Offsetting obstacles of any shape for robot motion planning

Md Nasir Uddin Laskar, Hoang Huu Viet, Seung Yoon Choi, Ishtiaq Ahmed, Sungyoung Lee and Tae Choong Chung

Robotica / *FirstView* Article / April 2014, pp 1 - 19
DOI: 10.1017/S0263574714000630, Published online: 19 March 2014

Robotica: page 1 of 19. © Cambridge University Press 2014
doi:10.1017/S0263574714000630

Offsetting obstacles of any shape for robot motion planning

Md Nasir Uddin Laskar†, Hoang Huu Viet§,
Seung Yoon Choi†, Ishtiaq Ahmed†, Sungyoung Lee‡
and Tae Choong Chung†*

†*Artificial Intelligence Lab., Department of Computer Engineering, Kyung Hee University, Gyeonggi-do 446-701, South Korea*
§*Department of Information Technology, Vinh University, 182-Le Duan, Vinh City, Nghe An, Vietnam*
‡*Ubiquitous Computing Lab., Department of Computer Engineering, Kyung Hee University, Gyeonggi-do 446-701, South Korea*

(Accepted February 13, 2014)

SUMMARY
We present an algorithm for offsetting the workspace obstacles of a circular robot. Our method has two major steps: It finds the raw offset curve for both lines and circular arcs, and then removes the global invalid loops to find the final offset. To generate the raw offset curve and remove global invalid loops, $O(n)$ and $O((n+k)\log m)$ computational times are needed respectively, where n is the number of vertices in the original polygon, k is the number of self-intersections and m is the number of segments in the raw offset curve, where $m \leq n$. Any local invalid loops are removed before generating the raw offset curve by invoking a pair-wise intersection detection test (PIDT). In the PIDT, two intersecting entities are checked immediately after they are computed, and if the test is positive, portions of the intersecting segments are removed. Our method works for conventional polygons as well as the polygons that contain circular arcs. Our algorithm is simple and very fast, as each sub-process of the algorithm can be completed in linear time except the last one, which is nearly linear. Therefore, the overall complexity of the algorithm is nearly linear. By applying our simple and efficient approach, offsetting obstacles of any shape make it possible to construct a configuration space that ensures optimized motion planning.

KEYWORDS: Mobile robot; Obstacle; Polygon offsetting; Motion planning; Configuration space; Global invalid loop.

1. Introduction
Offsetting creates a new object similar to a given object, but at a specific distance, which is called the offset distance. Polygon offsetting is classified into two approaches: inner and outer offsetting. Although both of the approaches share common properties, they also have some differences. For example, the offset direction becomes opposite from one offset to another. This study is concerned with offsetting a polygon to facilitate efficient motion planning of a circular robot with radius d (which will work as an offset distance) in a workspace with obstacles of any shape. To plan a motion, the robot must compute the outer offsets of the workspace obstacles. In this paper, our theory and implementations deal with the case of outer offsetting in particular, which is a similar operation to the Minkowski sum.

In $\mathbb{R}^2$, the construction of the Minkowski sum of a polygon with a disc is called polygon offsetting. Given two sets, P and Q, their Minkowski sum is the set of points $\{p+q \mid p \in P,\ q \in Q\}$. The Minkowski sum is a widely used technique in computer-aided design and manufacturing (CAD/CAM), pocket machining, robot motion planning, and related research areas.

* Corresponding author. E-mail: tcchung@khu.ac.kr

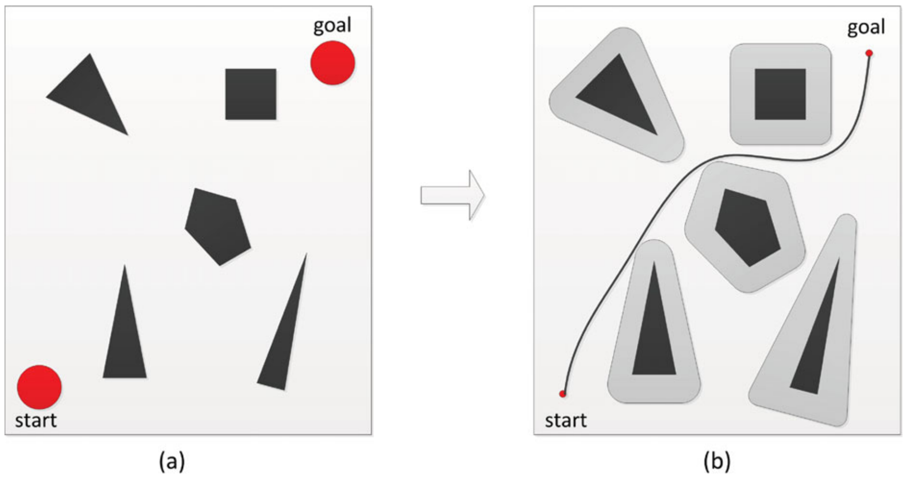

Fig. 1. (Colour online) Polygon offsetting in robot motion planning: (a) Workspace of a circular robot with polygonal obstacles and (b) configuration space of the robot after offsetting the workspace obstacles. A circular robot in the workspace is represented by a point in the configuration space. A sample path is also shown from the start to the goal position.

According to Wein *et al.*[1], a robot path has to be short, clear and smooth. A path should be the shortest possible convenient route; the distance of any point on the path to the closest obstacle should not be lower than some predefined value, and the path should not contain any sharp turns. Offsetting polygonal obstacles is a method of constructing the representation of a robot and obstacles in the configuration space to find the optimum path. Figure 1 illustrates the workspace and configuration space of a translating circular robot.

1.1. Related works: polygon offsetting

Research regarding the offsetting of a two-dimensional (2D) point-sequence curve (PS-curve) is classified into four categories[2]: pair-wise, level set, Voronoi diagram and bisector-based technique. Among these, the pair-wise offsetting technique has received more attention than the others. In this method, only the offset of the two end points of a segment is computed along their normal direction according to the offset distance and, finally, any invalid loops are removed. Choi and Park[3] proposed a pair-wise offset algorithm for a polygon with no islands, and introduced a pair-wise interference-detection (PWID) test to remove any local invalid loops. During the PWID test, their method compares each pair of elementary offset segments for interference.

Wein[4] presented a variant of the Minkowski sum with a disc to compute the exact and approximate offsets of polygons. This method decomposes the non-convex polygon into sub-polygons with only convex vertices, and finds the union of all of the sub-offsets. The decomposition technique offsets the intermediate edges twice, even though they do not take part in the final offset as they are not contour edges. Also, computing the union is a computationally expensive task. Another variant of the Minkowski sum of two polygons is proposed by Guibas and Seidel[5] to compute the convolution of two bodies. The controlled linear perturbation (CLP) approach is presented in ref. [6], and it is demonstrated on the Minkowski sums of polyhedra. Assigning inconsistent truth values to the predicates can cause errors with this method.

To generate the initial offset curve, a bisector method is introduced by Wong and Wong[7] that can automatically bridge the islands with the main profile. A segment of two adjacent intersecting bisectors is removed to handle local invalid loops. It takes $O(n^2)$ time to compute distance between two PS-curves, where n is the total number of curves. After that, Kim[8] and Kim *et al.*[9] used the bisector method to calculate the raw offset curve. They used the direction every time to report global invalid loops.

Our algorithm is a hybrid of the pair-wise and bisection methods. We find the offset points of a line segment along its normal direction to the offset distance d and then, to remove the local invalid loops, apply local geometry to check whether two adjacent segments intersect, as the bisector method does for two adjacent bisectors. If they do, portions of these two intersecting segments are removed.

The Voronoi diagram implementation of polygon offsetting is another popular approach. Bo[10] introduced a recursive method to compute a trimmed offset of a polygon by constructing a topological structure of all of Voronoi edges. Held[11] presented an $O(n \log n)$ algorithm to generate the Voronoi diagram of curvilinear polygons and then find the offset from the Voronoi diagram in $O(n)$ time, where n is the number of boundaries of the polygon. A circular arc extension to Held[11] is performed based on a randomized incremental insertion.[12] The Voronoi diagrams of the points, line segments and circular arcs in 2D Euclidean space are computed in $O(n \log n)$ times.

McMains *et al.*[13] presented an algorithm to build thin-walled parts in a fused deposition modeling machine. Chen and Mcmains[14] applied winding numbers to offset polygons. Their approach calculates the offset of multiple, non-overlapping polygons with islands and takes $O((n+k)\log n)$ time and $O(n+k)$ space, where n is the number of vertices in the input polygon and k is the number of self-intersections in the raw offset curve. Blomgren[15] first introduced the offsetting of a non-uniform rational B-spline (NURBS) curve by offsetting the control polygon and generating a new curve that had the same weights and knot vector as the base curve. Klass[16] approximates cubic splines at every segment separately by computing the inner control points based on the curvatures.

1.2. *Related works: robot motion planning*

Robot motion planning has been studied extensively in robotics literature for many years. There is a rich set of motion planning algorithms: grid-based search, cell decomposition, Minkowski sum, potential field method, sampling-based and some others. These algorithms deal with two approaches: sensor-based and model-based[17] techniques. The first generates information through the sensors in real time, and the second one generates a mathematical model of the environment. Among the variety of motion planning algorithms, in this paper, we will focus on finding a mathematical model of the robot environment that is very close to the method that employs the Minkowski sum.

A methodology for robot path planning among polygonal obstacles in a dynamic environment using mathematical modeling was introduced in refs. [17] and [18]. These offsets contradict the smooth path properties of an optimized path in the workspace as described in Wein *et al.*,[1] and cause the robot to execute many sharp turns in obstacle corners. However, these methods cannot handle a polygon that consists of both lines and arcs.

An approach to the coverage path planning problem, called Boustrophedon Cellular Decomposition (BCD), was proposed by Choset.[19] In this method, obstacles are considered to be polygons, and the accessible area of the robot is decomposed into non-overlapping cells so that the union of all of the cells forms the whole accessible area. Another approach to the online complete coverage problem was proposed in ref. [20]. Schwartz and Sharir[21] were the first to apply an exact cell decomposition technique to solve the motion planning problem by using their famous piano mover's problem. A trapezoidal map of the free space with polygonal obstacles and path planning from the road map were presented by Berg *et al.*[22]

Two common approaches are used for the environments in which the future locations of moving agents are known: adding a time dimension to the configuration space, or separating the spatial and temporal planning problems.[23] When the future locations are unknown, the planning problem is solved locally[24] or in conjunction with a global planner that guides the robot toward a goal.[16, 25]

For robot motion planning in the Euclidean space, different algorithms are proposed based on computing the Minkowski sum and the representation of the configuration space obstacles. The motion of a polygonal robot in an environment with polygonal obstacles is planned successfully by Kedem and Sharir.[27] By obtaining a set of deterministic samples, Varadhan *et al.*[28] presented an algorithm to find complete path for a translating polyhedral robot in 3D space. It constructs a roadmap of the free space without finding the explicit representation of the workspace. A technique of modeling a polygonal obstacle to plan the paths for three circular robots is presented in ref. [29].

1.3. *Motivation and contributions*

Most offsetting methods concentrate only on convex polygons. They divide non-convex polygons into convex sub-polygons, and offset them separately. Computing the Minkowski sum of two convex polytopes can have $O(n^2)$ time complexity,[5] where n is the number of features of the polytopes. The Minkowski sum of two non-convex polyhedra costs even more, and can have a combinatorial complexity as high as $O(n^6)$. Moreover, calculating the union of pair-wise Minkowski sums of sub-polygons is a computationally expensive task. If a non-convex polygon is divided into m sub-polygons, their union can have $O(m^3)$ time complexity.[28] If a polygon consists of a lot of non-convex vertices, the value of m becomes higher along with, and so is the computational complexity.

In this paper, we present a new algorithm that ensures the efficient offsetting of arbitrarily shaped objects in approximately linear time. Our algorithm handles non-convex objects very efficiently and thus a big performance improvement is achieved in this case. Along with recovering the drawbacks mentioned above, we use simple geometry to remove local invalid loops and our method can also handle objects that contain circular arcs. Thus, our algorithm can be used to offset any type of shapes.

1.4. Organization

The remainder of the paper is organized in the following manner. In Section 2, we describe some preliminary definitions and assumptions necessary for the main algorithm. Section 3 describes our method in two subsections for offsetting lines and arcs. The technique to handle global invalid loops is presented in Section 4. In Section 5, we present the experimental results and the performance analysis of our algorithm, and finally we conclude in Section 6.

2. Preliminary Definitions

2.1. Workspace and configuration space

In robot motion planning, the robot environment is classified into sensor-based and model-based systems.[17] Sensor-based systems generate information through sensors, and model-based systems use mathematical modeling of the environment through mathematical entities, such as polygons. Our algorithm is intended for the model-based systems that do not require any coloring methods of image processing, rather the accessible region $\mathcal{C}_{\text{free}}$ and the inaccessible region $\mathcal{C}_{\text{obs}}$ are determined from the concept of configuration space (Fig. 1(b)) as follows:

Let the robot be represented as a point $(x,\ y) \in \mathbb{R}^2$ in the continuous Cartesian plane, and the *configuration* q be defined as $q = [x,\ y,\ \theta]^T$, where (x, y) is the coordinate position of the robot's center in the Cartesian plane, and θ is its heading angle. When the robot moves on a workspace $\mathcal{W} \in \mathbb{R}^2$, not all regions of $\mathcal{W}$ are accessible to it. A configuration q of robot $\mathcal{A}$ is said to be valid if robot region $\mathcal{A}(q)$ is in $\mathcal{W}$ as

$$\mathcal{C} = \{\forall q \mid \mathcal{A}(q) \subset \mathcal{W}\}. \tag{1}$$

The set of all valid configurations of robot $\mathcal{A}$ is defined as its *free space*. Configuration q in $\mathcal{W}$ that robot $\mathcal{A}$ cannot access is called invalid, or it corresponds to *obstacles*. Let $\mathcal{O}_i$ be an obstacle in $\mathcal{W}$, and the union of all $\mathcal{O}_i$ becomes the obstacle region in $\mathcal{W}$. The configuration space obstacles or inaccessible region $\mathcal{C}_{\text{obs}}$ is found by offsetting the workspace obstacles based on the robot radius. It is a set of all configurations of robot $\mathcal{A}$ at which the robot region $\mathcal{A}(q)$ is in contact, or is overlapping, with an obstacle region $\mathcal{O}_i$ as

$$\mathcal{C}_{\text{obs}} = \{\forall q,\ i \mid \mathcal{A}(q) \cap O_i \neq \emptyset\}. \tag{2}$$

Since the obstacle region prohibits certain configurations of the robot, the free space $\mathcal{C}_{\text{free}}$ or accessible region of the robot is defined as

$$\mathcal{C}_{\text{free}} = \{\, q \in \mathcal{C} \mid \mathcal{A}(q) \cap (\bigcup_i \mathcal{O}_i) = \emptyset\}. \tag{3}$$

To plan a motion for the robot, a connectivity roadmap of $\mathcal{C}_{\text{free}}$ can be constructed by computing deterministic samples in $\mathcal{C}_{\text{free}}$.

2.2. Offsetting shapes

Offsetting a polygon is the process of computing the set of points that are at a constant distance d along the normal vector of polygon edges. If the curve $C(t)$ with parameter t is expressed as

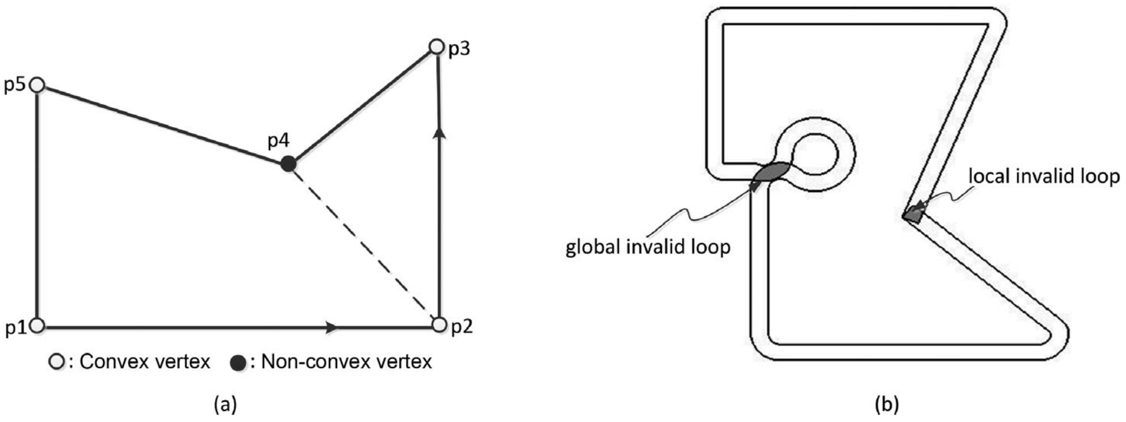

Fig. 2. (a) Convex and non-convex vertices. Conventional methods decompose the polygon along the dotted line and (b) local and global invalid loops (shown in dark).

$C(t) = (Cx(t), Cy(t))$, its offsetting curve $C_d(t)$ with a constant distance d is defined as

$$C_d(t) = C(t) + d\,N(t), \tag{4}$$

where $N(t)$ is the unit normal vector at t, and the constant distance d is called the offset distance. By applying local geometry, this paper deals with the offsetting task for a circular robot with radius d.

Definition 1 (Vertex type). A point (or a vertex) p_i is convex if the angle formed by p_i inside the polygon holds the condition $\angle(p_{i-1}, p_i, p_{i+1}) < \pi$, or a left turn is made at this vertex while marching along the contour, as in Fig. 2(a). Vertex p_i is non-convex if $\angle(p_{i-1}, p_i, p_{i+1}) > \pi$, or a right turn is made at this vertex while marching along the contour.

Definition 2 (Curve type). A PS-curve consists of lines and circular arcs. The curve types for each segment are defined as *line* and *arc*. The segments are treated differently based on their curve types.

Definition 3 (Invalid loop). A loop is caused by self-intersections and must be removed in order to find a valid offset curve. As shown in Fig. 2(b), a loop that is bounded by a single self-intersection point is called a *local invalid loop*, while a *global invalid loop* is bounded by multiple self-intersection points.

The intersection of two offset segments causes invalid loops. The pair-wise intersection detection test (PIDT) is a process to check whether two consecutive offset segments intersect. Two adjacent offset lines of a convex vertex do not intersect, but the adjacent offset lines of a non-convex vertex do. Two offset lines of the form $a_1x + b_1y = c_1$ and $a_2x + b_2y = c_2$, where a_i, b_i and c_i are real constants and both a_i and b_i are not simultaneously zero, are checked for their determinant $a_1b_2 - a_2b_1$. If $(a_1b_2 - a_2b_1) \neq 0$ then the lines intersect at a point and cause a local invalid loop. To determine the intersection between a circular arc and a line segment, we compute the minimum Euclidean distance from the line to the circle center. If the minimum Euclidean distance from the line to the circle center is less than the radius, then the line and arc segment intersect.

3. Computing Offsets

The input to our algorithm is a PS-curve with rational coordinates oriented counterclockwise, a curve type for each segment and the robot radius d. The curve type is used to track the segments, whether these are lines or arcs. The robot radius d works as an offset distance and we treat polygon vertices differently depending on whether a vertex is convex or non-convex.

A polygon consists of line and arc segments. For lines, the inputs are the endpoints and the curve type. For arcs, the arc start point, endpoint and its center are given, along with the curve type. If the curve type is a line, offset the line along its normal direction by the offset distance d. In this step, a vertex is checked to determine whether it is convex or non-convex. If the vertex is convex, the endpoints of the nearest offset segments are connected by a CCW-oriented arc centered on this

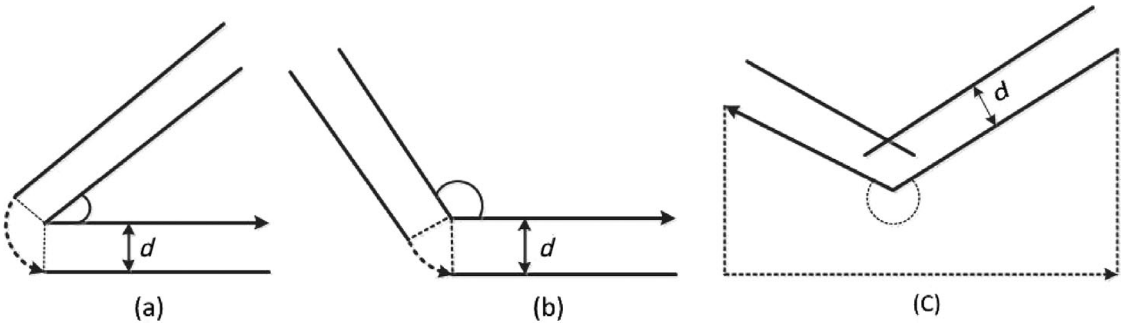

Fig. 3. Possible scenarios for offsetting polygon lines: (a) Acute angle, (b) obtuse angle and (c) reflex angle.

vertex; and if the vertex is non-convex, the endpoints of the nearest offset edges are connected and the intersection of the two offset lines is computed by invoking the PIDT to remove local invalid loops. If the curve type is an arc, the computation of the arc radius is straightforward. The start and end points are computed based on the offset distance and the arc radius. The PIDT is invoked to check whether this newly computed inflated arc creates any local invalid loops. If there is a positive result of PIDT, the intersection points must be found and a portion must be removed in order to avoid local invalid loops. In this way, the raw offset curve is generated, and then a check is done for any global invalid loops.

Any geometric shape can be represented with line and arc segments. Our method can be used to compute efficient offsets for both lines and arcs, so we can construct offsets for any obstacle shapes. We describe our algorithm in separate sections for offsetting lines and circular arcs.

Throughout this paper, we indicate the points (or vertices) of the given polygon as p_i and the offset points as p'_j, where $i, j \epsilon \mathbb{N}$. Since, in most of the cases, the vertex of the original polygon is associated with two points of the offset polygon, i and j are not necessarily equal.

3.1. *Offset lines*

A polygon can have convex as well as non-convex vertices. Figure 3 demonstrates the possible offsetting cases for both convex and non-convex vertices. Let a polygon P in the Cartesian plane have n points $p_0(x_0, y_0)$, $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ through $p_{n-1}(x_{n-1}, y_{n-1})$. Point $p_i(x_i, y_i)$ is directed to $p_{i+1}(x_{i+1}, y_{i+1})$. To offset the lines of polygon P, we shift each line segment by the offset distance d along its normal direction or to the right side of the edge. As a result, we obtain n disconnected offset edges with $2n$ new points $p'_j(x'_j, y'_j)$ parallel to the original lines. For example, if we offset line p_1p_2, we find the offset line $p'_1p'_2$, for line p_2p_3 we find the offset line $p'_3p'_4$, and so for line $p_{n-1}p_0$ we end up with the offset line $p'_{2n-2}p'_{2n-1}$. Now we need to check whether a vertex is convex or non-convex according to Algorithm 1.

Algorithm 1 tests the convexity of vertex p_i by computing the angle $p_\theta = \angle(p_{i-1}, p_i, p_{i+1})$ at that vertex. A vertex is considered differently if it is the start or end of a polygon arc. In Algorithm 1, n is the number of vertices and β is the intermediate angle without considering the polygon direction. The input vector components of point p_i are (Ax, Ay, Az) and (Bx, By, Bz), where (Ax, Ay, Az) is associated with vector $\overrightarrow{p_{i-1}p_i}$, and (Bx, By, Bz) is associated with vector $\overrightarrow{p_ip_{i+1}}$.

Algorithm 1 applies the theory of vector cross-product to vectors $\overrightarrow{p_{i-1}p_i}$ and $\overrightarrow{p_ip_{i+1}}$ to find angle p_θ based on the direction of the polygon (usually CCW). The output of this algorithm is the convexity of vertex p_i along with its angle p_θ formed by the adjacent two lines. Every vertex has a z-component associated with it, which indicates the turning direction of the polygon at this vertex. Initially, the z-values of any vertex are zero, and computing the cross-product $(p_i - p_{i-1}) \times (p_{i+1} - p_i)$ provides a non-zero value along the z-direction. The direction of the vector cross-product is perpendicular to both the vectors being multiplied and the vectors normal to the plane containing them. If the z-value is positive, then $p_\theta < \pi$ and we are turning left. If it is negative, then $p_\theta > \pi$ and we are turning right. Therefore, convex and non-convex vertices produce positive and negative z-values respectively. For example, let the coordinates of the polygon in Fig. 2(a) be $p_1 = (-2, -1)$, $p_2 = (2, -1)$, $p_3 = (2, 3)$, $p_4 = (0, 1)$ and $p_5 = (-2, 2.5)$. Vertex p_2 is convex because angle $\angle(p_1, p_2, p_3)$ at p_2 is $\frac{\pi}{2}$ and

Algorithm 1: Convexity test.

```
input  : $(Ax, Ay, Az)$ and $(Bx, By, Bz)$; the vector components of two lines associated with $p_i$
output: convexity of vertices and angle produced by them, $p_\theta = \angle(p_{i-1}, p_i, p_{i+1})$
begin
    $zComponent \leftarrow 0$
    for $i = 1$ to $n$ do
        $zComponent \leftarrow AxBy - AyBx$
        $\beta \leftarrow \arcsin(\frac{|(p_i - p_{i-1}) \times (p_{i+1} - p_i)|}{|p_i - p_{i-1}|\,|p_{i+1} - p_i|})$
        if $(zComponent > 0)$ then
            $p_\theta \leftarrow \beta$
            return convex
        end
        else
            $p_\theta \leftarrow (2\pi - \beta)$
            return non-convex
        end
    end
end
```

$z_2 = 16$. For vertex p_4, the polygon angle $\angle(p_3, p_4, p_5)$ at p_4 is $\frac{17\pi}{11}$ and $z_4 = -7$, so p_4 is non-convex.

If the vertex is convex as in Figs. 3(a) and (b), where the outside lines are always the offset lines, then the two new adjacent points of the offset edges will be connected by a circular arc centered in the corresponding vertex with radius d, which is robot's radius. For a vertex p_i, we draw an arc centered at p_i between the points p'_{2i-2} and p'_{2i-1}, which are the endpoints of the adjacent offset edges, induced by $p_{i-1}p_i$ and $p_i p_{i+1}$ respectively. The angle that defines such a circular arc is $\pi - \angle(p_{i-1}, p_i, p_{i+1})$.

Consider that the line supporting $p_1 p_2$ is $ax + by + c = 0$, where $a, b, c \in \mathbb{R}$. The representation of the offset edge that is based on the locus of all points lying at the offset distance d from the line $ax + by + c = 0$ is given by

$$d = \frac{|ax + by + c|}{\sqrt{(a^2 + b^2)}}. \tag{5}$$

Algorithm 2 describes the process of finding two points that lie parallel to a given line segment at distance d. The line segments $p_i(x_i, y_i)\,p'_j(x'_j, y'_j)$ and $p_{i+1}(x_{i+1}, y_{i+1})\,p'_{j+1}(x'_{j+1}, y'_{j+1})$ are perpendicular to the given line segment $p_i(x_i, y_i)\,p_{i+1}(x_{i+1}, y_{i+1})$. d_x and d_y are the vector components of $\overrightarrow{p_i p_{i+1}}$, and its perpendicular vector components are x_{perp} and y_{perp}. The normalized vector components x'_{perp} and y'_{perp} are calculated by applying the vector magnitude i_{len} of the vector $\overrightarrow{p_i p_{i+1}}$. The offset coordinates x'_j and y'_j are found from the corresponding point (x_i, y_i) of the original polygon according to

$$\begin{aligned} x'_j &= x_i + x'_{\text{perp}}, \\ y'_j &= y_i + y'_{\text{perp}}. \end{aligned} \tag{6}$$

For n lines of a polygon, we find n disconnected offset segments with $2n$ end points, as one point in the original polygon corresponds to the endpoints of two offset lines. The starting point of a given line segment $p_i(x_i, y_i)$ results in $p'_j(x'_j, y'_j)$ at distance d along its normal direction, and the endpoint $p_{i+1}(x_{i+1}, y_{i+1})$ results in $p'_{j+1}(x'_{j+1}, y'_{j+1})$ at the same distance d.

The computation of the offset segment of a given line is shown in Fig. 4(a). Let the given line be $p_1 p_2$, and we want to compute the corresponding offset line $p'_1 p'_2$. The slope $m_1 = \frac{y_2 - y_1}{x_2 - x_1}$ of line $p_1 p_2$ is equal to the slope of line $p'_1 p'_2$ because the offset line is parallel to the given line. If two lines are cut by a transversal so that the corresponding angles are congruent, then the lines are parallel. We find two points p'_1 and p'_2 such that $p'_j(x'_j, y'_j)$ lies on the perpendicular line segments of $p_1 p_2$

Algorithm 2: Offset lines.

```
input  : $p_i(x_i,\ y_i)$, $p_{i+1}(x_{i+1},\ y_{i+1})$ and offset distance $d$, which is the robot radius
output: $p'_j(x'_j, y'_j)$ and $p'_{j+1}(x'_{j+1}, y'_{j+1})$ such that $p'_j p'_{j+1}$ is parallel to $p_i p_{i+1}$ with given
        distance $d$

begin
    $k \leftarrow 1$
    for $i = 0$ to $(n - 1)$ do
        $d_x \leftarrow x_{i+1} - x_i$
        $d_y \leftarrow y_{i+1} - y_i$
        $x_{perp} \leftarrow d_y$
        $y_{perp} \leftarrow -d_x$
        $i_{len} \leftarrow \sqrt{x_{perp}^2 + y_{perp}^2}$
        $x'_{perp} \leftarrow (x_{perp} * d) / i_{len}$
        $y'_{perp} \leftarrow (y_{perp} * d) / i_{len}$
        for $j = k$ to $k + 1$ do
            $x'_j \leftarrow x_i + x'_{perp}$
            $y'_j \leftarrow y_i + y'_{perp}$
            $x'_{j+1} \leftarrow x_{i+1} + x'_{perp}$
            $y'_{j+1} \leftarrow y_{i+1} + y'_{perp}$
        end
        $k \leftarrow j + 1$
    end
end
```

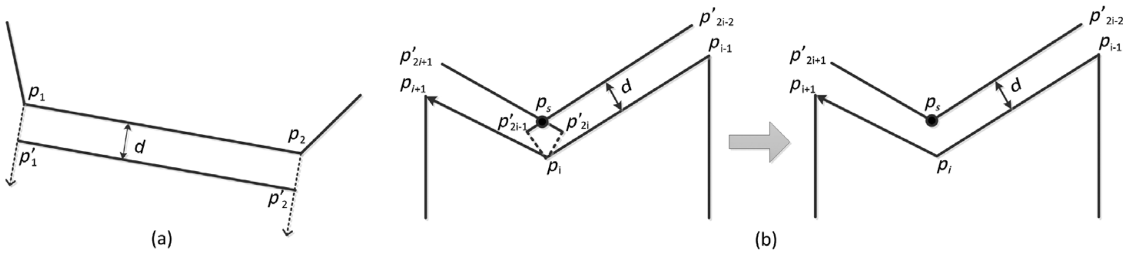

Fig. 4. (a) Computing offset edge based on the polygon line segment and (b) local invalid loop for non-convex vertex.

with slope $m_i = -\frac{x_2 - x_1}{y_2 - y_1}$. The vector $\overrightarrow{p_2 p_1}$ is regarded as $d_x = x_2 - x_1$ and $d_y = y_2 - y_1$. A vector perpendicular to $\overrightarrow{p_2 p_1}$ has the parameters $x_{\text{perp}} = d_y$ and $y_{\text{perp}} = -d_x$. The vector magnitude i_{len} is calculated from $\sqrt{x_{\text{perp}}^2 + y_{\text{perp}}^2}$.

If the vertex is not convex, as in Fig. 3(c), the PIDT results are positive, as the offset lines of the two adjacent lines of the non-convex vertex intersect each other. Since we are dealing with the outer offsetting only, it is most likely that only non-convex vertices will cause two consecutive offset lines to intersect. For a non-convex vertex p_i, we find an intersection of two offset lines $p'_{2i-2}p'_{2i-1}$ and $p'_{2i}p'_{2i+1}$ that causes a local invalid loop as shown in Fig. 4(b). We need to compute this intersection point to remove the local invalid loop caused by offset lines. If the intersection point is p_s, from p_s to p'_{2i-1} and from p'_{2i} to p_s will be removed respectively for the offset lines $p'_{2i-2}p'_{2i-1}$ and $p'_{2i}p'_{2i+1}$.

Consider that two offset lines $p'_{2i-2}p'_{2i-1}$ and $p'_{2i}p'_{2i+1}$ are of the form $a_1x + b_1y = c_1$ and $a_2x + b_2y = c_2$ respectively, where a_i, b_i and c_i are constants and $a_i, b_i \neq 0$. Then the determinant Δ will be $a_1b_2 - a_2b_1$. If the intersection point is $p_s(x_s, y_s)$, we find $x_s = (b_2c_1 - b_1c_2)/\Delta$ and $y_s = (a_1c_2 - a_2c_1)/\Delta$.

The number of offset segments is usually less than the number of total input segments. The following explains why the number of valid segments m in a raw offset curve is less than the total segments n in the input polygon.

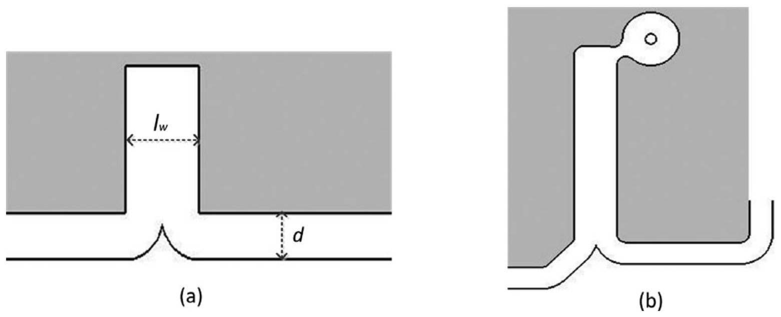

Fig. 5. Coincident offset segments, $l_w < 2d$.

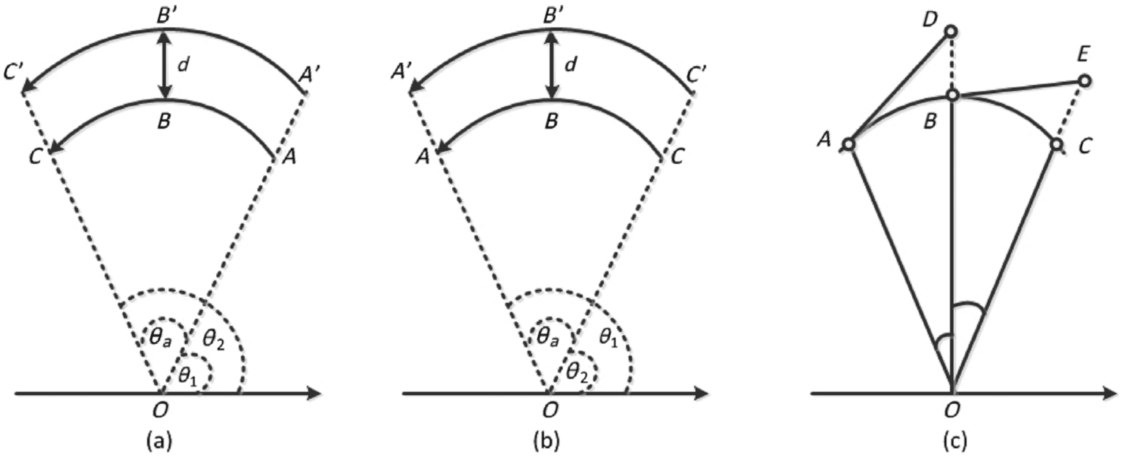

Fig. 6. Offsetting circular arcs: (a) CCW arc, $\theta_2 > \theta_1$, (b) CW arc, $\theta_2 < \theta_1$ and (c) draw the arc for offsets.

Coincident offset segment: If the length of a line segment l_w, as shown in Fig. 5, is less than twice the length of the offset distance d, then that line segment will not be considered for offsetting. In Fig. 5(a), the length of the line segment $l_w = 30$ mm and the offset distance $d = 15.5$ mm, so the coincident offset segment for the line is not shown. The PIDT finds the intersections and can avoid coincident offset segments all the time. Figure 5(b) is a complex scenario with a coincident offset segment in which the offsetting of the circular arc is displayed, even though the coincident segment is avoided before that. This indicates the fact that a coincident offset segment does not influence the segment next to it.

3.2. *Offset circular arcs*

Arcs are handled quite differently from lines. The input PS-curve has arc endpoints, center and the direction of the circle along with the curve type. Start angle θ_1, end angle θ_2 and arc radius r are calculated from the given inputs. Local invalid loops are removed by invoking the PIDT and then finding the intersections of offset arc segment with its neighboring offset segments.

According to Figs. 6(a) and (b), a circular arc can be clockwise (CW) or counterclockwise (CCW). Points $A(x_1, y_1)$, $C(x_2, y_2)$ and $O(c_x, c_y)$ are the start, end and center points, respectively, of the arcs A, B, C. Points $A'(x'_1, y'_1)$ and $C'(x'_2, y'_2)$ are the corresponding offset points for A and C respectively. The start angle θ_1 is calculated from OA as $\theta_1 = \tan^{-1}(\frac{y_1 - c_y}{x_1 - c_x})$, and the end angle θ_2 is calculated from OC based on $\theta_2 = \tan^{-1}(\frac{y_2 - c_y}{x_2 - c_x})$. The arc angle $\theta_a = \theta_2 - \theta_1$. For a CCW arc, $\theta_2 > \theta_1$, and for a CW arc, $\theta_2 < \theta_1$. If θ_1 and θ_2 become negative, then $\theta_1 = \theta_1 + 2\pi$ and $\theta_2 = \theta_2 + 2\pi$.

Finding the arc radius r is straightforward from the arc center (c_x, c_y) and the start (or end) point of the arc as $r = \sqrt{(c_x - x_1)^2 + (c_y - y_1)^2}$. If we consider the circle equation as

$$x^2 + y^2 + 2gx + 2fy + c = 0, \tag{7}$$

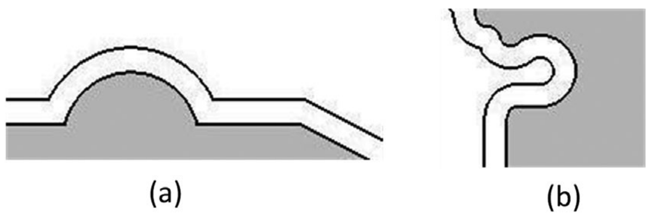

Fig. 7. Selecting the offset distance for CW and CCW arcs: (a) In CCW arc, center lies inside the polygon area and offset radius is longer $(r + d)$. (b) In CW arc, center lies outside the polygon area and offset radius is shorter $(r - d)$.

where g, f and c are constants, $(-g, -f)$ is the circle center and we can find the arc radius $OA = r$ from $\sqrt{g^2 + f^2 - c}$. However, the radius of the offset arc OA' depends on the arc direction. Although the curve direction is always counterclockwise, the direction of an arc can be clockwise or counterclockwise in that curve. This is because an arc center can even lie outside the polygon region. A CCW arc has its center inside the polygon region and the radius is $(r + d)$, where d is the offset distance, as shown in Fig. 7(a), while a CW arc center resides outside the polygon area and the radius is $(r - d)$ as shown in Fig. 7(b). The following derivation is performed considering the arc direction toward CCW, so $OA' = (r + d)$ and a similar derivation can be done for a CW arc considering $OA' = (r - d)$.

The start and end points of the offset arc are calculated according to θ_1, θ_2 and r. The start point $A'(x'_1, y'_1)$ of the offset arc $A'B'C'$ is calculated as

$$\begin{aligned} x'_1 &= c_x + (r + d)\cos\theta_1, \\ y'_1 &= c_y + (r + d)\sin\theta_1. \end{aligned} \tag{8}$$

To find the next point to (x'_1, y'_1), we apply the theory of circular motion in physics. Figure 6(c) shows the algorithm to efficiently draw a circular arc. We compute the magnitude and direction of the tangential and radial motions. $\overrightarrow{AD}$ is the perpendicular vector of $\overrightarrow{AO}$ and is tangent to the circle. The length AD is easily found by multiplying with $\tan\theta$. Adding this tangential vector to $\overrightarrow{AO}$ gives DO. To get BO, DO is multiplied with $\cos\theta$. Starting from A, we have found a new point B. A similar computation is done for BE to find the next point C. Both tangential and radial factors are constant and can be pre-calculated, so there is no need to compute them in every iteration.

Let θ be the angular length of a segment ahead of the previous point and x_t be the tangent of θ. If N_s is the number of segments (500 in our case) in the arc between the start angle and the end angle, $\theta = \theta_a / N_s$. We can easily turn vector $\overrightarrow{OA}$ by $\frac{\pi}{2}$ radians in the CW direction that becomes tangent to the circle. The components of vector $\overrightarrow{OD}$ are calculated as

$$\begin{aligned} x'_{\text{new}} &= x'_1 + x_t \tan\theta, \\ y'_{\text{new}} &= y'_1 + y_t \tan\theta, \end{aligned} \tag{9}$$

where $(x'_{\text{new}}, y'_{\text{new}})$ is point D, θ is the angular length of a small segment of the arc, and x_t and y_t are the tangent vector components of the radial vector (x_1, y_1). Vector $\overrightarrow{OB}$ is calculated as

$$\begin{aligned} x_{\text{new}} &= x'_{\text{new}} \cos\theta, \\ y_{\text{new}} &= y'_{\text{new}} \cos\theta, \end{aligned} \tag{10}$$

where $(x_{\text{new}}, y_{\text{new}})$ is point B. At this point we are back to where we started and repeating this process will fill in the entire angular length θ_a. Algorithm 3 describes the process of offsetting the circular arc.

Now the PIDT will determine whether this offset arc intersects its neighboring offset segments. According to Fig. 8, one of three scenarios can happen: (i) both the offset segments attached to the offset arc come from a single line as in Fig. 8(a), (ii) the offset arc intersects two different lines in its two sides as in Fig. 8(b) and (iii) an offset arc is a neighbor of another arc as in Fig. 8(c).

Algorithm 3: Offset circular arcs.

```
input  : $arcDirection$, $(x_1,\ y_1)$, $(x_2,\ y_2)$ and $(c_x,\ c_y)$; arc direction, start point, end point and
         center of the arc respectively.
output : An inflated arc between start and end point.
begin
    $r \leftarrow \sqrt{(c_x - x_1)^2 + (c_y - y_1)^2}$
    if ($arcDirection == CW$) then
        $r \leftarrow r - d$
    end
    else
        $r \leftarrow r + d$
    end
    $\theta_1 \leftarrow \tan^{-1}(\frac{y_1 - c_y}{x_1 - c_x})$
    if ($\theta_1 < 0$) then
        $\theta_1 \leftarrow \theta_1 + 2\pi$
    end
    $\theta_2 \leftarrow \tan^{-1}(\frac{y_2 - c_y}{x_2 - c_x})$
    if ($\theta_2 < 0$) then
        $\theta_2 \leftarrow \theta_2 + 2\pi$
    end
    $\theta_a \leftarrow \theta_2 - \theta_1$
    $\theta \leftarrow \frac{\theta_a}{N_s}$
    for $i = 1\ to\ N_s$ do
        $x'_1 \leftarrow c_x + r\cos\theta_1$
        $y'_1 \leftarrow c_y + r\sin\theta_1$
        $x_t \leftarrow -y'_1$
        $y_t \leftarrow x'_1$
        $x'_{new} \leftarrow x_{start} + x_t\tan\theta$
        $y'_{new} \leftarrow y_{start} + y_t\tan\theta$
        $x_{new} \leftarrow x'_{new}\cos\theta$
        $y_{new} \leftarrow y'_{new}\cos\theta$
    end
end
```

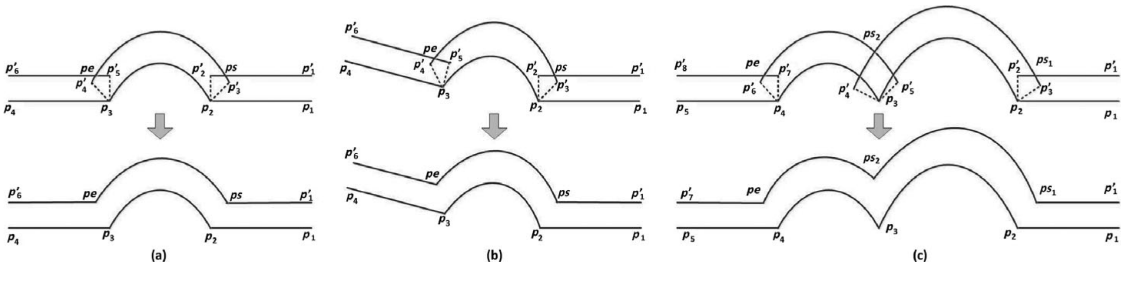

Fig. 8. Removing local invalid loops for arc offsets: (a) Arc within a same line, (b) arc with different lines at its two ends and (c) two consecutive arcs in the main polygon.

To find the intersection point for cases (i) and (ii) above, consider Figs. 8(a) and (b), where the starting point p_s (and ending point p_e) of the offset arc $p_s p_e$ intersects the offset line $p'_1 p'_2$ before (and $p'_3 p'_4$ after) the arc itself. As a result $p_s p'_2 p_2 p'_3$ and $p_e p'_4 p_3 p'_5$ form invalid loops, which we call local invalid loops. For the final offsetting, the line segments $p_s p'_2$ and $p'_3 p_e$ will have to be removed. To do this, finding the points p_s and p_e is important. To calculate point p_s, we solve the circle equation containing arc $p_s p_e$ and line $p'_1 p'_2$, and to find p_e, we solve the same circle equation

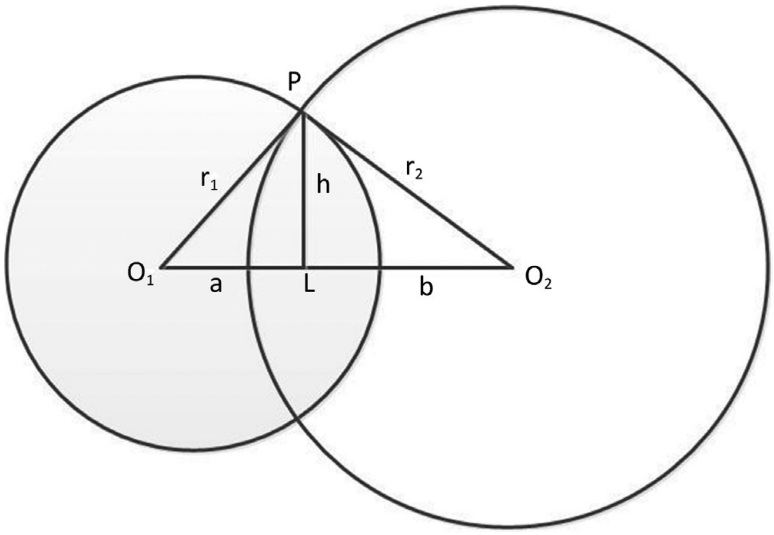

Fig. 9. Computing the intersection of two arcs.

with line $p'_3p'_4$. If lines p_1p_2 and p_3p_4 of the main polygon are the segments from the same straight line as in Fig. 8(a), it is sufficient to solve the circle equation with any of the $p'_1p'_2$ and $p'_3p'_4$ offset lines, as the intersection of a line and a circle gives us both intersecting points. In this case we can save some computations to calculate p_s and p_e.

We now explain how to compute the intersection points $p_s(x_s, y_s)$ and $p_e(x_e, y_e)$ of an offset line segment $p'_1p'_2$ with its neighboring offset arc as shown in Figs. 8(a) and (b). The Euclidean distance $l_{p'_1p'_2}$ from p'_1 to p'_2 is $l_{p'_1p'_2} = \sqrt{(x'_1 - x'_2)^2 + (y'_1 - y'_2)^2}$, and the direction vectors $\overrightarrow{D} = \overrightarrow{p'_1p'_2}$ from p'_1 to p'_2 are $D_x = (x'_2 - x'_1)/l_{p'_1p'_2}$ and $D_y = (y'_2 - y'_1)/l_{p'_1p'_2}$. Now, for the parameter $t \in [0, 1]$, the line equation of $p'_1p'_2$ will be $x = t\ D_x\ +\ x'_1$ and $y = t\ D_y\ +\ y'_1$. The parameter value of t for the closest point to the circle (that contains the arc) center $(c_x,\ c_y)$ will be $t = D_x(c_x - x'_1) + D_y(c_y - y'_1)$, which is the projection of the circle center on the line from p'_1 to p'_2. Consider the point $p'_i(p'_{ix},\ p'_{iy})$ on $p'_1p'_2$ which is closest to the circle center. The coordinates will be $p'_{ix} = tDx + x'_1$ and $p'_{iy} = tD_y + y'_1$. The Euclidean distance from p'_i to the circle center is $l'_{pi} = \sqrt{(p'_{ix} - c_x)^2 + (p'_{iy} - c_y)^2}$, so the intersection coordinates of p_s are $x_s = (t - dt)\ D_x\ +\ x'_1$ and $y_s = (t - dt)\ D_y\ +\ y'_1$, where dt is the Euclidean distance from parameter t to the offset circle intersecting points p_s or p_e, and the second intersection point p_e will be $x_e = (t + dt)\ D_x\ +\ x'_1$ and $y_e = (t + dt)\ D_y\ +\ y'_1$.

For the case in which two consecutive arcs arrive in the given polygon (Fig. 8(c)), two circle equations are solved to find the self-intersecting points as shown in Fig. 9. Let the centers of the first and second circles be $O_1(x_1, y_1)$ and $O_2(x_2, y_2)$ respectively. The radii of the circles are r_1 and r_2 respectively. We are going to find the intersection point $P(x, y)$. From Fig. 9, we also see that $h = PL$, $a = O_1L$ and $b = O_2L$. Applying the theory of ref. [30], the intersection point $P(x, y)$ of the two circles is

$$\begin{aligned} x &= x_l - \frac{h(y_2 - y_1)}{a+b}, \\ y &= y_l + \frac{h(x_2 - x_1)}{a+b}, \end{aligned} \tag{11}$$

where (x_l, y_l) is the coordinate of point L.

The following explains why the number of valid segments m is less than the total number of segments n in a raw offset curve in the case of arc offsetting.

Arc Radius and offset distance of a CW arc: If an arc is centered outside the polygon area and the offset distance is equal to or more than the arc radius ($d \geq r$), then this arc is not considered for offsetting. As shown in Fig. 10(b), the arc from the original polygon will not be considered for offsetting if the length of the offset distance exceeds the length of the arc radius. Figure 10(a) shows that an arc in the original polygon centered outside the polygon area will be considered for offsetting only if the offset radius is more than the offset distance.

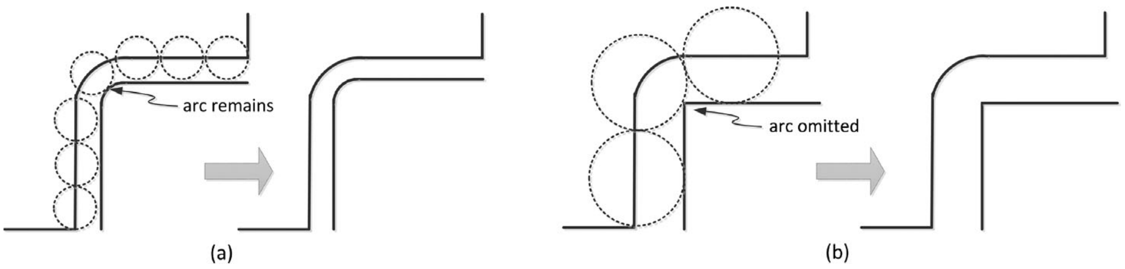

Fig. 10. (a) Offset distance is less than the arc radius ($d < r$) and (b) offset distance is larger than the arc radius ($d \geq r$).

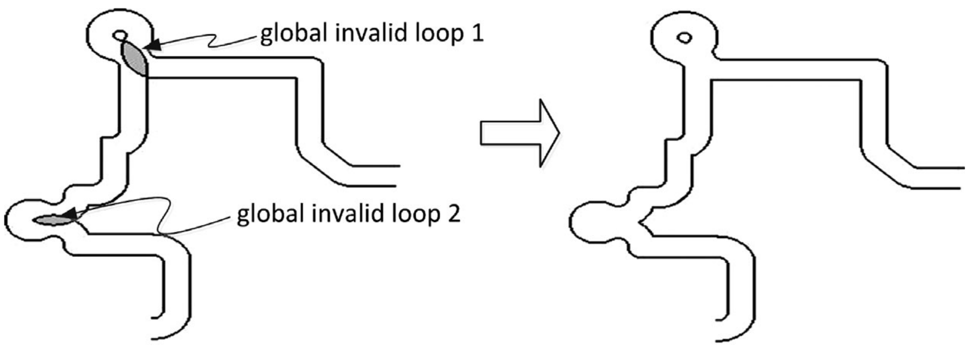

Fig. 11. Global invalid loops (gray) and the final offset after removing them.

4. Handling Global Invalid Loops

Usually, invalid loops occur at the self-intersection points of raw offset curves. Finding the intersections between two offset lines or between an offset line and an offset arc can prevent local invalid loops before generating the raw offset curve, but global invalid loops may still exist. Figure 11 is a part of a pocket profile that shows the demonstration of two global invalid loops and the final offsetting after removing them. The radius of loop one is 80.04 mm, and the radius of the invalid loop two is 65.004 mm, while the offset distance d is 65 mm.

In a raw offset curve, it is always necessary to find the self-intersecting points to remove global invalid loops. Instead of using the brute-force algorithm of $O(m^2)$, where m is the total number of segments in the raw offset curve, we use the well-known "Bentley-Ottmann Algorithm"[31] to find the global self-intersecting points, which requires $O((n + k) \log m)$ time complexity, where n is the total number of vertices in the original polygon, k is the number of self-intersections and m is the number of valid segments in the raw offset curve, where $m \leq n$. We need to check only for the global self-intersection points instead of checking for all local and global self-intersection points because the local self-intersection points have already been removed before the raw offset curve is generated by PIDT.

5. Experimental Results and Analysis

5.1. Experimental results

We implemented our algorithm in GLUT for Win32 version 3.7.6 on a personal computer with a 3.00-GHz Pentium(R) D CPU and 2.00-GB RAM. We have tested our algorithm for various types of polygons, and it can successfully handle any shape of geometric object. In the literature section, we have mentioned for several times that most of the algorithms cannot handle non-convex shapes and circular arcs. However, our method deals with such problems very efficiently with minimum computational cost, as we have seen so far.

Figure 12 demonstrates the whole process of our proposed method. The input polygon has total 11 vertices ($n = 11$). Among them, six are convex, three are non-convex and the remaining two are the start and end points of the only arc. Figure 12(a) shows the original polygon. Figure 12(b) shows

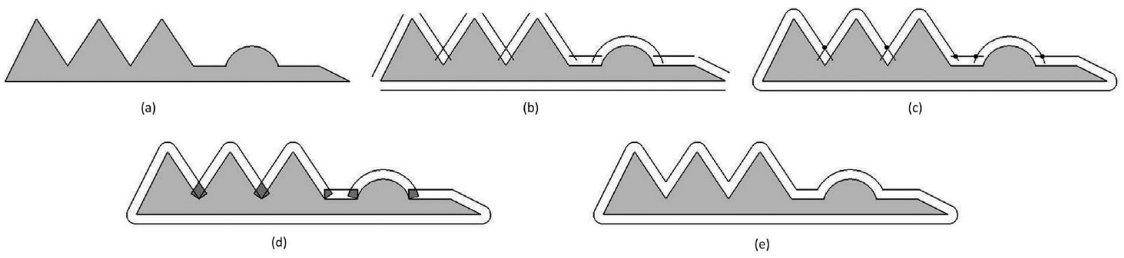

Fig. 12. Output of a polygon consists of convex vertex, non-convex vertex and arc: (a) Input polygon (gray), (b) disconnected offset segments for each lines and arcs, (c) for convex vertex, adjacent two offset points are connected by circular arcs of radius d. Self-intersection points caused by non-convex vertices and circular arc are indicated by black circles, (d) raw offset curve with local invalid loops (dark gray), (e) final output.

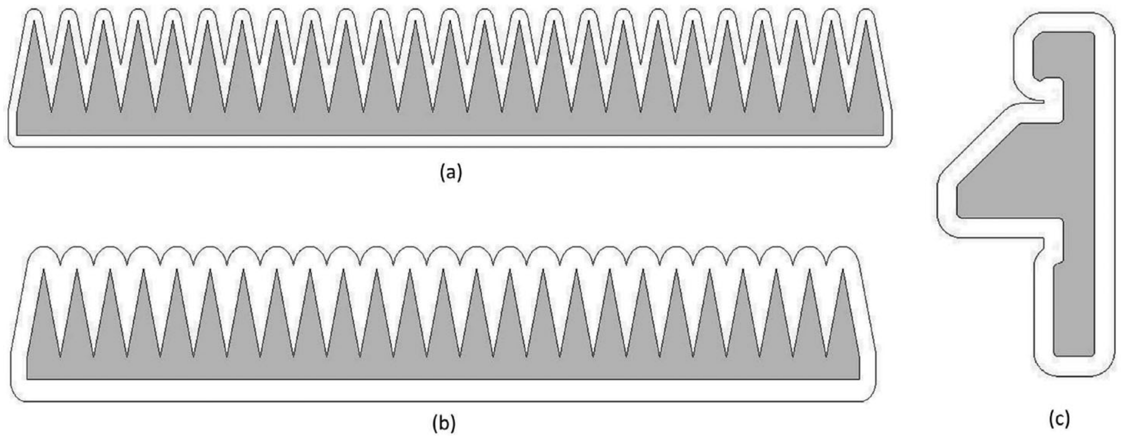

Fig. 13. Polygon consists of convex and non-convex vertices: (a) comb output, using offset distance, $d = 50$, (b) comb output, using bigger offset distance, $d = 100$, and (c) pillar.

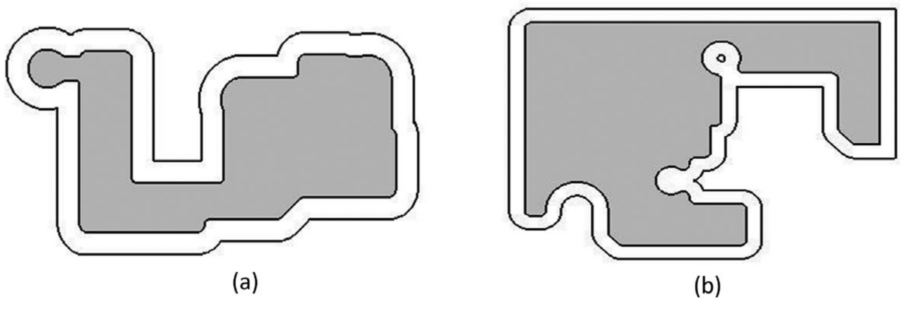

Fig. 14. (a) Shape of a duck, created with line and arc segments and (b) a pocket profile consists of arbitrary line and arc segments.

the disconnected offset segments for the lines and arcs based on the offset distance $d = 50$. Based on the type of the PS-curve, the angle between two neighboring segments is checked. For a convex vertex, the two new adjacent points of the offset edges are connected by a circular arc centered in the corresponding vertex, p_i, as shown in Fig. 12(c). The pair-wise self-intersections are also shown for non-convex vertices and circular arcs. A circular arc causes self-intersections with its two adjacent offset segments at its both ends. Figure 12(d) is the raw offset curve for the original polygon with local invalid loops. Usually we remove local invalid loops before generating raw offset curves so that in the raw offset curve only global invalid loops remain. However, in Fig. 12(d), the local invalid loops are kept for a better understanding of the whole process of getting the final output. After removing the local invalid loops, 12(e) shows the final output.

Figure 13 shows the output of a comb for different d values and the case of a pillar. Figure 14(a) is the shape of a duck with 19 lines and 66 arc segments, and Fig 14(b) is a pocket profile with nine

Table I. Running time (measured in milliseconds) for various types of input polygons.

Input polygon	Size (n)	Non-convex vertex (n_{nc})	Arc segment	Offset distance (d)	Running time (ms)
Output 1	11	3	1	75	2.2065
Comb	53	24	0	100	7.1836
Pillar	32	0	13	65	4.5705
Duck	85	5	14	50	23.7883
Pocket	66	9	25	50	21.2514

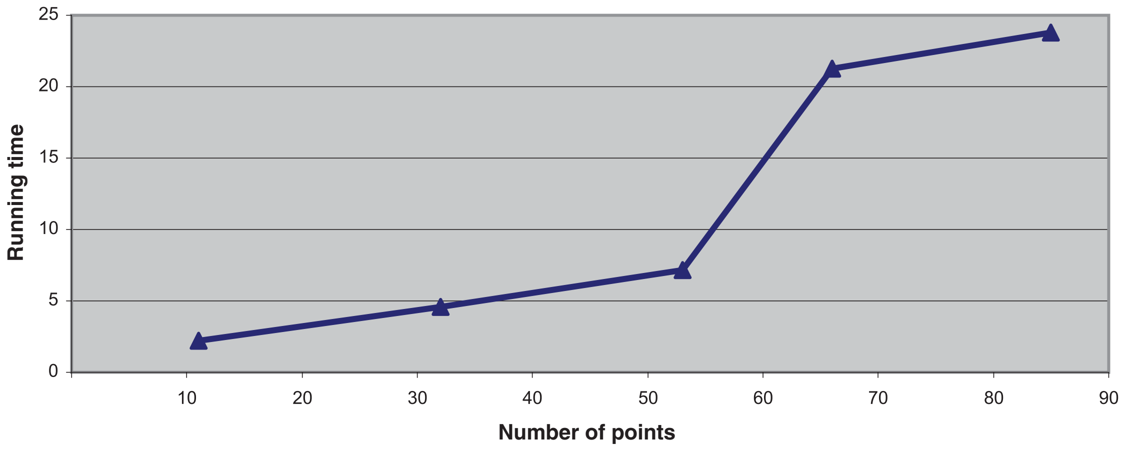

Fig. 15. (Colour online) Running times for the outputs.

non-convex vertices and 29 arc segments. The polygon statistics and algorithm execution times are shown in Table I. Since the number of non-convex vertices and arc segments present in the polygon greatly influences the computation time, we mentioned them in Table I. Data representing the time taken to construct the offset are listed as the average of the maximum time taken by a single input from multiple executions. Figure 15 is the graph drawn for the running time in terms of the size of the shapes. The output shapes listed in Table I show that the running time is approximately linear relative to the input size n. One may observe from the graph that the running time of the pocket profile is comparatively higher than other inputs. This is because of the large number of arc segments present in the pocket profile.

The relationship between the algorithm running time and the value of the offset distance is a performance issue for the polygon offsetting task. It is expected that the running time should not increase drastically for a bigger d value. We experimented with our algorithm for different d values for all the shapes shown in this section, and we found that our algorithm achieves this criterion. Figs. 13(a) and (b) show the comb output found using the offset distance of 50 mm and 100 mm respectively. Table II shows the running time based on different offset distances for the comb output. Figure 16 shows the plotted result of Table II. Offset distance d has impact on offset shapes. Offset shapes look similar to the original object for smaller d values. For increased d values, offset shapes can even be quite different than the original object. Larger d values cause increased number of intersections in the offset segments. For complex shapes, it imposes more coincident offset segments as in Fig. 5. It leads to extra computation, hence the running time increases.

The last example shown in Fig. 17 is part of a complex geometric shape in an industrial PCB design process. The shape has 527 points, 105 lines and 124 arc segments. For each line and arc segment, raw-offset curve is generated. The offset distance is 40 mm. Local invalid loops are removed immediately after generating the raw-offset curve by invoking PIDT. Global invalid loops are then detected and removed to find the final offsetting. It takes total 87.1472 ms to compute the offset.

With the same experiment settings, Table III shows the computation time comparison of our algorithm with that of Wein.[4] Figure 18 is the plotted result of Table III that shows the superiority of our method over the approximate offset construction methods. The method described in ref. [4] constructs the exact and approximate offset polygons by using the conic and circle/segment

Table II. Running time for comb output based on different d values.

Offset distance (d)	Running time (ms)
10	0.0154
25	3.0128
40	4.7452
50	5.5406
65	5.8863
80	6.2571
90	6.9486
100	7.1829

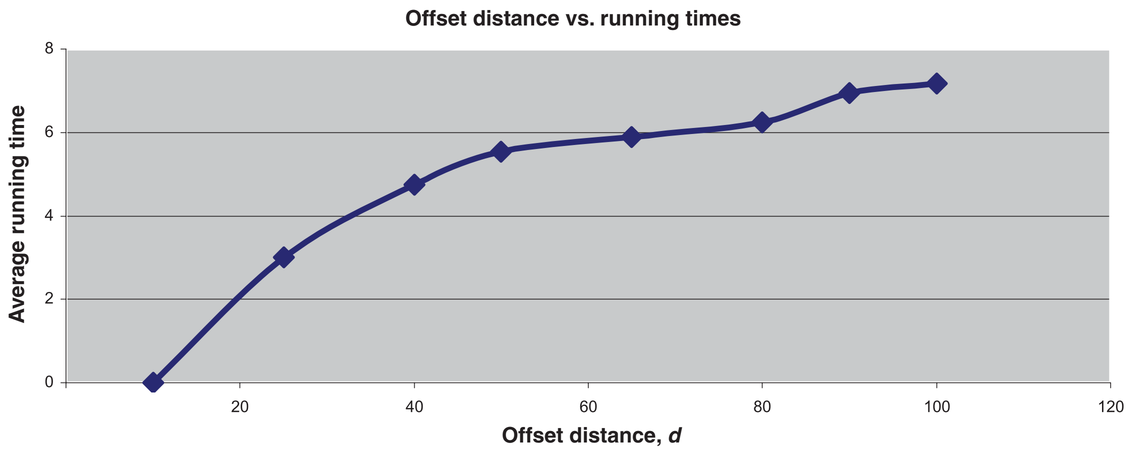

Fig. 16. (Colour online) Plotted result of Table II. Effect of offset distance on the comb output.

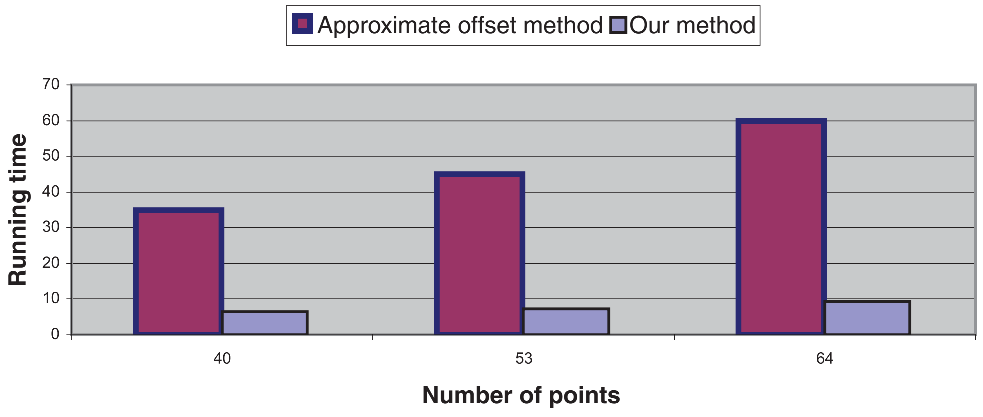

Fig. 17. (Colour online) A schematic of a real PCB design with its offset output.

traits-classes respectively. Computing and manipulating the intersection points of two arcs in the circle/segment traits-class is more efficient than in the conic-traits class, so the approximate construction is considerably faster than the exact construction of the offsets. We compare our method with the approximate construction of offset polygons, and our method shows superior performance. To the best of our knowledge, so far, our method constructs offset polygons in minimal computation time.

5.2. Complexity analysis

The input of the algorithm is a polygon as a PS-curve, and the output is an offset polygon. The whole process is performed in three steps: (1) raw offset curve generation, (2) checking for local invalid loops by invoking the PIDT and (3) removal of global invalid loops.

Table III. Running time comparison with other methods.

Input polygon	Size (n)	Non-convex vertex (n_{nc})	Offset distance (d)	Running time (ms) Approximate offset construction	Running time (ms) Our method
Comb	53	24	25	45	3.1026
Wheel	40	14	50	35	2.5547
Spiked	64	40	50	60	4.2691

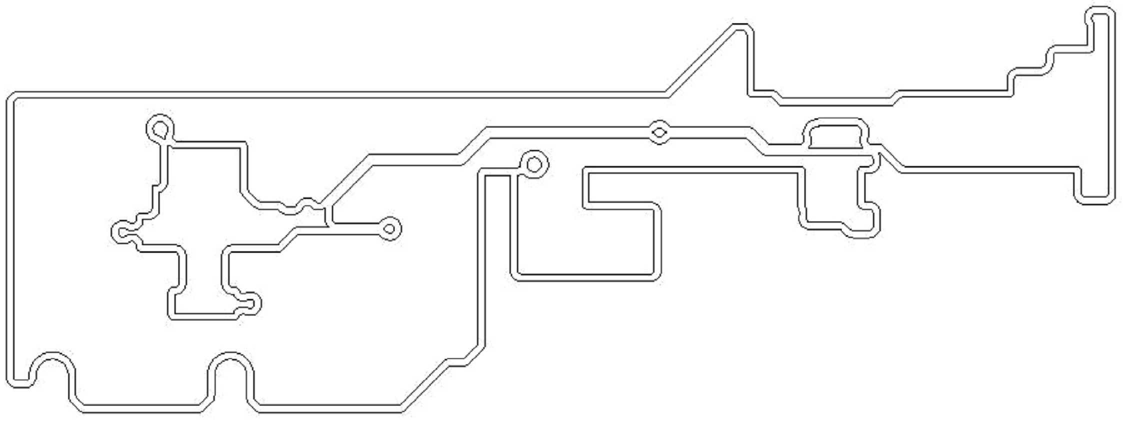

Fig. 18. Running time comparison with approximate offset construction method.

To generate the raw offset curve, each vertex is inserted only once. The time complexity to construct the raw offset curve is $O(n)$, where n is the number of vertices in the input PS-curve. This step also takes $O(n)$ space.

For the second part, two adjacent offset segments are continuously checked for intersections if they come from non-convex vertices. Since we are dealing with outer offsetting, three situations will create local invalid loops: (i) a non-convex vertex (Fig. 4(b)), (ii) type variations of two adjacent offset curves, for example, if a line is adjacent to an arc or *vice versa* (Figs. 8(a) and (b)), and (iii) two adjacent offset arcs (Fig. 8(c)). A polygon has n total vertices, and among them n_{nc} are non-convex and n_a vertices contain arcs, so checking for local invalid loops with the PIDT will cost $O(n_{\text{nc}} + n_a)$ time, where $(n_{\text{nc}} + n_a) < n$.

In the third step, most of the time is spent reporting the self-intersections. The time complexity to remove the global invalid loops is $O((n + k)\log m)$ and uses $O(n + k)$ space, where n is the total number of vertices in the original polygon, k is the number of self-intersections, and m is the number of segments in the raw offset curve, where $m \leq n$. In practice, the number of global invalid loops is very small and makes the complexity $O(n \log m)$. Moreover, the PS-curves contain a large number of interfering segments and the value of m becomes much smaller than n, so the time for this step becomes nearly linear.

Above all, overall time complexity of the algorithm with its all three steps can be completed in $O(n \log m)$ time, in which n is the total number of points in the input PS-curves and m is the total number of segments in the raw-offset curves, where $m \leq n$.

6. Conclusions and the Future Works

A new algorithm to offset arbitrary shapes for the efficient motion planning of a circular robot is presented in this paper. The algorithm is able to deal with convex and non-convex shapes as well as arcs in an object. Line and arc segments are treated differently, and local invalid loops are removed before generating the raw offset curve. Our method is simple, mathematically well defined and produces consistent results. It handles the non-convex shapes very efficiently where prior methods impose redundant computations. The overall time complexity of the algorithm is approximately linear.

From geometric point of view, any object shape can be constructed from basic geometric primitives. Line and arc segments can be used to represent any kind of shapes. This paper considers the arcs to be circular. In the future, other types of arcs, such as elliptical, parabolic and hyperbolic arcs, can be integrated with this method.

Acknowledgments

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0012609). This research was also partly supported by the National IT Industry Promotion Agency (NIPA) in 2013 (Global IT Talents Program).

References

1. R. Wein, J. P. Berg and D. Halperin, "The Visibility-Voronoi Complex and Its Applications," *Proceedings of the Twenty-First Annual Symposium on Computational Geometry (SCG)*, Eindhoven, The Netherlands (2005) pp. 63–72.
2. L. Zhiwei, F. Jianzhong and G. Wenfeng, "A robust 2d point-sequence curve offset algorithm with multiple islands for contour-parallel tool path," *Comput. Aided Des.* **45**, 657–660 (2013).
3. B. Choi and S. Park, "A pair-wise offset algorithm for 2d point-sequence curve," *Comput Aided Des.* **31**, 735–745 (1999).
4. R. Wein, "Exact and approximate construction of offset polygons," *Comput. Aided Des.* **39**, 518–527 (2007).
5. L. J. Guibas and R. Seidel, "Computing convolutions by reciprocal search," *Discrete Comput. Geom.* **2**, 175–193 (1987).
6. V. Milenkovic, E. Sacks and M.-H. Kyung, "Robust Minkowski Sums of Polyhedra via Controlled Linear Perturbation," *ACM Symposium of Solid and Physical Modeling (SPM 10)*, Haifa, Israel (2010) pp. 23–30.
7. T. N. Wong and K. W. Wong, "Toolpath generation for arbitrary pockets with islands," *Int. J. Adv. Manuf. Technol.* **12**, 174–179 (1996).
8. H.-C. Kim, "Tool path generation for contour parallel milling with incomplete mesh model," *Int. J. Adv. Manuf. Technol.* **48**, 443–454 (2010).
9. H.-C. Kim, S.-G. Lee and M.-Y. Yang, "A new offset algorithm for closed 2d lines with islands," *Int. J. Adv. Manuf. Technol.* **29**, 1169–1177 (2006).
10. Q. Bo, "Recursive polygon offset computing for rapid prototyping applications based on Voronoi diagrams," *Int. J. Adv. Manuf. Technol.* **49**, 1019–1028 (2010).
11. M. Held, "Voronoi diagrams and offset curves of curvilinear polygons," *Comput. Aided Des.* **30**(4), 287–300 (1998).
12. M. Held and S. Huber, "Topology-oriented incremental computation of Voronoi diagrams of circular arcs and straight-line segments," *Comput. Aided Des.* **41**, 327–338 (2009).
13. S. McMains, J. Smith, J. Wang and C. Sequin, "Layered Manufacturing of Thin-Walled Parts," *Proceedings of ASME Design Engineering Technical Conference*, Baltimore, MD (2000) pp. 343–356.
14. X. Chen and S. McMains, "Polygon Offsetting by Computing Winding Numbers," *Proceedings of ASME International Design Engineering Technology Conference (IDETC)*, California, USA (2005) pp. 565–575.
15. R. Blomgren, "B-spline Curves," **In**: *Class Notes, Boeing Document B-7150-BB-WP-281/D-441.2*, CASE/SME Seminar, Michigan, USA (1981).
16. R. Klass, "An offset spline approximation for plane cubic splines," *J. Comput. Aided Des.* **15**(5), 297–299 (1983).
17. W. Esquivel and L. Chaiang, "Nonholonomic path planning among obstacles subject to curvature restrictions," *Robotica* **20**(1), 49–58 (2002).
18. A. Srivastava, D. Kartikey, U. Srivastava and S. Rajesh, "Nonholonomic sortest robot path planning in a dynamic environment using polygonal obstacles," *Proceedings of International Conference on Industrial Technology (ICIT)*, Viña del Mar, Chile (2010) pp. 553–558.
19. H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Int. J. Auton. Robots* **9**(1), 247–253 (2000).
20. H. H. Viet, V.-H. Dang, M. N. U. Laskar and T. C. Chung, "Ba*: An online complete coverage algorithm for cleaning robots," *Int. J. Appl. Intell.* (2012) doi:10.1007/s10489-012-0406-4.
21. J. T. Schwartz and M. Sharir, "On the piano movers problem: Ii. general techniques for computing topological properties of real algebraic manifolds," *Adv. Appl. Maths.* **4**, 298–351 (1983).
22. M. Berg, O. Cheong, M. Kreveld and M. Overmars, *Computational Geometry: Algorithms and Applications* (Springer, New York, NY, 2008).
23. S. M. LaValle, *Planning Algorithms* (Cambridge Univesity Press, Cambridge, UK, 2006).
24. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, *Principles of Robot Motion* (MIT Press, Cambridge, MA, 2007).
25. A. Clodic, V. Montreuil, R. Alami and R. Chatila, "A Decisional Framework for Autonomous Robots Interacting with Humans," *Proceedings of the IEEE International Workshop on Robot Human Interaction Communication*, Nashville, USA (2005) pp. 543–548.

26. B. Xu, D. Stilwell and A. Kurdila, “A Receding Horizon Controller for Motion Planning in the Presence of Moving Obstacles,” *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Alaska, USA (2010) pp. 974–980.
27. K. Kedem and M. Sharir, “An Automatic Motion Planning System for a Convex Polygonal Mobile Robot in 2D Polygonal Space,” *ACM Symposium on Computational Geometry*, Urbana-Champaign, IL (1998) pp. 329–340.
28. G. Varadhan, S. Krishnan, T. V. Sriram and D. Manocha, “A simple algorithm for complete motion planning of translating polyhedral robots,” *Int. J. Robot. Res.* **24**(11), 983–995 (2005).
29. A. Sudsang, F. Rothganger and J. Ponce, “Motion planning for disc-shaped robots pushing a polygonal object in the plane,” *IEEE Trans. Robot. Autom.* **18**(4), 550–562 (2002).
30. P. Bourke, ‘Intersection of two circles,” available at: http://paulbourke.net/geometry/circlesphere/ (1997) (accessed February 1, 2014).
31. J. L. Bentley and T. Ottmann, “Algorithms for reporting and counting geometric intersections,” *IEEE Trans. Comput.* **C-28**(9), 643–647 (1979).