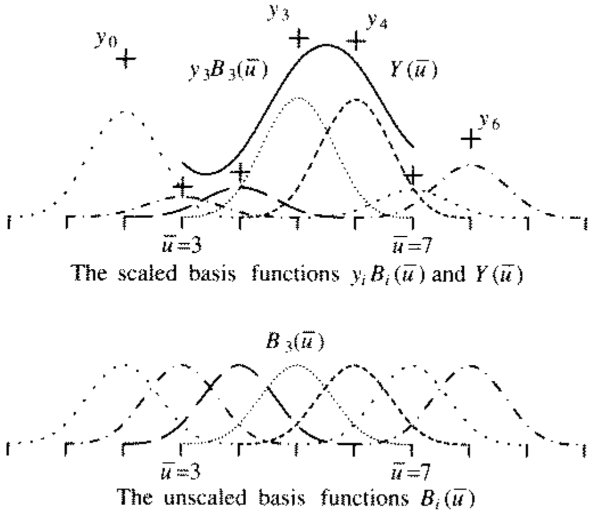

Figure 3: A B-Spline curve and its basis functions

sult, each robot has a geometrically different yet physically consistent view of the workspace. Each robot also, computes a visibility graph for its version of the workcell. These visibility graphs can be constructed in $O(n^2 \log n)$ where n is the number of vertices in the original graph. More efficient algorithms have been proposed that bring this down to $O(m+n \log n)$ where m is the number of edges in the visibility graph. In the worst case m is in $O(n^2)$ but in better cases it can be as small as $O(n)$.

Eventually, the workcell management agent will make a request to the path generation tool to find a path between two points for a particular robot. The path generation tool, using the previously computed visibility graph, calculates the shortest path using Euclidean distance as an edge weighting metric. Searching a visibility graph for its shortest path can be done using various techniques. Dijkstra's or the A^* algorithm can find the path or return failure in $O(n^2)$, or if the list of vertices is pre-sorted, in $O(m \log n)$ where m is the number of vertices in the visibility graph. The resulting list of vertices in the path is passed on to the trajectory generation tool for conversion into a smooth trajectory. We have chosen visibility graphs and shortest paths because they are straightforward to compute and are complete.

3 Trajectory Generation Tool

The trajectory generation tool ignores the location of obstacles in the workcell. The path planner has already handled the generation of a collision free path between the start and the goal configuration. This is an important concept to remember, since this decoupling of path planning from trajectory generation has the affect of simplifying both the path generation and trajectory generation tools.

3.1 B-Splines

A B-Spline is composed of multiple B-spline basis functions denoted by $B_{i,k}(\bar{u})$ where i is the interval number and k is the order of the B-spline basis function. Since in this paper we concern ourselves with uniform B-splines, the intervals are defined by a uniformly spaced parametric knot sequence. The form of the basis function is dependent upon its order, where order k B-spline basis functions are curves of degree $k-1$. A single B-spline basis function is constructed as k piecewise curve segments, all of degree $k-1$. These segments are constrained to connect to one another with C^{k-2} continuity and have unit area beneath the basis function.

The entire B-spline curve, $Q(\bar{u})$, is composed of a vertex weighted sum of the basis functions and can be written as

$$Q(\bar{u}) = \sum_i V_i B_{i,k}(\bar{u})$$

where the V_i's are the control vertices. In Figure 3 we have a cubic, fourth order, B-spline, with seven control vertices denoted by y_i. It requires four basis functions to properly define each cubic curve segment. Hence there are three more basis functions (and three more control vertices) than there are curve segments. Each basis function is nonzero over exactly four parametric intervals. The leftmost basis function extends three additional intervals to the left of the curve, and the rightmost three to the right. Summarizing: there are $m+1$ control vertices, $m+1$ basis functions, $m-2$ curve segments bounded by $m-1$ knots, and $m-1+3+3 = m+5$ knots altogether. The curve is generated as $\bar{u}$ runs from $\bar{u}_3$ to $\bar{u}_{m+1}$. For lack of space, we direct the reader to several other sources [3, 6, 8, 9, 18] for further discussions of B-splines.

3.2 Forming a B-Spline Trajectory

Continuity is the most important property of the path generation tool. By using a B-splines of the appropriate degree/order, the continuity of the final path can be adjust higher or lower depending upon the constraint parameters of the individual robotic system. However, we propose the use of a continuous jerk path since it strongly relates to the impact characteristic of the loading of the system [13] as well as the rate of change of force producing acceleration [10].

Since we desire C^3 continuity (continuous jerk), we adopt the use of fifth order ($k = 5$) B-splines. The basis function is of fourth degree and composed of five piecewise segments denoted by b_i. We use a set of 25 equations and 25 unknowns that represent the continuity and unit area constraints of the basis function to solve for its five segments. Notice that each segment of the basis function is of degree four.

$$\left.\begin{array}{lcl} b_{-0} & = & \frac{1}{24}u^4 \\ b_{-1} & = & \frac{1}{24}(1+4u+6u^2+4u^3-4u^4) \\ b_{-2} & = & \frac{1}{24}(11+12u-6u^2-12u^3+6u^4) \\ b_{-3} & = & \frac{1}{24}(11-12u-6u^2+12u^3-4u^4) \\ b_{-4} & = & \frac{1}{24}(u-1)^4 \end{array}\right\} \quad (1)$$

We do not use these equation in the actual evaluation of B-splines. Instead we use the Cox-DeBoor recurrence relation [6]. This recurrence, shown in Equations 2 and 3, can be generalized to compute a B-spline basis of any order and derivative.

$$\frac{d^j}{d^j u} V_i B_{i,k} = \sum_i \nu_{i,j+1} B_{i,k-j} \quad (2)$$

where

$$\nu_{r,j+1} = \begin{cases} V_i & \text{for } j = 0 \\ \frac{\nu_{r,j}-\nu_{r-1,j}}{(U_{r+k-j}-U_r)/(k-j)} & \text{for } j > 0 \end{cases} \quad (3)$$

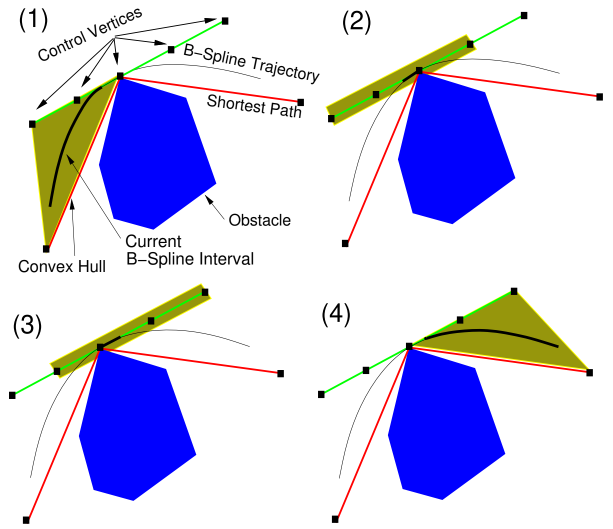

Figure 4: The convex hull of each segment (shaded) and the fourth order B-spline ($k = 4$) for a path rounding an obstacle

We develop two methods for transforming path vertices into B-spline control vertices. First we concern ourselves with the explosion of interior vertices along the path and then the selection of end-condition control vertices.

3.2.1 Interior Control Vertices

We will explode each interior vertex along the path provided by the path planing tool into $O(k)$ B-spline control vertices. This construction is done by adding $2(k-2)+1$ control vertices along a line segment L and two vertices at other calculated positions. The placement of the line segment L and the other two points allow us to insure that the resulting trajectory will observe the physical limitations of the robotic system. The reason for the placement of control vertices along a line segment is that the convex hull of every series of k consecutive vertices will lie outside of the obstacle. This is shown for each interval in Figure 4.

We show the construction of the actual line segment L in Figure 5. We first characterize the velocity and acceleration limits of the physical robotic system with a circle of radius r. This circle represents a conservative estimate of the minimum radius of curvature path that the physical robot can traverse. Again B-splines do not in general interpolate the actual vertices and hence the radius of the circle must be conservative. To optimize the shape of the B-spline, its curvature can be calculated and matched to the minimum radius of

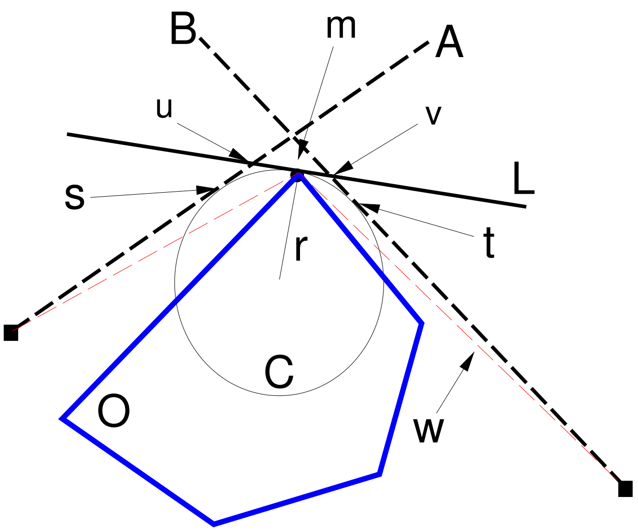

Figure 5: Construction of line segment L and placement of control vertices

curvature of the velocity and acceleration constraints. We use the circle itself to construct a first order approximation of optimal curvature.

We place the circle C such that its center lies equidistant from the two segments of the shortest path, w, and coincident to the vertex m on the obstacle, O. We form line segment A extending from the previous vertex of the shortest path and tangent to the circle C at point s. Similarly, we form the line segment B as extending from the following shortest path vertex and tangent to the circle C at point t. We construct the line L as passing through the vertex m and tangent to the circle C at that point. The placement of control vertices is then as follows: one at point s, $2(k-2)$ along L between points u and v, one at point m, and one at point t. In this construction u is the intersection point of segments A and L. Similarly v is the intersection point of segments B and L.

Placing the control vertices such that we "hug" corners of obstacles is actually desirable. Many path planning techniques result in paths which are "equidistant" from all of the obstacles, essentially traversing down the middle of the free space. While this is good for a single robot system, it seriously degrades performance in a multi-robot system. By "hugging" the vertices of obstacles, we minimally clutter the workcell transportation corridors, allowing room for passage of other robots at the same time.

3.2.2 End Condition Control Vertices

Most trajectories ramp up from a complete stop and come to a complete stop at their termination. This necessitates the construction of the end conditions of the B-spline to have position specification (i.e. end vertex interpolation), and some derivatives at zero. We can achieve this by applying the technique of phantom vertices developed by Barsky [2]. In essence this is a method for determining the correct placement of extra control vertices before the first vertex and after the last vertex to achieve some specified condition(s) at the ends of the B-spline curve. For the fifth order B-spline we have calculated the following phantom vertex values for the beginning of the curve segment. We do not explore the end conditions at the end of the curve since they are symmetric to those at the beginning.

- Start vertex interpolation:

$$V_{-1} = 13V_0 - 11V_1 - V_2$$

- Start vertex interpolation and zero velocity:

$$\begin{aligned} V_{-1} &= \frac{-V_1}{4}V_1 + \frac{5}{4}V_0 \\ V_{-2} &= \frac{7}{4}V_1 - \frac{3}{4}V_0 \end{aligned}$$

- Start vertex interpolation, zero velocity, and zero acceleration (triple vertex):

$$V_{-1} = V_{-2} = V_{-3} = V_0$$

Based on the desired end conditions, the trajectory generation tool transparently inserts these phantom vertices into the list of control vertices, yielding the desired effect at the ends of the B-spline curve. If we make a path modification while the robot is in motion, these phantom vertices can be easily recomputed to maintain the end-condition constraints.

3.3 Robot Controller

Robot controllers are typically quite diverse by nature of their underlying hardware differences. Recent approaches to standardize such controller interfaces [14] have yet to gain popularity. However, our trajectory generation control tool provides a wealth of output information, providing position, velocity, acceleration, and jerk verses time. These are easy to compute using the recurrence relation of Equations 2 and 3. Almost any commercially available robot system can use one or more of these pieces of information to drive its low level controller. This flexibility makes our system well suited for a variety of robots.

4 Multiple Robot Interactions

The complex interactions between all of the multiple robots is handled automatically by the tools discussed in our approach. A particular property of B-spline curves is local control, by which we mean that altering the position of a single control vertex causes only part of the curve to change. An added benefit of local control is that it minimizes the computational overhead required to recompute a curve after a control vertex has been moved since only a small portion of the curve has changed.

For example if we have two moving robots, X and Y, each traversing the workcell, it may be the case that robot X's B-spline path moves into a new B-spline interval. This means that one control vertex is removed from its convex-hull and one vertex added. This in turn changes the convex-hull of robot X. This may result in an alteration to the shortest path of robot Y. Robot X is guaranteed not to affect the positions of the control vertices that are currently being used to evaluate the B-spline path for robot Y because those vertices form the convex-hull for robot Y. These vertices make robot Y appear as a fixed obstacle to robot X. Clearly, robot X is not allowed to alter the appearance of a fixed obstacle. Robot Y is currently in transport across the workcell, but its future path will collide with robot X. The path planner has made the necessary changes along the future shortest path to avoid such a collision. Since all of the future B-spline basis functions for the original control vertices are zero, we can remove those vertices and replace them with new control vertices that avoid the collision. Robot Y will make the path correction when it starts evaluating the B-spline basis functions over those new control vertices. In addition we are guaranteed by the properties of B-splines that the path, although altered, will continue to be C^{k-2}. This same argument is applicable to any number of robots. This ability to dynamically alter paths and maintain both continuity and end-condition constraints is a major benefit of our system.

5 Workcell Management Agent

Use of a single agent to guide the actions within a workcell can be compared to the actions of a memory manager in a virtual memory system. In a computer operating system, memory is allocated to various processes. When the physical memory is exhausted, the memory manager must choose a piece of memory to reside in a less-desirable space, such as on a magnetic medium. However, when the user process wishes to access that piece of memory again, the manager must be able to return it to its previous location and state.

In the case of the workcell management agent, the resource is not memory, but workcell real-estate and tools. This high-level management agent must be able to deliver the various resources to a robot that requests them and also retrieve them back if the robot gets "greedy." In our system we are mainly concerned with the allocation of space. In a workcell with fixed tools this reduces to tool management since only the robot that is in the area containing a tool can use that tool.

5.1 Locality of Assembly Operations

Typical assembly tasks exhibit locality. An action in a local region is followed by a transport to a new local region where the next operation is performed. Common sites in a workcell are easy to identify from an assembly sequence:

1. Part enters the workcell on a <u>conveyor</u>
2. Robot transports part through <u>sensors</u> to localize and identify part
3. Part is transported to <u>fixturing platform #1</u> to perform a sub-assembly.
4. Sub-assembly is transported to a <u>holding area</u> until the rest of the sub-assemblies are completed
5. Sub-assembly is moved to <u>fixturing platform #2</u> for final assembly
6. Final assembly is transported to <u>sensors</u> for inspection
7. Final inspected assembly is transported to <u>conveyor</u> and leaves the workcell

Each of the operations above takes place in a rather small area of the workcell. By giving exclusive rights to each of these areas after a request is made, the workcell manager maintains state information about the progress of work. If a urgently needed part comes into the workcell and requests to be transported to a fixturing device that's in use, the robot currently using the device can be interrupted and priority given to the robot associated with the higher-priority task, in this case the urgent transport of a part to a fixturing device. Again, since operations are local, the other robots in the workcell will be unaffected by this operation.

5.2 Allocation of Space

The workcell management agent is, in essence, a workcell space manager, maintaining robot ownership rights over space within the workcell as well as performing arbitration between robots over space. Any time up until a robot releases ownership of its space, it may move freely within that space. This is useful because once a robot has been granted ownership of a space, the collision avoidance problem has been solved locally.

At some point in the assembly sequence, a robot will desire to move to a new position in the workcell, call it the goal for that robot. In order for that robot to perform that movement it must be granted "free passage" by the workcell management agent. It does this by making a request to the workcell management agent for passage to its goal. The agent will respond with one of the following:

- **Grant**: There is a collision free path path for the robot to move through. This is true if the path planner returns with a non-empty path to the goal.

- **Fail** In this case the path planner has returned an empty path list. The robot may do another task or wait and try its request later. The workcell management agent maintains a request table so that repeated requests will eventually be granted. This request table keeps a robot from being starved out of all of the resources in the workcell. In addition this will avoid most deadlock condition that may develop as a result of dependencies within the assembly sequence. Some deadlock conditions can only be avoided by extremely high-level task planning at the user level. However, use of the workcell visualization tool is extremely useful in identifying such problems in the assembly sequence.

- **Wait** This is similar to the case above, in the sense that the path planner has returned an empty path list. However, the workcell management agent has decided that one or more of the obstacles preventing the completion of the path are low-priority or idle robots. These are robots that the workcell management agent can send an irrefutable request to move to a clear of the workcell. A **Grant** signal is sent when the path has been cleared. This is essentially a page swap. We have a resource, memory (space), owned by a stale, non-accessed process (robot), and a request to use the resource by another process (robot). The wait signal is just the page-fault until we can swap out the old memory (clear the robots from the path).

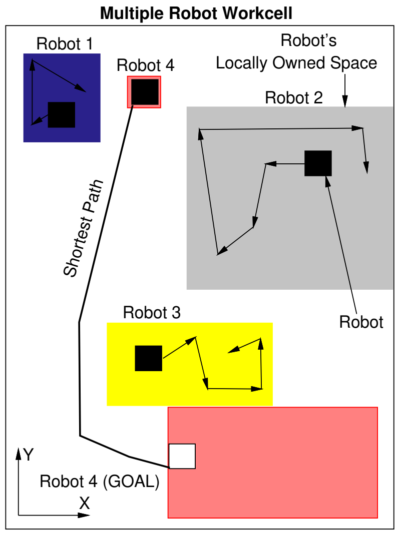

Figure 6: A typical multiple robot workcell depicting allocated space

A typical state of workcell space during an assembly sequence is depicted in Figure 6. In this figure there are four robots, three of which are performing particular sub-assembly tasks in a locally owned region (shown shaded). These qualify as task robots. The fourth is invoking the help of the workcell management agent to plan a trajectory from its own space to another. This robot is an idle robot, however, once it begins moving it will be a transport robot. We combine two of the previous techniques. First the path planner will return the shortest path to the goal. Then the trajectory generation tool will produce a collision free smooth path that will also provide convex regions of containment for the moving robot to help the other robots reduce it to a fixed obstacle. The workcell management agent arbitrates disputes and monitors all of the space management to insure consistency with the other robots.

6 Workcell Visualization Tool

The visualization tool provides a graphical representation of the robot and workcell state. This tool allows a user to display any of the visibility graphs for each of the robots along with its shortest path, convex hull, location, B-spline interval, B-spline path and other useful information for debugging parallel assembly sequences. The power of this tool is in allowing a user to watch the various interactions within the workcell during an assembly task, and as a result make better decisions about workcell tool placement and assembly sequencing.

7 Conclusion and Results

We have developed a generalized tool for high-level on-line collision avoidance exploiting the continuity, locality, and convex-hull properties of B-splines. Throughput with such a system is dramatically increased by allowing robots to perform operations in parallel while freeing the user from the tedium of handling the highly complex interactions between the various robots. In addition our approach exhibits the ability to dynamically alter paths in real-time and maintain both continuity and end-condition constraints imposed by the system on each robot. We have developed this within the framework of an industrial assembly robot and parallelized assembly sequences to increase production throughput. A typical pick-and-place/peg-in-hole assembly technique [16] adopted this system and was parallelized, resulting in a 40 percent reduction in assembly time. Finally, we demonstrate a user visualization tool for our system.

References

[1] Neejraj Aggarwal and Kikuo Fujimura. Motion planning amidst planar moving obstacles. In *IEEE International Conference on Robotics and Automation*, pages 2153–2158, 1994.

[2] Brian Barsky. End conditions and boundry conditions for uniform B-spline curve and surface representations. In *Computers in Industry*, volume 3(1&2), March 1982.

[3] Richard Bartels, John Beatty, and Brian Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[4] Stephen J. Buckley. Fast motion planning for multiple moving robots. In *IEEE International Conference on Robotics and Automation*, pages 322–326, 1989.

[5] P.J. Choi, J.A. Rice, and J.C. Cesarone. Kinematics of an infinitely flexible robot arm. In *Journal of Robotics Systems*, volume 3, pages 407–425, 1993.

[6] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[7] Michael Erdmann and T. Lozano-Pérez. On multiple moving objects. In *IEEE International Conference on Robotics and Automation*, pages 1419–1424, 1986.

[8] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 1988.

[9] James Foley and Andries van Dam. *Introduction to Computer Graphics*. Addison-Wesley, 1990.

[10] C. W. Ham, E. J. Crane, and W.L. Rogers. *Mechanics of Machinery*. McGraw-Hill, 1958.

[11] Susan Hert and Vladimir Lumelsky. The ties that bind: Motion planning for multiple tethered robots. In *IEEE International Conference on Robotics and Automation*, pages 2734–2741, 1994.

[12] Chun-Shin Lin, Po-Rong Chang, and J.Y.S. Luh. Formulation and optimization of cubic polynomial joint trajectories for industrial robots. In *IEEE Transations on Automation Control*, volume 28, December 1983.

[13] Hamilton Mabie and Fred Ocvirk. *Mechanisms and Dynamics of Machinery*. John Wiley and Sons, 1975.

[14] Ed Nicolson. Standardizing I/O for mechatronic systems (SIOMS) using real-time unix device drivers. In *IEEE International Conference on Robotics and Automation*, May 1994.

[15] D. Parsons and J. Canny. Motion planning for multiple mobile robots. In *IEEE Conference on Robotics and Automation*, pages 8–13, 1990.

[16] Eric Paulos and John Canny. Accurate insertion strategies using simple optical sensors. In *IEEE International Conference on Robotics and Automation*, pages 1656–1662, May 1994.

[17] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *IEEE Conference on Foundations of Computer Science*, pages 144–154, 1985.

[18] I.J. Schoenberg. *Selected Papers Volume 2*. Birkhauser, 1988.

[19] Dan Simon and Can Isik. Optimal trigonometric robot joint trajectories. In *Robotica*, volume 9, pages 379–386, 1991.

[20] Stuart Thompson and Rajnikant Patel. Formulation of joint trajectories for industrial robots using B-splines. In *IEEE Transations on Industrial Electronics*, volume 34, May 1987.

[21] Takash Tsubouchi and Suguru Arimoto. Behavior of a mobile robot navigated by an iterated forecast and planning scheme in the presence of multiple moving obstacles. In *IEEE International Conference on Robotics and Automation*, pages 2470–2475, 1994.

[22] Seung-Jong Yi and Kyuil Kim. Effect of tension parameters and intervals on splines-under tension based robot trajectory planning. In *Journal of Robotics System*, volume 11, pages 91–102, 1994.