2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Free Space Mapping and Motion Planning in Configuration Space for Mobile Manipulators

James Ward and Jayantha Katupitiya
ARC Centre of Excellence for Autonomous Systems
School of Mechanical and Manufacturing Engineering
The University of New South Wales, Australia
james.ward@student.unsw.edu.au, j.katupitiya@unsw.edu.au

Abstract— **A new method for calculating the collision free areas of a manipulator's configuration space (C-space) is demonstrated in this paper. Rather than mapping a point obstacle to an n-dimensional surface in the C-space, the dual problem is considered. That is, regions of the C-space that are guaranteed to be collision free are determined. Whilst some configurations that are collision free are not captured by this method, its speed and low memory usage make it ideal for motion planning in the C-space. It is especially useful for mobile manipulators such as wall climbing robots, where the surrounding terrain changes with each step the robot takes. Being able to compute the mapping of Cartesian to C-space representations of objects in real time is essential, as the workspace for these robots is not static and therefore cannot be precalculated. Once the free areas of C-space have been determined, more conventional path planning techniques which have been developed for point robots moving around forbidden areas can be used for the generation of a motion plan. This means that motion planning can be guaranteed to have an exhaustive search through all possible paths and return the lowest cost path - something that has previously been difficult to achieve with other manipulator motion planning techniques. The technique is demonstrated for a 2-degree of freedom planar manipulator, and will be extended in the future to apply to higher dimensional C-spaces. Further developments are suggested for further optimisation of the technique, most notably the idea of partitioning the C-space.**

I. Introduction

Motion planning for robotic manipulators is made difficult by the combination of the facts that the robot changes shape in different configurations, and that all points of the robot's structure must avoid the terrain, not just the end effector. Because of this, traditional planning techniques used for non-deformable robots cannot be readily applied to manipulators.

The configuration space (C-space) of an n-degree of freedom robot represents every combination of actuator positions as a point in an n-dimensional space. If obstacles can be represented in the C-space, then motion planning can be performed in the C-space, which is far simpler because the robot is represented by a single point at any given time. This approach was initially described in [1] and further developed in [2]. A method for 6-degree of freedom manipulators was described by [3] but was not implemented for a real or simulated system, so no information about calculation times are available. The focus of subsequent work was on completely mapping the obstacle into the C-space. [4] presents a geometric method for mapping.

All of the previously mentioned methods do not include information about the computational requirements. However, this work was built upon by others who were able to create practical implementations of C-space obstacle mapping. A method for describing the obstacles within the C-space analytically was developed in [5]. Techniques for bitmap representation of obstacles are demonstrated in [6]–[8].

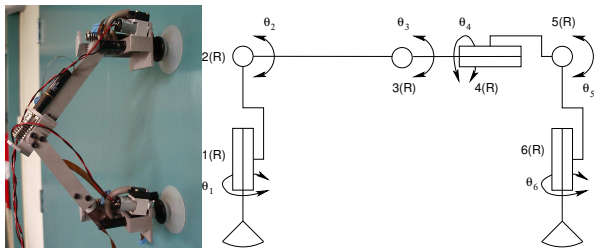
Further developments were made by [9] where they demonstrated a method to reduce the memory requirements of these types of techniques by storing C-space data using octrees. Even then, typical applications of the technique require around 10 hours of processing time and 125Mb of memory.

All of these techniques have long preprocessing times and large memory requirements. This may not be as critical when the manipulator is operating in an essentially static environment. However, more and more manipulators operate in dynamic environments or are themselves mobile. The field of wall climbing robots is a good example of the type of application where the ability to motion plan in real time when operating in a changing environment is key. Fig 1. shows a prototype wall climbing robot being developed by the authors.

In this paper we present an approach to the problem that can be run in real time with little memory overhead. The dual problem of determining guaranteed free areas within the C-space is tackled, rather than trying to map the C-space obstacle directly. This leads to significant advantages for real world applications. It also lends itself to further work about how these freespaces and forbidden areas partition the C-space, making some areas inaccessible and therefore necessary to include in the motion plan search algorithm leading to further time savings. These key differences in approach have led to very promising results.

II. C-Space Obstacles

In order to perform motion planning in the configuration space, rather than in Cartesian space, it is necessary to represent all obstacles in the configuration space. Each point in the Cartesian space will map to a set of points in the



(a) Suction cup testing (b) Kinematic configuration

Fig. 1. Prototype 6-degree of freedom wall climbing robot

C-space. These points become the forbidden areas of the C-space. Only when all obstacles have been mapped can motion planning begin.

Several techniques have been developed to tackle this problem. For 2-degree of freedom planar manipulators, a solution to finding the C-space obstacles is given by [10]. They state that their solution is complete and efficient for low dimension C-spaces, but is unlikely to scale well for higher dimension C-spaces. [5] also describes an analytical solution to the problem. For each point obstacle in the Cartesian space, a set of parametrised equations is developed for each link of the robot. The parameter describes the position along the robot for which a collision with the obstacle is calculated, using the inverse kinematics equations for the robot. To determine whether a point in the C-space represents a collision with the obstacles, each one of these equations must be checked. For an n-degrees of freedom robot moving among m point obstacles, this will require nm checks for each point in the C-space.

The technique described in [9] relies on extensive pre-processing to create a comprehensive lookup table for the relationship between points in C-space and Cartesian space. The Cartesian space is discretised and every configuration which results in some part of the robot intersecting a cell is calculated, and repeated for every cell in the Cartesian space. Once the robot is operating, real obstacles can be located in Cartesian space and the lookup table used to determine C-space values that would result in a collision with this obstacle. This is repeated for all obstacles so that all of the forbidden regions are determined.

A. Computational Considerations

As we are trying to develop a method for online motion planning for use with mobile manipulators, there are some important computational considerations.

The first consideration is storage space. Once the forbidden C-space areas are determined they need to be stored so that the motion planner can access this information and avoid these areas. For an n-degree of freedom robot, the C-space will be n dimensional. Any point in the C-space will generally require an n-tuple to completely specify it. For a 5-degree of freedom robot with a configuration space discretised into 1 degree cells, this will result in $360^5 = 6.0 \times 10^{12}$ cells. If each cell requires 1 byte to specify it this means 5.5Tb of memory would be required to represent the entire C-space. Even if only the obstacles were mapped and they made up 0.1% of the C-space, 5.5Gb of memory would still be needed, and would have to be searched to calculate motion plans. This is clearly infeasible. It should be noted that more efficient storage schemes can be designed (indeed [9] uses octrees for this very reason), but the fact remains that representing obstacles in the C-space in this way is very memory intensive. Storage schemes which can reduce the memory required do so at the expense of increased search time because the computer must decompress the relevant data as it is searching it.

Computation time is the second important issue to consider. Mobile manipulators operate in constantly changing workspaces. This can be because they are moving relative to the obstacles, or because previously unknown obstacles become known as new sensor information arrives. Any C-space mapping technique must be able to build and update maps in real time so that the robot can continue its mission uninterrupted by delays for navigational computations.

Both of these problems are caused because the C-space obstacle is being mapped (and therefore stored) in the discretised C-space representation. As the number of dimensions and fineness of the discretisation of the C-space increase, so too do the computation and storage requirements. The technique presented in Section III avoids this problem.

III. Free Space Mapping

If we map an obstacle from the Cartesian space to the C-space we must make sure that every point in the C-space that would result in a collision is stored in our representation. If we miss a single point, the motion planner may develop a motion plan that results in a collision. Therefore we expend large amounts of computation time and memory to ensure that our mapping is complete.

We can overcome this problem by addressing the dual problem. That is, rather than determining the areas of the C-space that result in a collision with the object, we determine the C-space areas that are guaranteed to be clear of a collision. We call these areas *freespaces* and they are far faster to calculate, store and work with. In doing this our motion planner will lose access to some C-space points which are collision free but not included in the calculated freespaces, but the advantages in time and memory more than make up for this.

A. Method

Freespace mapping relies on identifying boundary conditions for the obstacle in C-space and using those boundary values to establish guaranteed collision free regions (freespaces). The determination of boundary conditions can be specified in any number of ways. Each method will have a certain cost in terms of calculation time and storage memory required, but will generate different amounts of usable freespace for a given obstacle. As more and more freespace boundary conditions are defined, the freespace

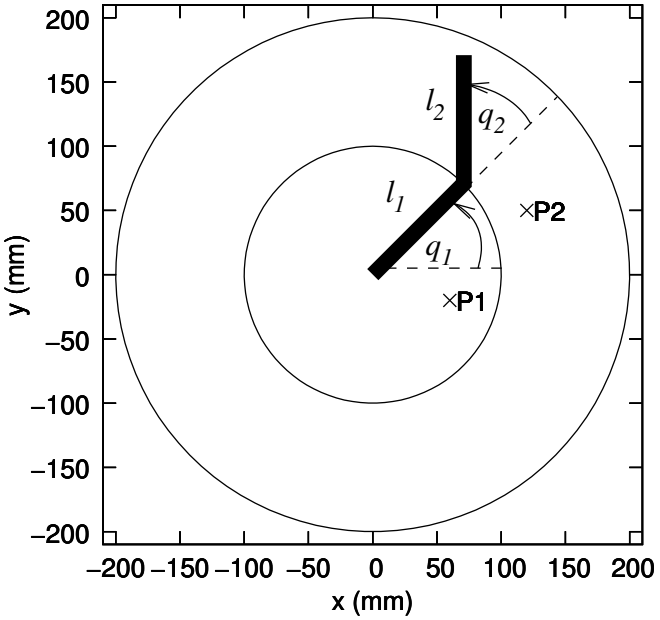


Fig. 2. Planar manipulator and obstacles in Cartesian space

mapping technique approaches the C-space obstacle mapping technique. That is, the entire C-space obstacle is completely defined and the amount of uncaptured freespace tends to zero, at the expense of memory and computation time.

An example is given for a 2-degree of freedom planar manipulator. This was chosen for the ease of demonstration. Obviously further work is needed to extend this to higher degree of freedom manipulators, culminating in a motion planning solution for the 5-degree of freedom wall climbing robot.

1) Individual Obstacles: The first step in freespace mapping is to determine the freespace of each individual point object. The freespace of a point obstacle is generated from the union of freespace primitives. These primitives are n dimensional prisms, where n is the number of degrees of freedom of the manipulator. This means that each freespace primitive is completely defined by specifying the coordinates of two diagonally opposite corners. Furthermore, a point in the C-space can be tested to see whether it lies inside the primitive by checking that each of its ordinates lie between the diagonally opposite vertices of the primitive.

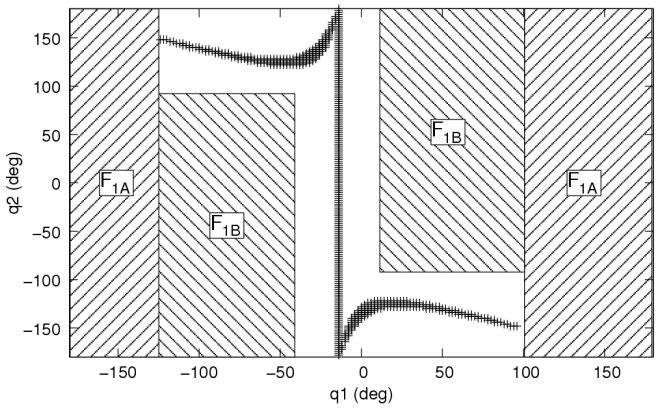
The following notation is used to define an n-dimensional freespace primitive.

$$F = F\{({}^1q_{min}, {}^1q_{max}), ({}^2q_{min}, {}^2q_{max}), ..., ({}^nq_{min}, {}^nq_{max})\} \tag{1}$$

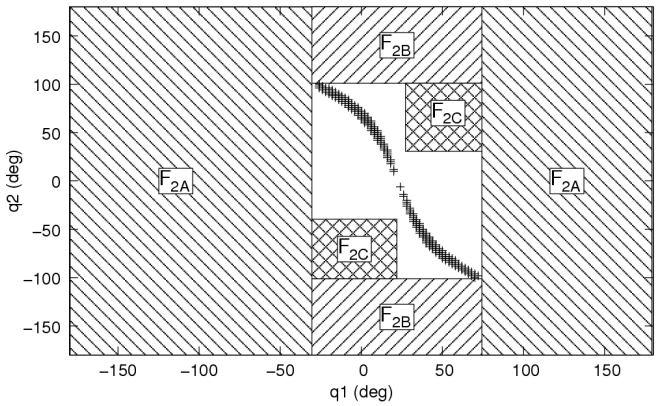
A configuration, $q = \{q_1, q_2, ..., q_n\}$, lies inside the freespace primitive if

$${}^iq_{min} < q_i < {}^iq_{max}, i = 1, 2, ..., n \tag{2}$$

Consider the two points, P_1 and P_2 in Fig. 2. In keeping with the nomenclature introduced by [5] we will refer to



(a) Obstacle P_1



(b) Obstacle P_2

Fig. 3. C-space obstacles and associated freespaces. Freespace primitives must take account of the width of the links, hence there is space between the primitives and the curve representing the centre of each link colliding with the obstacle.

freespaces of obstacles inside the reach of the first link as *type-1 freespaces*, and freespaces of obstacles that can only be reached by the second link as *type-2 freespaces*.

a) Type-1 Freespaces: The obstacle $P_1(x, y)$ lies within the workspace of the first link of the manipulator, so it is possible for both the first and second links of the manipulator to make contact with it.

The distance from the origin to the point P_1 is defined by:

$$r = \sqrt{x^2 + y^2} \tag{3}$$

We determine the value of q_1 that will cause a collision between the obstacle and the first link. This is called the q_1 offset angle, and is referred to as $\bar{q_1}$.

$$\tan(\bar{q_1}) = \frac{y}{x} \tag{4}$$

We then determine the values of q_1 which put the obstacle outside of the reach of the second link. This means that in this range of q_1 values, any value of q_2 will not result in a collision.

$$\hat{q_1} = \cos^{-1}\left(\frac{l_1^2 + r^2 - l_2^2}{2rl_1}\right) \tag{5}$$

The first pair of freespace primitives is therefore given by:

$$F_{1A} = \begin{cases} F\{(-180^\circ + \epsilon, \bar{q_1} - \hat{q_1}), (-180^\circ - \epsilon, 180^\circ + \epsilon)\} \\ F\{(\bar{q_1} + \hat{q_1}, 180^\circ + \epsilon), (-180^\circ - \epsilon, 180^\circ + \epsilon)\} \end{cases} \tag{6}$$

As with all freespace primitives, additional tests must be made to see if the values of $\bar{q_1} \pm \hat{q_1}$ lie outside of the range $[-180, 180]$ and suitable adjustments made if so. We use the value $180^\circ + \epsilon$, where ϵ is some small value, rather than 180° for parameters that have no bounds because the test for whether a configuration lies within a freespace primitive uses the $<$ operator. In these cases a value of 180° does lie within the freespace, and the test performed in (2) needs to use boundary values which reflect this fact.

As can be seen from the C-space obstacle shown in Fig. 3(a), the values of q_2 which cause the second link to collide with the obstacle have two stationary points (ie a maximum and a minimum). Beyond these values of q_2 the configuration is collision free, as long as the first link does not collide with the object. By differentiating the expression for q_2 to find the stationary points and denoting the values at the stationary points as $\hat{q_2}$:

$$\hat{q_2} = \tan^{-1}\left(\frac{r}{\sqrt{l_1^2 - r^2}}\right) \tag{7}$$

At these values we must also take account of the width of the second link:

$$\tilde{q_2} = \tan^{-1}\left(\frac{w_1}{2\sqrt{l_1^2 - r^2}}\right) \tag{8}$$

The offset angle gives the value of q_1 for which the middle of the first link would hit the obstacle. The width of the link must also be taken into consideration. The angle subtended at the first joint by the link at the point of contact is given by:

$$\tilde{q_1} = \tan^{-1}\left(\frac{w_1}{2r}\right) \tag{9}$$

As all values of q_1 within the range $(\bar{q_1} - \tilde{q_1}, \bar{q_1} + \tilde{q_1})$ result in a collision with the obstacle, regardless of the value of q_2. Therefore no freespace primitives can be defined in this region. With the q_2 values found in (7) and (8) the second pair of freespace primitives can be defined:

$$F_{1B} = \begin{cases} F\{(\bar{q_1} - \hat{q_1}, \bar{q_1} - \tilde{q_1}), (-180^\circ - \epsilon, 180^\circ - \hat{q_2} - \tilde{q_2})\} \\ F\{(\bar{q_1} + \tilde{q_1}, \bar{q_1} + \hat{q_1}), (-180^\circ + \hat{q_2} + \tilde{q_2}, 180^\circ + \epsilon)\} \end{cases} \tag{10}$$

The union of these pairs of freespace primitives [(6), (10)] forms the type-1 freespace for obstacle P_1 shown in Fig. 3(a). As noted previously, it would be possible to define more freespace primitives by performing more calculations (and using more memory and processor time), but the two pairs of primitives defined in this section capture the majority of the true freespace and therefore make a good compromise.

b) Type-2 Freespaces: As $P_2(x, y)$ lies outside of the range of the first link of the robot, it will generate a type-2 freespace. As was the case with the type-1 freespace, we first determine the q_1 offset angle $\bar{q_1}$.

$$\tan(\bar{q_1}) = \frac{y}{x}$$

Now we calculate the values of q_1 which put the obstacle outside of the reach of the second link, just as for the type-1 freespace. As before in (5):

$$\hat{q_1} = \cos^{-1}\left(\frac{l_1^2 + r^2 - l_2^2}{2rl_1}\right)$$

This results in a pair of freespace primitives:

$$F_{2A} = \begin{cases} F\{(-180^\circ - \epsilon, \bar{q_1} - \hat{q_1}), (-180^\circ - \epsilon, 180^\circ + \epsilon)\} \\ F\{(\bar{q_1} + \hat{q_1}, 180^\circ + \epsilon), (-180^\circ - \epsilon, 180^\circ + \epsilon)\} \end{cases} \tag{11}$$

Within the range of q_1 values that were not included as part of the previous primitives, it is possible to find more freespace primitives. There is a range of values of q_2 for which the maximum reach of the manipulator is less than the distance from the origin to the obstacle. In this case, the manipulator cannot collide with the obstacle. We will now determine these values.

$$\hat{q_2} = \cos^{-1}\left(\frac{l_1^2 + r^2 - l_2^2}{2l_1l_2}\right) \tag{12}$$

The corresponding freespace primitives are:

$$F_{2B} = \begin{cases} F\{(\bar{q_1} - \hat{q_1}, \bar{q_1} + \hat{q_1}), (-180^\circ - \epsilon, -\hat{q_2})\} \\ F\{(\bar{q_1} - \hat{q_1}, \bar{q_1} + \hat{q_1}), (\hat{q_2}, 180^\circ + \epsilon)\} \end{cases} \tag{13}$$

These two pairs of primitives leave a large square area in the center of the C-space as being unavailable. It is clear from the C-space obstacle curve that some of this area is available. We can add two more freespace primitives which represent the cases when the second joint is bending away from the obstacle, ie $q_2 > 0, q_1 > \bar{q_1}$ and $q_2 < 0, q_1 < \bar{q_1}$. Because the links of a real manipulator have width, this must also be taken into account. The angle subtended by the link at the second joint is given by:

$$\tilde{q_2} = \cos^{-1}\left(\frac{w_1}{2(r - l_1)}\right) \tag{14}$$

This final pair of freespaces is defined by:

$$F_{2C} = \begin{cases} F\{(\bar{q_1} - \hat{q_1}, \bar{q_1}), (-\hat{q_2}, -\tilde{q_2})\} \\ F\{(q_1, \bar{q_1} + \hat{q_1}), (\tilde{q_2}, \hat{q_2})\} \end{cases} \tag{15}$$

The union of these three pairs of freespace primitives [(11), (13), (15)] forms the type-2 freespace and is shown for obstacle P_2 in Fig. 3(b).

2) Multiple Obstacles: Once the freespaces for the individual obstacles has been determined, it is necessary to combine them to form the complete freespace for the robot's environment.

In order to intersect two freespace primitives F_1 and F_2 to form $F' = F_1 \cap F_2$:

$$\begin{aligned} F_1 \cap F_2 = F \{ & \big(\max(F_1 : {}^1q_{min}, F_2 : {}^1q_{min}), \\ & \min(F_1 : {}^1q_{max}, F_2 : {}^1q_{max})\big), \\ & ..., \\ & (\max(F_1 : {}^nq_{min}, F_2 : {}^nq_{min}), \\ & \min(F_1 : {}^nq_{max}, F_2 : {}^nq_{max}))\} \end{aligned} \tag{16}$$

If any pair of limits in F' is such that ${}^nq_{min} \geq {}^nq_{max}$ we discard the new primitive as there is no intersection.

The process for finding the intersection of two freespaces is to iterate over all of the freespace primitives in the first freespace and intersect them with all of the freespace primitives in the second freespace. If the freespace $S_1 = \cup_{i=1}^{n} {}^1F_i$ and $S_2 = \cup_{j=1}^{m} {}^2F_j$, then the freespace representing the intersection $S' = S_1 \cap S_2$ is defined as:

$$S_1 \cap S_2 = \cup_{i=1}^{n} \cup_{j=1}^{m} ({}^1F_i \cap {}^2F_j) \tag{17}$$

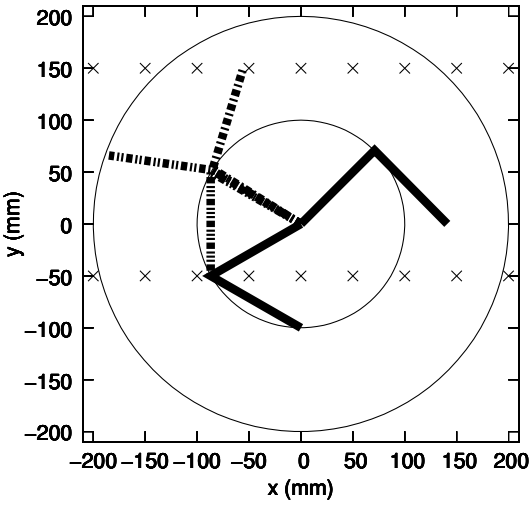
When more than two obstacles are involved, each freespace is successively intersected with the previous intersection. That is,

$$\begin{aligned} S' = & S_1 \cap S_2 \cap S_3 \cap ... \cap S_n \\ = & ((((S_1 \cap S_2) \cap S_3) \cap ...) \cap S_n) \end{aligned} \tag{18}$$

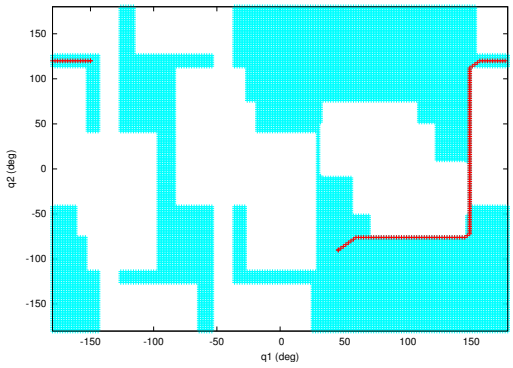
Fig. 4. shows the complete freespace for a 2-degree of freedom planar manipulator moving among several point obstacles. The freespace comprises 94 freespace primitives to represent the effect of the 18 point obstacles. As each primitive can be represented by pairs of floating point numbers, this is a memory use of 1.5kB assuming 4B for each float. Generating the freespace primitives took 19ms on an Athlon64 3500+.

IV. Motion Planning

Once the freespace for the surrounding terrain has been calculated a wide variety of techniques can be used to generate the motion plan. Because this motion planning is occurring in the configuration space and the configuration of the robot at any time is represented by a single point in this space, many of the path planning techniques that have been developed for mobile robots can be used. The motion plan shown in Fig 4(b) was generated in approximately 1 second by using the A* search algorithm [11], using a cost function that minimises total actuator movement. Different cost functions will result in different shaped paths through the C-space. The ability to use these well studied algorithms is one of the great attractions of C-space obstacle mapping.



(a) Robot and obstacles. The dashed lines are the intermediate configurations between the solid initial and goal positions.



(b) Freespace and motion plan

Fig. 4. Entire freespace

Such algorithms are guaranteed to find the lowest cost motion plan if one exists. The cost function can be defined as desired - for a wall climbing robot it might select be the lowest total actuator movement, lowest time to execute the plan or the plan that has the lowest moment about the attached foot in order to decrease the likelihood of the robot detaching from the wall. Cost functions like these cannot be incorporated into the more prevalent probablistic motion planners such as rapidly-exploring random trees [12], random walks [13] and probablistic roadmap planners [14]. These planners must also perform all of their collision checking in the Cartesian space, which is computationally burdensome. Efficient techniques have been developed to do this [15], but they still cannot compete with motion planning in the C-space. Their advantage is that they require no preprocessing and can therefore be used in complex and dynamic environments, where previous C-space mapping schemes could not.

It should be noted that previous attempts have been extremely memory intensive because they have to discretise the C-space with the resolution that is required by the search algorithm to be run. Freespace mapping neatly avoids this

situation because it defines free regions of C-space. The discretisation of the C-space only occurs when the algorithm is running, and cells of C-space are only allocated memory as they are needed. In this way we are making use of the main strength of path planners such as A*, namely that they do not exhaustively search the entire space to find the lowest cost path. As much of the C-space does not need to be searched, it does not need to have memory allocated to it and entire process is very processor and memory efficient.

There is great scope for varying the search algorithms used after freespace mapping is complete, because the mapping process and calculated freespace are entirely independent of the technique used to find the motion plan within the freespace. It would be possible to run the search algorithm several times at different resolutions in order to determine if a viable motion plan exists and then to refine it as the robot is spending time executing the initial coarse plan. This type of hierarchical search is described in [16], but could not be applied to manipulator arms until now.

If freespace mapping is to be used in an application such as a wall climbing robot where the sensors are continually updating the terrain data (ie adding more obstacles to the map and requiring changes to the calculated freespace), a search algorithm such as D* [17] which is designed for dynamic environments would be more appropriate.

The initial configuration can also be used to make the motion planning process more efficient. A given freespace is composed of many freespace primitives. To find a path from inside one primitive to another, the two primitives must overlap or be joined by a series of overlapping primitives. Any primitives that are not joined to the primitive containing the initial configuration in this way cannot be reached from the initial configuration. Accordingly, these primitives do not need to be in the list of primitives to be tested when a new neighboring node is generated and tested to see whether it is inside one of the freespace primitives. Reducing the number of primitives to test against will reduce the computation time required for the motion planning process. This process of *partitioning* the freespace will become more useful as the number of dimensions of the C-space increases, and requires further investigation.

V. Conclusion

Freespace mapping provides a very fast and effective first step for manipulator motion planning within the configuration space. In the future it will applied to higher degrees of freedom. As the degrees of freedom increase there will also be a need to optimise the technique to provide the preferred trade off between processing speed and completeness of freespace determination. Freespace mapping provides a step forward in terms of the processing time and memory required to represent obstacles within the robot's C-space, but further work is required to generalise to higher dimensional problems.

There are further optimisations to this technique that could be investigated, including freespace partitioning and different methods for search algorithm application to the freespace. The most efficient method for representing non-point obstacles should also be investigated, as it may not be necessary to map every point on the obstacle if certain key points can be used to map the entire freespace of the obstacle.

Acknowledgements

This work is supported in part by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Government.

References

[1] T. Lozano-Perez, "Spatial planning: A configuration space approach," *Computers, IEEE Transactions on*, vol. C-32, no. 2, pp. 108–120, 1983.

[2] T. Lozano-Perez, "A simple motion-planning algorithm for general robot manipulators," *Robotics and Automation, IEEE Journal of [legacy, pre - 1988]*, vol. 3, no. 3, pp. 224–238, 1987.

[3] B. R. Donald, "A search algorithm for motion planning with six degrees of freedom," *Artificial Intelligence*, vol. 31, pp. 295–353, Mar. 1987.

[4] R. Brost, "Computing metric and topological properties of configuration-space obstacles," in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, vol. 1, pp. 170–176, 1989.

[5] C. Zhao, M. Farooq, and M. Bayoumi, "Analytical solution for configuration space obstacle computation and representation," in *Industrial Electronics, Control, and Instrumentation, 1995., Proceedings of the 1995 IEEE IECON 21st International Conference on*, vol. 2, pp. 1278–1283, 1995.

[6] L. Kavraki, "Computation of configuration-space obstacles using the fast fouriertransform," *Robotics and Automation, IEEE Transactions on*, vol. 11, no. 3, pp. 408–413, 1995.

[7] B. Curto, V. Moreno, and F. Blanco, "A general method for c-space evaluation and its application toarticulated robots," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 1, pp. 24–31, 2002.

[8] B. Curto and V. Moreno, "Mathematical formalism for the fast evaluation of the configuration," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pp. 194–199, 1997.

[9] X. Wu, Q. Li, and K. Heng, "A new algorithm for construction of discretized configuration space obstacle and collision detection of manipulators," in *Advanced Robotics, 2005. ICAR '05. Proceedings., 12th International Conference on*, pp. 90–95, 2005.

[10] A. Maciejewski and J. Fox, "Path planning and the topology of configuration space," *Robotics and Automation, IEEE Transactions on*, vol. 9, no. 4, pp. 444–456, 1993.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[12] J. Kuffner, J.J. and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 2, pp. 995–1001, 2000.

[13] S. Carpin and G. Pillonetto, "Robot motion planning using adaptive random walks," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, pp. 3809–3814, 2003.

[14] R. Geraerts and M. H. Overmars, "Sampling and node adding in probabilistic roadmap planners," *Robotics and Autonomous Systems*, vol. 54, pp. 165–173, Feb. 2006.

[15] F. Schwarzer, M. Saha, and J.-C. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, vol. 21, no. 3, pp. 338–353, 2005.

[16] A. Autere, "Hierarchical A* based path planning – a case study," *Knowledge-Based Systems*, vol. 15, pp. 53–66, Jan. 2002.

[17] A. Stentz, "The focussed D* algorithm for real-time replanning," in *Artificial Intelligence, 1995. Proceedings., 1995 International Joint Conference on*, pp. 1652–1659, August 1995.