VLSI-Efficient Schemes for High-Speed
Construction of Tangent Graph

S. K. Lam, K. Sridharan and T. Srikanthan

(Corresponding author)
Lam Siew-Kei
Centre for High Performance Embedded Systems
Nanyang Technological University
50, Nanyang Drive, Research TechnoPlaza
3rd Storey, BorderX Block
SINGAPORE 637553
Tel: (65) 6790 6643
Fax: (65) 6792 0774
Email: assklam@ntu.edu.sg

K. Sridharan
Department of Electrical Engineering
Indian Institute of Technology Madras
Chennai – 600 036, INDIA
Email: sridhara@iitm.ac.in

T. Srikanthan
Centre for High Performance Embedded Systems
Nanyang Technological University
50, Nanyang Drive, Research TechnoPlaza
3rd Storey, BorderX Block
SINGAPORE 637553
Email: astsrikan@ntu.edu.sg
VLSI-Efficient Schemes for High-Speed Construction of Tangent Graph

Abstract:
Tangent graph based data structure has been readily used in motion planning for mobile robots and robot manipulators. The complexity of the tangent graph grows exponentially as the robot’s configuration space increases in dimension. The ability to construct larger number of tangents at high-speed thus becomes crucial to facilitate dynamic motion planning where on-line avoidance is necessary. In this paper, we present efficient schemes for construction of tangent graphs for an environment consisting of both non-convex and convex obstacles. The proposed technique for tangent graph construction is based on a gradient computation approach that encompasses binary search, logarithmic approximation, and half-plane computation modules. The modules were ported to VLSI using commercial tools. Synthesis results show that each module has a latency of only 7.2 ns and a total chip area of about 7K NAND gates, thus demonstrating that the proposed techniques are highly appropriate for tangent graph computations in real-time applications.

Keywords: Robot motion planning, tangent graph, Very Large Scale Integration

1 Introduction
The ability to automatically plan collision-free motions in an obstacle-filled environment is an attractive feature of a mobile robot or robot manipulator [12][21]. A number of approaches for motion planning have been presented in the past, where most of them utilise the concept of configuration space (C-space) [13][17][18]. The C-space can be defined as the space of configurations of a moving object, which encompasses a set of parameters that completely specify the position of every point on the object. The C-space of a mobile robot consists of a two-dimensional plane containing polygonal obstacles. For a robot manipulator, the configurations are specified by the set of joint angles, so a robot’s C-space is its joint space. The obstacles in the robot’s workspace (C-obstacles) can be mapped into the C-space, and various techniques using potential field, cell decomposition and roadmap-based
approaches can be employed to compute the collision-free paths from a start to end point.

The visibility graph developed by Lozano-Perez and Wesley [18] is a roadmap approach that has been widely implemented for motion planning on the C-space. In a two-dimensional C-space, the visibility graph can be constructed by forming visibility segments between obstacle vertices including the start and end points. Various path search algorithms can then be employed to identify an optimal collision-free path in the C-space. The visibility graph and its variants continue to be of tremendous interest from the viewpoint of algorithmic geometry [8]. Since the process of constructing a visibility graph consumes time and storage space, parallel algorithms on general-purpose architectures have been proposed in [3]. The algorithms are efficient from an asymptotic viewpoint. In particular, the authors present $O(\log n)$ time algorithms for solving several visibility problems on an $n$-node polygon in the CREW-PRAM model of computation. Empirical performance details are not available. Alternative proposals presented in [6][7][12][13][14][15][16] aim to reduce the complexity of the visibility graph without affecting the search result.

Liu et. al. [14][15][16] proposed the tangent graph (T-graph), whereby only the visibility segments that are tangents to the obstacles are constructed. This reduced form of the visibility graph can lead to less computation time without affecting the optimality of the solution path. The two-dimensional C-space, which is well suited for mobile navigation, can be extended to higher dimensions for robot manipulators. However, the higher dimension C-space poses additional complexity to the construction of T-graph as the C-obstacles contain curved shapes. Unless the tangent endpoints are restricted to a resolution interval, the number of tangents between a point and an obstacle is very large. The number of tangents increases exponentially in the order of $O(r^{n-1})$, where $r$ is the C-space resolution and $n$ is the C-space dimension ($n > 2$). Limiting the resolution will lead to less computation for the construction of T-graph, but the optimality of the solution path may be compromised.

Doyle et. al. [2] presented a path planning strategy for a robot manipulator in high dimensional C-space that constructs the tangents during the path search process.
Heuristics are applied to construct a limited set of tangents from a point $v$ to a two-dimensional obstacle plane defined by $v$, the obstacle centroid and destination point. The tangents are computed by considering the set of lines from point $v$ to the cross-section of the obstacle, which intersects with the plane. The two lines with the greatest angle between them are the required tangents. As suggested in the paper, more optimal results can be achieved by defining more planes rotated along the line from the current position to the goal.

It is apparent that a high-speed computation engine for constructing large number of tangents will lead to more optimal solutions without posing a performance bottleneck to the search process. In addition, the ability to compute these tangents at high-speed will lend itself well to solving the problem of dynamic motion planning, which is often substantially harder than the static path-planning problem [20]. In dynamic environments, on-line obstacle avoidance is required and the high-speed construction of tangent graphs is imperative to the system’s reliability.

The significance of a hardware implementation for autonomous robot motion planning has long been recognised [4]. Research based on distributed representation of the C-space [20] and cellular automata architectures [22] have been conducted. The authors in [22] present an approach using Voronoi diagrams for path planning. While Voronoi diagrams can help in generating a path for a robot that maintains a safe distance from all obstacles for path planning purposes, their construction is non-trivial and the metric used for the construction influences the quality of the diagram. A significant limitation for the VLSI implementation of the cellular automata architecture is the high pin count imposed by the interconnection network. The pin count increases in proportion to the number of cells. Moreover, since the number of cells that can be housed in a single chip is limited, a number of such chips have to be employed in order to cover large input dimensions [22].

In this paper, we present a novel strategy for constructing the tangent graph, which employs binary search, logarithmic approximation and half plane computation methods. The strategy lends itself well to VLSI implementation, which facilitates high-performance computation. The binary search module provides a means to reduce the complexity of tangent computations, and the logarithmic approximation simplifies
the complex arithmetic operations, which leads to more cost-effective hardware. The half-plane computation is required to validate whether the computed tangents intersect with other C-obstacles [2]. Synthesis results on a 0.35-micron process show that each module has an average latency of only 7.2 ns, and occupying a total area occupied is only 7K NAND gates. Performance comparison of the proposed hardware implementation with a previous reported technique for tangent graph generation shows an estimated speed-up of 19K times.

2 Gradient-Based Tangent Construction

In this section, we present a tangent graph computation strategy using binary search, logarithmic approximation, and half-plane computation approaches. We will limit our discussion to the problem of constructing tangents between a point \( v \) to a polygonal obstacle \( P \) in a two-dimensional C-space. It is noteworthy that the proposed method can be easily extended to C-space of higher dimensions using the concepts discussed in [2]. The following definitions and theorem in [7] associated with the tangent graph generation are appropriate for our discussion. The proof of Theorem 1 is presented in [7] and is hence omitted here.

**Definition 1:** \( HP_{+sg} \) and \( HP_{-sg} \) represent the positive and negative half-planes that result from dividing a two-dimensional space into two regions along the line \( sg \). A third point is necessary to render either half-plane positive or negative.

**Definition 2:** A vertex \( v \) of an \( r \)-sided polygon is said to be the farthest front vertex with respect to a point \( x \) if all vertices \( v_i \) for \( i=1, 2, ..., r \) are confined entirely either to \( HP_{+xv} \) or to \( HP_{-xv} \).

**Theorem 1:** Relative to a point \( x \) in the plane, the two farthest front vertices of polygon \( P \) comprise the optimal pair of available via points to circumvent \( P \).

The proposed technique is based on the observation that the farthest front vertices for a given point \( v \) are those for which the line gradient (tangents from \( v \)) is either the maximum or the minimum. The gradient computation and the maximum/minimum identification are performed for each of the vertices on the polygon \( P \), and the
complexity of computing the farthest front vertices can be done in time proportional to the number of vertices in P (i.e. \(O(m)\) if \(m\) is the maximum number of vertices in each polygon). If we assume the case of a higher dimensional C-space, where \(p\) number of two-dimensional planes is defined to obtain optimal path solutions [2], the complexity of computing the farthest front vertices of the C-obstacle is in the order of \(O(mp)\).

2.1 Binary Search for Reducing Complexity of Gradient Computation

We can perform a binary search defined suitably to reduce the asymptotic complexity of the gradient computations. We assume that the coordinates of the vertices of polygon P are stored in counter-clockwise order in an array and the logarithms used in the discussion are in base 2.

Consider Fig. 1 and a ray \(R\) from the external point \(v\) through the first vertex in the array, namely \(v_1\). This ray passes through the point \(k\) on the boundary of P. To determine the two farthest front vertices, we first compute the intersection point \(k\) in \(O(\log m)\) time. The intersection point \(k\) along with \(v_1\) allows us to look at two portions, one above that contains one of the farthest front vertices and one below that contains the other farthest front vertex. Note that this division of the polygon into two pieces one on either side of the line joining \(k\) and \(v_1\), holds even if \(v\) had been located elsewhere (for example to the right of P or below P and so on). Also, angles of rays to

![Figure 1: Illustration for binary search to find farthest front vertices \(v_5\) and \(v_{11}\)](image-url)
vertices from \( v \) measured with respect to ray \( R \) would be positive on one side and negative on the other. Computing \( k \) therefore allows us to compare counter-clockwise angles among themselves to find maximum while also permitting comparison of clockwise angles among themselves to find minimum.

We determine if the middle vertex in the array, denoted by \( v_{\text{mid}} \) (\( v_7 \) in Figure 1), lies on the same side as \( v_2 \) with respect to the line containing \( R \) joining \( v \) and \( v_1 \). This can be checked in constant time by looking at the signs using the half-plane equation given by (1) that simply expresses the fact that a point \((x,y)\) lies on a line joining \((x_a,y_a)\) and \((x_b,y_b)\) (if two points lie on the same side of line joining \((x_a,y_a)\) and \((x_b,y_b)\), then we have two inequalities of the same type). If they lie on the same side, then we can neglect the counter-clockwise chain of vertices between \( v_{\text{mid}} \) and \( v_2 \) since it does not contain the intersection point \( k \). Otherwise (if they lie on opposite sides of the line), we omit the counter-clockwise chain of vertices between \( v_{\text{mid+1}} \) and \( v_m \).

\[
\left( x \times y_a - x \times y_b \right) + \left( y \times x_b - y \times x_a \right) - \left( y_a \times x_b - y_b \times x_a \right) + \left( x_a \times y_b - x_b \times y_a \right) = 0 \tag{1}
\]

Suppose that they lie on the same side of line containing \( R \). Then we examine if \( v_{\lceil m/4 \rceil} \) (the ceiling symbol has been introduced to get an integer index) is on the same side as \( v_{\text{mid}} \) and if it is so, we can now neglect this counter-clockwise chain between \( v_{\text{mid}} \) and \( v_{\lceil m/4 \rceil} \) since by similar arguments, it will not contain the intersection point \( k \).

The process continues until we reach the edge containing \( k \). While storing the intersection point \( k \), we can also store the two vertices of the edge containing \( k \) since they will be useful to find the farthest front vertices. This binary search procedure works in \( O(\log m) \) time where \( m \) is the number of vertices of \( P \). Suppose \( v \) had been to the ‘right’ of \( P \) and the vertex \( v_l \) was encountered before the ray \( R \) pierced the polygon \( P \). Then \( k \) will lie to the left of \( v_l \). This, however, is not a problem as far as computing coordinates of the intersection point \( k \) is concerned.

Using the intersection point \( k \), we find the farthest front vertices. Consider Fig. 1 again (the arguments for other locations of the external point \( v \) would be similar) and let the vertex preceding intersection point \( k \) in the counter-clockwise order be denoted
by \( v_s \). The middle vertex in this counter-clockwise chain starting from \( v_1 \) may be denoted by \( v_{s/2} \). Let the counter-clockwise angle made by \( R \) with a ray from \( v \) through \( v_s \) be denoted by \( \theta_s \). Similarly, let the angle made with \( v_{s/2} \) be denoted by \( \theta_{s/2} \) (assuming \( s \) to be an even number) and the one with \( v_{(s/2)+1} \) be denoted by \( \theta_{(s/2)+1} \). If \( \theta_{s/2} > \theta_s \) and \( \theta_{(s/2)+1} > \theta_{s/2} \), then it is clear that the maximum angle vertex lies between \( v_{s/2} \) and \( v_s \). On the other hand, if \( \theta_{s/2} > \theta_s \) and \( \theta_{s/2} > \theta_{(s/2)+1} \), then the maximum angle vertex lies between \( v_1 \) and \( v_{s/2} \).

We can continue to bisect the interval this way until we reach the vertex to which the ray from \( v \) is such that all other vertices lie “below” the half-plane. By a similar procedure, we can find the vertex to which the ray from \( v \) makes the minimum angle. These once again take \( O(\log m) \) time. It should be noted that the binary search cannot be performed using simply the \( x \) or \( y \) coordinates of vertices even though they exhibit a certain monotonic behaviour since the farthest front vertices are not necessarily the vertices with maximum \( x \) or \( y \) coordinate. Hence, the complexity of constructing the tangents from a point \( v \) to a two-dimensional polygon using the binary search approach is reduced to \( O(\log m) \).

### 2.2 Logarithmic Approximation For Gradient Computation

The angles comparison in the previous section is equivalent to comparing the gradients of the lines between the point \( v \) and the vertices of the polygon. The conventional method of computing the gradient \( s \) using equation (2) incurs a division, which does not lend itself well to hardware-efficient implementation.

\[
s = \frac{y_a - y_h}{x_a - x_h} \tag{2}
\]

However, the actual gradient values of all the vertices are not required. The proposed solution employs the logarithmic function that exhibits a monotonic characteristic with the conventional computation of \( s \). As shown in (3), logarithm computations relieve us of the costly divider in hardware. Note that although the gradient can
assume a positive or negative value, the polarity information is not shown in (3). A method to incorporate polarity into the final implementation will be discussed later.

\[
\log_2 s = \log_2 \left( \left| y_a - y_b \right| \right) - \log_2 \left( \left| x_a - x_b \right| \right)
\]  

Upon identifying the farthest front vertices, each of the nodes in the graph for the environment stores the corresponding to each of the obstacles in logarithm representation ($\log_2 s$). The visibility segments from $v$ to the polygon corresponding to the maximum and minimum gradient, represent the tangents required for the path search process. Efficient computation of the logarithm will be discussed in section 3.

2.3 Validating the Computed Tangents

As described in [2], the computed tangents must be validated as they may intersect with other obstacles. Consider the case in Fig. 2, where the segment connecting node $v$ to one of its farthest front vertex $v_1$ of obstacle $P_2$ is obstructed by obstacle $P_1$. These obstructed links should be eliminated as they form invalid tangent they will entail undesirable collisions. We propose a solution to validate the computed tangents using the half-plane equation in (1) as shown in Fig. 2.

**Theorem 2**: Let us denote by $l$ the segment formed between a node $v$ and its farthest front vertex $v_{ffv}$ that resides on obstacle $O$. Node $v$ also forms links with farthest front vertices $v_{max}$ and $v_{min}$ on obstacle $O'$ (where $O' \neq O$) and the maximum and minimum gradient of these segments (in logarithmic representation) are denoted as $s_{max}$ and $s_{min}$ respectively.

An obstacle $O'$ is said to obstruct a segment $l$ if both of the following are true:

1. When the gradient of $l$ lies between $s_{max}$ and $s_{min}$.
2. When the co-ordinates of vertices $v_{max}$ and $v_{min}$ are substituted into the half plane equation (1) as $(x_a, y_a)$ and $(x_b, y_b)$ along with the vertices $v$ and $v_{ffv}$ individually, vertices $v$ and $v_{ffv}$ are in different half-plane regions.
Proof:

1. If the gradient of $l$ does not lie between the $s_{\text{max}}$ and $s_{\text{min}}$, then the gradient of $l$ must be greater or less than $s_{\text{max}}$ and $s_{\text{min}}$. This implies that $l$ does not intersect $O'$ and therefore, segment $l$ cannot be obstructed by obstacle $O'$.

2. If the gradient of $l$ lies between $s_{\text{max}}$ and $s_{\text{min}}$, two cases arise. In the first case, vertex $v_{ffv}$ and $v$ are located in the same half-plane region with respect to the line $v_{\text{max}}v_{\text{min}}$. Segment $l$ cannot be obstructed by $O'$ as segment $l$ does not intersect with the line $v_{\text{max}}v_{\text{min}}$. In the second case, vertex $v_{ffv}$ and $v$ are located in different half-plane regions relative to line $v_{\text{max}}v_{\text{min}}$. Segment $l$ intersects with the line $v_{\text{max}}v_{\text{min}}$ and thus, it is obstructed by $O'$.

Fig. 2: Problem and solution to the logarithm approximation method to generate visibility graph

In the example above, let’s assume that the farthest front vertices for obstacle $P_1$ have been computed to be $v_{P1}^1$ and $v_{P1}^2$. The link formed between $v$ and $v_{P1}^1$ is obstructed by $P_1$ since (1) the gradient of $v v_{P1}^1$ lies between the gradient of $v v_{P1}^1$ and $v v_{P1}^2$ and (2) $v_{P1}^1$ lies on a different half-plane region from $v$ with respect to the line formed with end-points $v_{P1}^1$ and $v_{P1}^2$. On the other hand, $v_{P2}^2$ is not obstructed by $P_1$ since it does not satisfy the first condition of Theorem 2. The overall algorithm can be summarized as follows.

Step 1: Perform binary search to find the farthest front vertices for each graph node $v$ for each obstacle. Store the gradients using logarithmic representation.

Step 2: Use the half-plane equation to eliminate obstructed segments for each node $v$. 

10
The validation of the tangents in Step 2 must be performed for every obstacle in the environment. If the number of C-obstacles is $c$, and each obstacle can be defined by $p$ number of planes, the second step has a complexity of $O(cp)$. Hence, the overall asymptotic complexity of the sequential tangents generation scheme for a point $v$ is $O(p \log m + cp)$. The algorithm given exhibits a high degree of parallelism. While Step 2 can only be executed after Step 1, operations within each of these steps can be performed independently. It is noteworthy that the degree of parallelism is limited only by the constraints imposed on hardware resources. With adequate hardware resources, binary search can be done in parallel. Similarly, one can find obstructed links for each node $v$ in parallel.

2.4 Pre-processing of Non-Convex Obstacles

The proposed techniques for gradient-based tangent construction discussed so far are for an environment comprising of convex-shaped obstacles. For the approach to be applicable to environments with both convex and non-convex objects, we propose a pre-processing step for non-convex obstacles.

The pre-processing step aims to construct a convex boundary for the non-convex obstacles. It is noteworthy that when neither the source nor target locations reside in the non-convex region of an obstacle, optimal traversals will not involve the non-convex regions of obstacles. Hence, a pre-processing step to construct the convex boundary for each of the non-convex objects will lead to the deployment of the proposed gradient-based tangent construction method for computing path traversals. A rectangular bounding box could be employed for the construction of such convex boundaries. In situations whereby the source or target location resides within the non-convex regions of the obstacles, an additional step is required to move that location to the corresponding convex boundary along the trajectory of the shortest path between the source and destination nodes. This will provide for a simple but effective solution to compute traversal paths.

Methods of identifying the convex boundary of a non-convex obstacle can vary based on the level of required accuracy and computational complexity of the path traversal process. For example, the pre-processing stage can include methods to identify the
convex hull of non-convex obstacles. This will lead to optimal path traversals but require a higher computational complexity. Alternatively, simpler methods can be adopted to identify a rectangular bounding box for the non-convex obstacles when the accuracy of path traversals can be compromised in place of computational efficiency. In the section on performance analysis in this paper, we show that the rectangular bounding box approach for transforming non-convex obstacles is highly compute-efficient.

3 Logarithmic Approximation Technique

In Section 2.2, we proposed replacing the divider that is commonly employed for gradient computation with a logarithmic function. In this section, we introduce a new hardware-efficient technique to convert the conventional number representation to the logarithm number representation.

Binary logarithms are widely researched due to the desired efficiency in complex arithmetic computations. However, the conversion from conventional number representation to the logarithm number representation and vice-versa must be performed at high-speed with desired accuracy. These conversions always involve approximations, resulting in inaccuracies, and therefore the usage of the binary logarithm is ideal in specific applications only [23]. Some interesting conversion methods from the conventional number representation to the logarithm number representation can be found in [5][9][10][11][23].

Let $N$ be the natural binary number $z_{i+1}z_{i+2} \cdots z_0$ and let $z_t$ be the most significant non-zero bit of $N$. The binary logarithm of this number is shown in (4) where $0 \leq x < 1$, $t$ is the characteristic of the logarithm and $a = \log_2(1+x)$ constitutes the mantissa.

$$\begin{align*}
\log_2 N &= \log_2 \left(2^t \left(1 + \sum_{i=0}^{t-1} 2^i z_i \right) \right) = \log_2 \left(2^t (1 + x) \right) = t + a \\
\text{As explained in [9], the binary logarithm of } N \text{ can be linearly approximated by } a = \log_2(1+x) \approx x \text{ and the maximum error for this conversion is approximately 0.086.}
\end{align*}$$
The existing conversion algorithms can be roughly classified into two categories: Look-up table (LUT) based and Computation based [23]. The LUT approach benefits from high-speed but the VLSI area increases enormously when the data length is increased. The computation approach performs the conversion using multiplication and/or division operations. The disadvantage of this approach is its slow operation speed. We now propose a new method to convert the conventional number representation to the logarithm number representation, by employing a combination of the two general approaches mentioned above, to compute the logarithmic gradients as described in (3). The anti-logarithm function is not necessary, as the actual gradient value is not required (we can work with the logarithm of the gradients).

Although the hardware implementation of the conventional linear approximation method explained in [9] is simple, it does not lend itself well to applications that require high accuracy due to its significant approximation error. In this section, we describe a method of improving the accuracy of the conventional linear approximation approach using minimal LUT and a small computation unit. Equation (4) implies that given an \( n \)-bit number, the mantissa of the binary logarithm is governed by at most \( n-1 \) bits after the most significant non-zero bit \( z_t \). Thus, there is at most \( 2^{n-1} \) unique mantissa for an \( n \)-bit number.

Our method employs a piecewise linear approximation technique by grouping all the mantissa of the binary logarithm \( N \) based on \( l \) number of binary digits following \( z_t \), namely \( z_{t-1}z_{t-2}\ldots z_{t-l} \). Each group consists of a unique range of mantissas as shown in Fig. 3 where \( l = 3 \). Hence, given a binary number \( N \), the \( l \) binary digits can be used to determine which group the mantissa resides in. Let’s denote these \( l \) bits as the LUTI (Look-Up Table Index) bits. The remaining bits \( M \), in particular \( z_{t-3}z_{t-4}\ldots z_{t-\omega} \), are used to compute the approximate value of the mantissa within that group, where \( m \) is the number of \( M \) bits after the characteristic and LUTI bits that are used to calculate a higher conversion precision within each mantissa group.
The LUT values comprise $2^l$ entries corresponding to the minimum mantissa of each LUTI group, and are denoted as $F_{\text{LUTI}}^{\text{min}}$, and another $2^l$ entries corresponding to the unique quotient $D$ of each LUTI group, in which the values are computed as shown in (5), where $F_{\text{LUTI}}^{\text{max}}$ is the maximum mantissa of each LUTI group.

$$D = \left( F_{\text{LUTI}}^{\text{max}} - F_{\text{LUTI}}^{\text{min}} \right) / (2^m - 1)$$

(5)

The approximated mantissa $a'$ of the binary logarithm of $N$ can then be computed as shown in (6), where $M$ is $\sum_{i=0}^{l-1} 2^i z_i \times 2^j$ and $j = m + l - t$.

$$a' = F_{\text{LUTI}}^{\text{min}} + (D \times M)$$

(6)

Fig. 4 shows plot of the approximated error for all values when $n = 8$. The maximum error is 0.0022, which is much lower than the linear approximation method. Moreover, majority of the errors are of the same polarity and so this would not lead to an accumulated error in the subtraction of equation (3). Since the maximum approximation error is less than $2^{-8}$, it is sufficient for an 8-bit accuracy approximation.
An 8-bit fractional accuracy translates into an approximated resolution $e$ of 0.0039 ($2^{-8}$). For example in Fig. 5, if the actual difference in the gradients of $l_1$ and $l_2$ is less than or equal to $e$ ($|l_1 - l_2| \leq e$), then the gradients are assumed to be equal.

Since this leads to the interpretation that both the endpoints of $l_1$ and $l_2$ are farthest front vertices, the traversal of the robot can be made to maintain an offset of $d$ from the object in order to avoid collision. Since the error with the proposed logarithm approximation method is considerably small, the method is suitable for use in optimal path planning.

4 Hardware Implementation

In this section, the hardware implementation of the modules for tangent construction is described. We present the architectures for three main computation units of the system, which are the Binary Search Unit, the Half-Plane Computation Unit and the Logarithmic Approximation Unit. Fig. 6 illustrates the overview architecture of the system.
The external memory stores the co-ordinates of all the vertices, the vertices of the respective obstacles, and the maximum/minimum gradients along with the farthest front vertices of each vertex with respect to the obstacles in the C-space.

![Diagram of the tangent computation system](image)

The Binary Search Unit is used in two stages of the algorithm. In the first stage, it is used along with the Half Plane Computation Unit to identify the edge of an obstacle that intersects with a line adjoining a vertex and the first vertex of the obstacle. In the second stage, the Binary Search Unit is used along with the Logarithmic Approximation Unit to determine the maximum and minimum gradients. Fig. 7 describes the computational core of the Binary Search Unit.
The Binary Search Unit comprises of two registers to store the current offset value (which is the value of the previous iteration divided by 2), a one-bit shifter to emulate the divide by 2 operations, some adders to compute the new index of the vertex to be considered, and a multiplexer to select the appropriate index for the different processes. Initially, the offset register is loaded with the length of the binary search array. In each of the subsequent iterations, a new index is obtained and the registers are updated with the current offset and index values.

Based on equation (7), a possible implementation of the Half Plane Computation Unit is illustrated in Fig. 8. The complexity of the architecture for the half-plane can be reduced by normalising the equation in (1) to either \((x_a,y_a)\) or \((x_b,y_b)\). The Half Plane Computation Unit is also used along with the Final Tangent Computation Unit (which comprises of a comparator and some registers) to validate the computed tangents.

\[
\begin{align*}
\text{From 1, } & \quad (x \times y_a) - (x \times y_b) + (y \times x_b) - (y \times x_a) - (y_a \times x_b) + (x_a \times y_b) \\
& = (x - x_b) \times (y_a - y_b) - (x - x_b) \times (y_b - y_a) + (y - y_b) \times (x_b - x_a) - (y - y_b) \times (x_a - x_b) + (x_a - x_b) \times (y_b - y_b) \\
& = (x - x_b) \times (y_a - y_b) - (y - y_b) \times (x_a - x_b)
\end{align*}
\] (7)
Lastly, we look at the Logarithmic Approximation Unit, which is shown in Fig. 9. It is based on a system, which requires 8-bit binary data conversion. Since the objective of this unit is to determine the logarithmic approximation of the gradient with end-points of vertices \((x_a, y_a)\) and \((x_b, y_b)\) (as explained in equation (3)), a subtraction is first performed on the \(x\) and \(y\) values respectively. The priority encoder determines the position of the first non-zero bit \(z_t\) from the results of the subtraction. By determining the characteristic \(t\), the appropriate values from the LUT and \(M\) can be obtained. As explained in the previous sub-section, the look-up table consists of 8 entries for \(F_{\text{LUT}}^{\text{min}}\) and another set of 8 entries for \(D\).

Using equation (6), the approximate values to be subtracted can be obtained. These values are then substituted into (3) to determine the maximum/minimum gradient. Note that the sign bits from \((x_a, y_a)\) and \((x_b, y_b)\) are used to determine whether the gradient (in logarithmic representation) is positive or negative.
The modules were implemented in VHDL and synthesized using Synopsys Design Compiler 2001.08-SP1 with Compass Passport CB35OS142 standard cell library in 0.35-micron CMOS process. The operating conditions are set to normal case with the input delays as the set-up time of a data register and the output load as four times the driving load of a data register. Fig. 10 shows the synthesis results of the critical path are area utilization for the Binary Search Unit, half Plane Computation Unit and Logarithmic Approximation Unit. With the unit for the area as a 2-input NAND gate, the total area (including the combinational, non-combinational and net interconnect area) was found to be only about 7K gates. The latency for each of the modules is only about 7.2 ns and hence, the data obtained suggest that the proposed technique for tangent computations lends itself well to cost-effective and high performance implementation.

<table>
<thead>
<tr>
<th>Modules</th>
<th>Area (NAND gate)</th>
<th>Latency (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Comb</td>
<td>Non-Comb</td>
</tr>
<tr>
<td>Binary Search</td>
<td>808.91</td>
<td>297.56</td>
</tr>
<tr>
<td>Half Plane Computation</td>
<td>2451.26</td>
<td>356.19</td>
</tr>
<tr>
<td>Logarithmic Approximation</td>
<td>983.27</td>
<td>203.26</td>
</tr>
</tbody>
</table>

Fig. 10: Synthesis results for Binary Search, Half Plane Computation and Approximation Unit
5 Performance Comparison

In this section, we will estimate the performance gain of the proposed hardware implementation over an existing approach that is reported in [16]. Fig. 11 shows the environment that is used for the performance analysis. The environment consists of mostly convex shaped obstacles and we assume that the source and target locations do not reside within the non-convex regions of the obstacles. As discussed in Section 2.4, the pre-processing stage involves the formation of rectangular bounding boxes for the non-convex obstacles.

![Fig. 11: Environment for performance estimation](image)

Let’s denote by \( n \) the number of vertices in the environment, \( m \) the maximum number of vertices of an obstacle, \( c \) the number of obstacles in the environment, and \( u \) the number of non-convex obstacles. Fig. 12 shows the maximum number of clock cycles required by the various stages of the proposed implementation based on the parameters \( n, m, c, \) and \( u \). It is assumed that a clock cycle is sufficient for a complete computation cycle of the modules in Fig. 7, 8 and 9. Hence, the clock cycle must have a minimum period of 7.2 ns.

<table>
<thead>
<tr>
<th>Computation Stage</th>
<th>Number of clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Identification of the rectangular bounding box of non-convex obstacles</td>
<td>( 10mu )</td>
</tr>
<tr>
<td>2) Computation of the intersection point ( k ) (see Fig. 1)</td>
<td>( 3n \log m )</td>
</tr>
<tr>
<td>3) Computation of the farthest front vertices</td>
<td>( n(2 + 8\log m) )</td>
</tr>
<tr>
<td>4) Validation of computed tangents</td>
<td>( 7nc )</td>
</tr>
</tbody>
</table>

![Fig. 12: Maximum clock cycles for proposed implementation](image)
In the proposed implementation, the execution time to compute the tangents for the environment in Fig. 11 with a clock frequency of 100 MHz (10 ns) is approximately 0.130 ms. When compared to the moving line algorithm reported in [16], which requires a total of 2500 ms running on the Apollo DN 3500, an order of magnitude improvement in speed-up is observed.

In order to project the performance improvement over a recent microprocessor such as the Pentium IV 1.7 GHz processor, we use the LINPACK benchmark Mflop/s data for the Apollo DN 3500 and Pentium IV in [1]. Based on this, the projected performance of the moving line algorithm running on an Intel Pentium IV processor is estimated to be about 1.72 ms, confirming that the proposed architecture is about 13 times faster. Moreover, the proposed architecture does not require the extensive resource support of the Pentium IV processor. This together with the potential for additional speed-up through parallel computations makes the proposed technique appropriate for low-cost applications that demand high-performance.

When compared to the technique presented in [16], the proposed approach can achieve a significant performance speed-up for the following reasons:

1. It adopts a simple approach to eliminate tangent computations of non-convex obstacles.
2. Only a reduced number of vertices are involved due to the binary search method.
3. It employs a hardware efficient logarithmic approximation technique for gradient computations that is fast with tolerable error. The modules for binary search and validation of computed tangents can also be easily implemented in hardware.

Since the proposed technique relies on logarithmic approximations, and computations are performed on the rectangular bounding box for non-convex obstacles, there may be a small amount of error. However, as discussed in Section 3, the maximum error incurred from the logarithmic approximation is very small and can be easily accounted for within the predefined traversal offset. Moreover, it has been shown to be well within the required tolerance for motion planning of mobile robots or robot manipulators.
The reliability of the proposed hardware implementation for tangent graph construction is also high as it eliminates any intermediate layers such as an operating system. Unlike general-purpose processor based implementations, the proposed hardware alternative lends well for a highly responsive system with relatively lower clock rates, thereby resulting in a highly reliable and stable solution, which is crucial to robust path planning applications.

6 Conclusions

VLSI-efficient techniques to generate the tangents for motion planning of mobile robots and robot manipulators have been proposed. The proposed method employs an efficient binary search and logarithmic approximation technique to compute the farthest front vertices of the polygonal obstacles and it is shown to cope with non-convex obstacles. The computed tangents are then validated using the half plane computation approach. The maximum error for the approximation method is only 0.0022 and it requires a very small look-up table to compute base-2 logarithm of 8-bit natural numbers. In addition, it has been demonstrated that the employment of the binary search method leads to notable reduction in the number of logarithmic approximation computations, particularly for high dimension configuration space. Techniques based on half-plane computations lend itself well for the rapid identification and elimination of segments that are obstructed by other objects. Suitable architectures have been developed and synthesis results show that the architectures lend well to cost effective and high performance realisations. Moreover, they exhibit a high degree of parallelism at the architectural level and are thus well suited to high-speed dynamic motion planning.

References


Lam Siew-Kei has been with NTU (Nanyang Technological University, Singapore) since 1994, where he received his B.A.Sc (Honors) and M.Eng in Computer Engineering. He currently holds an appointment as Research Associate in the Centre for High Performance Embedded Systems and has worked on a number of challenging projects that involved the porting of complex algorithms in VLSI. He is also familiar with rapid prototyping and ASIC design flow methodologies. His research interests include algorithms-to-architectural translations, high-speed arithmetic units, and embedded system design algorithms and methodologies. He is a member of IEEE.

Dr. K. Sridharan graduated with a B.E (Honors) from Regional Engineering College (now NIT), Tiruchirapalli, India in 1987. He then received his M.S from Indian Institute of Science, Bangalore in 1990 and his PhD from Rensselaer Polytechnic Institute, Troy, New York in 1995. From March 1996 to May 2001, he was with the Department of Computer Science and Engineering, IIT Guwahati, India as an Assistant Professor. Since June 2001, he has been with the Department of Electrical Engineering, IIT Madras, India where he is presently an Associate Professor. He was a visiting staff member at the School of Computer Engineering, Nanyang Technological University, Singapore from May 2000 to May 2001. Dr. Sridharan was a joint recipient of the IEEE Vincent-Bendix Award. He also received the IEEE Outstanding Leader Award for contributions to his student branch. He has been a reviewer for IEEE Transactions on Robotics and Automation, IEEE Transactions on Education and Journal of Robotic Systems. He is a Senior Member of IEEE.

Dr. Thambipillai Srikanthan has been with NTU (Nanyang Technological University, Singapore) since 1991, where he holds a joint appointment as Associate Professor and Director of the Centre for High Performance Embedded Systems. He received his B.Sc. (Hons) in Computer and Control Systems and PhD System Modelling and Information Systems Engineering from Coventry University, United Kingdom. His research interests include System integration methodologies, Architectural Translations of Compute Intensive Algorithms, high-speed techniques for Image processing and dynamic routing. Dr. Srikanthan has published more than 150 technical papers and has served a number of administrative roles during his academic career. He is the founder and Director of the Centre for High Performance Embedded Systems, which is now a University level research Centre at NTU. He is a corporate member of the IEE and a senior member of the IEEE.