

Points-based Safe Path Planning of Continuum Robots

Regular Paper

Khuram Shahzad^{1*}, Sohail Iqbal¹ and Peter Bloodsworth¹

¹ National University of Sciences and Technology (NUST), Islamabad, Pakistan

*Corresponding author(s) E-mail: 12mscshahzad@seecs.edu.pk

Received 04 January 2015; Accepted 10 May 2015

DOI: 10.5772/60857

© 2015 Author(s). Licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Continuum robots exhibit great potential in a number of challenging applications where traditional rigid link robots pose certain limitations, e.g., working in unstructured environments. In order to enable the usage of continuum robots in safety-critical applications, such as surgery and nuclear decontamination, it is extremely important to ensure a safe path for the robot's movement. Existing algorithms for continuum robot path planning have certain limitations that need to be addressed. These include the fact that none of the algorithms provide safety assurance parameters and control for path planning. They are computationally expensive, applicable to a specific type of continuum robots, and mostly they do not incorporate design and kinematics constraints. In this paper, we propose a points-based path planning (PoPP) algorithm for continuum robots that computes the path by imposing safety constraints and improves upon the limitations of existing approaches. In the algorithm, we exploit the constant curvature-bending property of continuum robots in their path planning process. The algorithm is computationally efficient and provides a good tradeoff between accuracy and efficiency that can be implemented to enable the safety-critical application of continuum robots. This algorithm also provides information regarding path volume and flexibility in movement. Simulation results confirm that the algorithm possesses promising potential

for all types of continuum robots (following the constant curvature-bending property). We believe that this effectively balances the desired safety and efficiency requirements.

Keywords Continuum robots, Points-based Path Planning, Curvature, Constant Curvature, Robot kinematics

1. Introduction

Robot designers have numerous design options when designing their machines. With time, the number of such design options continues to increase and/or be refined based upon technological advancements, inspiration from nature, and the designer's own imagination [1]. A number of different types of traditional rigid link robots have been proposed in the past. The design of these traditional robots is based upon a human arm-inspired mechanism, which consists of serially arranged rigid links or a joints-based architecture. Such traditional rigid link robots work fine in applications in which the environment is predefined and well-structured, but they have limitations in unstructured and congested workspaces [2]. To overcome the limitations of traditional rigid link robots, a large amount of research in continuum robotics has been published recently [2]. Continuum robots do not have rigid links/joints but present

continuous flexibility in their structure. Because of their continuously bending architecture, continuum robots can manoeuvre through an unstructured congested workspace and can overcome the limitations posed by traditional robots [2, 3, 4].

In the literature, different types of continuum robot designs have been proposed that can be categorized as tendon-based, continuously bending actuator-based, concentric tubes-based, steerable needle-based or variable stiffness-based continuum robots [1, 4, 5]. Irrespective of their underlying physical architecture, the common property of the approximately constant curvature-bending of the backbone is exhibited by all types of continuum robots [4, 5]. This property has been exploited by different researchers to propose different designs and kinematics models of continuum robots [4, 5]. They exhibit significant potential in many challenging applications where traditional rigid links/joints-based robots have certain limitations. Examples of such applications involve working in obstacle-filled environments and minimally invasive surgery. For a robot to move from a given start location to a goal, the robot has to follow a path. There are certain constraints that the robot needs to follow during path planning. These constraints are: goal seeking, obstacle avoidance, non-penetration, spot the shortest path and energy minimization of the robot. For traditional rigid link robots, many path planning algorithms exist which solve a certain range of problems [6, 7]. For continuum robots, because of their continuously bending architecture and kinematics constraints, path planning brings many challenges that need to be addressed. Safety assurance is extremely important for the approval and usage of continuum robots in safety-critical applications, such as surgery. ISO 9000 standard norms have been updated to comply with the constraints of medical systems in the European Community. These standards allow the placement of such devices (e.g., continuum robots) in the market and their use for surgical applications.

Various research groups have proposed different methodologies for the path planning of continuum robots. However, there are certain limitations with existing approaches that need to be addressed. M. Moll and L.E. Kavraki proposed minimal energy curves-based path planning algorithms for deformable objects [8, 9, 10]. They introduced a new adaptive parametrization method for the representation of curves or configurations. Based on parametrization, they proposed methods for computing minimal energy curves, i.e., Lagrangian dynamics, null space sampling and constrained optimization techniques. They also proposed an algorithm for finding the sequence of minimal energy configurations between the start and goal configurations. The proposed methodology does not incorporate the obstacle avoidance and kinematics constraints that continuum robots have. This methodology is computationally expensive and is based on continuous deformation for specific shape-based path planning.

R. Gayle et al. proposed a medical image processing-based path planning algorithm for continuum robots [11]. During path planning, the algorithm checks for possible collisions between the robot and obstacles at each step. A special overlap test (2.5D) is used for finding the splitting surface between the robot and obstacles. The proposed algorithm does not incorporate the kinematics and design constraints of continuum robots. The algorithm is computationally expensive and uses a GPU-based collision detection algorithm. This may result in limited accuracy due to a lack of image-space resolution. In addition, the algorithm does not provide safety measures and control for safety-critical applications. E. S. Conkur and R. Gurbuz proposed a path planning algorithm for snake-like robots [12]. The algorithm consists of two steps, firstly it computes the smoothly curved path consisting of points from a given start to a goal point, and secondly the robot performs a serpentine motion on the previously computed path consisting of a number of points. The path trace from a given start point to a goal point is computed using the numerical potential field method (PFM). However, the proposed algorithm provides a discrete serial links-based path, and does not incorporate safety parameters. L. Lyons et al. proposed a motion planning algorithm for active cannulae-type continuum robots [13]. In the proposed algorithm, given the start and goal coordinates with obstacle information, the motion planning problem is formulated as a nonlinear constrained optimization problem. The PFM has been used for obstacle avoidance and for finding the optimal configuration of an active cannula. The limitation of the proposed methodology is that it is intended for use in image-guided procedures where the target and obstacles can be segmented from pre-operative images. The methodology is robot-specific (limited to active cannulae), computationally expensive, and does not provide accommodation for safety parameters and control.

I. Godage et al. proposed a path planning algorithm for multi-section continuum arms [14]. The algorithm was originally presented by Maciejewski et al. in [15] for the path planning of rigid link robots. This original algorithm was modified for continuum robots by incorporating a finite number of proximity sensors mounted along the length of the continuum robot to provide distal information. This information is used to find the repelling velocities of different sections of continuum robots, which are then used for goal seeking and obstacle avoidance. The proposed algorithm takes inputs from multiple sensors placed along the sections of the continuum robot, which might not be desirable for certain applications [1, 16]. The algorithm also requires a points-based path trace as input. It is robot-specific, computationally expensive and does not provide safety parameters and control for safety-critical applications. C. Bergeles and P. Dupont proposed an algorithm for planning stable paths for concentric tube robots [17]. The rapidly-exploring random tree (RRT*) algorithm, present-

ed by S. Karaman et al. in [18], is used for the path planning of the robot. For goal seeking, their algorithm navigates the robot through a sequence of locally stable configurations. Their work makes a good contribution to finding locally stable configurations for the path planning of active cannulae. However, the algorithm is robot-specific (limited to active cannulae), based on a variable curvature, computationally expensive, and does not incorporate safety parameters and control [17].

The main contribution of this paper is the proposed PoPP algorithm for continuum robots, which computes the locally optimal path by imposing safety constraints and improves upon the limitations that have been identified in existing approaches. The PoPP algorithm comprises two steps. Firstly, it computes the points-based path trace. Secondly, it finds the constant curvature (circular) curves-based continuous path using the points-based path trace as input. Both of these algorithms are internally used by the PoPP algorithm to generate a constant curvature curves-based continuous path for the continuum robot.

Organization: The rest of the paper is organized as follows: In Section 2, we present our PoPP methodology. Different approaches for computing the points-based trace of the path are presented in Section 2.1. The constant curvature curve-based continuous path planning algorithm is presented in Section 2.2. Section 3 discusses the proposed methodology, with its advantages and disadvantages, and makes a comparison with existing algorithms. The conclusion and possible future directions appear in Section 4.

2. The Points-based Path Planning Algorithm

Continuum robots can interact and perform certain tasks in a workspace by using their tip/head as an end-effector. Hence, the tip/head is considered as a point of interest of continuum robots. Continuum robots emulate snake-like motion. The path planning of continuum robots in static environments is fairly simple because the rest of the body of the robot will most probably follow the track made by the head of the robot. By finding the track of the head of the continuum robot, we can find the path of the whole robot by traversing the head trace. Based on this idea, we propose algorithms to find the points-based path trace. The problem statement for the points-based path trace algorithm is: given start and goal points, find all the intermediate points that connect the start point and the goal point such that no intermediate point collides with any obstacle and the resulting path trace satisfies the imposed safety constraints.

The PoPP algorithm is divided into two steps: points-based path trace computation and constant curvature continuous path computation. For computing a points-based path trace, we propose two algorithms in Section 2.1 - one uses a sine method (SM) and the other uses the PFM. Both these approaches differ based upon their obstacle avoidance strategy. The SM exploits the information from the shortest

distance vector to the obstacle, and then the vector to the goal point, while the PFM exploits the traditional PFM. Both approaches return a path trace connecting the start point and the goal point satisfying the constraints of obstacle avoidance as well as the safety parameters. For a continuum robot to follow a certain path, it needs to be continuous and the motion produced should be based on a constant curvature [4, 5]. This is so that the path satisfies the design and kinematics constraints of continuum robots. The points-based path trace returned by both approaches is discrete and does not satisfy the design and kinematics constraints for continuum robots. Hence, for this purpose, in Section 2.2, we propose a constant curvature curves-based continuous path planning algorithm. This algorithm takes the points-based trace of the path (resulting from the algorithms proposed in Section 2.1) as input and provides us with a continuous constant curvature curves-based path that satisfies the design and kinematics constraints of a continuum robot. The notation and symbols used in this paper are mostly defined locally in the algorithms; however, some of the globally-used nomenclature is defined in Table 1.

p_s	Start point
p_g	Goal point
$step$	Step size. Minimum length of the section of continuum robot
min_marg_dist	Minimum marginal distance to be maintained from obstacles. Measured from the centre-line/backbone of the continuum robot.
\emptyset	Empty set
θ	Bending or central angle
ϵ	Small constant value, e.g., 0.1
r	Radius of circular arc
$\kappa = \frac{1}{r}$	Curvature of circular arc
δ	Deviation angle or equal angle of isosceles triangle
ω	Unequal angle of isosceles triangle
C, C'	Centre of circle

Table 1. Nomenclature used in this paper

2.1 The Points-based Path Trace

We propose two algorithms for computing a points-based path trace. Both algorithms take the start point, the goal point, the step size and the minimum marginal distance required from obstacles as input, and return the sequence of points connecting the start point and the goal point, such that none of the intermediate points collide with any obstacle and that it satisfies the safety parameters. Let us present each approach, one after another.

Algorithm 1 ComputeCRPath using SM: Computes the sequence of intermediate points connecting the start point and the goal point such that all the intermediate points satisfy the imposed safety constraints.

Input: $p_s, p_g, step, min_marg_dist$

Output: Sequence of points connecting p_s and p_g such that the intermediate points satisfy the imposed constraints, e.g., obstacle avoidance, safety parameter

Basic Idea:

path := p_s

head := p_s

while (distance between head and p_g) < step **do**

p_i := Intermediate point at distance step away from head in the direction of p_g

p_i^* := handleObstacle(head, p_i , p_g , min_marg_dist)

if (No obstacle AND p_i^* moving away) **then**

return \emptyset

else

 Append p_i^* at the end of path array

 Update head pointer at p_i^*

end if

end while

return Append p_g at the end of path array

2.1.1 The Sine Method Algorithm

The basic idea of the sine method (SM) is explained in Algorithm 1. The algorithm first initializes the path with the start point p_s and places the head pointer over start point p_s . Next, we keep on iterating until the distance between the head and goal point p_g is greater than or equal to the step. With each iteration, the head pointer is updated with the intermediate point. In the body of the iteration loop, we find the intermediate point p_i at a distance step away from the head in the direction of p_g . The intermediate point p_i will be a shorter distance from the goal point than the head point, with a difference of step, and will be moving linearly towards the goal point p_g . We then run the handleObstacle() function (Algorithm 2) over the newly computed intermediate point p_i . The handleObstacle() function returns the obstacle-free intermediate point p_i^* if p_i was colliding with any obstacle. If p_i was not colliding or else influenced by any obstacle, then it returns the same point p_i at the output as p_i^* . Afterwards, we check whether there is influence by an obstacle on the intermediate point and whether, instead of getting closer to the goal state, we are moving away from the goal state; then, we return \emptyset to show that the connection attempt has failed, i.e., that the path cannot be found. If this is not the case, then we accept p_i^* as the intermediate configuration from head to p_g . Hence, we update the path trace and head coordinates with the intermediate point p_i^* . The algorithm will keep iterating until it finds all the intermediate points from the start point p_s to the goal point p_g or else the connection attempt fails. Finally, the algorithm appends the goal point p_g with the path trace and returns the complete points-based path trace as output.

Algorithm 2 handleObstacle: provides the obstacle-free point if that point is colliding with or being influenced by an obstacle.

Input: head or previous point of the intermediate point p_{i-1} ; the intermediate point p_i , p_g , min_marg_dist

Output: Obstacle-free point

Basic idea:

if (p_i not colliding with the obstacle AND away, than min_marg_dist distance from the obstacle) **then**

return p_i

else

θ := current angle of p_i , such that p_{i-1} is the centre of the circle and p_i is a point on the circle

\vec{vec}_{p_g} := vector from p_{i-1} to p_g

\vec{vec}_{sp} := vector from p_{i-1} to the shortest distance point on the obstacle

if sign of $\vec{vec}_{p_g} \times \vec{vec}_{sp} > 0$ **then**

 direction := clockwise

else

 direction := counterclockwise

end if

$p_{i_temp} := p_i$

while (p_{i_temp} is colliding with the obstacle OR less than min_marg_dist distance from the obstacle) **do**

if (direction is counter-clockwise) **then**

 Increment θ by ϵ

else

 Decrement θ by ϵ

end if

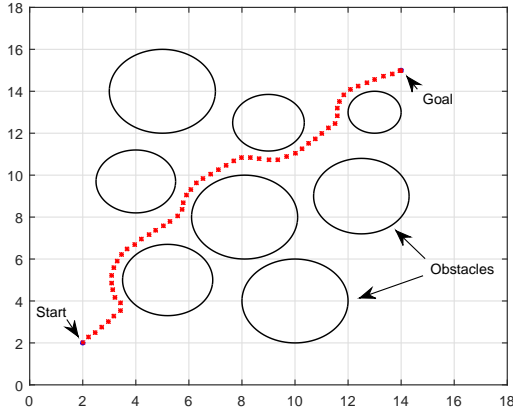
p_{i_temp} := point on the circle with updated θ

end while

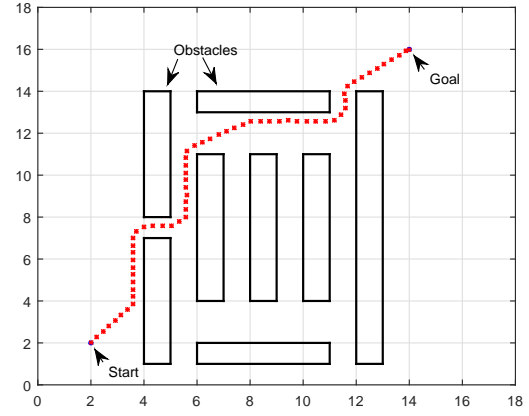
return p_{i_temp}

end if

In the Algorithm 1, the handleObstacle() function takes the head point, the intermediate point, the goal point and the minimum distance required from the obstacles as input, and returns the obstacle-free safe intermediate point as its output. The basic idea is explained in Algorithm 2. The algorithm first checks whether the intermediate point p_i is not colliding with any obstacle and is away than min_marg_dist from the obstacle, i.e., the intermediate point is a safe point. If so, then return the same point p_i as the output point. If it is colliding with or being influenced by the obstacle, then find the current angle (θ) of the intermediate p_i , such that p_{i-1} is the centre of the circle and p_i is a point on the circle. To avoid the obstacle, we either move clockwise or counter-clockwise. Hence, to find the direction to avoid the obstacle (local optimal), we compute the vector \vec{vec}_{p_g} (the vector from p_{i-1} to p_g) and the vector \vec{vec}_{sp} (the vector from p_{i-1} to the shortest distance point on the obstacle). We then compute the cross-product $\vec{vec}_{p_g} \times \vec{vec}_{sp}$ using the determinants method. Based on the sign of $\vec{vec}_{p_g} \times \vec{vec}_{sp}$, we decide in which direction to move in order to avoid the obstacle. If the sign of $\vec{vec}_{p_g} \times \vec{vec}_{sp}$ is greater than 0, then we move in a clockwise direction or else move in a counter-clockwise direction to avoid the obstacle. Next, while in the loop, we keep on rotating p_i around the head, i.e., p_{i-1} in the signalled direction, by increasing/decreasing



(a) Path planning through a mathematically defined environment with circular obstacles in a $18\text{cm} \times 18\text{cm}$ space.



(b) Path planning through a mathematically defined maze environment.

Figure 1. Working of the SM for computing the points-based path trace in different environments

(by a small constant ϵ , e.g., 0.1) the angle of p_i , unless p_i satisfies the safety constraints. In the end, we return the safe point as the output of the algorithm. The sign of $\vec{vec}_{pg} \times \vec{vec}_{sp}$ can also be determined using the sine of the angle between the vectors \vec{vec}_{pg} and \vec{vec}_{sp} , i.e., the angle (α) from \vec{vec}_{pg} to \vec{vec}_{sp} in the counter-clockwise direction. Even in the cross-product $\vec{vec}_{pg} \times \vec{vec}_{sp} = |\vec{vec}_{pg}| \cdot |\vec{vec}_{sp}| \cdot \sin(\alpha) \cdot \hat{n}$ (where \hat{n} is a unit perpendicular vector to the plane containing \vec{vec}_{pg} and \vec{vec}_{sp}), it is the sine of α which determines the sign of the cross-product. Because of this fact, we named this approach the "sine method". To avoid the complexity of trigonometric operations, we preferred using the cross-product with the determinants method.

Property	Sine method	Tangent bug
Obstacle handling	Exploits the angle information between the shortest distance vector towards the obstacle and the vector towards the goal for computing the intermediate safe-path point by rotating in either clockwise or counter-clockwise around the shortest-distanced obstacle previous safe point (head).	Computes the tangent points with the obstacle using the range sensor placed at the head. Move to the shortest-distance tangent point using the heuristic distance with the sum of the distance from the tangent point and the shortest-distanced obstacle from the goal.
Obstacle avoidance	Iterate the obstacle handling mechanism (as illustrated in the above point).	Starts the boundary-following behaviour in the same way as the Bug 1 and Bug 2 algorithms.
Leaving obstacle	Leaves the obstacle tangentially towards the goal.	Leaves the obstacle from the shortest-distanced point between the blocking obstacle and the goal.

Table 2. Differences between the SM and tangent bug algorithms

Without loss of generality, assume that we have a continuum robot with a section(s) with a diameter of 2mm and that we want to plan a path through a mathematically defined environment of $180\text{mm} \times 180\text{mm}$ such that robot maintains a marginal distance of 3mm from the obstacles boundary. For the purpose of simulations, the obstacles' coordinates are known to the program; however, the algorithm takes the obstacles as unknown and finds the points-based path trace from the start to the goal points such that the path satisfies the imposed safety constraints. The algorithm has two safety parameters, i.e., the step size and the minimum marginal distance from the obstacles. The minimum marginal distance will be the sum of the radius of the continuum robot and the minimum distance to be maintained from the obstacles. For the above problem, the minimum marginal distance (min_marg_dist) will be passed as $1\text{mm} + 3\text{mm} = 4\text{mm}$. This is the minimum min_marg_dist value to be passed to the algorithm and the step size should be less than this value or else the robot may violate the safety constraints. For the above problem, if we pass the step size as 3.5mm and run the SM algorithm, then Figure 1 illustrates the resulting points-based path trace. If we decrease the step size value, then the resulting path becomes closer to the absolute path and reduces the path error. If we increase the step size value, then the resulting path behaves the other way around, i.e., it increase the path error. The other safety parameter, i.e., min_marg_dist , maintains the provided safety distance from the obstacle for the resulting points-based path trace.

It is to be noted that there exist some apparent similarities between the SM algorithm and the tangent bug algorithm [21], e.g., the motion towards the goal and the boundary following behaviour. However, both are theoretically different from each other, as illustrated in Table 2. The SM algorithm works more efficiently in avoiding convex obstacles, does not require the computation of tangent points, and leaves the obstacle more efficiently, resulting in a shorter distance compared to the tangent bug algorithm.

2.1.2 The Potential Field Method Algorithm

The SM algorithm results in a path trace with fewer oscillations; however, it has certain limitations, e.g., it provides a local optimal path, poses limitations in environments with special convex obstacles, and is somewhat time-expensive around the obstacles boundary. To address these limitations posed by the SM, we present an alternative approach to determine the points-based path trace. This approach is based on the customization of the traditional PFM originally presented by O. Khatib [19]. In the PFM, the goal generates an attractive potential field while the obstacle generates a repulsive potential field. The path direction is computed by combining the two potential fields and following the force induced by the new, combined field to reach the goal by avoiding obstacles. The basic idea is explained in Algorithm 3.

Algorithm 3 ComputeCRPath using PFM: computes the sequence of points connecting the starting point and the goal point such that the intermediate points satisfy the imposed constraints.

Input: $p_s, p_g, step, min_marg_dist$

Output: Sequence of points connecting p_s and p_g such that the intermediate points satisfy the imposed constraints, e.g., obstacle avoidance, safety parameter

Basic Idea:

$path := p_s$

$head := p_s$

while (distance between $head$ and p_g) < $step$ **do**

$\vec{PF}_{att} := attractiveField()$

$\vec{PF}_{rep} := repulsiveField()$

$\vec{PF}_{total} := \vec{PF}_{att} + \vec{PF}_{rep}$

$\widehat{direct} := -1 * (\text{unit vector of } \vec{PF}_{total})$

$head := head + (step * \widehat{direct})$

 Append updated head at the end of the $path$ array

end while

return $path$ after appending p_g at the end of the $path$ array

Algorithm 3. first initializes the path with the start point p_s and places the head pointer over the start point p_s . Next, the algorithm keeps on iterating until the distance between the $head$ and the goal point p_g is greater than or equal to the $step$. With each iteration, the $head$ pointer and path array will be updated with the intermediate path point. In the body of the iteration loop, we first find the attractive potential field vector \vec{PF}_{att} generated by the goal over the $head$ using the $attractiveField()$ function. This function returns the vector of the attractive potential field originating at $head$ by the goal. We then compute the repulsive potential field vector \vec{PF}_{rep} generated by obstacles over the $head$ using the $repulsiveField()$ function. This function returns the vector of the repulsive potential field originating at $head$ by the obstacles. The potential field values are taken between 0 and 1, inclusively. The value of the

attractive potential field will be low (approaching 0) at points further from the goal as the $head$ moves closer towards the goal where the potential field approaches 1. The same methodology is adopted for the repulsive potential field generated by the obstacles. The value of the repulsive potential field will be around zero at distances greater than the marginal distance away from the obstacles, and approaches 1 as the $head$ approaches the boundary of the obstacles at a distance less than the marginal distance from the obstacles.

After computing the attractive and repulsive potential fields, we compute the total potential field PF_{total} at the head point by summing-up both the potential fields. We then find the direction unit vector \widehat{direct} using the total potential field vector \vec{PF}_{total} . Using the direction vector \widehat{direct} , we update the coordinates of the head with the intermediate safety moves between the $head$ and p_g satisfying the constraints. For this, we find the point at the distance $step$ away from the $head$ in the direction of the unit vector \widehat{direct} . We then update the path trace by appending the updated $head$ coordinates. The algorithm keeps on iterating until it finds all the intermediate points from the start point p_s to the goal point p_g . Finally, the algorithm appends the goal point p_g with the path trace and returns the complete points-based path trace as output. The safety parameters work in the same way as they do for the SM algorithm.

2.2 The Constant Curvature Curves-based Continuous Path Planning Algorithm

Continuum robots do not contain any rigid links or joints and consist of a continuously bending architecture [4]; hence, for a continuum robot to follow a particular path, the path needs to be continuous. Additionally, a common property of the approximately constant curvature-bending of the backbone is virtually exhibited by all types of continuum robots [4, 5]. Hence, the motion produced for the continuum manipulator should be a constant curvature (circular) curves-based path movement, such that the path satisfies the design and kinematics constraints of continuum robots. The path trace resulting from the approaches presented in Section 2.1 is discrete and does not satisfy the design and kinematics constraints of continuum robots. Moreover, as discussed, continuum robots cannot follow such a path because of their design and kinematics constraints. To compute the path that satisfies the design and kinematics constraints of continuum robots, we have proposed a path planning algorithm based on constant curvature curves. The algorithm takes the points-based path trace (resulting from the algorithms presented in Section 2.1) and the initial tangent/orientation vector as input, and provides a constant curvature moves-based continuous path as output. The resulting path now satisfies the design and kinematics constraints of continuum robots. The algorithm is based on fitting the constant curvature

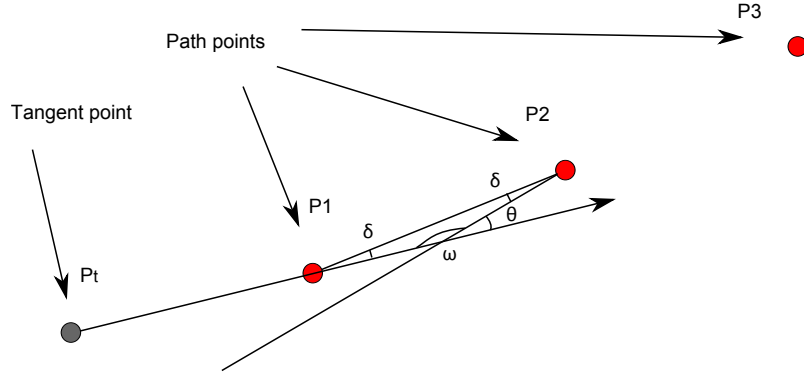


Figure 2. Fitting an isosceles triangle between points P_1 and P_2 to find the degree of curvature (θ)

curves between path points by maintaining the continuity of the path. The curvature is defined as the degree to which the curves deviate from being a straight line. For a curve, the rate at which the tangent vector changes direction with respect to the arc length is called the "curvature" (κ) [20]. In circular curves, the rate of change of the tangent vector's direction is constant; hence, these are called "constant curvature curves". The curvature is the reciprocal of the radius [3, 20]. Circles with a smaller curvature imply a larger radius, and vice versa. For a circle of radius r , the curvature κ can be computed as follows:

$$\kappa = \frac{1}{r} \quad (1)$$

Next, we define the theoretical aspects of the constant curvature curves-based continuous path planning algorithm by exploiting it geometrically. For input, we take the path points shown in Figure 2. To fit the constant curvature curve between path points P_1 and P_2 (highlighted in Figure 2), we need the initial direction to maintain the continuity of the path. This initial direction can be maintained using the tangent vector at point P_1 . This vector will be the tangent on the previously fitted curve, e.g., between P_0 and P_1 , at point P_1 . The constant curvature curve fitted between path points P_1 and P_2 needs to maintain the continuity with the path. The tangent vector at point P_1 on the previously fitted curve should guide this continuity and help us in pruning out all the other curves that do not satisfy the continuity constraints. Hence, we take point P_t such that it guides the tangent at point P_1 , as depicted in Figure 2. There are other methodologies to maintain the tangent [20], but we have used a points-driven tangent vector for maintaining continuity. This will also, later on, help in the selection of curves with the same curvature to satisfy the continuity constraint.

The vector from P_t to P_1 ($\vec{P_tP_1}$) is the tangent on previously fitted curve, e.g., between P_0 and P_1 , at point P_1 . $\vec{P_tP_1}$ will also be the tangent on the curve between P_1 and P_2 at point P_1 . With this methodology, we also provide the initial direction or tangent of the first section of the continuum

robot with the tip placed at point P_s . This initial direction/orientation guides the continuity of the path between the path points. For the purpose of this paper, a straight line-orientation (zero curvature) of the continuum robot is placed as a guiding vector for the path to maintain continuity. This orientation is placed such that the head of the first section of the continuum robot is at point P_s and the tail at the tangent point is passed to the algorithm. We can place any orientation of the continuum robot given the initial tangent vector at P_s .

We exploit the properties of an isosceles triangle to fit the constant curvature continuous curve between path points P_1 and P_2 . We find the bending angle or central angle (θ) of the curve to be fitted between P_1 and P_2 . This angle is also referred as the "degree of curvature". Using angle θ , we find the radius and hence the curvature of the curve using Equation (1). We fit the isosceles triangle such that one of the equal edges follows the tangent vector $\vec{P_tP_1}$, while the unequal edge is the line between P_1 and P_2 (as shown in Figure 2). Using the information of the angle δ between vectors $\vec{P_tP_1}$ and $\vec{P_1P_2}$, we find the angle θ adjacent to the unequal angle ω of the isosceles triangle. This has been represented pictorially in the Figure 2.

The unequal angle (ω) of the isosceles triangle can be computed using $\omega = 180 - 2(\delta)$. Based upon ω , we can compute the degree of curvature using $\theta = 180 - \omega$. The angle θ can be directly computed from δ as follows:

$$\theta = 180 - (180 - 2(\delta)) = 2\delta \quad (2)$$

The angle θ known as the bending angle or the central angle of the curve provides information for the circular curve between points P_1 and P_2 . To find the curvature of the curve, we first need to find the radius of the curve to be fitted between P_1 and P_2 . The radius (r) can be computed by exploiting the properties of the isosceles triangle CP_1P_2 and the right-angled triangle CP_mP_2 , as demonstrated in Figure 3.

tangent vector $\vec{P_{t1}P_{t2}}$, we can repeat the same methodology for points P_2 and P_3 , and so on. If we run the above approach over the points-based path trace depicted in Figure 1 (a), then the resulting constant curvature curves-based continuous path for a continuum robot with a diameter of $2mm$ is as presented in Figure 5.

3. Discussion

In this section, we compare our proposed algorithm with existing algorithms and analyse the novelties of our research work. The PoPP algorithm computed the path by imposing safety constraints and catered for the limitations of existing approaches. The algorithm has two safety parameters: the step size and the minimum marginal distance. The step size should be less than the minimum marginal distance so as to ensure that the continuum robot does not collide with an obstacle. If we set the step size as so small that it becomes negligible when compared to the minimum marginal distance, then the path becomes very close to the true path and the path error becomes negligible. There exists a good trade-off between accuracy and efficiency. By decreasing the step size, the path error decreases as well, but it increases the running time of the algorithm. This works the other way around when we increase the step size. The second safety parameter - the minimum marginal distance - maintains the safety distance from obstacles. We proposed two algorithms for computing the points-based path trace, i.e., the SM algorithm and the PFM algorithm. The SM algorithm computes the path trace with fewer oscillations, better smoothness and fewer chances of getting stuck in the local minima. However, it provides the local optimal path, works efficiently for

avoiding convex obstacles, needs improvements for avoiding special concave obstacles, and is somewhat time-expensive around the obstacles boundary. On the other hand, the PFM algorithm addresses the limitations of the SM algorithm, to some extent, but the path generated may have oscillations and may become stuck in the local minima. The PFM algorithm is faster compared to the SM algorithm, and enormous research has been carried out regarding its applications in traditional rigid link robots which can be incorporated for an efficient path trace. From Figure 6, we observed that the SM algorithm is slightly more time-consuming compared to the PFM algorithm.

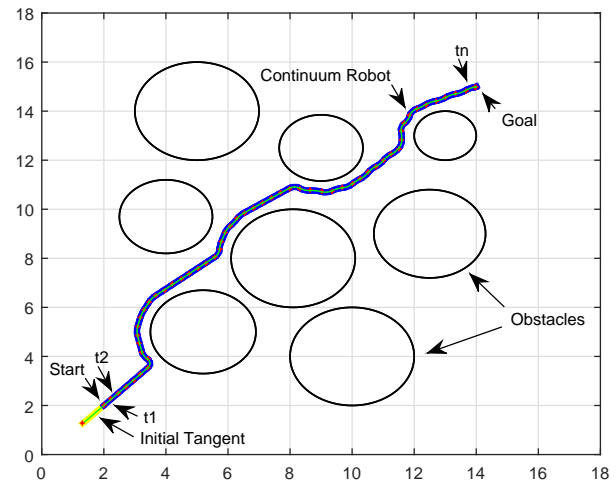


Figure 5. Constant curvature curves-based continuous path for a continuum robot with a diameter of $2mm$ in a $18cm \times 18cm$ space

Algorithm	Robot Independent	Safety	Obstacle Avoidance	Time	Kinematics & design constraints	3D	Dynamic Workspace	Globally Optimal
PoPP	Yes	Yes	Yes	Approx 0.47 Sec	Yes	No	No	No
Minimal energy curves based path planning	Yes	No	No	–	No	Yes	No	No
GPU based path planning	Yes	No	Yes	Few Hours	No	Yes	No	Yes
Points-based path planning for snake-like robots	No	No	Yes	–	No	No	No	Yes
Lyons motion planning algorithm for active cannulae	No	No	Yes	One Min for three sections	Yes	Yes	No	No
Sensors-based path planning algorithm	No	No	Yes	20 Sec for three sections	Yes	Yes	Yes	No
RRT*-based path planning algorithm	No	No	Yes	Order of Sec for three sections	No	Yes	No	No

Table 3. Comparison of PoPP with existing algorithms

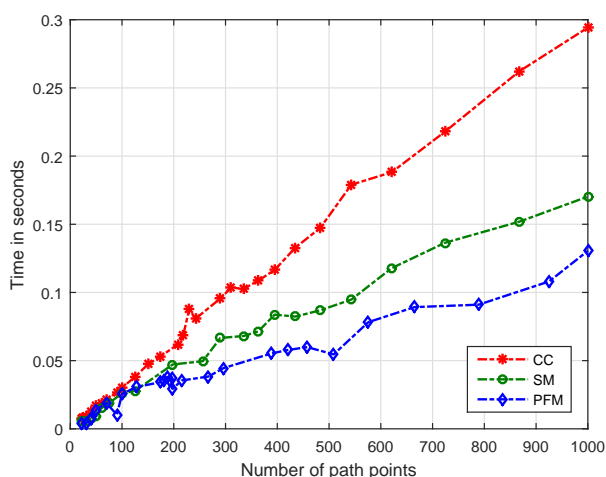


Figure 6. Time complexities graph for points-based path trace algorithms (SM algorithm and potential field algorithm) and constant curvature curves-based continuous path planning (CC)

Continuum robots consist of continuously bending architectures [4]; hence the path of continuum robots needs to be continuous. Furthermore, the backbones of all types of continuum robots can be visualized as a serially arranged, approximately constant, curvature curves-based configuration [4, 5]. Hence, the path moves should consist of constant curvature (circular) curves-based path moves such that the path satisfies the design and kinematics constraints of continuum robots. The points-based path trace resulting from the approaches presented in Section 2.1 (SM algorithm and PFM algorithm) is discrete and does not satisfy the design and kinematics constraints of continuum robots. This path trace is used by the constant curvature curves-based continuous path planning algorithm as input and computed the constant curvature curves-based continuous path for the continuum robots. From Figure 5, we can visualize that the path generated by the constant curvature curves-based continuous path planning algorithm is continuous and is based on constant curvature curves-based path moves. Hence, the resulting path satisfies the design and kinematics constraints of continuum robots. The smoothness and curvatures of the path depends upon the smoothness of the points-based path trace passed as input to the constant curvature curves-based continuous path planning algorithm. Fewer oscillations and curvature changes in the points-based path trace result in a smoother constant curvature curves-based continuous path. Figure 5 shows the configuration of an n -section continuum robot at different time slots, following the initial configuration t_1, t_2, \dots, t_n . For the path in Figure 5, $n=56$ (with 57 path points). Figure 7 shows the curvature variations of the path at different time slots. These curvature variations can be used for computing the volume of the path hose and dexterity for the robot's movement at a certain time slot. From Figure 6, we can observe the time complexity of the constant curvature curves-based continuous path planning algorithm (CC). We observed that the constant curvature curve fitting algorithm takes approximately 0.3 seconds for a path consisting of 1,000 path points. MATLAB 7.10 R2010a has been used with a 2.5 GHz

PC processor for generating the simulations and computing the time complexities of the algorithms proposed in this paper.

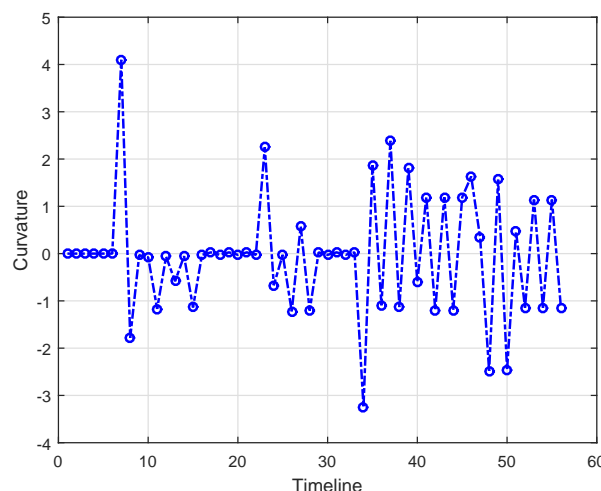


Figure 7. Curvature of the path at different time slots

The PoPP algorithm takes approximately 0.47 ($0.3 + 0.17$) seconds for computing a complete path consisting of 1,000 path points (Figure 6). The PoPP algorithm has been demonstrated to be more efficient compared to existing algorithms (see Table 3 for a comparison). This is because, firstly, it is points-driven and, secondly, it does not find the configurations of all the sections of the continuum robot for different time slots. Rather, the body of the continuum robot follows the path trace made by the head (first section) of the continuum robot. As such, the algorithm simply finds the configurations of the first/head section of the continuum robot, while the rest of the body follows the path trace of the head section in consecutive time slots. The 1,000 points-based path can be visualized as any number of sections of a continuum robot, ranging from 1-999, following the path in different time slots. Hence, the algorithm exhibits sound potential for applications with continuum robots with any number of sections to achieve the desired safety with speed. Additionally, the algorithm is applicable to all types of continuum robots satisfying the property of constant curvature-bending for their manipulators. This also addresses the limitations of existing algorithms for the path planning of continuum robots, which are applicable to specific types of continuum robots with a limited number of sections (mostly three) and which are computationally expensive (ranging from seconds to hours) with limited accuracy.

4. Conclusion

In this paper, we proposed a PoPP algorithm for continuum robots in two-dimensional environments. Existing algorithms for the path planning of continuum robots are robot-specific and do not provide safety measures, assurance or control. They are computational-

ly expensive and mostly they do not incorporate design and kinematics constraints. Safety assurance is extremely important for the approval and usage of continuum robots in safety-critical applications. In the PoPP algorithm, we exploited the constant curvature backbone bending property of continuum robots, and provided a constant curvature curves-based continuous path satisfying the safety, design and kinematics constraints of continuum robots. The algorithm also addresses the limitations of existing algorithms for the path planning of continuum robots. We have confirmed that the algorithm also provides information about the volume of the path hose and flexibility of movement at different time slots. The algorithm exhibits promising potential for applications with all types of continuum robots - following the constant curvature backbone bending property - to achieve the desired safety with efficiency. In addition, the algorithm provides a good trade-off between accuracy and efficiency that can be exploited for safety-critical applications of continuum robots, e.g., minimally-invasive surgeries. Future research is in progress to optimize the performance of the points-based path trace algorithms and the constant curvature continuous path planning algorithm.

5. Acknowledgements

The authors would like to thank National University of Sciences and Technology (NUST), Islamabad, Pakistan for their support to complete this research project.

6. References

- [1] Cianchetti, M., Ranzani, T., Gerboni, G., Nanayakkara, T., Althoefer, K., Dasgupta, P., Menciassi, A. (2014) Soft robotics technologies to address shortcomings in today's minimally invasive surgery: the stiff-flop approach. In: *Soft Robotics*, 2014. pp. 122–131.
- [2] Cowan, L.S., Walker, I.D. (2008) "Soft" continuum robots - the interaction of continuous and discrete elements. In: *Artificial Life XI*, 2008. pp. 126–133.
- [3] Iqbal, S., Mohammed, S., Amirat, Y. (2009) A guaranteed approach for kinematic analysis of continuum robot based catheter. In: *Robotics and Biomimetics (ROBIO)*, 2009 IEEE International. doi: 10.1109/ROBIO.2009.5420395, Dec. 2009. pp. 1573–1578.
- [4] Walker, I.D. (2013) Continuous backbone "continuum" robot manipulators. *ISRN Robotics*. Available from: <http://dx.doi.org/10.5402/2013/726506>. doi: 10.5402/2013/726506, Vol. 2013: 19, 2013. Accessed on 8 May 2014.
- [5] Webster, R.J., Jones, B.A. (2010) Design and kinematic modeling of constant curvature continuum robots: a review. *International Journal of Robotics Research*. doi:10.1177/0278364910368147, ISSN: 0278-3649, Vol. 29(13), Nov. 2010; Thousand Oaks, CA, USA. pp. 1661–1683.
- [6] Gigras, Y., Gupta, K. (2012) Artificial intelligence in robot path planning. In: *International Journal of Soft Computing and Engineering (IJSCE)*, 2012. pp. 2231–2307.
- [7] Oroko, J., Ikua, B. (2012) Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: a review. *Mechanical Engineering Conference on Sustainable Research and Innovation Proceedings*. ISSN:2079-6226, Vol. 4, 2012.
- [8] Moll, M., Kavraki, L.E. (2004) Path planning for minimal energy curves of constant length. In: *IEEE International Conference on Robotics and Automation (ICRA)*. doi:10.1109/ROBOT.2004.1307489, ISSN:1050-4729, Apr. 2004. pp. 2826–2831.
- [9] Moll, M., Kavraki, L.E. (2005) Path planning for variable resolution minimal-energy curves of constant length. In: *IEEE International Conference on Robotics and Automation (ICRA)*. doi:10.1109/ROBOT.2005.1570428, Apr. 2005. pp. 2130–2135.
- [10] Moll, M., Kavraki, L.E. (2006) Path planning for deformable linear objects. *IEEE Transactions on Robotics*. doi:10.1109/TRO.2006.878933, ISSN: 1552-3098, Vol. 22(4), Aug. 2006. pp. 625–636.
- [11] Gayle, R., Segars, P., Lin, M.C., Manocha, D. (2005) Path planning for deformable robots in complex environments. In: *Robotics: Science and Systems*, 2005, Citeseer. pp. 225–232.
- [12] Conkur, E.S., Gurbuz, R. (2008) Path planning algorithm for snake-like robots. *Information Technology and Control*, Kaunas, Technologija. Vol. 37(2), 2008, Citeseer. pp. 159–162.
- [13] Lyons, L.A., Webster, R.J., Alterovitz, R. (2009) Motion planning for active cannulas. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. doi:10.1109/IROS.2009.5354249, Oct. 2009. pp. 801–806.
- [14] Godage, I.S., Branson, D.T., Guglielmino, E., Caldwell, D.J. (2012) Path planning for multisection continuum arms. In: *International Conference on Mechatronics and Automation (ICMA)*. doi: 10.1109/ICMA.2012.6283423, Aug. 2012. pp. 1208–1213.
- [15] Maciejewski, A.A., Klein, C.A. (1985) Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*. Vol. 4(3), 1985. pp. 109–117.
- [16] Croom, J.M., Rucker, D.C., Romano, J.M., Webster, R.J. (2010) Visual sensing of continuum robot shape using self-organizing maps. In: *IEEE International Conference on Robotics and Automation (ICRA)*. doi:10.1109/ROBOT.2010.5509461, ISSN:1050-4729, May 2010. pp. 4591–4596.

- [17] Bergeles, C., Dupont, P.E. (2013) Planning stable paths for concentric tube robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). doi:10.1109/IROS.2013.6696792, ISSN:2153-0858, Nov. 2013. pp. 3077–3082.
- [18] Karaman, S., Frazzoli, E. (2011) Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, Vol. 30(7), 2011. pp. 846–894.
- [19] Khatib, O. (1986) Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.* doi:10.1177/027836498600500106, ISSN:0278-3649, Vol. 5(1), Apr. 1986. pp. 90–98.
- [20] Thomas, G.B., Finney, R.L. (1996) *Calculus and analytic geometry*. ISBN:0-201-53174-7, Addison-Wesley, 9th Edition, 1996. pp. 876–892.
- [21] Choset, M.H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W., Kavraki, L.E., Thrun, S. (2005) *Principles of robot motion: theory, algorithms and implementation*. ISBN:978-0262033275, MIT Press, 2005. pp. 17–38.