Proceedings of the 1991 IEEE
International Conference on Robotics and Automation
Sacramento, California - April 1991

Parallel Robot Motion Planning

Tomás Lozano-Pérez,
Patrick A. O'Donnell

MIT Artificial Intelligence Laboratory

Abstract

This paper presents a fast, parallel method for computing configuration space maps. The method is made possible by recognizing that one can compute a family of primitive maps which can be combined by superposition based on the distribution of real obstacles. We use the fast configuration space computation to implement a global six degrees-of-freedom path search for a Puma robot.

1 Introduction

In this paper we present a simple global motion-planning algorithm that can take advantage of the capabilities of massively-parallel machines. The key idea that makes this algorithm possible was first articulated in W. Newman's doctoral thesis [10]. Newman noted that, for most robot kinematics, one can precompute a one-parameter family of **primitive configuration space maps**, indexed only by radial distance to the robot base. One can then obtain complex configuration space maps by superposition of these primitive obstacle maps. This result is a generalization of the union property of the configuration space, where the configuration space obstacle for a union of objects is the union of the configuration space obstacles of the individual objects [7]. The superposition of primitive maps is quite general and can be used to construct configuration space maps for the first three links of the majority of existing industrial robots. We have exploited an extension of this property to develop parallel algorithms for computing the configuration space maps for six degrees-of-freedom wrist-decoupled robots.

The primitive configuration space maps can be represented as bitmaps. The advantage of a bitmap representation is that it provides a simple and efficient means of superimposing precomputed maps. The drawbacks of a bitmap representation are that the memory requirements limit the practical resolution of the map, and it forces a discretization of the configuration space in all dimensions.

Nevertheless, under a few simplifying assumptions, notably that the robot have few degrees-of-freedom (typically three) and that the required position resolution is limited, one can construct complex configuration space maps in a few seconds on a modern workstation [10, 2]. Although some previous planners [8, 4] are almost this fast on comparable problems, the bitmap superposition approach leads to very simple algorithms and is ideal for parallel implementation.

We have implemented this motion planner for the first three degrees-of-freedom of a Puma robot in *Lisp™ on a Thinking Machines' Connection Machine with 8K processors. The time to build a $32 \times 32 \times 32$ configuration space is approximately 0.3 sec; for a $64 \times 64 \times 64$ configuration space, the time is approximately two seconds. In both cases, the running time is independent of the number of obstacle bits of the input obstacle maps. Increasing the number of physical processors in the Connection Machine, which can have up to 64K processors, would provide a linear speedup.

We have also implemented a six degree-of-freedom version of the algorithm. This algorithm performs a sequential search of the six dimensional configuration space, building three-dimensional cross sections in parallel. In the example illustrated in Figure 1 a path was found in approximately three minutes.

2 Relation to Previous work

There has been a great deal of published work on motion planning during the past twenty years. A number of approximate motion planning algorithms have been implemented and tested; the survey [9] lists many of these. The survey [12] describes work on exact combinatorial algorithms.

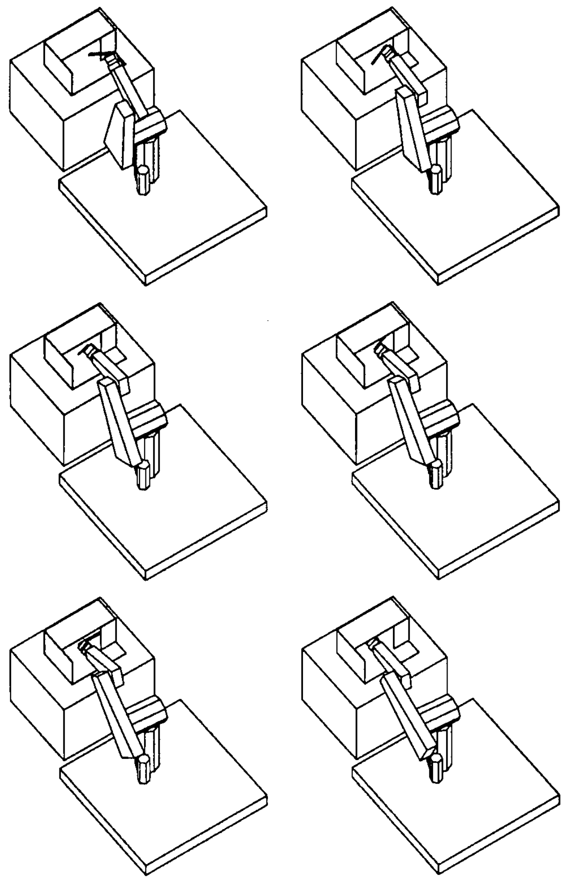

Figure 1: A six degree of freedom path for a Puma robot found with this algorithm. The task is to insert a 12 inch ruler into a partially closed cardboard box. The sequence is left to right, top to bottom.

A number of recent approaches that are worthy of note are: Local methods—[1] uses a combination of a low-dimensional Voronoi-like diagram and randomized search to explore large degree of freedom configuration spaces, [5] uses a variant of the potential field method that uses linearized constraints in the configuration space rather than pseudo-forces. Global methods—[6, 3] show how to compute explicit configuration space boundaries for manipulators with several degrees of freedom, avoiding the sampling techniques of previous methods.

We are aware of very few parallel algorithms for motion planning, except Pollard [11].

3 Primitive configuration space maps

Our approach exploits the rotational symmetry in the kinematic structure of most robots to allow precomputation of the configuration space maps for simple obstacles [10, 2]. We will illustrate this idea first for a two-link planar revolute robot, then for the first three links of a Puma-like robot, next for the first three links of a variety of other robot types, and finally for wrist-decoupled six degrees-of-freedom robots. The treatment of the first three links of robots parallels that in [10, 2]; the treatment of wrist-decoupled robots is new. In this section we describe the general approach. Section 4 generalizes the approach to six degree-of-freedom grippers. In Section 5 we show the parallel algorithms.

3.1 A two-link revolute robot

Consider the two-link planar manipulator shown in Figure 2(a). The configuration of this manipulator is specified by the two joint angles θ_2, θ_3. (The choice of joint angle subscripts 2 and 3, instead of 1 and 2, is to retain consistency with the three-dimensional robot we will discuss later.) In Figure 2(a), we can also see a point obstacle whose polar coordinates are r, ϕ. The configuration space obstacle corresponding to collisions between this point obstacle and the last link is shown in Figure 2(b). This configuration space obstacle is made up of a single locus, described by the following parametric equations with parameter s (which is the distance on the second link where the point intersects):

$$\theta_3 = \cos^{-1}\left(\frac{r^2 - l_2^2 - s^2}{2l_2 s}\right) \quad (1)$$

$$\theta_2 = \phi - \operatorname{atan}(s\sin\theta_3, l_2 + s\cos\theta_3) \quad (2)$$

The crucial observation is that ϕ enters into these equations only as an offset in θ_2. The only obstacle parameter that affects the shape of the locus is r. The configuration space obstacle for another point with the same r but different ϕ can be derived from the one in Figure 2(b) simply by shifting in the θ_2 direction.

This property follows from the rotational symmetry of the problem; the choice of the zero value for θ_2 is totally arbitrary. Had we chosen the base frame to be rotated so that the ϕ coordinate for the obstacle point was zero, we would not have changed the geometry of the problem in any way. A rotation of the base coordinate system amounts to a shift in the configuration space. Therefore, we should expect the effect of a coordinate system rotation to manifest as a shift in the configuration space; it does.

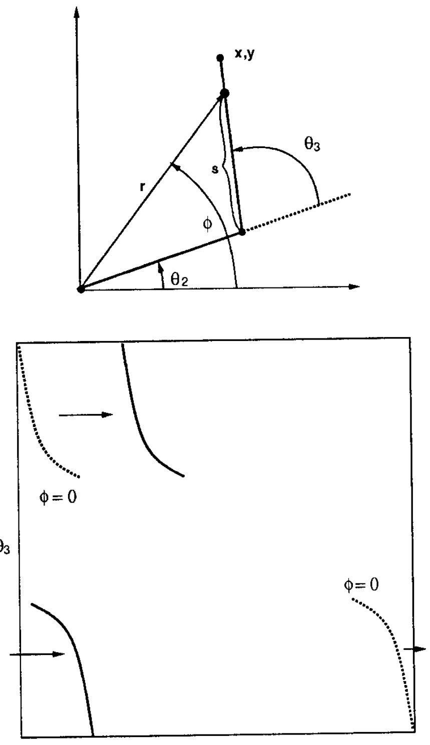

Figure 2: (a) A two-link planar manipulator. θ_2 and θ_3 are the joint variables. A point obstacle is shown at polar coordinates (r, ϕ). Configurations which collide depend on the parameters s, r, and ϕ. ϕ only affects the configuration as an offset of θ_2. (b) The configuration space obstacle of a point on the x-axis (dotted). As the polar angle, ϕ of the point increases, the configuration space obstacle shifts along θ_2, but does not change shape (solid).

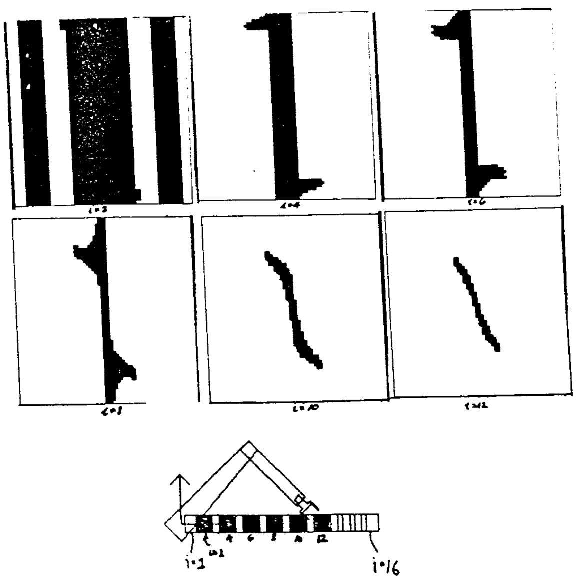

Figure 3: Primitive maps for some values of r for a planar robot with polygonal links.

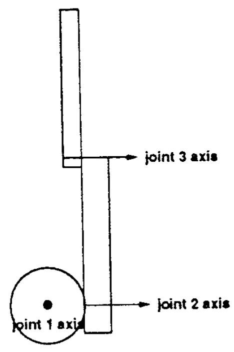

Figure 4: Top view of a Puma-like robot with a shoulder offset.

Given this property, we can characterize the **primitive configuration space maps** for a given manipulator by storing the configuration space generated from point obstacles with $\phi = 0$ and different values of r. Each of these maps, for this simple manipulator, is characterized by a single locus. The primitive maps form a one-parameter family of curves in the configuration space with radial distance r as a family parameter.

The fact that the shape of the configuration space obstacles for an obstacle point depends only on r is not limited to manipulator links modeled as lines; it holds for any link shape. It follows directly from the rotational symmetry of the problem. Therefore, one can build the primitive configuration space maps for each range of r using any existing configuration space algorithm. Since this only needs to be done once for a given robot, there is no particular need for efficiency. Figure 3 shows some primitive configuration space maps for links with more complex shapes. These maps can be used instead of the obstacle maps for line links shown in Figure 2. Similarly, the extension from point obstacles to physical primitives is trivial, provided the primitive obstacle is symmetric around the relevant joint axis of the robot.

3.2 The third dimension

We describe our algorithms for a Puma-like robot, with two arm links operating in a plane which is offset from the shoulder axis by a distance h (see Figure 4). (Actually, they operate in a planar slab—see below.) Let us, for the moment, neglect the actual shape and displacements of the arm links and assume that both links reside in a plane. Then, given a point obstacle at x, y, z (with $\sqrt{x^2+y^2} \geq h$), there are two values of θ_1 for which that point will lie in the plane of the arm. Let $r' = \sqrt{x^2+y^2-h^2}$ and

$$\begin{aligned} \psi &= \operatorname{atan}(y, x) + \operatorname{atan}(\pm r', h) - \frac{\pi}{2} && (3) \\ \phi &= \operatorname{atan}(z, \pm r') && (4) \\ r &= \sqrt{r'^2 + z^2} && (5) \end{aligned}$$

Note that there are two sets of r, ϕ, ψ , one for each of the two roots for r'. These correspond to the two values of θ_1 which cause the plane of the arm links to intersect the point obstacle.

As in the planar case, the primitive configuration space obstacles form a one-parameter family of curves. In fact, it is exactly the same family, since the effect of θ_1 is merely to pick a plane within which the obstacle interaction happens. These equations ignore the shape of of the arm out of the plane. For the Puma, the links are a constant width w perpendicular to their plane of motion. In that case, there are two *ranges* of ψ values over which the effects of a point obstacle are felt. The range can be approximated by treating the obstacle point as a sphere of diameter (at least) w.

If the shape of the link out of the plane of motion were not a constant width, then we would need to use three-dimensional primitive configuration space obstacles. To see this, consider attaching a spherical protrusion to the side of a planar link. Assume an obstacle point on the x axis. Then, depending on the value of θ_1, the link shape is either the shape of the planar link or a circle. Of course, this has drastic effects on the primitive obstacles, requiring the use of three-dimensional primitive maps so as to capture the variations as a function of θ_1.

3.3 Other robot kinematics

The approach described in the preceding sections is not limited to Puma-like robots. Whenever one has intersecting joint axes, one obtains the symmetries that these algorithms exploit.

4 Grippers and wrists

We have limited our attention thus far to the first three links of a robot. In this section we extend our ideas to full six degree-of-freedom robots. We will show that for "wrist-decoupled" robots the applicability of families of primitive obstacle maps extends to the grippers. We will also show how a simple search strategy can make use of fast three degree-of-freedom configuration space computation to implement a full six degree-of-freedom path search.

In the majority of non-redundant linked robots, the first three links serve to position the robot's **wrist**, a point where the last two or three joint axes intersect. It is well known that this type of wrist structure ensures the existence of a closed-form solution to the robot's kinematics. For related reasons, this design also ensures that the computation of the configuration space obstacles for the robot's gripper can be effectively decoupled from the computation of the obstacles for the arm. Therefore, we will assume that we are dealing only with "wrist-decoupled" robots. We can exploit the same kind of rotational symmetry that we have used to construct the configuration space around the base of the robot to characterize the obstacles for the robot gripper (and its payload). Furthermore, this approach is applicable to wrist-decoupled robots independent of the kinematic arrangement of the first three links.

The difficulty of dealing with the gripper is that the symmetry exists about the position of the "wrist," that is, the tip of the arm and not its base. Therefore, we have to perform a computation at each wrist position. This should not be surprising; it simply recognizes the fact that the gripper has six degrees-of-freedom. Fortunately, the computation is the same at each wrist position. Only the obstacles found in the neighborhood of each wrist position differ. This type of computation is, in principle, ideal for SIMD machines such as the Connection Machine. In practice, memory limitations have forced us to pursue a somewhat different approach for the six dimensional case (see Section 4.3). We will illustrate our approach in the planar case, where it is easier to visualize.

4.1 Planar grippers

Consider a simple planar robot gripper. We are interested in determining the legal combinations of x, y, θ values for the gripper. That is, we want to construct a three-dimensional configuration space. Note that we are using values of θ, the orientation of the gripper about the global x axis, and not the values of the last joint angle. This choice is essential to enable us to consider collisions of the gripper independently of collisions of the arm. Our strategy is to characterize the forbidden positions and orientations of the "floating" gripper independent of any arm constraints.

In our implementation, we characterize the range of forbidden wrist angles for a given wrist position, x, y. This process is analogous to that of building the configuration space obstacles for the planar revolute manipulator. Earlier, we noted that rotations of the base coordinate frame of the revolute manipulator did not affect the shape of the configuration space obstacles, but only their position in the configuration space. In the case of a floating planar gripper, neither translations of the wrist nor rotations about the wrist affect the shape of the configuration space obstacles. Only the distance between the point obstacle and the wrist matters. Therefore, the primitive obstacles for the gripper (at a particular wrist position) form a one-parameter family of one-dimensional configuration space obstacles, representing forbidden values of θ and indexed by distance from the wrist position.

4.2 Three-dimensional grippers

The three-dimensional case is analogous to the planar case. The difference is that now we have to deal with three wrist angles and three displacements. We examine the case of a gripper mounted on a spherical wrist. The extension of these remarks to other types of wrist construction is straightforward.

We will use the angles α, β, γ to represent the orientation of the gripper relative to the global cartesian frame. The wrist angle α corresponds to the angle ψ in the spherical coordinate representation. The wrist angle β corresponds to the spherical angle ϕ. The wrist angle γ represents rotation about the axis defined by α and β. Given the $\theta_1, \theta_2, \theta_3$ joint angles that determine a wrist position, these wrist angles can be converted to joint angles for the wrist.

One crucial issue in handling a three-dimensional gripper is that the Euler-type angle specification used for the gripper only has one angle whose origin can be specified arbitrarily, namely α. Therefore, we require a two-parameter family of three-dimensional primitive maps, parameterized by values of β and r.

It is easy to see that the geometry of the system is unchanged when the base coordinate system is rotated by an arbitrary angle. Just as with the first link of the two-link robot, it follows that the shape of the primitive obstacles is independent of α. However, once the axis of rotation for α is chosen, a natural origin for β is defined; the obstacles are not independent of β. The three-dimensional configuration space obstacle corresponding to a point changes in its θ_4 extent as the β coordinate of the point changes.

Compared to the primitive obstacle maps for the arm, the maps for the gripper include an extra dimension of the map and an extra parameter describing the family of maps. Although this would appear to increase the complexity of the problem, it will be seen that the simplicity of the algorithm to compute the configuration space will not be affected. The fundamentals remain identical to those for the configuration space of the arm.

4.3 On-demand computation of obstacle maps

In general, one has to be wary of computing high-dimensional obstacle maps, as they require memory and computation that is exponential in the degrees-of-freedom. A *single* full six degree-of-freedom configuration space bitmap with a sampling resolution of 11 degrees per index would consume a 128 megabytes of storage. Even our Connection Machine, as currently configured, contains only half that amount of memory. Our approach has been to compute obstacle maps only as needed to find a path. In particular, we construct the complete obstacle map for the three arm degrees-of-freedom and compute the maps for the wrist as necessary. We compute the wrist maps for only a single x, y, z position of the wrist at a time, only requiring a three degree-of-freedom map for the orientation of the wrist. The three degree-of-freedom maps consume a more managable 4K bytes each.

Any safe path for the robot must lie in the free space of the configuration space of the arm degrees-of-freedom. We can thus use that configuration space to direct a sequential search for a full six degree-of-freedom path for the arm plus gripper. The search generates a path for the arm within the configuration space of the arm, and for each step of the arm's path, a motion of the wrist is computed which maintains safety from collision due to the gripper. This is easily accomplished by comparing the gripper configuration spaces of neigboring configurations along the path to identify common free space. With the gripper in any configuration in the common free space, the arm motion for that path step will be safe. All that remains is to find a path in the gripper configuration space into the common free space.

The parallel implementation of our approach (see Section 5) is sufficiently fast in computing configuration spaces that we can compute the gripper obstacle maps on demand as the search proceeds. As the search strategy tries each new step along the arm path, it computes the gripper map for the new configuration to compare it with the map for the current configuration. Whenever there is a common free space between the maps and a path for the gripper into the common free space, the search can consider that new step as a possibility. If no common free space or path can be found, that branch of the search reaches a dead-end.

5 Parallel Algorithms

Our parallel configuration space algorithm runs in time that depends only on the resolution desired. It is independent of the number of obstacles. The algorithm is as simple as it can possibly be, but the implementation is slightly tricky because the organization of the data in the SIMD machine must be carefully laid out to take full advantage of the parallelism offered.

The fundamental algorithm is simply this: form a cover of the actual obstacles using primitive obstacles. Begin with a configuration space map initialized with the joint limits of the arm. For each primitive obstacle in the cover, select the primitive map indexed by r—the radial distance of the obstacle, shift the map by ϕ—the polar angle of the obstacle, and add it into the configuration space map being constructed. That's all.

The fundamental algorithm is unchanged whether we're generating the configuration space map for the two-link planar robot, the three-dimensional Puma-like robot or the general 3D gripper. The layout of the data in the Connection Machine is the only difference, as described in this section.

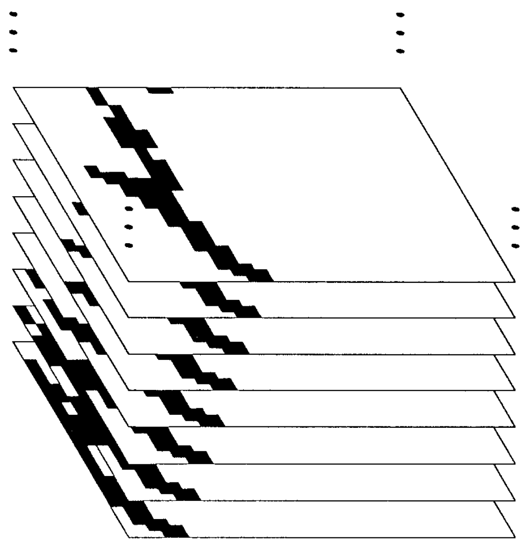
Figure 5: The primitive maps as stored in the Connection Machine. This stack illustrates the storage for a single value of r. Each plane represents a θ_2-θ_3 plane indexed by a particular value of ϕ. (For the resolution shown, there would be 32 of these planes.) The map is shifted according to the ϕ index of the processor. The r-dimension (not shown) controls the shape of the primitive configuration space obstacle.

5.1 The Connection Machine

Our Connection Machine has 8192 physical processors which can be configured on demand to simulate a conceptually unbounded number of virtual processors arranged in an arbitrarily dimensioned grid. (Each physical processor simulates a number of virtual processors.) This capability suggests several natural mappings of parameters of the path planning problem to axes of the virtual Connection Machine. Unfortunately, large numbers of virtual processors (large ratios of virtual-to-physical processors) make unreasonable demands on the available physical memory of the machine, and also increase program running time approximately linearly with the virtual processor ratio. As a result, our actual implementation trades off natural implementation strategy against memory and time.

The Connection Machine offers an interprocessor communication function which our parallel algorithms exploit. This is a **scan** operation, which combines data from all processors along a specified dimension of the processor grid using an associative arithmetic function, such as logical-or. As a special case, it can also copy the value from one processor to all processors along that dimension. This scan operation runs in essentially constant time.

5.2 Data representation

We need to represent three data structures in the Connection Machine: the bitmap representing the cartesian obstacles, the primitive obstacle bitmaps, and the configuration space bitmap. It is natural to associate the primitive obstacle bitmaps and the configuration space bitmap with a two-dimensional processor grid, with each processor representing one bit. It is not immediately obvious how to represent the cartesian obstacles to efficiently exploit the machine.

Since we index the family of primitive maps by r, and shift them by ϕ, where r and ϕ are the polar coordinates of the cartesian obstacles, a natural representation is to form a bitmap with axes r and ϕ, where each bit will be on if there is an obstacle in cartesian space at those coordinates.

The polar-coordinate obstacle bitmap is denoted

$O_p(r,\phi)$. The family of primitive bitmaps is denoted $P[r](\theta_2,\theta_3)$, where θ_2 and θ_3 are quantized along the axes of the bitmaps. The configuration space map is denoted $C(\theta_2,\theta_3)$.

We configure the Connection Machine into a *four*-dimensional, $n \times n \times n \times n$ grid, indexed by θ_2, θ_3, r and ϕ. The reason for this will be seen in the procedure for computing the configuration space, which will consist of one scan along each of the dimensions of the Connection Machine.

To see the layout of the data in the CM, consider a two-dimensional sub-grid indexed by a particular pair of (r,ϕ)—a θ_2-θ_3 plane. (Figure 5.) (Remember that in these algorithms, θ_2 and θ_3 are quantized.) Within this sub-grid, we store the primitive obstacle map, $P[r]$, offset along the θ_2-axis by the value of ϕ used to select this sub-grid. The effect is that we will store in a processor with indices $(\theta_2,\theta_3,r,\phi)$ the primitive obstacle bit $P[r](\theta_2-\phi,\theta_3)$. (The subtraction $\theta_2-\phi$ must, of course, be computed modulo n.) Call this the P-bit. Each primitive obstacle map, $P[r]$ will be represented in the Connection Machine n times, once for each value of ϕ. Since the primitive maps depend only on the robot and the resolution of the quantization, the maps need to be downloaded into the CM only once.

The two-dimensional obstacle bitmap, $O_p(r,\phi)$, is duplicated for all values of θ_2 and θ_3; that is, each processor, $(\theta_2,\theta_3,r,\phi)$, stores the obstacle bit for coordinates (r,ϕ). For each r and ϕ the entire θ_2-θ_3 plane is either 1 or 0, depending on $O_p(r,\phi)$. Call this the O_p-bit.

The final configuration space, $C(\theta_2,\theta_3)$, will be generated in the two-dimensional, θ_2-θ_3 sub-grid selected by $(r=0)\wedge(\phi=0)$.

5.3 Parallel algorithm for the two-link robot

Assuming that the obstacle bitmap has been stored in the Connection Machine, the computation of the configuration space only needs to combine the primitive maps as selected by the obstacle bitmap. The O_p-bit selects the desired (appropriately shifted) primitive maps, and a scan operation with logical-or combination along the r- and ϕ-axes generates the configuration space map.

To store the obstacle bitmap in the Connection Machine in the format we described above, we download the two-dimensional bitmap into a two-dimensional subgrid of the Connection Machine, then copy the bitmap along the remaining two dimensions.

In full, the algorithm can be written as:

```
**procedure** two-link-from-$r\phi$-bitmap-parallel
**begin**
  **download-2d**($O_p$, $r$-axis, $\phi$-axis)
  $O_p \leftarrow$ **scan(copy**, $O_p$, $\theta_2$-axis)
  $O_p \leftarrow$ **scan(copy**, $O_p$, $\theta_3$-axis)
  $C \leftarrow$ **scan(logior**, $P \wedge O_p$, $r$-axis, **toward-zero)**
  $C \leftarrow$ **scan(logior**, $C$, $\phi$-axis, **toward-zero)**
  $C \leftarrow C \vee$ **joint-limit-map**
**end**
```

Once the obstacle bitmap is in the Connection Machine, this algorithm runs in essentially constant time, since only the scan operations are used. Rather than downloading the obstacle bitmap, O_p, it can be computed directly in the Connection Machine in time that is linear in the number of obstacles.

One way to visualize this algorithm is to imagine a two-dimensional $\theta_2 \times \theta_3$ grid. At each grid point is stored a duplicate copy of the two-dimensional $O_p(r,\phi)$ obstacle bitmap. Also at each grid point is stored n^2 bits from the primitive obstacle bitmaps: $P[r](\theta_2-\phi,\theta_3)$, for all values of r and ϕ. At each grid point, the C-bit is turned on if and only if for some r and ϕ the O_p-bit is on and the correct P-bit is on.

Another way to view the algorithm is as a projection from four dimensions to two of the primitive maps which are selected by the obstacle bits.

5.4 The third dimension

Conceptually, extending the two-link, two-dimensional parallel algorithm to one for a three dimensional, Puma-like robot is trivial. As mentioned above, the first joint rotation, θ_1, simply selects which plane the remaining two links operate in, and the algorithm operates within that plane identically to the two-dimensional case. Thus, we need simply add a single dimension to our processor grid and represent our original obstacle bitmap in a kind of spherical coordinate system indexed by (r,ϕ,ψ) triples. This spheric-coordinate obstacle map is denoted by $O_s(r,\phi,\psi)$.[1] The algorithm is identical to the two-dimensional case, with the substitution of O_s for O_p, using a three-dimensional download function, and resulting in a three-dimensional configuration space.

Unfortunately, five dimensions with 32 processors on each axis (a resolution of 11 degrees) requires 32 million virtual processors, or a virtual processor ratio of 4096 : 1 on our machine. This ratio is totally unreasonable in terms of both execution time and memory availability. The algorithm is modified to alleviate this problem by

[1]These are not the familiar spherical coordinates, since the shoulder offset of the Puma robot yields two values for ψ and ϕ for a single point. As with O_p, the O_s bitmap is easily computed directly, since the mapping $(r,\phi,\psi)\rightarrow(x,y,z)$ is one-to-one, and each (r,ϕ,ψ) domain can be easily tested for intersection with obstacles.

combining the bits for each data structure along θ_3-dimension into bit-vectors within each processor. (Note that the only operation along the θ_3-axis is to copy the obstacle bitmap. By collapsing this dimension, this scan becomes unnecessary.) Also, instead of storing all the shifted copies of the primitive maps in the ϕ-dimension, we loop (serially) over all values of ϕ, using the nearest-neighbor communcation facility of the Connection Machine to shift the primitive maps, combining the maps into the configuration space at each step, as indicated by the O_s bitmap. This modification of the algorithm still runs in time that depends only on the resolution, though now it increases linearly with the resolution. At the end, the configuration space is represented by a vector (the θ_2-dimension) of bit-vectors (the θ_3-dimension).

5.5 Parallel algorithm for the three-dimensional gripper

The parallel algorithm for the three-dimensional gripper is as easy to construct as the three-dimensional arm was from the two-dimensional arm. This is because the algorithm is, in essence, identical. The primary difference is simply that, as discussed in Section 4.2, the primitive obstacle maps are three-dimensional, and they are indexed by two parameters, β and r, as well as being shifted by α. Conceptually, this leads to a six-dimensional processor grid, with axes θ_4, θ_5, θ_6, α, β, and r. In the implementation this is reduced to three dimensions by looping over α and compressing the θ_5 and θ_6 dimensions into a two-dimensional bitmap in each processor. Other than the size of the bitmap stored in each processor, the algorithm is identical to the three-dimensional arm algorithm. (For the arm, n bits are stored in a vector; for the gripper n^2 bits are stored in a two-dimensional bitmap.)

Acknowledgments

This work was funded primarily by the Office of Naval Research under University Research Initiative contract N00014-86-K-0685. Additional support was provided by an NSF Presidential Young Investigator Award to Lozano-Pérez.

References

[1] J. Barraquand and J. C. Latombe. A monte-carlo algorithm for path planning with many degrees of freedom. In *1990 International Conference on Robotics and Automation*, pages 1712–1717, Cincinnati, Ohio, 1990.

[2] M. S. Branicky and W. S. Newman. Rapid computation of configuration space obstacles. In *IEEE International Conference on Robotics and Automation*, Cincinatti, 1990.

[3] J. R. Dooley and J. M. McCarthy. Parameterized descriptions of the joint space obstacles for a 5r closed chain robot. In *1990 International Conference on Robotics and Automation*, pages 1536–1541, Cincinnati, Ohio, 1990.

[4] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *IEEE International Conference on Robotics and Automation*, Atlanta, March 1984.

[5] B. Faverjon and P. Tournassoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. In *1987 International Conference on Robotics and Automation*, pages 1152–1159, 1987.

[6] Q. J. Ge and J. M. McCarthy. An algebraic formulation of configuration space obstacles for spatial robots. In *1990 International Conference on Robotics and Automation*, pages 1542–1547, Cincinnati, Ohio, 1990.

[7] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transaction on Computers*, C-32(2):108–120, February 1983.

[8] T. Lozano-Pérez. A simple motion planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation*, RA-3(3):224–238, June 1987.

[9] J. M. McCarthy and R. M. C. Bodduluri. A bibliography on robot kinematcis, workspace analysis, and path planning. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *Robotics Review 1*. MIT Press, 1989.

[10] W. S. Newman. *High-Speed Robot Control in Complex Environments.* PhD thesis, Massachusetts Institute of Technology, Dept. of Mechanical Engr., 1987.

[11] N. Pollard. The grasping problem: towards task-level programming for an articulated hand. Master's thesis, Massachusetts Institute of Technology, Dept. of Elec. Engr. and Computer Science, May 1989.

[12] C. K. Yap. Algorithmic motion planning. In J. T. Schwartz and C. K. Yap, editors, *Advances in Robotics: Volume 1*. Lawrence Erlbaum Associates, 1987.