Abstract: - The Bug algorithm is a local path planning methodology which detects the nearest obstacle as a mobile robot moves towards a target with limited information about the environment. It uses obstacle border as guidance toward the target. In Bug algorithm, the robot circumnavigates the obstacle till it finds certain condition to fulfill algorithm criteria to leave the obstacle toward target point.

This paper introduces an approach utilizing a new algorithm called PointBug that attempts to minimize the use of outer perimeter of an obstacle (obstacle border) by looking for a few important points on the outer perimeter of obstacle area as a turning point to target and finally generates a complete path from source to target. The less use of outer perimeter of obstacle area produces shorter total path length taken by a mobile robot.

This approach is then compared with other existing selected local path planning algorithm for total distance and a guarantee to reach the target.

Key-Words: - Path Planning, bug algorithm, autonomous robot, sensor based, mobile robot

1 Introduction
Path planning is one of the most important elements for mobile robot. Path planning is the determination of a path that a robot must take in order to pass over each point in an environment [1, 2] and path is a plan of geometric locus of the points in a given space where the robot has to pass through [3]. Generally, the problem of path planning is about finding paths by connecting different locations in an environment such as graph, maze and road. Path planning “enables” mobile robots to see the obstacle and generate an optimum path so as to avoid them.

The general problem of path planning for mobile robots is defined as the search for a path which a robot (with specified geometry) has to follow in a described environment, in order to reach a particular position and orientation B, given an initial position and orientation. As mobile robot is not a point in space, it has to determine the correct direction or perform a proper movement to reach destination and this is called manoeuvring planning.

PointBug Algorithm tries to reduce the usage of outer parameter of obstacle as implemented in Bug Algorithm. As an example, in Bug1, the coverage of circumnavigating of obstacle is more than the size of perimeter of the obstacle and meanwhile for Bug2, the maximum coverage is equal to the total perimeter size of obstacle. By avoiding circumnavigating the obstacle, the problem PointBug is to find next point to go toward target point. It has to determine where the next point should be located on the outer parameter of obstacle. In PointBug Algorithm robot is assumed equipped with an infinite range sensor, odometer and digital compass with ideal positioning. The range sensor gives a reading to controller for interval period and action is executed based on the difference in reading of two sequences of times. Then robot moves to the new sudden point according to the angle of robot rotation and limited by the odometer.

2 Bug Algorithms
Bug algorithms are well known mobile robot navigation method for local path planning with minimum sensor and simple algorithm. James Ng and Thomas Bräunl listed about eleven types of bug algorithms [4]. The most commonly used and referred in mobile robot path planning are Bug1 [5], Bug2 [5], DistBug [6], VisBug [7] and TangentBug [8]. Others bug algorithms are Alg1 [9], Alg2 [9], Class1 [10], Rev [11], Rev2 [11], OneBug [4] and LeaveBug [4].
The variations of bug algorithms showed the effort toward shorter path planning, shorter timing, simpler algorithm and better performance. Bug1 moves from start point toward target point by hitting and circumnavigating the obstacle then leaving the leave point. Bug2 has similar behaviour except it is guided by m-line where m-line is used as leaving point and hitting point. Bug1 is considered overcautious and having coverage more than the full perimeter of the obstacle yet effective meanwhile Bug2 is inefficient in some cases such as local loops but shorter coverage compared to Bug1.

The first bug family algorithm that incorporates a range sensor is Visbug [7] which calculates shortcuts relative to the path generated by the Bug2 algorithm from or to m-line. Alg1 [12, 13] improved Bug2 weakness is that it can trace the same path twice by storing the sequence of hit points occurring within an actual path to the goal. These storing data are used to generate shorter paths by choosing opposite direction to follow an obstacle boundary when a hit point is encountered for the second time. The same researcher introduced the Alg2 [12] to improve Alg1 by ignoring the m-line of Bug2 with new leaving condition. The Alg1 and Alg2 still face a reverse procedure problem where after encountering a visited point that causes loop, a mobile robot follows an uncertain obstacle by an opposite direction until it can leave the obstacle. [14] solved the reverse procedure problem in Alg1 and Alg2 by introducing a mixing reverse procedure with alternating following method to create shorter average bound of path length and named the algorithm as Rev1 and Rev2. Alternating following method is defined as independently, if a robot always changes a direction following an uncertain obstacle alternatively, the robot arrives at a destination earlier on average and there will decrease probability for the robot to join a loop around a destination.

Other bug algorithms that also incorporate range sensors are DistBug algorithm and TangentBug Algorithm. The DistBug algorithm is a local path planning algorithm that guarantees convergence and will find a path if one exists. It requires its own position by using odometry, goal position and range sensor data. To guarantee convergence to the target, the DistBug algorithm needs a small amount of global information for updating dmin(T) and for determining that the robot completed a loop around an obstacle. The value of dmin(T) can be extracted directly from the visual information. This guarantee convergence using updating dmin(T) value makes problem in determining accuracy because the value of dmin(T) is taken from direct global visual information.

Meanwhile, the TangentBug is another variations of DistBug that improves the Bug2 algorithm in that it determines a shorter path to the goal using a range sensor with a 360 degree infinite orientation resolution [15]. Tangent Bug incorporates range sensors from zero to infinity to detect obstacles. When an obstacle is detected, the robot will start moving around the obstacle and will continue its motion-toward target point routine as soon as it has cleared the obstacle. During following boundary, it records the minimal distance to target dmin(T) which determines obstacle leaving and reaching condition. While the robot is moving towards target, d(x,T) decreases monotonically and boundary following attempts to escape from a local minimum of d(x,T). The robot constructs a local tangent graph (LTG) based on its sensors’ immediate readings. The LTG is constantly updated and it is used by the robot to decide the next motion. The disadvantage of this algorithm is requiring robot to scan 3600 before making decision to move to the next target. The latest variant of TangentBug is LadyBug [16] which incorporates bio-inspired heuristics to improve the robot trajectory in real time based on Ladybugs hunt for aphids for a group of networked mobile robots. Figure 1 shows the different of paths taken among foure bug algorithms.

An extension to classical Bug based algorithms called Sensbug was introduced which can produce an effective path in an unknown environment with both stationary and movable obstacles [17]. CBug applies the Bug1 behaviour in its algorithm with online navigation algorithm for a size D disc robot moving in general planar environments [18, 19]. The algorithm searches a series of expanding ellipses with focal starting point and target point, and its total path length is at most quadratic in length of the shortest offline path [20]. K-Bug algorithm [21] consumes global information such as obstacles geometry and position to select the waypoint among the vertex of obstacles that caused collisions. This algorithm does not use sensor to get the environment information and the information needs to change every time the environment is changed.
3 Pointbug Algorithm

PointBug, recently developed navigates a point of robot in planar of unknown environment which is filled with stationary obstacles of any shape. It determines where the next point to move toward target from a starting point. The next point is determined by output of range sensor which detects the sudden change in distance from sensor to the nearest obstacle. The sudden change of range sensor output is considered inconstant reading of distance either it is increasing or decreasing. It can be from infinity to certain value or certain value to infinity or certain value to a certain value where the difference value, \( \Delta d \) is defined. If value of \( \Delta d \) is defined for 1cm, any reading from range sensor from interval time, \( t_n \) to \( t_{n+1} \) which detects the different in range for 1cm and above is considered a sudden point.

![Fig. 2: Range sensor is detecting an obstacle from left to right and right to left.](image)

The robot is capable to scan the environment using range sensor by rotating itself from 00 up to 3600 at a constant speed. The initial position of robot is facing straight to the target point and then the robot rotates left or right searching for sudden point. After the first sudden point is found, the rotation direction of the robot is according to position of straight line between current sudden point and target point or \( d_{\min} \) line. The rotation direction of robot is always toward position of \( d_{\min} \) line. The value of \( d_{\min} \) is the shortest distance in one straight line between sudden point and target point and its value is always recorded every time the robot reaches new sudden point. The robot always ignores the sensor reading at rotation of 1800 to avoid detection of previous sudden point making the robot return to previous sudden point from its current point. If there is no sudden point found within a single 3600 rotation, the target is considered unavailable and the robot stops immediately.

The pseudo code of the algorithm as follows:

```
While Not Target
    If robot rotation <= 360
        Robot rotates right of left according to position of \( d_{\min} \)
        If sudden point
            If 180 degree rotation
                Ignore reading
            /* to avoid robot return to previous point */
            Else
                Get distance from current sudden point to next sudden point
                Get angle of robot rotation
                Move to new point according to distance and rotation angle
                Record New \( d_{\min} \) value
            End if
        End if
    Else
        Robot Stop /* No sudden point and exit loop */
    End if
End while
Robot Stop /* Robot successfully reaches target */
```

![Fig. 3: Sudden points on different surfaces detected by range sensor.](image)

Figure 2 shows a range sensor scanning a pentagon shaped obstacle from A to E with a graph showing the distance produced from range sensor in cm from A to E. The C line is perpendicular to the surface of obstacle which is the shortest distance detected to the obstacle. The value of distance increases constantly from C to B and from C to D. From point B to A from the graph, the value of distance is suddenly increased almost twice and from point D to E the value of distance is suddenly...
increased from a few centimeters to infinity. The point A and E are the sudden points and considered the points where the robot will move for the next point. Figure 3 shows the sudden points are detected on different shape of obstacles.

![Figure 3: Sudden points detected on different shape of obstacles.](image)

Fig. 4: Trajectory generated by the PointBug to solve local minima problem. The shadowed area is scanned area.

Fig. 4 shows how the algorithm is working in an environment to solve local minima problem by detecting sudden points from a starting point to target point. The robot first faces the target point at the starting point and then rotates from point A until it finds a sudden point at point B. Robot then move to point B and at point B, it rotates to the right direction to find next sudden point because the dmin line is located right side of current robot direction and finds new sudden point at C. Robot rotates to the right again at point C and finds new sudden point at D. At point D, the robot still rotates to the right and finds last sudden point and stop at target point.

Table 1: Explanation on how sudden points are found from Starting point to Target from the Fig 4.

<table>
<thead>
<tr>
<th>Point of Movement</th>
<th>Sudden point Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A to B</td>
<td>B (from certain value to infinity)</td>
</tr>
<tr>
<td>B to C</td>
<td>C (from infinity to certain value)</td>
</tr>
<tr>
<td>C to D</td>
<td>D (from infinity to certain value)</td>
</tr>
<tr>
<td>D to target</td>
<td>Infinity to target</td>
</tr>
</tbody>
</table>

3.1 Algorithm Analysis

The main goal of the algorithm is to generate a continuous path from start (S) to the point target (T) and S and T is fixed. The distance between two points is denoted as d(A, B) and for this case specifically, d(S, T) = D, where D is a constant. d(An, B) signifies that point A is located at nth sudden point on the way to T, and P is total length of connected sudden points from S to T. The line (S, T) is called the main line or m-line.

As all path generated by the algorithm are straight lines, robot position is measured by d(x, y) and the total distance can be calculate by totalling all straight lines distance those connect sudden points.

\[
P = \sum_{n=1}^{s+1} (A_{n-1}, A_n)
\]

In PointBug algorithm, every sudden point found will produce a logical triangle which is formed from three points namely target point, current sudden point and previous sudden point. The line between target point and current sudden point is dmin line and its values are accumulated in an array starting from 0 which is distance from starting point and target point up to last sudden point before meeting target point. Value dmin[0] is assigned manually and it is the initial value required to run the algorithm. The values of dmin[1] to dmin[n] are obtained from cosines rule except dmin[0].

\[
a^2 = b^2 + c^2 - 2bc \cos A
\]

if a is d\text{\_min} then;

\[
d_{\text{min}} = \sqrt{b^2 + c^2 - 2bc \cos A}
\]

In equation (3), the value of b is distance between current sudden point and previous sudden point which is obtained from range sensor and the value of c is previous value of dmin, then dmin[n] is;

\[
d_{\text{min}}[n] = \sqrt{b^2 + d_{\text{min}}[n-1]^2 - 2b \times d_{\text{min}}[n-1] \times \cos A}
\]

The value of \(\angle A\) is obtained from rotation of the robot from current direction to next direction if the robot located on the starting point, otherwise:

\[
\angle A = 180 - \angle \text{Adj} - \angle \text{Rot} \quad \text{if} \; \angle A \leq 90
\]

\[
\angle A = \angle \text{Adj} + \angle \text{Rot} \quad \text{if} \; \angle A > 90
\]

where is \(\angle \text{Adj}\) Adjacent angle of triangle and \(\angle \text{Rot}\) is the robot rotation angle. \(\angle \text{Adj}\) value is obtained from sine rule;

\[
\frac{b}{\sin B} = \frac{a}{\sin A}
\]

If \(\sin B\) is \(\angle \text{Adj}\) then
\angle Adj = \sin^{-1} \left( \frac{b \sin A}{a} \right)
\tag{7}

where the \( b \) is previous \( d_{\text{min}} \) value and \( a \) is current \( d_{\text{min}} \) value.

Lemma 1: if \( d_{\text{min}}[n] = 0 \), the robot currently is on the target point.

Proof: \( d_{\text{min}}[n] \) is the minimum distance between sudden point and target point. If its value is zero means the sudden point is on the target point, the value of \( A \) is zero and the value of previous \( d_{\text{min}}[n] \) is equal to distance between current sudden point and previous sudden point or \( c \). Let’s say value of previous \( d_{\text{min}}[n] \) is \( b \), and from equation (3), the value of \( d_{\text{min}}[n] \) is:

\[
d_{\text{min}}[n] = \sqrt{b^2 + b^2 - 2bb\cos 0}
\]

\[
d_{\text{min}}[n] = \sqrt{2b^2 - 2b^2}
\]

\[
d_{\text{min}}[n] = 0
\]

4 Simulation and Results
The simulation of point to point bug algorithm is carried out using ActionScript 2.0 on Adobe Flash CS3. The algorithm is simulated on three types of environments namely free environment, maze based environment and office like environment.

Fig. 5 shows that sudden points are generated at every vertex of the rectangle. In this environment, the algorithm generates the shortest path from starting point to target point.

Fig. 6 and figure 7 show comparison of three algorithm namely Distbug, TangenBug and PointBug with each robot equipped with unlimited sensor range. TangentBug and PointBug produced almost the same result but TangentBug makes a little obstacle following increases the total path distance taken compared to PointBug algorithm. In an office like environment, tangentBug algorithm outperforms other algorithms. The performance of pointBug varies according to types of environment and position of obstacle.

5 Discussion and Conclusion
This paper compares 2 main path planning algorithms from the bug algorithm family. This new approach of path planning, The Point to Point Sensor Based Path Planning algorithm is a new approach that behaves similar to other algorithms in the bug family. However, the Point to Point Sensor Based Path Planning Algorithm needs very minimal amount of prior information namely \( d_{\text{min}}(T) \) and \( \theta_{rel}(T) \) compared to other bug algorithms such as DistBug and VisualBug algorithm which need global information to update value of \( d_{\text{min}}(T) \) during the boundary following and
determine completion of a loop of robot to ensure convergence to target.

The algorithm can operate in dynamic environment as well because information about the environment can be obtained immediately from the range sensor during the movement of the robot. The performance of the algorithm depends on total sudden points detected. The less number of sudden points detected is better. Thus, whether it outperforms other bug algorithms depends on obstacles in the environment as if the obstacle is a circle, it will produce less sudden point since a circle has no vertex. The total vertex in obstacle affects the total sudden points.

6 Acknowledgement

The authors gratefully acknowledge and thank University Technology MARA, Malaysia (UiTM) for providing a grant for this research. [600-RMI/ST/DANA 5/3/Dst (234/2009)].

References:


