Fast Transformations to Provide Simple Geometric Models of Moving Objects

David Adrian Sanders[1(✉)], Giles Tewkesbury[1], and Alexander Gegov[2]

[1] School of Engineering, University of Portsmouth, Portsmouth, UK
david.sanders@port.ac.uk
[2] School of Computing, University of Portsmouth, Portsmouth, UK

Abstract. Models are compared for use with a sensor system working in real time (*in this case a simple image processing system*). A static robot work-cell is modelled as several solid polyhedra. This model is updated as new objects enter or leave the work-place. Similar 2-D slices in joint space, and spheres and simple polhedra are used to model these objects. The three models are compared for their ability to be updated with new information and for the efficiency of the whole system in accessing data concerning new objects. The system supplies data to a "Path Planner" containing a geometric model of the static environment and a robot. The robot structure is modelled as connected cylinders and spheres and the range of motion is quantised.

Keywords: Manufacturing · Navigation obstacles · Robot · Model

1 Introduction

Industrial processes are being improved to meet the requirements of lean and agile manufacturing. Navigation in these dynamic industrial environments is challenging, especially when the motion of the obstacles populating the environment is unknown beforehand and is updated at runtime. Although computers are getting faster and faster, real time applications still require efficient modelling and programming techniques. Some traditional motion planning approaches can be relatively slow when applied in real-time, whereas reactive navigation methods often have too short a look-ahead horizon. This paper presents simple but fast transformations to improve existing manufacturing processes by providing simple geometric models of objects moving through the workspaces of industrial robots. This can improve path and trajectory planning in real time.

A complex industrial environment consists of moving machinery, objects to be manipulated and worked, and obstacles to be avoided [1,2,3,4,5]. Free space available to the moving machinery depends on the accuracy of the models used for this changing environment. Robot navigation in dynamic environments is a challenging task [6], especially when the motion of obstacles is unknown beforehand. Traditional motion planning approaches can be too slow to be applied in real-time, whereas reactive navigation methods have generally a too short look-ahead horizon. Van-der-Stappen [7] presented an efficient paradigm for computing the exact solution of the motion planning

H. Liu et al. (Eds.): ICIRA 2015, Part I, LNAI 9244, pp. 604–615, 2015.
DOI: 10.1007/978-3-319-22879-2_55

problem with very few obstacles but he accepted that motion planning algorithms often use long worst-case running times [8]. These running times are why exact algorithms are rarely used in real-time. Processing speed depends on the complexity of the problem, the complexity of the models used and system processing speed. Process speed is increasing but problems are also becoming more complex.

Assumptions about size and distribution of obstacles leads to a significant reduction in complexity. The complexity of the free space is known to be linear in the number of obstacles and De-Berg [9] studied the complexity of the motion planning problem for a bounded-reach robot with few obstacles and Tang [10] investigated how to topologically and geometrically characterize the intersection relations between movable polygon models often used for manufacturing environments. Large presented real-time motion planning approaches based on the concept of the Non-Linear Vobst (NLVO) [11]. Given a predicted environment, velocities which lead to collisions with static and moving obstacles, were modeled.

The complexity of motion planning algorithms depends on the complexity of the models used for moving obstacles to provide the set of collision-free placements left to the robot. Complexity can be high, resulting in relatively long computing times. Reducing complexity reduces the running time of motion-planning-algorithms.

In this paper, models in two different spaces are considered and they are tested with a simple image processing system working in real-time. A static geometric model of a robot work-cell is held in computer memory as solid polyhedra. This static model is updated as new objects enter or leave the work-place. 2-D slices of joint space, spheres and simple polyhedra are used to model these objects. Spheres are suggested as the simplest models of dynamic obstacles. Halperin [12] devised techniques to manipulate a collection of loosely connected spheres in three-dimensional space. He analyzed sphere models and pointed to properties that make them easy to manipulate. He presented efficient algorithms for computing union boundaries.

The mapping from workspace to configuration space (or joint space) is important and then the avoidance of the obstacles in one or other space is then important. Sanders [1],[3,4,5] completed studies of geometric modelling techniques and regarded the following as meaningful criteria in depicting a robot and its work place: Fast intersection calculations, ease of use with path planning algorithms, fast model generation, low memory storage requirements, and efficiency. Sharma [13] for example recently studied minimum time needed to transfer vehicles from source to destination, avoiding conflicts with other vehicles. The other vehicles were effectively obstacles and a conflict occurred when the distance between any two vehicles was smaller than a velocity-dependent safety distance. Fiorini [14] presented a method for robot motion planning in dynamic environments in a velocity space. The models considered in this paper are compared for their ability to be updated with new information to provide data to planning systems like that. Efficiency in accessing data concerning new objects is also considered.

The robot machinery structure is modelled as connected cylinders and spheres and the range of motion is quantized. A fast sub-optimal path is to be derived using simplified models that avoids the modelled objects and seeks a direct path in terms of total actuator movement. The approach depends on inspecting a 3-D graph of quantised joint space.

The static model of the robot work-cell consists of solid polyhedra. The remaining free space model is updated as new objects penetrate or depart from the working volume. 2-D slices in joint space, multiple spheres, and six sided parallelepiped are used to model the dynamic objects in order to compare them.

2 Robot and Static Environment

Most computer representations of factory surroundings have flat surfaces and straight linear edges and this geometry resembles the objects often found in manufacturing work cells. These models are difficult to deal with in real-time. If both robot and dynamic objects are modelled by polyhedral shapes then the accuracy may be high but computation time is extended. The transformation of the static environment need only be made once though, so that computation time is not a problem. An accurate model was therefore selected and Polyhedra were used to model the static environment. The most influential factor in representing the robot was speed of intersection calculation (providing the model enclosed the whole robot). A large number of industrial robots have two major links, (an upper arm and a forearm) and three major joints (Base, Shoulder and Elbow). The simplest possible representation for this type of robot was two lines jointed at one end. Fixed distances from the lines were then defined as enclosing the outer casing of the robot. This gave two connected cylinders with hemispherical ends. The advantages of this representation were that the cylinders modelled the robot links efficiently and the intersection calculations between the robot arm and obstacles were simple. The end effector was then represented as a sphere with a radius adequate to surround the end effector. Work-pieces were included by increasing the radius of the sphere.

2.1 Dynamic Mapping

Speed of intersection calculation was compared for several models representing dynamic objects, that is objects that moved in the working volume of the robot. Three models compared favourably: 2-D slices in joint space, spheres and six sided parallelepiped in cartesian space. In all cases it was assumed that at least the 2-D cross section of the dynamic object in the X-Y plane and the height (Z) of the dynamic object was available from a sensor system (a simple vision system in this case). The dynamic objects were effectively only two and a half dimensional. That is, they had a two dimensional shape and a height. 3-D dynamic object shapes considered during the work described in this paper were cylinders and cubes. Parallelepiped models, spheres or similar 2-D planar slices in joint space modelled these 3-D shapes to a workable accuracy and in the case of the 2-D slices, more quickly in discretised 3-D space.

2-D slices are described here. Models were calculated by considering two pairs of boundaries: the angles of the base joint, $\theta 1$, which bounded the dynamic object ($\Theta 1min$ and $\Theta 1max$); and the maximum distance Dmax and minimum distance Dmin from the origin (maximum and minimum radii). The dynamic object was then modelled as a series of 2-D planar slices. The reference slice was calculated within a

boundary of a line from the Origin bounded by Dmax and Dmin and the limits of the Z axis. The "blocked" configurations for the shoulder and elbow joints θ2 and θ3 were then calculated for this bounded plane and copied for all θ1 within the two bounding angles, Θ1min and Θ1max. For the global path planning methods described in the literature, this reduced the number of searches and tests for "blocked" points. The major part of the algorithm was reduced to copying values within a 3-D graph. The dynamic object was first modelled as a 2-D rectangle as this was the simplest model which could be derived from the row and column limits of an object under a camera.

3 Transformation into Joint Space

Data were processed to transform dynamic objects into a joint configuration space. A point in cartesian space is not transformed into a point in joint space. If the point is within the working volume of the robot then it is transformed into one or more complex three dimensional shapes. These complicated profiles may be depicted within a computer as geometric shapes, units of space or by approximating the profiles by mathematical curves. The method selected in this work represented the dynamic objects as regions within joint space consisting of small units. The technique was not limited to any specific design of machinery and may be used with any number of degrees of freedom. The work described here was based on the implementation for the three major axes of a KUKA KR125 robot at Ford Motor Company. A graph was created which consisted of a three dimensional structure of unit regions. The 3-D graph had each dimension corresponding to a principal degree of freedom of the robot arm, Θ1, Θ2 and Θ3. The wrist configurations were not considered but these were included as being within a sphere. Each unit was initially set to `"clear"' status and the positions (in joint space) at which the robot intersected dynamic objects were then calculated. Each unit represented a range of configurations for the robot, in terms of, (Θ1cent, Θ2cent, Θ3cent), plus a degree of movement away from these central joint values. All units together represented the whole robot work-space and the number of units in the graph, NodeTotal, was given by:

$$\{(\Theta 1_{max}-\Theta 1_{min})/2 \text{ x } \delta\Theta 1\}*\{(\Theta 2_{max}-\Theta 2_{min})/2 \text{ x } \delta\Theta 2\}*\{(\Theta 3_{max}-\Theta 3_{min})/2*\delta\Theta 3\} \quad (1)$$

where, Θ1max, Θ1min = upper, lower limits of Θ1.
Θ2max, Θ2min = upper, lower limits of Θ2
Θ3max, Θ3min = upper, lower limits of Θ3.

If at any configuration in a unit, the robot intersected a dynamic object, then the unit was set to "blocked". If at all configurations within a unit the robot did not intersect a dynamic object then the unit remained "clear". The path planning problem for the global approach was then reduced to finding a series of neighbouring units between the START and GOAL configurations that were still "clear". If free space is assumed to be larger than blocked space then a fast method was to consider each

dynamic object and test for the nodes which could contain the transformed dynamic object. This was the method adopted and the algorithm was as follows:

For a node where the robot could intersect the dynamic object, recursively test all the neighbouring units to see if they are also within the reach of the robot.

Data structures were initialised to form a 3-D graph of joint space and trigonometric solutions were calculated. All units in the graph were set to `"clear"' status and flags were associated with each node. The static model of the work cell left a number of clear nodes that represented safe configurations that would not collide with the static environment. Dynamic object data was simulated or received from the vision system and the first task for the program was to read this data. The two and a half dimensional model was then created.

3.1 2-D Slices

Firstly the limits in x were increased by the radius of the upper-arm:

$$\text{StartRow\_clearance} = \text{StartRow} - \text{UpperRad\%} \tag{2}$$

$$\text{EnddRow\_clearance} = \text{EndRow} + \text{UpperRad\%} \tag{3}$$

The modulus of the ends and centre point on an edge StartCol were calculated. This is shown below for the furthest end from the Origin.

$$\text{Corner(TopLeft, Angle\%)} = \text{InvTan (StartCol / EndRow\_clearance)} \tag{4}$$

$$\text{Corner(TopLeft, Modulus\%)} = \sqrt{(\text{EndRow}^2 + \text{StartCol}^2)} \tag{5}$$

The following parameters of the model were found: the inside radius from the origin, (Dmin), outside radius from the origin, (Dmax), smallest base angle, (Θ1min), and largest base angle, (Θ1max).

If the dynamic object was matched to a template then the height of the dynamic object was extracted from the template, otherwise if the dynamic object height was unknown, the height (Z) was set to infinity. The segment was extrapolated to the Y axes so that calculation took place in the Y, Z plane. The modelled dynamic object was expanded by the radius of the robot's upper-arm in the Y and Z plane. Θ1 was set to its new lower limit and the inverse kinematic solution was found for all the points within the dynamic object, as shown in the following code:

```
FOR Yaxis = (Radius%(min%)UpperRad%) TO (Radius%(max%) +
UpperRad%)
  FOR Zaxis =  255 TO (Radius%(Z%) + UpperRad%)
       CALL InvKinematics
  NEXT Zaxis
NEXT Yaxis
```

The coordinates in Y and Z were converted to robotic joint angles using the inverse kinematic solution in the subroutine InvKinematics. Firstly the distance from the origin to the cartesian point (L3) and the angle to the point (CurvΘ) were calculated:

$$\text{CurvΘ} = \text{InvTan Zaxis / Yaxis} \tag{6}$$

$$\text{sqL3} = \text{Yaxis}^2 + \text{Zaxis}^2 \tag{7}$$

$$\text{L3} = \sqrt{\text{sqL3}} \tag{8}$$

The upper-arm was checked against L3 to see if a collision was possible. If within the reach of the upper-arm and if Θ2 was within its limit, then Θ2 was set to CurvΘ and Θ3 was set to "blocked" between its limits. If L3 was less than the Forearm plus upper-arm then the Forearm collided with the point. Θ2 and Θ3 were calculated using the cosine rule and if Θ2 and Θ3 were within their limits a flag was set to "blocked".

The method is being successfully used with sensors [15,16] mobile robots [17,18,19,20,21] and wheelchairs [22,23].

3.2 Spheres

The graph data structure described earlier was initialised. Limits of the graph corresponded to the angular limits for the robot's joints within the range of the work cell and obstacles outside this work-space were ignored. As the graph carried out intersection checks at a limited number of positions, only a limited number of trigonometric solutions were required and these were calculated at the start. Before the obstacles were calculated all the units in the graph had a flag set to `CLEAR' status. Four other flags were used with each node, these were: `New obstacle', `Forearm tested', `Upper arm tested', and `On list'. Each unit code was stored as one byte of computer memory in an array and the flags used one bit each. The obstacle data was received from a file or from the vision system and the first task for the program was to read this data.

The task was then split into two sub-tasks, firstly to calculate the upper arm and then to calculate the forearm blocked space on the graph. A configuration was calculated at which the part of the arm under consideration was closest to the obstacle centre. If the forearm was being considered, then the configuration where the Foretip was at the centre of the sphere was calculated. For the upper arm, the configuration was calculated for which the centre line of the upper arm pointed at the sphere centre. If the obstacle was within the reach of the link being tested, then this configuration was the first unit for the transformed obstacle. The base angle was calculated from the X,Y coordinates of the sphere. Firstly the modulus (L3) and the angle (SphΘ) from the robot to the centre of the sphere was calculated and a test was conducted to see if the sphere was out of range, in which case no further processing was necessary.

$$\text{Waist Angle } \Theta 1 = \text{InvTan}(Y/X) \tag{9}$$

$$\text{Modulus } XY = \sqrt{(X^2+Y^2)} \tag{10}$$

$$\text{SphΘ} = \text{InvTan}(Z/\text{ModulusXY}) \tag{11}$$

$$L3 = \sqrt{(X^2 + Y^2 + Z^2)} \tag{12}$$

The cosine rule was used to calculate the shoulder Θ2 and elbow Θ3 angles.
L1 = Upper-Arm = 220mm
L2 = ForeArm = 160mm

$$\Theta 3 = \text{InvCos}\left[(L1^2 + L2^2 - L3^2) / (2 * L1 * L2) \right] \tag{13}$$

$$\Theta 2 = \text{InvCos}\left[(L1^2 + L3^2 - L2^2) / (2 * L1 * L3) \right] + \text{Sph}\Theta \tag{14}$$

If the sphere centre was too close to the robot then Θ3 would exceed its lower limit (Θ3 < 90°). In this case Θ3 was set to 90° and Θ2 was calculated using InvTan:

```
If Θ3 < 90° THEN
    Θ3 = 90°
    Θ2 = InvTan ( L2 / L1) + SphΘ
END
```

This gave a starting configuration close to the centre. When the lower limit of Θ2 was exceeded, (Θ2 < -30°), the angle was set to minus 30° and the distance between the upper-arm and sphere centre was calculated (the modulus) using the subroutine FindModulus, from which the cosine could be used to find the new Θ3:

```
If -30° < Θ2 THEN
  Θ2 = -30°
  Θ3 = InvCos[(L1² + L2² - Modulus²) / (2* L1 * Modulus)]
END
```

The first configuration was set to blocked . Its neighbouring units were also tested and if they were set to blocked then their neighbours were checked. The position problem was solved using forward kinematic calculations and the minimum distance between the obstacle and the robot arm was calculated, (provided that it had not completed the calculation before). The method continued recursively until the whole obstacle transformation was found. All units were set to blocked, which had any two opposite neighbouring units which were also blocked. Any units which were on the edge of the now solid obstacle were recorded on a list. All the neighbours of the units on the list were tested, and the process repeated until the surface of the transformed sphere was completely defined.

Nodes which were blocked were stored on a list of units to be expanded later. When a unit was expanded it was retrieved from the list and new blocked points were added to the list. When all the nodes on the list were exhausted the obstacle transformation was complete. The most important consideration was processing speed. Times for calculating obstacles were recorded during the project and examples are presented in the results section.

3.3 Simple Polyhedral Shapes

Polyhedra are commonly used to model dynamic objects. The third modelling method described in this paper modelled the dynamic objects as simple six sided parallelepiped. This was the most elementary polyhedral model. The system found the position of the edges of the model in X and Y by calculating the limits of the rows and columns set by the vision program. If possible, the height of the object was then retrieved from an associated template. Edge positions were expanded with the model radius of the part of the robot under test (ie upper-arm or forearm), as demonstrated below for an expansion of the forearm in X.

$$\text{Expand\_XLow\%} = \text{EdgePosition\%(LowX\%)} - \text{ForRad} \tag{15}$$

$$\text{Expand\_XHigh\%} = \text{EdgePosition\%(HighX\%)} + \text{ForRad} \tag{16}$$

The cartesian coordinates of the arm were then tested against the expanded polyhedral edge limits.

4 Results

The most important consideration for the system was that it should be suitable for real time applications. Times for transforming dynamic objects were recorded during the project and as an example, the times for the three models to transform a large cube into joint space are shown in Table 1. The times were recorded with the Z axis of the cylinder at X = 0 mm and Y = 300 mm with respect to the origin.

Table 1. Transformation times for a cube.

Model	Time (Seconds)	Number of blocked nodes recorded.
One Sphere	8.8	2426
Two Spheres	14.1	2336
Simple Polyhedron	26.3	1983
2-D Slices	5.3	2489

The 2-D Slice Model: The advantage of modelling the dynamic object as a series of similar 2-D slices was that once the collision coordinates of Θ2 and Θ3 had been calculated for a particular Θ1 then these collisions could be repeated for the limits of Θ1 which collided with the dynamic object. This reduced the main processing task to copying data rather than calculating forward or reverse kinematic solutions. The representation of dynamic objects using similar 2-D slices was the fastest to transform into discrete 3-D joint configuration space.

Once a dynamic object increased above a certain size or was moved closer to the origin, part of the dynamic object intersected both the Upper Arm and ForeArm joint space. Thus the joint-space occupied by the dynamic object suddenly increased and

calculation time increased. For the transformation methods a graph of calculation time vs discrete work-space volume can be expected to be linear, that is the calculation time for a dynamic object was approximately proportional to the number of units tested, the total number of nodes being the work-space volume.

The computer time required for dynamic object transformations was short. The initial conversion time to model the static environment was slow; Up to three minutes of computer time depending on the complexity of the model, but the transformation was only performed when the system was powered up.

The Sphere Model: An obstacle was modelled first as a single sphere of the smallest radius which would enclose the obstacle. Later, if time allowed it was modelled by two smaller spheres and then four spheres. Nodes set to blocked associated with the first sphere tested usually also collided with other spheres. The forward kinematic solutions did not need to be recalculated for these nodes but the total calculation time increased with the number of spheres because the overhead of calculation for each sphere was greater than the saving in time achieved as the spheres became smaller. This meant the single sphere calculation was faster than the calculations for multiple spheres although the single sphere model was less accurate and had a larger volume. The problem when using more than one sphere was that the centre of several spheres would be set to blocked (with some surrounding nodes) after the expansion of the first sphere. As these nodes were blocked, later spheres were sometimes not retested so that many nodes were not added to the list.

5 Discussion

Transforming a geometric algorithm into an effective computer program is a difficult task. The accuracy of the models affected the performance of the Path Planner. High accuracy models required more computation time and therefore longer solution times. Low accuracy models required links or dynamic objects to be oversized to eliminate the chance of undetected collisions. Lowering the accuracy led to the rejection of valid solutions.

For dynamic models, speed of calculation was important. The simplest possible intersection calculations for the local methods were made using the sphere model. Calculation was reduced to finding the distance from the robot to a point and subtracting the radius of the sphere to give the distance to the surface of the sphere. Modelling with more than one sphere was considered. As the real environment for a robot becomes more complex so more spheres are needed for the model. It was considered how increasing the number of spheres might increase the accuracy of the model. A cubic number of spheres was used, i.e. 1, 8, 27, 64 etc. The spheres formed a regular pattern and were equal in size. An infinite number of spheres was required to model the cube completely but modelling objects using the same sized spheres was inefficient. For example, in modelling a cube using sixty-four spheres of the same size, eight of the spheres are totally enclosed and might easily be replaced by a single larger sphere without increasing the model volume.

To compare the modelling of obstacles using single and multiple spheres, as an example, a model of a cylinder using one and two spheres is compared. The volume of two spheres of radii 35 mm was compared to that of one sphere of 70 mm as shown below.

Volume of Two Spheres: 2 x 4/3 x π x 353 = 359,188 mm^3

Volume of One Sphere: 4/3 x π x 703 = 1,436,755 mm^3

The area of the two spheres would be much smaller except that the model of the robot must then be considered to find the union volume,

$$\text{Robot} \cup \text{Model}. \tag{17}$$

The Upper-arm model radius= 80mm so: Union radius for a single sphere is 70+ 80 = 150. Union radius for two spheres is 35+80 = 115. Union volume of a single sphere is 4/3 x π x 1503 = 14,137,167 mm^3. Union volume of two spheres is 2 x 4/3 x π x 1153 = 12,741,211 mm^3.

There was a similar number of collisions for both models. When points within the second sphere were not tested to see if they had collided during the calculations for a previous sphere, this partially explained the lack of improvement in processing time for the model using two spheres.

Considering the simple six sided parallelepiped model, the volume of the model for the horizontal cylinder was less than that of the 2-D slice model.

Parallelepiped Volume = $(60 + 160)^2$ x (140 +80) = 10,648,000 mm3. This potentially reduced the number of blocked nodes, but shape and therefore the calculations were more complex so calculation time increased.

6 Conclusions

Using the two dimensional slice model of the cylinder, Θ2 and Θ3 were only determined for a single slice. This reduced the processing time as this slice of "blocked" nodes was copied for all Θ1 within the bounding base joint angles.

The number of "blocked" nodes produced was similar to other models, so that the intersection volume was approximately the same as for the sphere and polyhedral models. This suggested an equivalent accuracy.

The method of modelling dynamic objects by similar 2-D slices had the fastest intersection calculation times. Using the 2-D slices described, software models of the dynamic work-place were quickly passed to the main computer by the vision system. Similar 2-D slices were less complex than polyhedra, only requiring the two bounding angles of the base joint Θ1, the inner and outer radius and a height (five items of data).

2-D slices in a joint actuator space are the most efficient of the three models considered and they represent dynamic obstacles at least as effectively as the other two models.

References

1. Sanders, D.A.: Recognizing shipbuilding parts using artificial neural networks and Fourier descriptors. Proc. Institution of Mechanical Engineers Part B-Journal of Eng. Man. **223**(3), 337–342 (2009)
2. Rasol, Z., Sanders, D.A.: An automatic system for simple spot welding tasks. Total Vehicle Technology Conf., pp. 263–272 (2001)
3. Sanders, D.A., Harris, P.: Image modelling for real time manufacturing applications using 2-D slices in joint space and simple polyhedra. Journal of Design and Manufacturing **3**, 21–27 (1993)
4. Sanders, D.A.: Real time geometric modelling using models in an actuator space and cartessian space. Journal of Robotic Systems **12**(1), 19–28 (1995)
5. Sanders, D.A., Lambert, G., Pevy, L.: Pre-locating corners in images in order to improve the extraction of Fourier descriptors and subsequent recognition of shipbuilding parts. Proc. Institution of Mechanical Engineers Part B-Journal of Eng. Man. **223**(9), 1217–1223 (2009)
6. Carpin, S.: Randomized motion planning: A tutorial. Int. Jrnl. of Robotics & Automation **21**(3), 184–196 (2006)
7. Berretty, R.-P., Overmars, M.H., van der Stappen, A.F.: Dynamic motion planning in low obstacle density environments. Computational Geometry **11**(3–4), 157–173 (1998)
8. Sanders, D.A., Moore, A., Luk, B.L.: A Joint Space Technique for Real Time Robot Path Planning. Robots in Unstructured Environments, IEEE, 91TH376-4, pp. 1683–1689 (1991). ISBN 0-7803-0078-5
9. de Berga, M., Katzb, M.J., Overmarsa, M.H., van der Stappena, A.F., Vleugels, J.: Models and motion planning. Computational Geometry **23**(1), 53–68 (2002)
10. Tang, K.: A geometric method for determining intersection relations between a movable convex object and a set of planar polygons. IEEE Transactions on Robotics **20**(4), 636–650 (2004)
11. Large, F., Laugier, C., Shiller, Z.: Navigation Among Moving Obstacles Using the NLVO: Principles and Applications to Intelligent Vehicles. Autonomous Robots **19**(2), 159–171 (2005)
12. Halperin, D., Overmars, M.H.: Spheres, molecules, and hidden surface removal. In: Proc. 10th Annual. Symp. on Computational Geometry, pp. 113–122 (1994)
13. Sharma, V., Savchenko, M., Frazzoli, E., Voulgaris, P.G.: Transfer time complexity of conflict-free vehicle routing with no communications. International Journal of Robotics Research **26**, 255–271 (2007)
14. Fiorini, P.: Robot motion planning among moving obstacles, PhD disertation, University of California (1995)
15. Sanders, D.A.: Environmental sensors and networks of sensors. Sensor Review **28**(4), 273–274 (2008)
16. Sanders, D.A., Lambert, G., Graham-Jones, J., et al.: A robotic welding system using image processing techniques and a CAD model to provide information to a multi-intelligent decision module. Assembly Automation **30**(4), 323–332 (2010)
17. Sanders, D.A.: Comparing ability to complete simple tele-operated rescue or maintenance mobile-robot tasks with and without a sensor system. Sensor Review **30**(1), 40–50 (2010)
18. Sanders, D.A.: Comparing speed to complete progressively more difficult mobile robot paths between human tele-operators and humans with sensor-systems to assist. Assembly Automation **29**(3), 230–248 (2009)

19. Sanders, D.A., Graham-Jones, J., Gegov, A.: Improving ability of tele-operators to complete progressively more difficult mobile robot paths using simple expert systems and ultrasonic sensors. Industrial Robot **37**(5), 431–440 (2010)
20. Sanders, D.A., Stott, I.J., Robinson, D.C., et al.: Analysis of successes and failures with a tele-operated mobile robot in various modes of operation. Robotica **30**, 973–988 (2012)
21. Sanders, D.A., Tewkesbury, G.E., Stott, I.J., et al.: Simple expert systems to improve an ultrasonic sensor-system for a tele-operated mobile-robot. Sensor Review **31**(3), 246–260 (2011)
22. Sanders, D.A., Langner, M., Tewkesbury, G.E.: Improving wheelchair-driving using a sensor system to control wheelchair-veer and variable-switches as an alternative to digital-switches or joysticks. Industrial Robot **37**(2), 157–167 (2010)
23. Sanders, D.A., Stott, I.J., Graham-Jones, J.: Expert system to interpret hand tremor and provide joystick position signals for powered wheelchairs with ultrasonic sensor systems. Industrial Robot **38**(6), 585–598 (2011)