Maximum Clearance Rapid Motion Planning Algorithm

Citation: S.S. Paliwal, R. Kala (2018) Maximum clearance rapid motion planning algorithm. Robotica 36(6): 882-903.

Final Version Available At: https://www.cambridge.org/core/journals/robotica/article/maximum-clearance-rapid-motion-planning-algorithm/F0E3C50897CF192967BCB1789E511CA3

***Abstract* -**This paper proposes a new path-planning algorithm which is close to the family of bug-algorithms. Path planning is one of the challenging problems in the area of service robotics. In practical applications, traditional methods have some limitations with respect to cost, efficiency, security, flexibility, portability, etc. Our proposed algorithm offers a computationally inexpensive goal oriented strategy by following a smooth and short trajectory. The paper also presents comparisons with other algorithms. In addition, the paper also presents a test bed which is created to test the algorithm. We have used a 2-wheeled differential drive robot for the navigation and only a single camera is used as a feedback sensor. Using an Extended Kalman Filter, we localize the robot efficiently in the map. Furthermore, we compare the actual path, predicted path and planned path to check the effectiveness of the control system.

***Keywords* -- Path Planning, Bug algorithm, Extended Kalman Filter, Homography, Localization, PID controller**.

Notations Used:

r: Radius of the circle.

$q(t)$: Centre of the circle at time *t*.

$B_r\big(q(t)\big)$: Circle of radius *r*, centred at $q(t)$

$d\big(q(t), q(t+1)\big)$: Distance between points $q(t)$ and $q(t+1)$

L: Line joining the source and goal location.

$r_{threshold}$: Minimum radius

r_{max} : Maximum radius for the initial obstacle-free circle

W^{obs} : Obstacle-prone workspace

τ: Path stored as a list of control points.

ϕ: Empty set

$q.R_z(r, \Theta)$: Rotation around the point q in the Z axis, with the radius *r*, at an angle of Θ

q_l, q_r : Propagating circle's centre for the left and right branch respectively

q_{lp}, q_{rp} : Pivot circle's centre for the left and right branch respectively

X_l, X_r: Rolling status of left and right branch respectively

p_l, p_r: Whether left and right branch have reached goal.

$\alpha = r(t+1)/r(t)$: An algorithm parameter, ratio of the new radius of propagating circle with respect to the previous radius.

I. Introduction

The problem of path planning is an integral part of mobile robots. The problem of path planning is to determine a path that the robot must take in order to achieve a desired configuration, and in case of a mobile robot, to reach the destination [1-3]. A path is a set of generated configurations in the configuration space which have to be sequentially followed to reach the goal [4]. Since the space may contain obstacles or constrained paths, planning should generate an optimum path which obeys every physical constraint present in the environment. Path planning is an NP-hard problem, where we can verify the optimality of the solution, but finding the optimal solution is very difficult and a computationally expensive problem. Also, the complexity increases exponentially with an increase in the dimensions of the environment (space). Thus, the path planning algorithm should be able to provide a feasible path and should also be computationally inexpensive.

Our paper provides a relatively fast algorithm, whose motivation comes from the following analogy. Suppose a person is lost in the middle of a forest, and all that he/she has is a compass and a map. And he/she knows the direction in which he/she has to head to find a safe location. Then, the best way to reach the destination will be to follow the shortest path, i.e. the one joining the current location and the destination. But, practically, life is not that easy, and there are obstacles in the way. Then the best possible way to avoid the obstacles, is to go around them, until again, we find the main track, and do this until we reach the destination.

The mathematical model of our algorithm falls roughly into the family of bug algorithms [5][6], which have two main modes of operation: following obstacle boundaries and moving towards the goal. But unlike the bug algorithm, we assume our robot to be contained inside a circle, which propagates towards the destination. The variable-sized circle used ensures that there is a maximum clearance and the obtained path is also relatively smooth. There is an algorithm parameter, α, which is a ratio and determines the radius of the circle in the next iteration. Now, if α is close to 1, the radius of the final circle becomes a linear function of minimum clearance between the obstacles. This results in a smoother path having a much better clearance. Further, while the Bug family of algorithms are entirely reactive in nature and can thus result in extremely poor paths, some deliberation is added into the designed algorithm to make the algorithm generate shorter and acceptable paths.

A good planning algorithm has various constraints like environment noise, hardware errors, etc. which come into the picture when they are applied to a physical system. Efficiently controlling the robot is one of the most challenging tasks for successful navigation of the robot. After getting a path, we can generate control signals and feed the robot, which it will use to navigate the path. Now, everything seems fine, until the question pops, what is the guarantee that the robot has reached the destination/desired state? Has the robot truly followed the desired path? Thus, the best control system should have a feedback mechanism, which can monitor the robot and can execute the desired plan in accordance to the situation. Precise localization of the robot is very important for an efficient feedback system [7]. Due to various sources of noise, the sensors of the robot may not always be correct. Thus, we use an Extended Kalman Filter [8][9], which is used to predict the best possible state of the robot, in case the sensors show fluctuant/inconsistent readings. Using a combination of

different sensors and filters, localization can be made very robust, which results in a better controlling of the system.

After getting the localized value from the filter, controlling should be done. The control system should be designed in accordance to all the physical constraints that the robot possesses. We have used the P, PD and PID variations of controllers in our test bed. The coefficients of the P, I and D components of the PID controller [10] are obtained by using a simulator, in which the real world robot is precisely mapped to the virtual robot. This is how the simulator's PID coefficients also work fine with the real physical robot.

The main intention behind the work is to make an algorithm that can be used by very cheap indigenously developed robots. Even though research has led to a lot of advancements in robotics, the adoption of robots in homes is limited due to different factors, one of them being the monetary cost. We see a future wherein cheap robots will be doing much of the household work. For the same this approach is designed and tested for very cheap robots which have a very high noise in actuation and sensing, and are assisted by a cheap external camera.

The algorithm can be seen as a hybrid of deliberative and reactive mechanisms for planning. The deliberative algorithms are computationally expensive, but optimal and complete; while the reactive algorithms are computationally less expensive, but miss out on the lines of optimality (and sometimes completeness). Various schemes for the fusion have been proposed in the literature realizing the relative pros and cons of the two methodologies. Most of these operate in two or multiple layers of hierarchy. Here a scheme is designed, that does not take two different algorithms as a deliberative and a reactive component; however, takes a reactive algorithm as a base and adds deliberation on top of it for judicious decision making. Not only does the scheme mix the relative pros and cons of deliberative and reactive planning, it also enables us to solve the problem of clearance in the selected reactive planner.

The main contribution of the work is to propose an algorithm which generates shorter paths and paths with a very high clearance as compared to the Bug algorithms, while taking a little extra computational time. The proposed algorithm is tested in an indigenously designed cheap mobile robot which is prone to extremely large actuation and sensing errors, and tested against low computational expense. The attributes of the algorithm perfectly match the requirements of such a robot. Further, a test-bed is made consisting of a video camera for feedback, an EKF for localization and a PID controller for control. Comparing the algorithm to other algorithms stresses upon the utility of the algorithm for low cost mobile robots with a high noise and limited computation facilities, where optimal paths may not be a strict criterion and paths with acceptable lengths may suffice.

There have been numerous attempts in developing fast motion planning algorithms. In a recent work by Salzman and Halperin [11], a hybrid fast algorithm, lower bound tree-RRT (LBT-RRT) was developed by combining a fast RRT algorithm and asymptotically optimal RRT* and RRG algorithms. An approximation factor was used to maintain the weights between RRT and RRG, and the resultant approach was shown to produce better results than RRT and executed faster than RRT*. In this single query approach, they used a combination of two roadmaps by maintaining the tree from RRG roadmap and an auxiliary graph (lower bound graph).

There has been a thorough investigation of robot path planning methods [12]. In a related work Sezer and Gokasan [13] proposed a mechanism for the robot to find large gaps between obstacles geometrically and made the robot go through these gaps. In another work, Alvarez and Sanchez [14]

also assess the gap using the robot's sensors and made the robot travel in the direction of the largest gap. The algorithms maximize clearance; however, the approach is purely reactive in nature, meaning that the robot can get stuck.

More interesting works are done in the use of deliberative approaches to motion planning, especially the approaches using hierarchical and multi-level decompositions. Such as the work of Zhang et al. [15], in which they proposed a new approach by using ACD (approximate cell decomposition) to divide the configuration space into cells and computed localized roadmaps by generating samples within these cells. Then, using localized roadmaps, a connectivity graph for adjacent cells with pseudo free edges was generated. Thus, using the connectivity of free space, an adaptive subdivision algorithm was proposed. In another work using multi-level decomposition approach [16], the LPA*(Lifelong planning A*) algorithm was made more robust and less computationally expensive. The algorithm utilized the pre-computed multiscale information of the environment to create the related search graph of a smaller size. Thus, it reduced the computational complexity of the LPA*.

While comparing the algorithm in terms of finding a path of maximum clearance, the best algorithm is the Voronoi diagram. The Voronoi diagram has a very large computation time when the map is available as a grid map with obstacles not defined as polygons. In contrast, the proposed algorithm visits a significantly lesser proportion of the space, giving results significantly earlier.

Another approach using hierarchical decomposition method was proposed by Cowlagi and Tsiotras [17]. They proposed a procedure in which a graph-search algorithm operated on a sequence of vertices and a lower level planner ensured consistency between the two levels of hierarchy by providing meaningful costs for the edge transitions of a higher-level planner using dynamically feasible and collision-free trajectories. The hierarchical solution of these two sub-problems were efficient. In another related work Kala et al. [18] used a coarser deliberative planner based on cell decomposition, and a reactive planner at the finer level for the navigation of the robot. The algorithm is largely-trap free, but the initial cell decomposition is highly resolution dependent and can be computationally expensive.

Apart from the motion planning algorithms, there has also been a lot of work on mapping, localization and control. In the work of Savkin and Li [19], a Pioneer-3dX robot mounted with range finder sensors was used for map building in an unknown environment having obstacles. A randomized algorithm was built for robot navigation. In a related work, William and Pinhas [20] worked on the confluence of mapping, localization and controlling of multi-robotic systems. They presented methods for integrated exploration and SLAM in multi robotic systems. Also, different benefits/trade-offs of multi-robot system design were also presented. In other work by Maddahi et al. [21] on control and calibration, the authors presented a method for calibrating a mobile robot having a differential drive, which corrects for the systematic errors. The proposed method was compared with the Universiry of Michigan Benchmark (UMBmark) odometry method on two sets of comparisons. The first set of tests, compared the workability and accuracy between the proposed method and UMBmark techniques, while the second test compared the performance of mobile robot, calibrated by any one of the techniques, while moving on an unknown path. In another related work by Hwang and Lee [22], laser range finder was used to build a 2-D map using motion-free ICP algorithm. The consecutive data frames acquired from the laser range finder were used to formulate a least square problem, having constrained rotation and translation. The optimal data acquired at each time-frame was integrated to build a robust 2-D map.

The paper is organized as follows. Section 2 gives an overview of the search and bug algorithms. Section 3 introduces our proposed algorithm. Then in Section 4, we describe our test-bed in detail. Section 5 describes the results and discussions of the proposed algorithm, along with comparisons with other algorithms. Finally Section 6 concludes the work.

2. Overview of Search and Bug Algorithms

In most of the algorithms, the effectiveness of the algorithm depends upon the resolution of the configurational space. The optimal algorithms, A* and Dijkstra are also resolution-optimal. The path planned by these algorithms is optimal within the limits of the resolution of the environment. This however can be improved by using multi-resolution quad tree techniques [23, 24], in which a two-dimensional environment is recursively decomposed into its four uniformly sized regions. The nodes of the tree are classified as a free node and an obstacle node. In a related work [25], the resolution was incrementally adapted to make an anytime algorithm using a graph search. The approaches reduce the computational time, but still the optimality of the path is lost. Recursively dividing the map creates the simplified grid, on which algorithms like A* would generate the path faster but the resolution is a function of the closest distance between the obstacles. Since the decomposition happens independent of the source and goal, so theoretically in the worst case it is similar to working on the un-decomposed original map for scattered small obstacles. Apart from this, verifying the sub-region (nodes) between obstacle and free node is a cumbersome process both computationally and memory wise.

Motivated by the same, in our algorithm, we use variable-sized circles to divide the configurational space in multiple-resolutions. Unlike the typical multi-resolution approaches, the aim here is to divide only a portion of the configuration space that is potentially on the way to the goal and therefore the proposed algorithm is a lot less computationally intensive.

While most of the algorithms are offline, for online processing, one of the oldest algorithms is the family of bug algorithms. The bug algorithms are the fundamental algorithms, which are complete as they guarantee to search the goal, if it is possible in the given space. They do not suffer from the problem of getting stuck in a local minima. Bug 1 algorithm is one of the oldest member of the bug family. The algorithm never gets struck in a local minima, but sometimes generates a path which takes the robot far away from the goal [6]. The algorithm moves the robot towards the goal using the motion to goal behaviour. In case of an encounter with an obstacle, the obstacle avoidance behaviour is invoked and the robot starts to move around the obstacle. While moving around the obstacle, the robot stores the distance between the current point and the goal. The robot stores the minimum distance point as it completes one revolution around the obstacle. Henceforth the robot again goes back to that point which recorded the minimum distance and navigates further using the motion to goal behaviour. The main drawback of this approach is that following the complete boundary of the obstacle can lead to a very long path.

Bug 2 algorithm is another improved version of the bug algorithm. Here the path is planned using a greedy approach [21]. The algorithm begins by moving towards the goal. The behaviour changes to obstacle avoidance whenever any obstacle is encountered and it moves around the obstacle. While moving along the boundary, the slope is calculated from that position towards the goal. Once the slope equals to the initial slope between the source and destination, it again starts moving towards the goal if such a motion is possible and the current point is closer than the previous such leaving point. Although the bug-2 works better than bug-1 in most cases, there can be cases where it does not perform better than bug 1. On encountering an obstacle, the robot has to choose between clockwise or

anti-clockwise traversal of the obstacle. A wrong choice in this algorithm can lead to significantly poor paths. The motivation behind the proposed algorithm is to add a little deliberation in the same methodology so as not to generate very poor paths. The advantages of a high clearance and a variable sized circular space decomposition are extra.

3. Proposed Algorithm

3.1. Overview of the Proposed Algorithm

Our proposed algorithm generates a feasible solution, while (non-strictly) optimizing two factors i.e. maximizing clearance factor of the path from obstacles and minimizing the processing time to generate the solution. The algorithm was initially inspired from the quad tree-approach using a multi-resolution technique [23][24]. In our proposed algorithm, we use a circle in place of square grids. The circle is made to propagate towards the goal (destination), along the shortest path. As the source is fixed, the initial circle encloses the robot at the source position. While traversing towards the destination, if the circle encounters any obstacle, then we use the analogy of the bug-2 algorithm [5][6], and go around the obstacle. But since our algorithm is an offline algorithm, we divide the circle at the junction and send two different circles along the surface (boundary) of the obstacle. Now, among the two branches of the circles, the one which reaches along the original line first (the line joining the source and destination) is accepted and the other branch is rejected. Now again the circle traces the path towards the goal and repeats the same steps if any obstacle is encountered. But suppose at any position both the circles of different branches are unable to find a path or get struck in a loop, then there is a condition of no path for that radius. Then we re-initialize the circle from the source with a new radius which is α times the previous radius. The same process is repeated until an obstacle free circle encompasses the destination or if the minimum permissible value of radius (threshold value) is reached.

Using a circle as the propagating virtual robot in our algorithm, the maximum clearance from the obstacle in ensured as the robot is enclosed inside the circle. The algorithm terminates when the threshold radius is reached. This ensures that the found path is feasible with respect to the physical dimensions of the robot. Moreover, the main utility of using a circular shape is that it can rotate around a pivot circle, while still guaranteeing clearance on the path rotated. This property is used for the propagation of the circle around the obstacle. This property is not met with any regular polygon. As an example consider a square as the shape chosen with a size of a. If a square is inside an obstacle and we take that as a pivot and rotate another square around the obstacle-prone square. Now at the corner, the rotating square has a distance of $\sqrt{2}a$ from the pivot square, so the clearance is larger than the clearance being maintained of a. While in case of a circle, the best bound of clearance will be the radius.

3.2. Technical Details

The algorithm is initialized by first determining the maximum obstacle-free region containing the circle around the source. To determine the radius of this circle, an initial circle with some radius δr is created around the source. The circle is checked for n (≈40), number of random points on its circumference for obstacle bound region. The radius of the circle is iteratively increased by δr, until the obstacle is detected or the goal is detected. Thus the maximum radius of the circle without obstacle region is taken as the initial propagating circle. If the goal is detected, then that gives a straight line joining the source and the goal as the path. In figure 1, the step-wise determination of the initial radius of the propagating circle is demonstrated. The algorithm uses the line joining the source

and goal as a heuristic to reach the goal. We define this line as the base line. The circles are propagated towards the goal along the base line.

In free space the circles are propagated along the line which is given by equation (1). Let r be radius of propagating circle, $q(t)$, q_g, q_s represent the coordinates of the centre of the circle, goal and source respectively at some iteration.

$$q(t+1) = q(t) + 2r \frac{q_g - q_s}{d(q_g - q_s)} \quad (1)$$

But when the obstacle-containing regions intersect the straight line joining the source and destination, the propagating circle which intersects the obstacle-containing region is converted into the pivot circle, $q_p(t)$, and the previous propagation circle, $q_{pp}(t)$, prior to the pivot circle, is rolled in a clockwise and an anticlockwise direction along the pivot circle as shown by equation (2).

$$q_{lp} = q_p + R_z\left(\frac{-\pi}{3}\right).[q_{pp} - q_p] \quad (2)$$

$$q_{rp} = q_p + R_z\left(\frac{\pi}{3}\right).[q_{pp} - q_p] \quad (3)$$

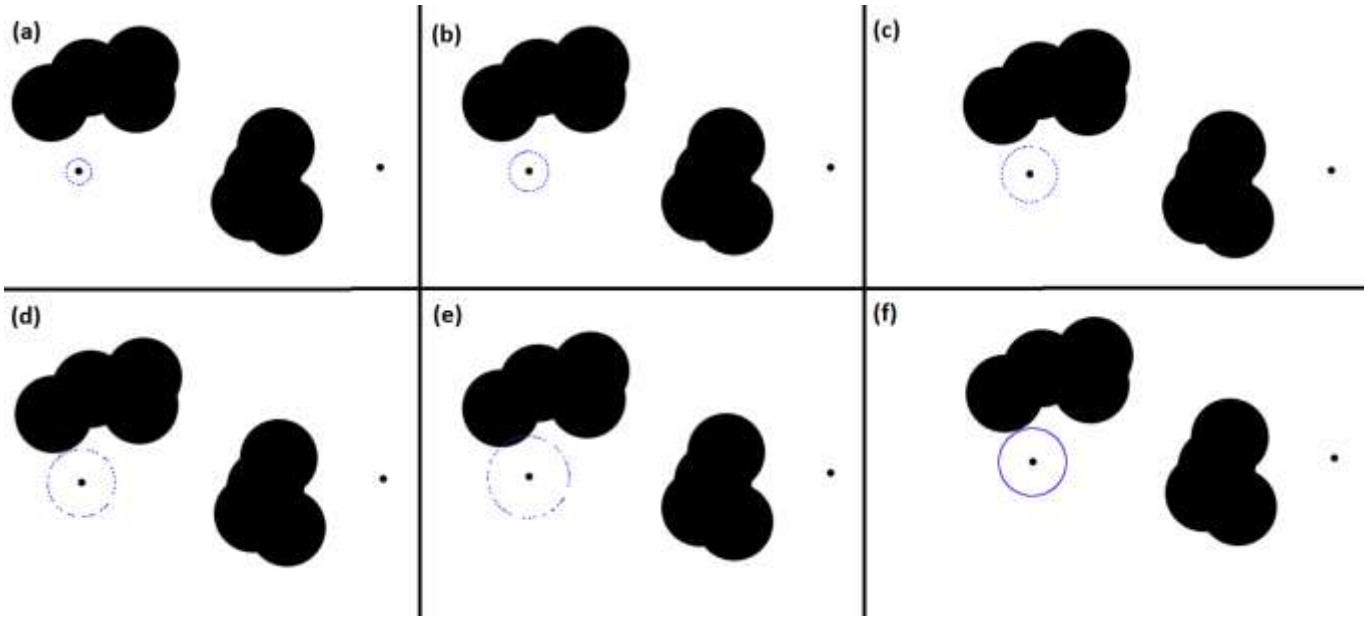

Figure 1. (From top left to bottom right) (a) Creating an initial circle (blue, dotted) and checking along the points (dots) on the circumference. (b -e) Iteratively increasing the radius and checking for obstacle region. (f) Maximum radius circle (solid) having obstacle free region is taken for initialization of the algorithm.

where $R_z(\theta)$, is the rotation matrix for θ angle of rotation around the Z-axis. These two circles form two branches. Again the circles are made to roll over the pivot circle in their respective directions. Now if the new propagating circle of the branch contains obstacle region, then that circle is made as the new pivot circle for that branch and the previous obstacle free propagating circle is made to roll over this new pivot circle. The process is repeated and finally both the propagating circles of the two branches move along the boundary of the obstacle. Figure 2 demonstrates how the propagating circle navigates around the obstacle after the initial maximum clearance radius is found. This approach is

also shown in the supplementary video for better understanding. Since the radius of the circles are fixed in one iteration, we have proved that there exists only an integral number of adjoining circles. This property also helps to maintain the list of visited circles around the given pivot circle and can be used to detect a loop.

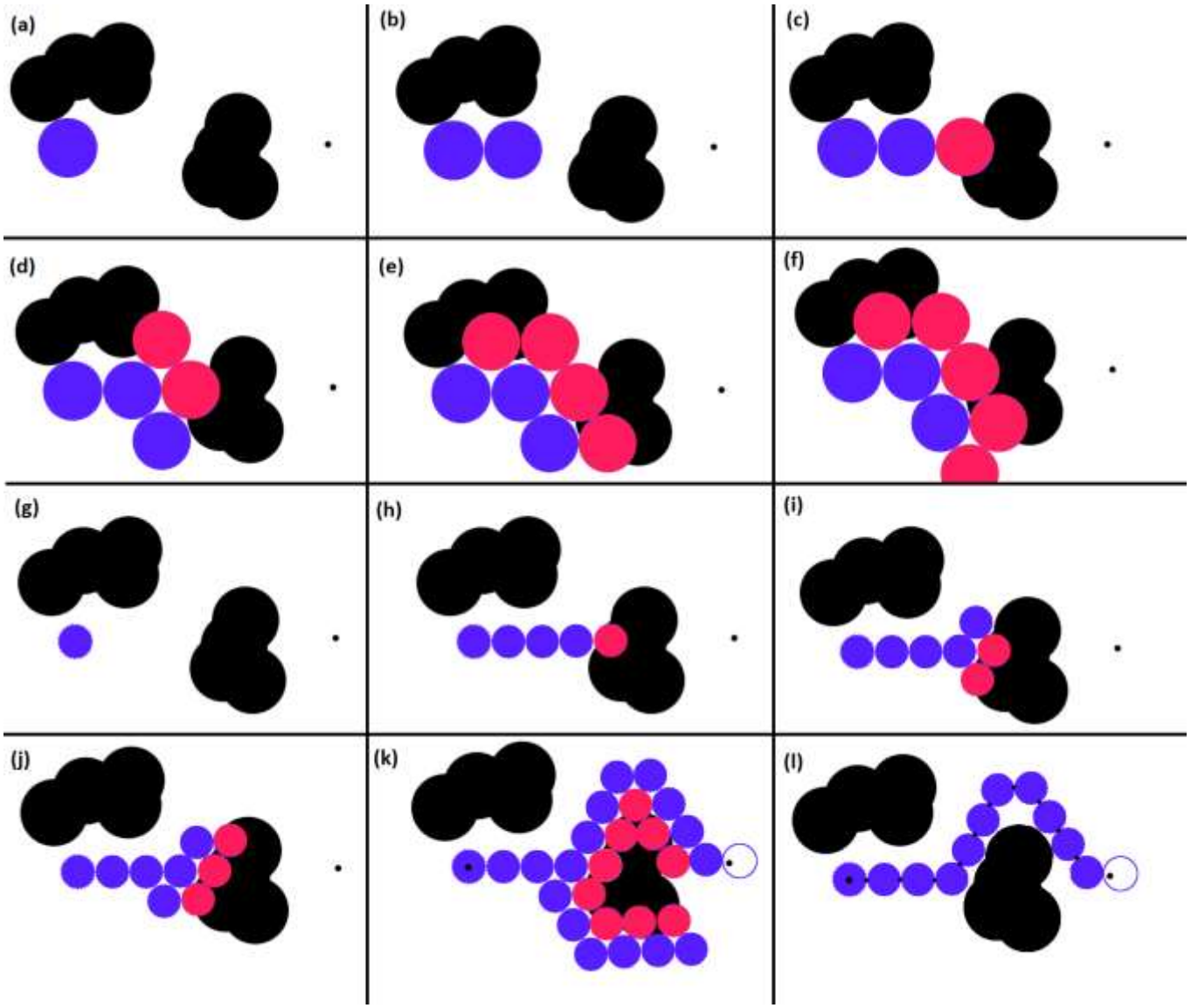

Figure 2. (From top left to bottom right) (a -b) Initializing initial propagating circle (blue) and propagating it towards the goal. (c) Obstacle containing circle is made as the pivot (red). (d) Rotation around the pivot is done and two circles are generated, (e-f) Two branches are propagated, but no free path is found. (g) A Circle with a new radius is initialized, (h-j) Circles are propagated towards the goal. (k-l) The upper branch finds the goal first and the touching points of the adjacent propagating circles are taken as the control points.

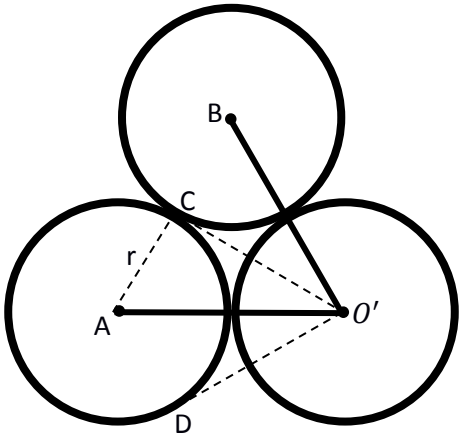

Figure 3. *Circle with centre O' is surrounded by two neighbouring circles*

Suppose there is circle with centre O' and radius *r*. The circles with centre *A* and centre *B* are its adjacent circles (figure 3). Let us take angle *AO'C* as θ. Since the angle *ACO'* is π/2, we can use trigonometry, to find θ from equations (4) and (5),

$$sin\theta = \frac{AC}{AO'} \tag{4}$$

$$\theta = sin^{-1}\frac{AC}{AO'} \tag{5}$$

$$\theta = sin^{-1}\frac{r}{r+r}$$

$$\theta = \frac{\pi}{6}$$

We know that the length of tangents to a circle from an external point are equal and the line joining the external point and the centre of the circle acts as a bisector of the angle created by the tangents. Thus, the angle *CO'D* will be π/3. This angle represents one neighbouring circle. Thus, if there are *n* circles in the neighbourhood of the circle, then we can say that the sum of angles casted by all the neighbouring circles is 2π. Simplifying it we get,

$$2\pi = n \times \frac{\pi}{3} \tag{6}$$

$$n = 6.$$

Thus, there are only 6 neighbouring circles possible which can touch the given circle. All the circles touch each other and roll at an angle of $\pi/3$ in case of an obstacle. Using this it can be intuitively observed that whatever is the path or whatever obstacle comes into the way during propagation, the circles will never overlap. Even in the worst case all the circles will be touching each other. This shows that we can divide the map in a finite set of unique circles. This property helps us to maintain the list of visited circles (state space), which is used to detect a loop.

Since our algorithm is offline, it gives the opportunity to handle the drawbacks of the bug-2 algorithm. At the junction where the circle first meets the obstacle, the pivot is set. Now at every such point, there are two ways possible, thus we create two branches of circles. The circles are rotated at an

angle of π/3 clockwise and π/3 anti-clockwise along the pivot for the two different branches. If any of the branch reaches the base line, we add that branch's control point to our original path and the other branch's control point is rejected. The algorithm again propagates along the base line until it reaches the goal. But there can be a case that among the two branches around the obstacle, no one finds a path back to the base line and gets stuck in a loop. Under that condition the algorithm reinitializes the circle from the start, with a different radius,

$$r_i = \alpha\, r_{i-1} \tag{7}$$

where $r_{i-1}$is the previous radius, and $0 < \alpha < 1$. α is the ratio factor by which the radius of the new circle depends on the previous circle.

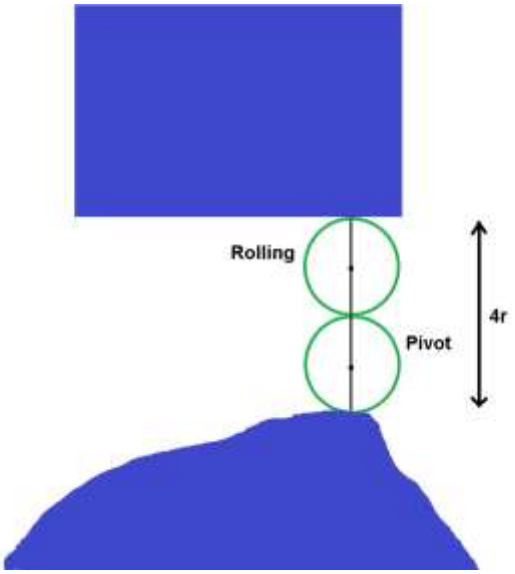

Figure 4. *Minimum clearance is equal to 4 × radius in the worst scenario*

The radius of the final circle (clearance), which successfully finds a way to the goal, is linearly related to the minimum clearance available in the configuration space between the obstacles. To prove this, suppose α is set very close to 1. Considering the worst case, at any iteration i, a circle with radius r_i just misses to find a path (figure 4). Then in the next iteration, the new radius will be marginally eligible to pass through the clearance. From the figure, it can be observed that the minimum radius of the circle will be one-fourth of the minimum global clearance possible, which lies along the way. No other worst case configuration is possible as compared to this. If no explicit conversion of the workspace to the configuration space is made by inflating the obstacles by the radius of the robot, the clearance between the obstacles should be at least two times the physical dimensions of the robot, in the worst case. If the workspace is however converted into a configuration space explicitly, smaller radii will also work.

3.3. Pseudo Code

Consider figure 5, as the 2-dimensional representation of environment or the map. The robot initializes from the initial position and there is goal position (destination). The blue coloured regions are the obstacle regions of the map. Let $B_r(q(t+1))$ be a circle with centre $q(t)$ and radius $2r$, given by (5). Further, let the line L be defined as a straight line joining the source (q_s) to the goal (q_g), given by (6).

$$B_r\big(q(t+1)\big) = \{x : d\big(x, q(t+1)\big) \leq 2r\} \tag{8}$$

$$L = \{q : \lambda q_s + (1-\lambda) q_g, 0 \leq \lambda \leq 1\} \tag{9}$$

We define a term pivot which is a circle around which the next propagating circles rotate in clockwise and anticlockwise directions creating two branches. The pseudo code of the approach is given in Algorithm 1. The algorithm can also be easily understood by the supplementary video.

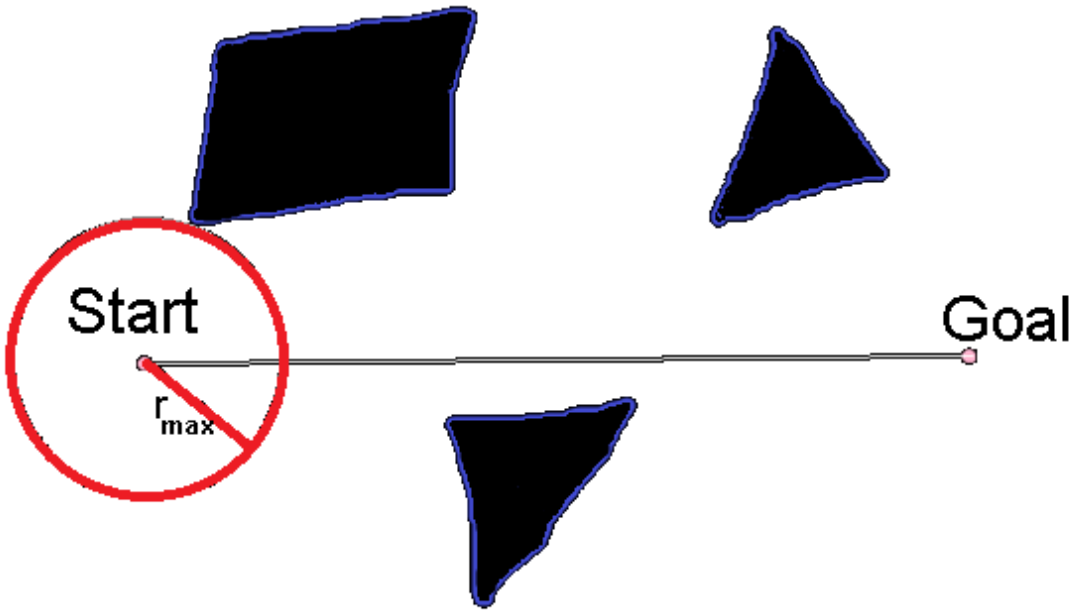

Figure 5. *Initialization phase. Circle with maximum permissible radius r.*

Algorithm 1

Input: $q_s = source, q_g = goal, r_{threshold}, \alpha$, *Output: Set of control points* τ

$t = 0,\ q(t) = q_s,\ r = r_{max},\ \tau = \phi$

loop:

if $r < r_{threshold}$, *return* ϕ

$q(t+1) = q(t) + 2r\frac{q_g - q_s}{d(q_g, q_s)}$

$Compute B_r\big(q(t+1)\big)$

if $B_r\big(q(t+1)\big) \cap W^{obs} = \phi$

$\tau \leftarrow \tau \cup q(t+1)$

else if $q_g \in B_r\big(q(t+1)\big) \wedge B_r\big(q(t+1)\big) \cap W^{obs} \neq \phi$

$t = 0,\ \tau = \phi,\ q(t) = q_s,\ r \leftarrow \alpha \times r$

else if $q_g \in B_r\big(q(t+1)\big)$

$\tau \leftarrow \tau \cup q(t+1)$, *break*

else if $B_r(q(t+1)) \cap W^{obs} \neq \phi$

$\tau_l = \phi, \tau_r = \phi, X_l = true, X_r = true, p_l = false, p_r = false$

$q_l = q_r = q_{lp} = q_{rp} = q(t+1), {q_l}' = {q_r}' = {q_{lp}}' = {q_{rp}}' = q(t)$

$ComputeB_r(q_{lp}), B_r(q_{rp}), B_r({q_{lp}}'), B_r({q_{rp}}')$

loop:

if X_l*:*

${q_l} \leftarrow {q_{lp}}.R_z\left(2r, \frac{-\pi}{3}\right)$

$ComputeB_r(q_l)$

$if\, q_l \in \tau_l, X_l = false$

$elseif\, B_r(q_l) \cap W^{obs} \neq \phi$

${q_l}' \leftarrow {q_{lp}}', {q_{lp}} \leftarrow q_l$

$elseif\, B_r(q_l) \cap W^{obs} = \phi$

$\tau_l \leftarrow \tau_l \cup q_l$

${q_{lp}}' \leftarrow q_l$

$if\, q_l \cap L \neq \phi$

$q(t+1) = q(t) + 2r\frac{q_g - q_s}{d(q_g, q_s)}$

$p_l \leftarrow true, \tau_l \leftarrow \tau_l \cup q_l$, *break*

if X_r*:*

${q_r} \leftarrow {q_{rp}}.R_z\left(2r, \frac{\pi}{3}\right)$

$B_r(q_r)$

$if\, q_r \in \tau_r, X_r = false$

$elseif\, B_r(q_r) \cap W^{obs} \neq \phi$

${q_r}' \leftarrow {q_{rp}}', {q_{rp}} \leftarrow q_r$

$elseif\, B_r(q_r) \cap W^{obs} = \phi$

$\tau_r \leftarrow \tau_r \cup q_r$

${q_{rp}}' \leftarrow q_r$

$if\, q_r \cap L \neq \phi$

$q(t+1) = q(t) + 2r\frac{q_g - q_s}{d(q_g, q_s)}$

$p_r \leftarrow true, \tau_r \leftarrow \tau_r \cup q_r$, *break*

if $X_l = false \wedge X_r = false$

$r \leftarrow \alpha \times r, t = 0$

$q(t) = q_s,\ \tau = \phi$, *break*

if $p_l = true,\ \tau \leftarrow \tau \cup \tau_l$

else if $p_r = true,\ \tau \leftarrow \tau \cup \tau_r$

The pseudo-code is given as Algorithm 1. In the proposed algorithm, we assume that we have the robot's initial position and we know the goal (destination). In the initialization part we set the α ratio factor and the threshold value, and initialize the list to store the control points of the path. Then we initialize our circle with centre at the source position with maximum possible obstacle free radius as shown in figure 5.

The line joining the source and destination is the shortest possible line. Our circle traverses along this line (line 7 from Algorithm 1). The new circle is checked - whether it is obstacle free, or if it contains

an obstacle or if it contains the goal. Now if the circle is obstacle-free (line 9), then we append the new position of the circle and continue with the loop.

If the circle is the repeated circle that has already been traversed or if it contains the goal along with an obstacle inside it (line 11), then there is no path possible, so we re-position the circle back to the source with the new radius which is alpha times the previous one. If the circle contains the goal (line 13) only, then we append the final position and break the loop.

If the circle contains an obstacle (line 15), then we have to go around the obstacle boundary. Thus, we make the obstacle circle as a pivot, along which the other circle will roll (figure 6). We initialize the two empty lists (line 16) to store the control points from the two different branches. After this we loop until there is a path back to the original line or there is no path. For the left branch, we get a new circle from the left pivot by rolling the previous non-obstacle circle over it in a clockwise direction (line 17, figure 6). The same is done with the right pivot in the right branch (line 18).

Now the new circle is checked for its status. If there is a repetition i.e. if it is struck in a loop, then the branch is terminated (line 23 and 35). If it contains an obstacle inside it (line 24 and 36), then the ball is set as the pivot ball and the previous ball is set as shown in figure 4. The other condition is when the circle is obstacle free (line 26 and 38), then we append the position of ball to the branch path list and set the current ball as the pivot's previous circle (figure 6). And the last condition, if the circle contains the segment of line (joining the source and destination, line 29 and 41), then the new position of the ball along the line is calculated, and the loop is broken.

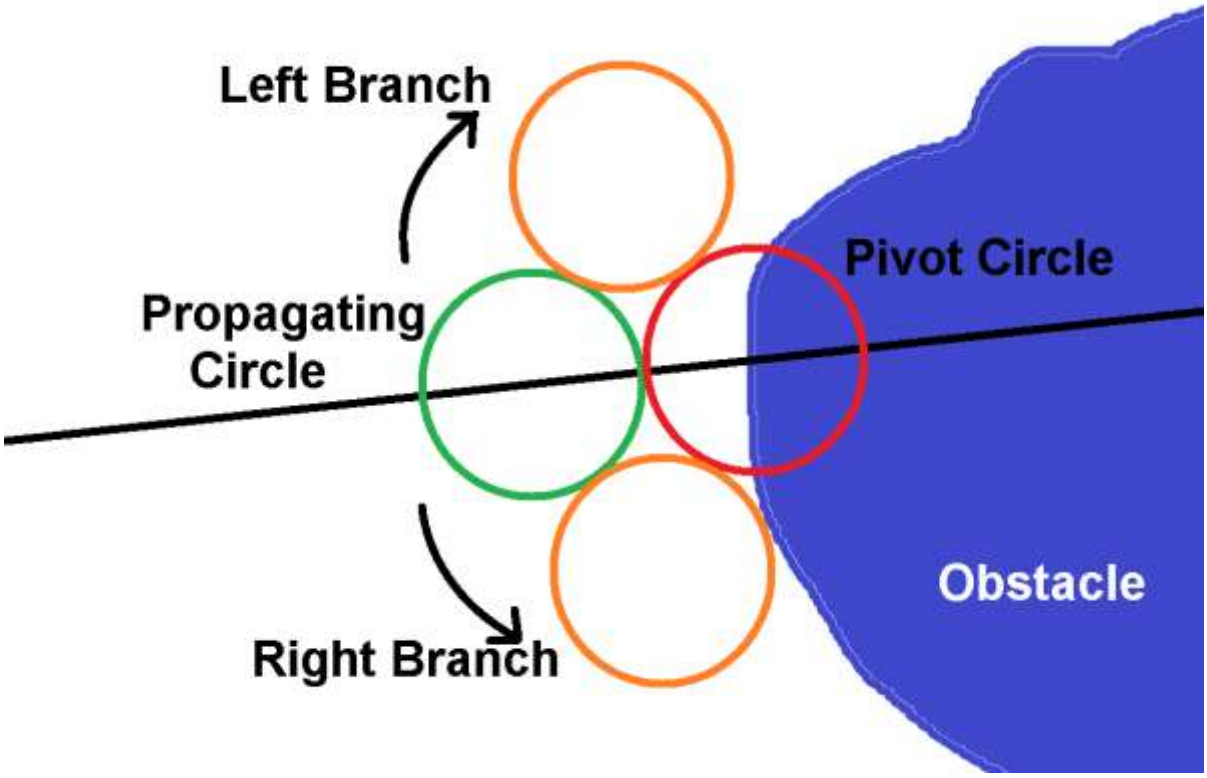

Figure 6. *Rotation around the pivot circle. Two branches emanate from the rolling circle in a clockwise and an anti-clockwise direction.*

If both the branches fail to find a path (line 44), then we reinitialize the circle from the source location and change the radius as alpha times the previous radius, remove all positions from the path, reinitialize it and break the inner loop. Finally, we append any one of the successful path from both branches, which successfully reached the main original line (joining source and destination, line 47-48). In any other case, the path is not appended. The loop is terminated if the goal is reached or if a minimum threshold value of radius is reached. Finally, on successful termination, the path (list) contains all the control points of the planned path.

3.4. Computational complexity

The complexity of our proposed algorithm can be discussed in the following parts:

We first initialize a maximum clearance circle around the source robot. This is done by constantly increasing the radius of the propagating circle, and checking for the presence of obstacles on the circle. The complexity for collision checking at the circle circumference of radius of r is $O(r)$ and hence the complexity of initial circle generation is $O(R^2)$, where R is the maximum radius of the propagating circle.

As the circle propagates towards the goal, it is constantly checked for collision checking. If r is the current radius of the propagating circle, the complexity of collision checking is $O(r^2)$.

For every obstacle, a left and a right branch is maintained. Every intersection point of the circle with the straight line to goal is a valid point to leave. Because the algorithm maintains two branches and whichever reaches earlier is used for the navigation till the leave point, in the worst case the path to overcome an obstacle i has a length of p_i travelled collectively by the successful branches. Here p_i is the perimeter of the obstacle i. Similarly a length of p_i would be travelled collectively by the unsuccessful branches. Further a length of D will be traversed in a straight line towards the goal by the robot, where D is the straight line distance from source to goal. Hence the total length of the path traced by the circle, summing up all the branches, in pursuit of a path is $D + 2\sum_i p_i$.

The collision checking happens at every step of the circle. The circle propagates in steps of size $2r$. The complexity for a particular radius r is given by

$$O\left(\frac{\mathrm{D} + 2\sum_i p_i}{2r} . r^2\right) = O\left(\left(\mathrm{D} + 2\sum_i p_i\right)\mathrm{r}\right)$$

We have initially taken $\alpha(<1)$ parameter, which defines the ratio of the new radius to the radius in the previous iteration. Assuming n iterations such that $\alpha^{n+1}R < R_{threshold}$, the total complexity of n iterations is given by Equation (3)

$$Complexity = (\mathrm{D} + 2\sum_i p_i)\mathrm{R}(1 + \alpha + \alpha^2 + \cdots + \alpha^n) \tag{3}$$

$$= \frac{(\mathrm{D} + 2\sum_i p_i)\mathrm{R}(1 - \alpha^n)}{1 - \alpha}$$

Although the number of iterations required in the algorithm depends upon α and the obstacle resolution, still the iteration parameter n is not large and can be assumed as a constant. A small tweak is to only check for collisions at the circle boundary instead of the complete circle, assuming the obstacles to be much larger than the size of the circle. In this case the radius *R* gets eliminated from the complexity.

Further analysis is also done using path length as a metric. Assuming $n_{i,}$ and p_i are the number of intersections and the perimeter of the i^{th} obstacle respectively. The bounds of path length and the bounds of computational time for path determination are compiled in table 2 and table 3 respectively.

3.5. Discussion on Completeness

An algorithm is complete if it finds a path in a finite time, under the condition that at least one path exists, or terminates with failure if no path exists. To prove the completeness of our theorem we assume that our robot is a point and there exists a path from start to goal (figure 7).

Suppose our algorithm is incomplete, so there exists a path from start to goal which is of a finite length and intersects obstacles a finite number of times. Under this scenario, since our algorithm is incomplete, it will be stuck in a loop and will not find a path.

Considering the case that it never terminates, the robot starts from the start position and moves along the direction of the goal. Let us call this angle as the heading angle. Now it encounters an obstacle and moves along the boundary. Since the algorithm will never terminate, the robot will be moving along the boundary forever. Thus, there will be no leaving point i.e. there will no point on the obstacle which makes the slope equal to the heading angle with the goal. However, there exists a hit point on the obstacle that the robot made on contacting the obstacle for the first time. The hit point of the obstacle has a slope equal to the heading angle.

The moving robot follows a closed loop around the boundary of the obstacle. Since there exists a hitting point on the obstacle boundary, the line joining the start and goal must also pass through that. Since the trajectory that the robot is following is closed, the obstacle must also have a closed boundary. Therefore, any line segment passing through a closed figure, must intersect it at even number of points (Jordan curve theorem). This contradicts the fact that there was no leaving point on the obstacle. Hence there exists a leaving point, which proves that robot will never get stuck in a loop. Thus, our algorithm is complete.

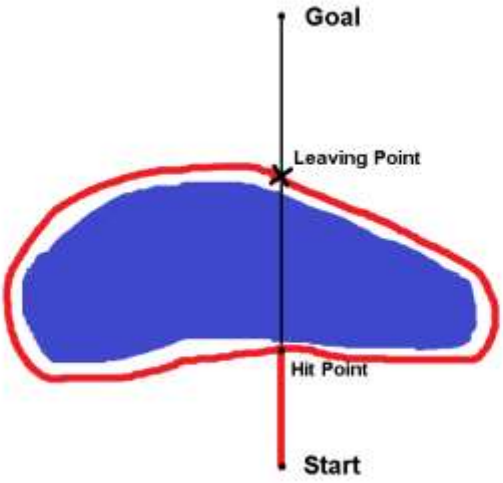

Figure 7. *Robot starts towards goal, and encounters obstacle at hit point.*

4. Creation of a Test Bed

To verify our algorithm, we created a cost-effective test bed. Our generic test-bed can be used to verify any general algorithm which follows the constraint of the physical robot. The robot which we have used is a differential 2-wheeled robot, powered by Raspberry Pi. There are no sensors used in the Robot and the DC motors are used on the wheels as the actuators. There is a fixed external camera placed globally, which is used to get the feedback for the robot. We have divided our test bed working in the following four sub-sections.

4.1. Mapping

The global fixed camera captures the image of the environment. It is assumed that the image is large enough that contains the robot, source and goal. However, we are required to generate a 2-D map, as the robot which we are using can travel only in a plane. Therefore, adding the third dimension is useless and will unnecessary increase the complexity. To convert the camera view to an overhead projection, we have used a Homography matrix. For this we calculated a 3×3 Homography matrix using the projection of an object from one epipolar line and mapping it with the same object along the other epipolar line shown in figure 8.

After the homography matrix is found, we multiply the input image, to get the complete overhead image. To classify among the robot, obstacle, etc., we have used colour filtering.

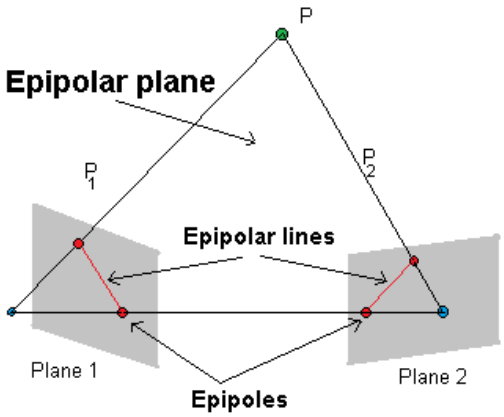

Figure 8. *Diagram showing the epipolar lines, for different perspectives.*

4.2. Localization

The localization of the robot is a difficult challenge in designing of a fully autonomous robot. Even after we locate our robot in the map using colour filtering, it is still inaccurate. What makes it difficult is the presence of uncertainty in both controlling and sensing of the robot. For example, sensing of the robot using a camera might not give the correct value if there is a change in the intensity of light, etc. Therefore, we use filters to extract the partial information from the sensors and system, and predict the filtered better path. We have used Extended Kalman filter in our test bed.

The Kalman Filter is a technique from the estimation theory that combines the information of different uncertain sources to obtain the values of variables of interest together with the uncertainty in these [8][9]. The Kalman filter requires a system model and a measurement model, along with the initial uncertainty present in the variables. Using these two models, the Kalman filter predicts the next state. However, the Kalman filter is valid only for linear systems and our model is non-linear. Therefore, we use the Extended Kalman filter, which is applicable for both linear and non-linear systems and works by locally linearizing the system.

4.2.1. System model - The system model describes how the state of the system changes due to the dynamics of the guidance system. In our differential 2-wheeled robot, we can get the control signals given for navigation. We can get the approximate velocity by mapping the PWM signals given to the real coordinate values. Using this we can get the relative displacement. Therefore, our model has the previous location value and the current relative displacement, and by using these we can have the approximated next state's value. In the model [26], the state at time *k* is given by,

$$x_k = \begin{bmatrix} x_{[x],k} \\ x_{[y],k} \\ x_{[\phi],k} \end{bmatrix} \tag{10}$$

where, *x*, *y* are the coordinate of the centre of robot, and φ is the orientation of the robot in the general coordinate system.

We can get the approximate relative movement of the two wheels. Using these, the relative displacement at any time *k* is given by,

$$u_k = \begin{bmatrix} u_{[\Delta D],k} \\ u_{[\Delta \phi],k} \end{bmatrix} \tag{11}$$

where, ΔD and $\Delta\varphi$ are relative displacement and relative change in orientation. Thus, using the previous state information and the current relative displacement, the location update equation becomes

$$x_k = f(x_{k-1}, u_{k-1}) = \begin{bmatrix} f_x(x_{k-1}, u_{k-1}) \\ f_y(x_{k-1}, u_{k-1}) \\ f_\phi(x_{k-1}, u_{k-1}) \end{bmatrix} \tag{12}$$

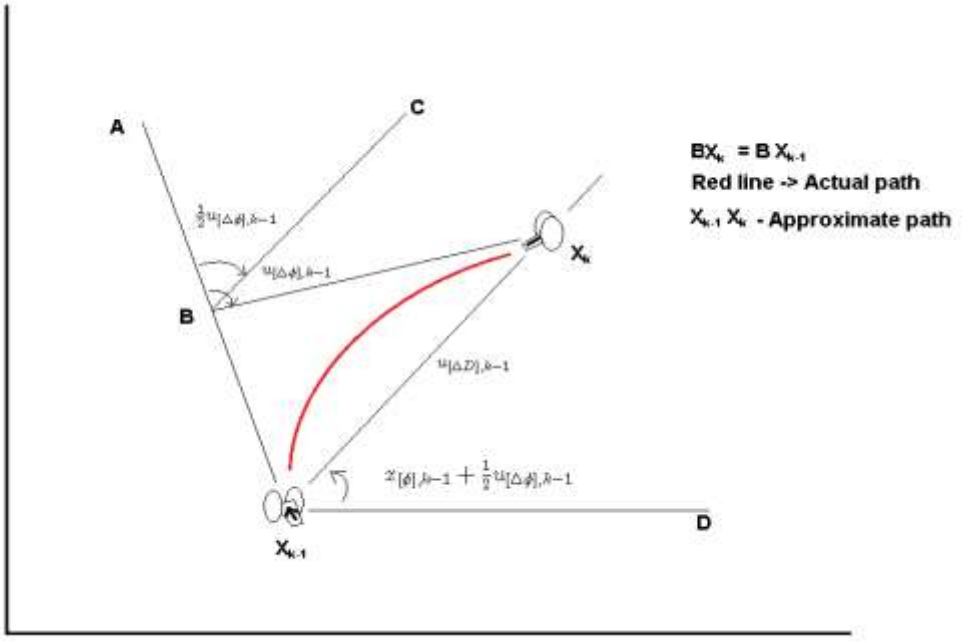

Figure 9. *Robot travelling from X_{k-1} to X_k along the red line*

As shown in figure 9, the robot moves from X_{k-1} to X_k, along the red line. It can be approximated to the distance $X_{k-1}X_k$ for a small time. The state X_k is derived using equations (10) -(12).

$$x_{[x],k} = f_x(x_{k-1}, u_{k-1}) = x_{[x],k-1} + u_{[\Delta D],k-1}.cos\left(x_{[\phi],k-1} + \frac{u_{[\Delta\phi],k-1}}{2}\right) \tag{13}$$

$$x_{[y],k} = f_y(x_{k-1}, u_{k-1}) = x_{[y],k-1} + u_{[\Delta D],k-1}.sin\left(x_{[\phi],k-1} + \frac{u_{[\Delta\phi],k-1}}{2}\right) \tag{14}$$

$$x_{[\phi],k} = f_\phi(x_{k-1}, u_{k-1}) = x_{[\phi],k-1} + u_{[\Delta\phi],k-1} \tag{15}$$

Since the robot will be disturbed by the external noises, we have modelled for the relative displacement noise by a random noise vector taken as a Gaussian distribution with zero mean.

Similarly, we model the system noise, which is indirectly related to relative displacement noise. The system noise is also modelled using a Gaussian distribution.

4.2.2. Measurement model - We have a fixed camera, which measures the full state of the system. Thus, the measurement Z_k becomes

$$z_k = \begin{bmatrix} z_{[x],k} \\ z_{[y],k} \\ z_{[\phi],k} \end{bmatrix} \tag{16}$$

where, *x*, *y* and φ measures the three states of the system. We model the noise in the measurement model using a Gaussian noise vector with a zero mean. This makes the measurement vector, Z_k , also Gaussian distributed.

Using the system model and the measurement model, we use the Extended Kalman filter to generate the path of the robot, which is used for controlling the trajectory of the robot.

4.3. Planning

We have already described the algorithm in the previous section. Our algorithm generates the control points in the map, which our robot will follow to reach the destination. After getting the path coordinates, it must be re-sampled. Thus, we interpolate equi-distant points between the control points.

Since the robot we are using has a linear velocity –angular velocity controller, the path has to be smooth. Although the circles used in our algorithm naturally smoothens the path, still for better control, we again smooth it.

To smooth a path, we have to optimize the path based on two criterions - distance between the original path and the new path, and second, optimizing the distance between the consecutive points of the path. For this, we use two parameter weights, data-weight and smoothness-weight. These two parameters decide the smoothness of the curve. We iterate in a loop minimizing based on these two factors, until the curve converges.

4.4. Control

We have used a Raspberry Pi board in our robot, which generates two independent PWM signals for the two different wheels of the robots. Using the variation of the PWM signals we assign different velocities to the two wheels, which is used by the robot to turn around. Since we have the desired trajectory, for our robot to move on it, we implant a virtual robot on the trajectory path. The virtual robot acts like a leader and leads the way. Our physical robot, follows the leader robot. The robot changes its orientation according to the change in the line of sight of the leader robot.

The kinematic equation we are using, is given by the equations (17) and (18).

$$v = \frac{v_{left} + v_{right}}{2} \tag{17}$$

$$\omega = \frac{v_{right} - v_{left}}{l} \tag{18}$$

where l is the distance between the wheels, v_l and v_r are the velocities of the left and right wheels of the robot. As our robot is guided by the change in orientation, using inverse kinematics we can get the V_{left} and V_{right} of the wheels of the robot. For any turn, which requires a turning angle φ, we get,

$$v_{right} = v_{base} + \frac{l * \omega}{2} \tag{19}$$

$$v_{left} = v_{base} - \frac{l * \omega}{2} \tag{20}$$

where, ω = φ /Δt, and v_{base} is the base velocity of the robot. The base velocity is the velocity of the centre of mass of the robot, and it is a function of the distance between the virtual robot and the real robot. Since our robot has some constraint with respect to the maximum velocity and the minimum velocity, the base velocity of the virtual robot should not exceed the base velocity of the physical robot. If the base velocity of the virtual robot is more than the physical robot, then there will be a gradual increase in the distance between the robots which will accumulate over time, and will lead to partial or complete missing of trajectory parts. Since we had calculated the Homography matrix, we already have the scale value between the real-world coordinates and the 2-D map coordinates. Thus, we adjust the base velocity of the virtual leader robot using that.

For best controlling, we have implemented a PID controller on the change of orientation, φ. The PID consists of three controllers: P (proportional) controller, I (integral) controller and D (derivative) controller. The proportional controller creates the correction in orientation in proportion with difference between the current orientation and the desired orientation.

$$P_t = K_p e_t \tag{21}$$

where P_t is the correction by proportional controller, for the error e_t, produced at time t and $K_p$is the proportional gain.

Although using only P controller works fine, still it will cause the system to oscillate, around the correct trajectory. Thus, to avoid this, we have implemented the D (Derivative) controller. In the derivative controller, we take the time derivative of the required controlling signal. In our case, we minimize the difference between the consecutive errors (control signals).

$$D_t = K_d(e_t - e_{t-\Delta t}) \tag{22}$$

Where D_t is the correction by the derivative controller, for the error e_t, where Δt is the frequency of the control system, and K_d is the derivative gain.

Although the above system works perfectly, but sometimes due to hardware errors, there can be a drift from the original trajectory. To avoid this, we use an integral controller, which accumulates the error over time and can correct it when the drift becomes large. The integral coefficient usually has a very small value; thus, it corrects slowly, on large drifts.

$$I_t = K_i \sum_{t=0} e_t \tag{23}$$

where I_t is the correction by integral controller for the error e_t produced at time *t* and $K_i$is the integral gain.

Although PID significantly improves the controlling of the system, finding the values of K_p, K_d and K_i is a very difficult task. It can be found by trial and error method, but that is not a feasible solution, which can be practically done. To overcome this, we created a simulator, which contains a differential 2-wheeled robot, with features like adding obstacles in it, changing dimensions of the map, changing physical dimension of the robot, etc. We can give a velocity to the individual wheels of the robot to move it.

To fetch the PID parameters the value for the real robot, we scale the simulator map with respect to the original world coordinate and then scale the simulator robot with respect to the actual robot. The frequency of the control signal is set the same in the simulator as in the real world. Since, now we have the replica of the physical robot, PID parameter value that we get in the simulator will also be applicable for the physical robot.

To get the PID parameter values, we then use the technique of twiddle. Twiddle initializes with any initial set of parameters of PID, and then, it adjusts the parameters one at a time to find an optimal set. We let the twiddle algorithm converge at a set of parameters, which is a local minima. The parameters found by using twiddle is applied to the physical robot controller.

5. Results and Discussions

We have tested our proposed algorithm on several different scenarios. The scenarios had many obstacles, ranging from simple convex to concave polygons. The results are shown in figure 10. The execution time was within a second using Intel core i5 processor. The algorithm is implemented in python 2.7.5. The simulation results show that our proposed algorithm performs successfully in configurational space in different scenarios. Figures10(c) and 10(e) show that the algorithm works well for convex obstacles and Figure 10(d) shows that the algorithm also works perfectly for mazes.

We have compared our algorithm with A*, Artificial Potential Field, Bug-1 and Bug-2 algorithms. The performance result of the comparison is shown in Figure 11. The algorithms are compared against the path length ratio metric, as detailed in Table 1. The path length ratio is the ratio between the path length and the Euclidean distance between the source and goal. The lesser the ratio, the better is the path, with the best value of 1 when the source and goal are directly connected. The choice of algorithms includes the A* algorithm which will naturally give near-optimal results, better than the proposed algorithm, however is computationally expensive. The choice of the algorithm was largely to understand if there is a huge loss in optimality due to limited deliberation in the algorithm. In the implemented version, no consideration to clearance or smoothness in A* was given, which can be added at some higher computational cost. Even though A* resulted in better paths (with more computation time), the difference is not excessive. The difference can be made smaller if the A* algorithm is made clearance conscious.

The other algorithms are more important from a comparison perspective due to their reactive nature, thus not limited by the computation time. The first two algorithms are the Bug algorithms which were the motivation behind the work and the choice was more importantly due to the complete nature of the algorithms. These algorithms clearly perform very poor in terms of path length. The upper bound and the lower bound of path lengths of bug algorithms are compared with our algorithm and compiled in

table 2, and also the bounds of computational time for determination of path length are presented in table 3. The clearance is not studied as a metric since the clearance for these algorithms is always 0 (or fixed constant). Practically a better algorithm is the Artificial Potential Field which results in much better paths. Testing was done only on the scenarios wherein the Artificial Potential Field Algorithm does not fail or get stuck. Even for such scenarios the proposed algorithm performs better using the path length as a metric. The clearance is stressed as a factor in all these metrics since the actuation and sensing uncertainties are assumed to be very high due to the low cost of the setup. The factor is not experimentally studied since none of the algorithms (baring potential field) has clearance has a non-fixed factor.

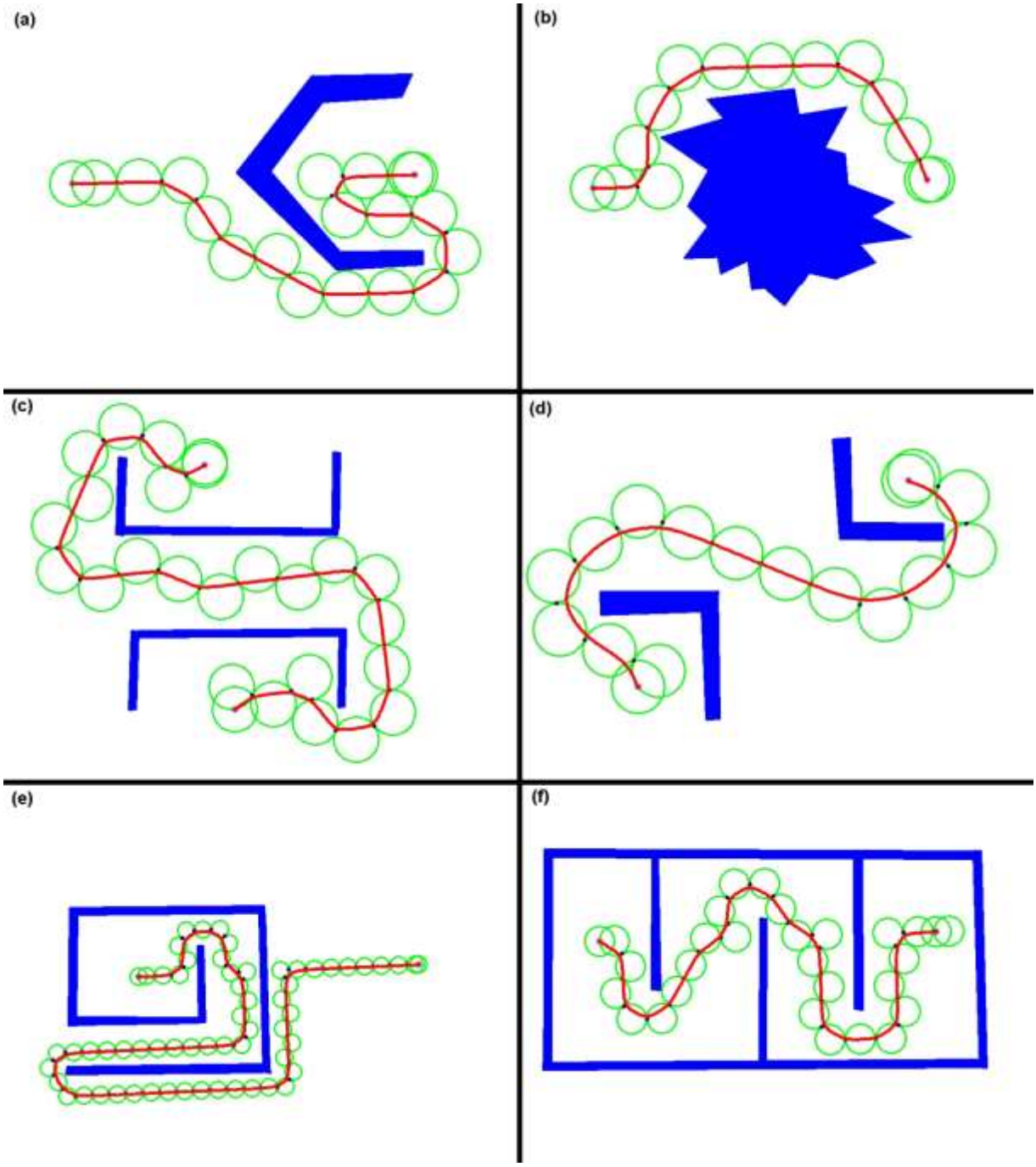

Figure 10. *Simulation results of the proposed algorithm in different configuration spaces.*

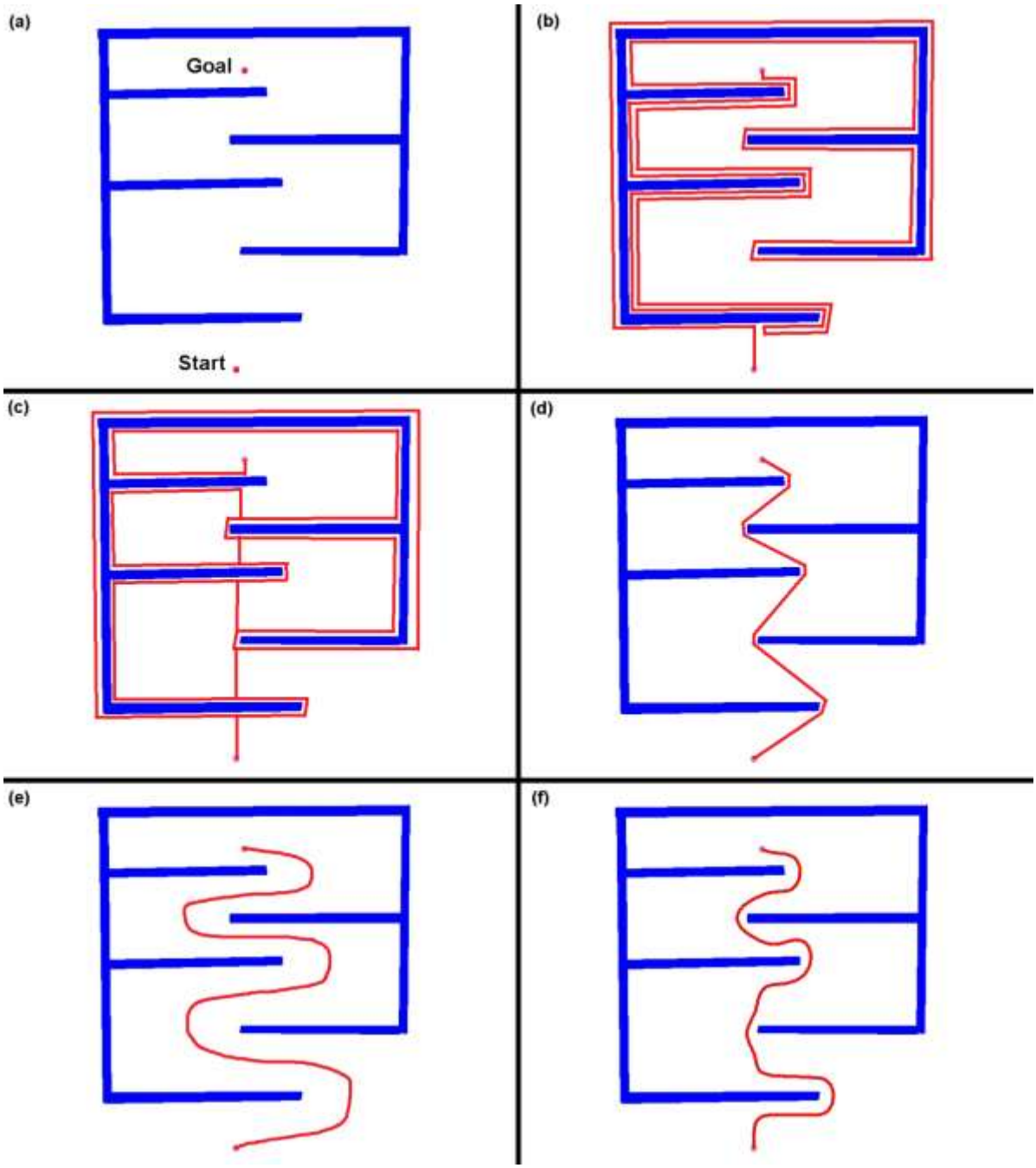

Figure 11. *(a) Scenario-1 showing start and destination. Comparison between the different algorithms are shown (b) Bug -1 algorithm. (c) Bug-2 algorithm. (d) A* algorithm. (e) Artificial potential Field algorithm. (f) The proposed algorithm.*

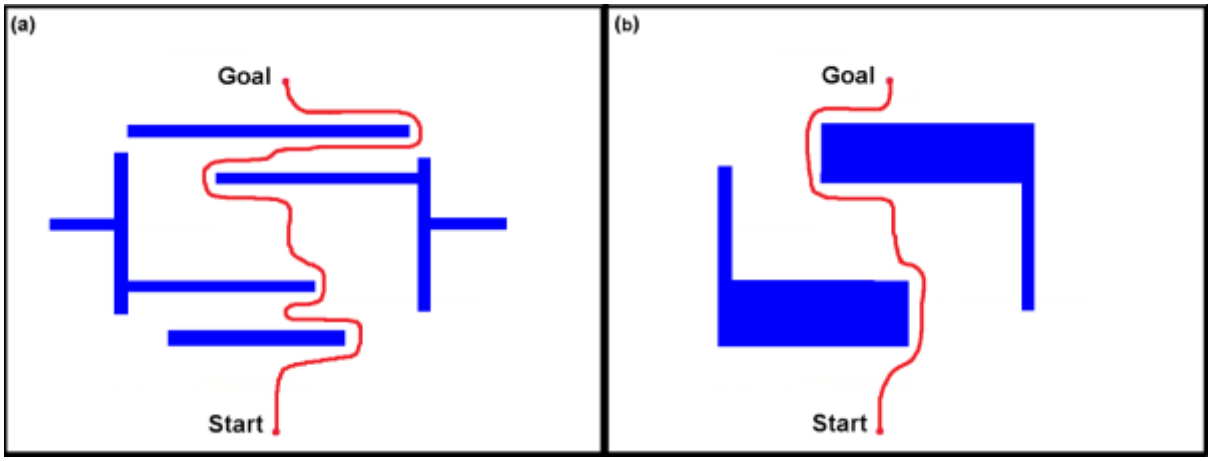

Figure 12. *Path found after smoothening the control points generated by the proposed algorithm. (a) Scenario -2 and (b) Scenario-3*

We have also tested the algorithm on our test bed. Figure 13 shows some of the overhead maps, in which our robot is located in the white circle. In the figure the blue points show the trajectory, which is obtained from the planning algorithm. A virtual leader robot is made to move on the blue points. The green points show the robot's position calculated using kinematic equations and the white points are the measured position/observed position from the external camera. These values are used by the Extended Kalman Filter to predict the position the robot. The red points in the figure is the path predicted by the filter. During the initial motion, there are some oscillations around the desired path. The oscillations die after some time when correction from I controller and D controller become effective.

In our testing, the average cross-track error found was of the order of 3-4 centimetres, keeping the motion under the physical constraints of the robot. The frame width of the robot is 15 centimetres and the distance between the wheels is 13 centimetres. The minimum clearance given in the map around the obstacle is about 31-32 centimetres. On an average, the path length which robot followed was about 88 centimetres with an error margin of 5% in deviation from the path.

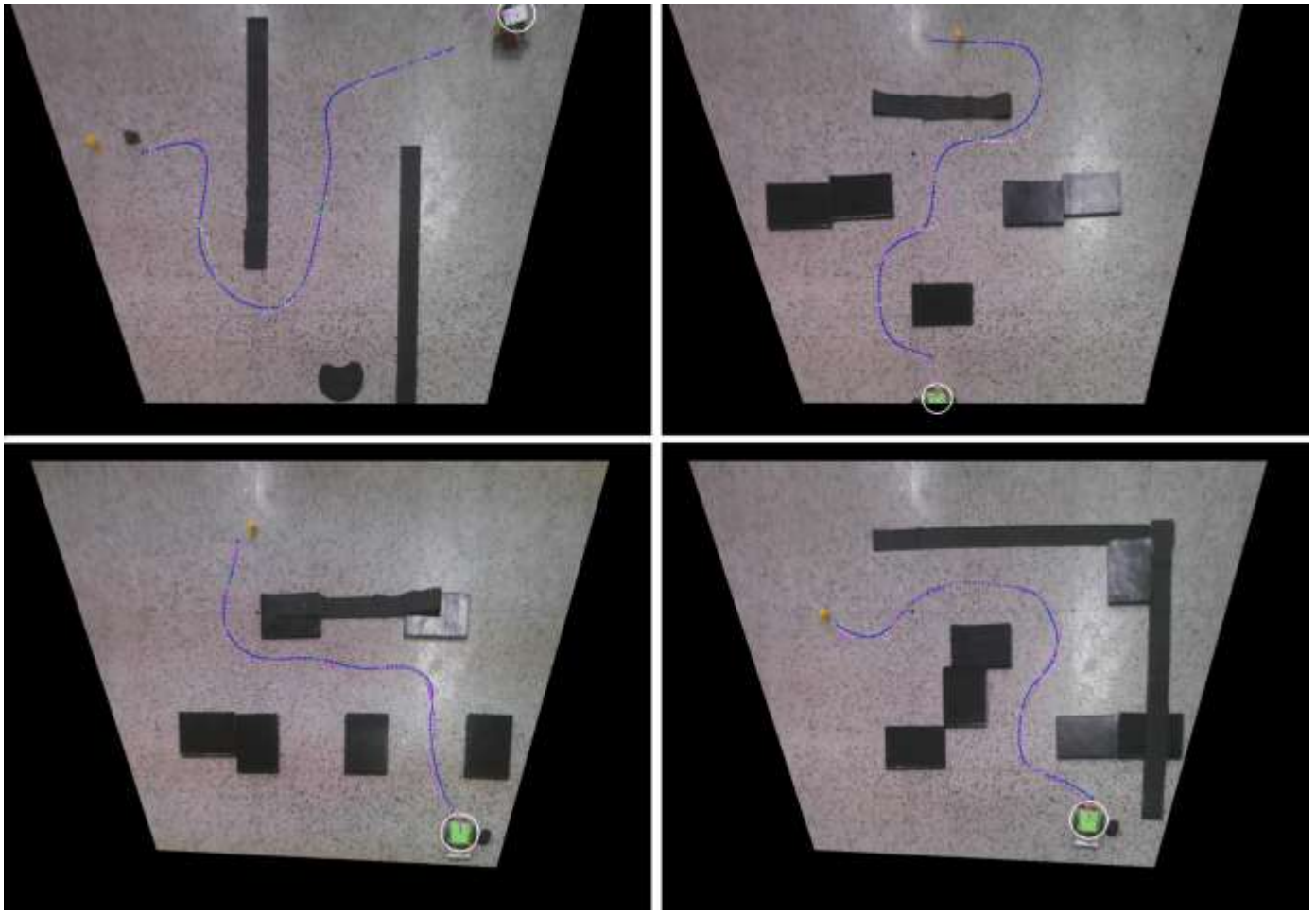

Figure 13. *Test bed demonstrating the trajectory following robot. Black covers are used as obstacles. The blue path is the planned path. White and green paths are the predicted paths from the measurement model and the system model respectively, and the red path is the actual travelled path of the robot.*

6. Conclusion

In this research we have developed a new offline path planning algorithm. The best part of the algorithm is that it can plan with respect to the physical dimensions of the robot in the workspace directly, without requiring an explicit conversion into the configuration space for mobile robots. It finds paths from the region having the maximum clearance. The radius of the final circle depends upon the minimum clearance available globally in the map. The radius is always greater than half of the minimum clearance possible. The path generated is in the form of finite control points, which can be later post-processed [17] for getting a simpler trajectory. Since the robots maintain a high clearance given by the radius of the propagating circle, any deviation produced from the desired trajectory which is less than the radius the circle is acceptable. The robot can navigate flawlessly with such an error margin, as the robot is not perfect with respect to odometry and other sensors.

Apart from the navigation advantages, the processing time of our algorithm is remarkably fast. Although it depends on the factor α ratio, it is still very fast as compared to the other algorithms. The algorithm proved to be the best if compared on the parameters of processing time, path length and clearance, simultaneously. The algorithm is complete and it never gets struck in a loop, unlike the potential field algorithms. If there exists a path which is in the reach of the robot, the algorithm guarantees to find it.

Many interesting improvements are still left for future works. Currently, while working on work space, the algorithm requires the clearance should be at least twice as that of the robot (worst case). This limit should be improved for more precise robot path planning. Due to small processing time and finite control points of paths, we can also use it for dynamic planning in case of moving obstacles. Apart from that, this algorithm can also be extended to 3-D space and faster planning can be done. The algorithm can be re-worked to generate more nearly-optimal paths. There are still lots of improvement which can be made in our existing model.

References

[1] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan. The power of a pebble: Exploring and mapping directed graphs, in Proceedings Annual Symposium on Foundations of Computer Science, 1-2, 1998

[2] R. Tiwari, A. Shukla, R. Kala: Intelligent Planning for Mobile Robotics: Algorithmic Approaches, 2013, IGI-Global, Hershey, PA.

[3] F. H. Clarke. Optimization and Nonsmooth Analysis. Springer-Verlag, Berlin, 1998.

[4] P. Fraigniaud, D. Ilcinkas, G. Peer, A. Pelc, and D. Peleg. Graph exploration by a finite automaton. Theoretical Computer Science,345(2-3):331–344, 2005

[5] V. J. Lumelsky. Sensing, Intelligence, Motion: How Robots and Humans Move in an Unstructured World. Wiley-Interscience, 2005

[6] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2:403–430, 1987.

[7] R.E. Kalman. Contributions to the theory of optimal control. Boletín de la Sociedad Matemática Mexicana, 5: 102–119, 1960.

[8] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering: 82(D), 35–45, 1960.

[9] R.E. Kalman; R.S. Bucy. New results in linear filtering and prediction theory. Journal of Basic Engineering: 83(1), 95–108, 1961.

[10] S. Bennett (1992). Chap. Process control: technology and theory in A history of control engineering, 1930-1955. IET. 50-60.

[11] O. Salzman and D. Halperin, "Asymptotically Near-Optimal RRT for Fast, High-Quality Motion Planning," in IEEE Transactions on Robotics, vol. 32, no. 3, pp. 473-483, June 2016.

[12] J.C. Latombe. Robot Motion Planning, Kluwer, 1991.

[13] V. Sezer, M. Gokasan: A Novel Obstacle Avoidance Algorithm: Follow the Gap Method. Robotics and Autonomous Systems, 60(9): 1123-1134, 2012.

[14] Alvarez and Sanchez: Reactive navigation in real environments using partial center of area method in Robotics and Autonomous Systems, 58(12): 1231-1237, 2010.

[15] L. Zhang, Y. J. Kim, D. Manocha. A hybrid approach for complete motion planning". in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.7-14, 2007.

[16] Y. Lu, X. Huo, O. Arslan, P. Tsiotras, Incremental Multi-Scale Search Algorithm for Dynamic Path Planning with Low Worst-Case Complexity. IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics 41(6) (2011):1556-1570.

[17] R.V. Cowlagi, P. Tsiotras, Hierarchical Motion Planning with Dynamical Feasibility Guarantees for Mobile Robotic Vehicles. IEEE Transactions on Robotics 28(2)(2012): 379-395

[18] R. Kala, A. Shukla, R. Tiwari (2010) Fusion of probabilistic A* algorithm and fuzzy inference system for robotic path planning. Artificial Intelligence Review, 33(4): 275-306.

[19] Savkin, A., & Li, H. A safe area search and map building algorithm for a wheeled mobile robot in complex unknown cluttered environments. Robotica, 36(1), 96-118, 2018.

[20] W. Rone, and P. Ben-Tzvi, (2013) 'Mapping, localization and motion planning in mobile multi-robotic systems', Robotica, 31(1), pp. 1–23. doi: 10.1017/S0263574712000021.

[21] Y. Maddahi, N. Sepehri , A. Maddahi, and M. Abdolmohammadi, 'Calibration of wheeled mobile robots with differential drive mechanisms: an experimental approach', Robotica, 30(6), pp. 1029–1039, 2012.

[22] Hwang, Y., & Lee, J. Robust 2D map building with motion-free ICP algorithm for mobile robot navigation. Robotica, 35(9), 1845-1863, 2017.

[23] S. Kambhampati and L. Davis. Multi resolution path planning for mobile robots. IEEE Journal on Robotics and Automation. 2 (3): 135-145, 1986.

[24] H. Noborio, T. Naniwa, and S. Arimoto. A quadtree-based path-planning algorithm for a mobile robot in Journal of Intelligent and Robotic Systems, 7(4): 574-576, 1990.

[25] R. Kala, A. Shukla, R. Tiwari (2011) Robotic path planning in static environment using hierarchical multi-neuron heuristic search and probability based fitness. Neurocomputing, 74(14-15): 2314-2335

[26] R. Negenborn : Robot Localization and Kalman Filters On finding your position in a noisy world, Master's Thesis, Utrecht University, 2003