International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 12(2020) pp. 212-221
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Trajectory planning in autonomous systems: A Recursive Tangent Algorithm

Adhiraj Shetty[1], **Annapurna Jonnalagadda**[2] **and Aswani Kumar Cherukuri**[3]

[1]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India
adhirajshetty97@gmail.com

[2]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India
jannapurna@gmail.com

[3]School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India
cherukuri@acm.org

Abstract: **Autonomous drones play a vital role in Disaster mitigation systems and commercial good delivery systems. The problem involves finding the shortest path between the delivery points while simultaneously avoiding stationary obstacles (for example high raised buildings) and moving obstacles like other drones. The path needs to be continuously changed based on the telemetry from other drones or based on the addition of new way-points. This is major issue in planning problems. Any algorithm will have to make complex choices like abandoning shortest paths to avoid collisions. In this paper we propose a tangent algorithm which chooses paths based on many performance measures like number of obstacles in current path and the future path and the distance to the next obstacle. The path has very few sharp turns and the locations of these turns are calculated during the path planning. This solves one of the major problems for fast-moving fixed wing systems.**
The performance evaluation on different environments demonstrates that the algorithm will be particularly faster in case of sparse obstacles since it always starts first by drawing a straight line between way-points and if there are no obstacles in the way then it can exit in a single step.

Keywords

Autonomous systems, Path planning,Recursive algorithm,Tangent algorithm

I. Introduction

There are a lot of applications of autonomous drones. The most important one is delivering food or medicine to people stuck in inaccessible areas due to natural disasters. There are many other applications like commercial deliveries or military surveillance. These tasks can be accomplished faster if multiple drones are sent together. The drones in this environment need to avoid each other and other stationary obstacles and find the shortest path to their next delivery point.
The research that is being dealt with in this composition surrounds path planning and collision avoidance in unmanned aerial vehicles. Each plane is sent a specific distance from the ground station after takeoff and will then orient itself to travel to the destination point. The focus here surrounds what happens should there be multiple stationary obstacles like tall structures or other drones flying around with their own different destinations. In recent times, this subject has become one of increasing interest to many. As such there are many preexisting algorithms one can consider but many of these require high computational power and still do not generate the most optimal path.
Delivery drones flying over cities will (in most cases) be flying at a height where not many stationary obstacles like buildings or trees will be in the way. We may have few no-fly zones like schools or residential areas where the drone will have to increase altitude or altogether avoid the area to prevent discomfort caused by noise from the drone. These no-fly zones can be modeled as cylinders with their base on the ground, radius and centre such that the entire area is covered and a suitable height. Even moving obstacles like other drones can be also modeled as cylinders centered at the drone. Most of the time the obstacles will be sparse as such the tangent algorithm will quickly return the straight line path in one step. Many of the pre-existing algorithms like RRT will go through many iterations even if there are no obstacles in the path, also the path generated by RRT will not be a straight line and will require smoothing.
The tangent algorithm is much simpler than existing concepts and generates shorter paths much faster if the area of operation is huge with sparse obstacles.This algorithm is particularly suitable for fixed-wing planes that cannot take sharp turns easily as it creates paths made up of straight lines with minimal turns. Also, the points with minor bends are stored during run time and can be smoothed out.
The aim behind the inception of this algorithm is to create a method for path planning and obstacle avoidance that makes use of simple geometry like circles and tangents. Therefore, students with basic knowledge of geometry and recursion can create working codes autonomous aerial systems. The aim was also to create an algorithm that can be suitable for fixed-

This paper is an extension of ”Recursive Tangent Algorithm for Path Planning in Autonomous Systems,” presented by authors at HIS 2019

wing system as in to generate the shortest path with minimum number of sharp turns in the fewest amount of iterations. Furthermore, we wished to create an algorithm where decision making process can be customized that is, by the use of different performance measures and different weights corresponding to the importance of each performance measure.
In this paper, we propose a new algorithm for path-planning and obstacle avoidance and comparison with other path-planning algorithms. Our main contributions are as follows:

- A new algorithm for path planning and obstacle avoidance in autonomous systems.
- The algorithm uses a speculative approach where it assumes a path and then corrects the path if there is a collision. Simulation results show that it is much faster in comparison to already existing algorithms like A* and RRT.
- Amongst all the algorithms mentioned in this paper, the tangent algorithm generates paths with the fewest amount of turns and so, will require the least amount of smoothing. The points where the path takes a turn are calculated during run-time and can be smoothed in parallel with the algorithm execution.
- We have implemented the algorithm to decide which path to take based on three performance measures. The measures have different weights signifying their importance in the calculation. These measures and their weights can be easily modified or new ones can be added in future implementations.
- We have also highlighted some future improvements in the algorithm with the help of parallel execution using threads.

II. RELATED RESEARCH

Many solutions have been formulated to both detect and avoid collisions among UAVs or between UAVs and other obstacles. These solutions are generally termed as 'algorithms' because they are a means to accomplishing an end by following a formulated process. Each of these algorithms has a different outlook for the problem and therefore its own benefits and downfalls.[1]

A. Decentralized collision avoidance: A method of collision avoidance that is more popular in larger and more powerful unmanned aerial systems. This family of algorithms takes a more 'divide and conquer' approach when faced with the issue of drone navigation. A centralized system acts as the single control unit that handles all of the navigational algorithm for all the units. The units (drones or planes), simply receive commands and go to the specified location. In a decentralized system, the vehicles have on-board processing power so they calculate their own paths. Each unit will either use radars or local broadcast communication to detect any nearby vehicles. Any impending collisions are dealt with locally that is, between the two units rather than on a global scale[11].

1) Generalized Roundabout Policy: This algorithm was developed using decentralized control[2]. It is a reactive algorithm which gives each individual plane a disc of reserved space centered on it. The disc is modeled with respect to the plane's maximum turning angle. It is rendered so that when the plane enters its maximum turn, the center of the disc does not change position. This means that not only can the disc change direction of motion as the plane does, but it is also able to stop moving when the plane starts circling at its maximum turn angle (drone copters can just stay stationary mid-air and do not need to circle). When the plane detects that it is about to enter another planes's disc space, it can change direction by circling around in order to avoid entering the foreign disc space. Sometimes one of the air-crafts might get caught in between two or more other air-crafts. When this situation occurs, the central aircraft can easily halt the motion of its disc and remain in place by circling inside its disc. It can move again once one or more of the air-crafts have freed themselves from the situation.

2) Multiple Party Collision Avoidance: Another documented algorithm that utilizes the divide and conquer approach is the Multiple Party Collision Avoidance algorithm[3]. Instead of dividing the airspace, this algorithm divides the air-crafts into a number of groups, or 'parties'. Each party contains the aircraft that are close to each other and have a chance of being involved in a collision in the near future. Every party is assigned a master plane from amongst the party members. The choice of a master does not affect the outcome whatsoever as in any plane can become the master. The master plane then begins to computes possibilities of future collisions in the group and attempts to resolve them.

Both the algorithms discussed under Decentralized collision avoidance are extremely effective in cases of high obstacle density that is when there are a lot of unmanned aerial vehicles(UAV) in a tight space. This is because dividing the problem and computing different components in parallel will be faster for a large problem, rather than solving the entire problem linearly. However, decentralizing a system takes up a lot more computational power than just using a single central unit and it might not be commercially feasible to have every UAV fitted with the necessary hardware.

B. Artificial Potential Fields: Artificial Potential Fields is a reactive algorithm that works based on the behaviour exhibited by charged particles in nature[4]. The process involves hypothetically giving each UAV a negative charge, and the next way-point a positive charge. In this model, all UAVs will repel each other and simultaneously be attracted to their way-points. In order to move the UAV, a force vector that is calculated from the potential field is generated at each step. The UAV can then applies this force to itself. The main drawback of this algorithm is the tendency of the UAVs to get stuck at points of local minima in the potential field instead of moving to the way-point(global minima).

C. Fuzzy Logic: Fuzzy logic is based on the idea that humans think in terms of concepts rather than exact numbers. The inputs are fuzzified, or made more general than specific numbers. This involves assigning terms such as Far or Near for the distance of the UAV from the goal and Negatively Small or Positively Large for the angle at which the goal is oriented with respect to the UAV[5]. These terms are then combined into a single value representing a velocity for the UAV's motion. Due to the algorithm's inexact nature, it is really important to implement the translation of values to concepts properly to get accurate results. However, experiments have shown that it performs well in both static and moving obstacle environments.

D. A* : A*(A-star) is a preemptive, path generating collision avoidance algorithm[6]. A* algorithm first divides the airspace into a grid of squares of same size, also called 'nodes'. The next step is to use a 'branching' algorithm to determine the most optimal path for every UAV. This is done by assigning a cost to every node that the UAV might pass through in its flight path and estimating the path with the least cost that is possible using the node under consideration. The algorithm also makes use of a process called 'bounding'. This restricts the algorithm to only branch out paths from the node with the least estimated cost, thus ensuring that only the most cost efficient path will be chosen.

E. RRT: A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search for way-points by randomly building a space-filling tree[7]. The tree is constructed incrementally from nodes drawn randomly in the search space. The search tree is biased to grow towards large unexplored areas in the space.

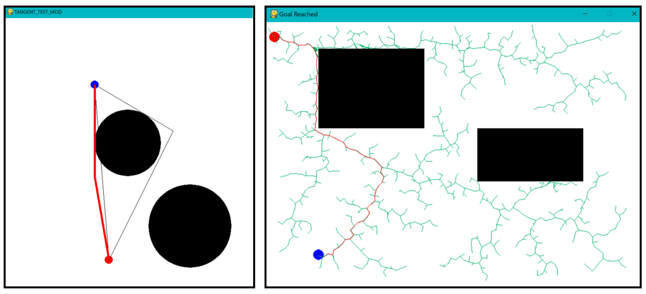


Figure. 1: **Pygame Simulation: Comparison between Tangent algorithm(left) and RRT(right) in situations with sparse or no obstacles in the way.**

Algorithms like Rapidly Exploring Random Tree (RRT) and A* algorithm take longer time even if there are sparse obstacles as shown in Fig.1 because they are based on randomly testing out each possibility. The tangent algorithm is a better solution as it directly tries to draw a straight line and finish the process, even in case of obstacles it makes minor deviations from the straight line path.
Also paths generated by RRT, A* and other algorithms that randomly move around may need a lot of smoothening as the paths are (in most cases) made up of small lines with different slopes. Each small line shows one decision made by the agent. But the tangent algorithm only needs smoothening at a few locations where we draw new tangents because it directly draws a single big line and tries to reach the next point in one step. Thus, autonomous drones will be safer as the chances of having to take sharp turns is less[12].
As can be seen in Fig.1 the RRT agent takes an unnecessarily longer path because a longer sub-tree happened to reach the goal first. Also, the path generated by RRT(in red) is not smooth and straight unlike the tangent algorithm because the RRT path is made up of small branches.

F. Dynamic Programming (DP)
Dynamic programming breaks down the problem into multiple related sub-problems. Solving the main problem requires the solutions from all the related sub-problems. The solution of sub-problems is stored in a memory structure for future needs[13]. In regards to UAV path-planning, the procedure of the DP algorithm is to calculate the distance to the goal from all the way-points on the map and the sub-problem is the pre-computed distance to the nearby way-points. Recurrence relation is a common method for demonstrating the relationship between a problem and its sub-problems. DP algorithms are the basis for many types of algorithms like Dijkstra's algorithm. In these types of algorithms, the lowest level or the smallest sub-problem is considered first, then the algorithm proceeds toward the higher level sub-problems, the main solution is thus found in an iterative fashion.

G. Greedy algorithm and Multi-Step Look-Ahead Policy (MSLAP)
A greedy algorithm is any algorithm that follows the problem-solving heuristic of choosing the best or locally optimal solution at each stage with the intent of finding a global optimum. Greedy algorithms can be characterized as being 'short sighted', and also as 'non-recoverable'. For many other problems, greedy algorithms fail to produce the optimal solution, and may even produce the unique worst possible solution. The MSLAP algorithm works by discretizing the UAV decision tree and then evaluating the different multi-step UAV path decisions for the most optimal performance. However, the main disadvantage is that the problem is NP-hard, this means that the computation time is highly dependent on the number of decision variables in the problem. Defining the size of decision possibilities for the UAV modulates the cardinality of set of decision variables. The major advantage of this algorithm is that a sub-optimal solution requires less computation time and hence is faster when compared to the optimal trajectory planning methods[14].

The Tangent Algorithm proposed in this paper also follows the Dynamic programming approach. The algorithm calls itself and in each call it draws a new tangent based on a greedy decision to get the best performance measures for that particular tangent and then passes the results to the next call. Further in the paper, we also propose a thread based approach to the tangent algorithm which is entirely based on dynamic programming where each tangent or sub-problem is passed on to a new thread.

H. Fringe algorithm
Path-planning solutions that apply fringe algorithms try to remove inefficiencies by making each data structure iterate over two sets of data: the frontier and the fringe. This means

that there will be two sets of lists, one which stores the current iteration and the other to store the next iteration. The Fringe algorithm can be shown to accelerate the search times by 10–40% when compared to the A* path-planning algorithm[15].

I. Approximate Reinforcement Learning (RL)

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, along with supervised learning and unsupervised learning. RL program learns to take actions with the goal to maximize a reward signal. The earned reward is the feedback for the next action. The agent can explore the environment for better path selection in the future. Value functions, functions of the states and performance measures estimate the degree of importance that the agent gives to a particular state [16].

J. Mixed Integer Linear Programming (MILP)

An integer programming problem is a mathematical optimization or feasibility algorithm in which some or all of the variables are restricted to be integers. In many settings the term refers to integer linear programming, in which the objective function and the constraints are linear. Linear programming maximizes (or minimizes) a linear objective function subject to one or more constraints. Mixed integer programming adds one additional condition that at least one of the variables can only take on integer values.
MILP is widely used for mathematical modeling and is known as a powerful tool for presenting optimal and near-optimal solutions. Implementation of MILP as a solution to the UAV path-planning while taking into consideration the probability of detection of other UAVs in the airspace is covered in [17].

K. Genetic Algorithm

GAs are search algorithms based on the model of evolution which seeks to imitate natural selection and survival of the fittest. Paths (each of which is encoded as a chromosome) are evolved by the genetic parse trees whose lengths changes throughout the run-time. The GA optimizes the population of paths based on the fitness landscape derived by a function called the objective function. The objective function checks the fitness for each solution. Different implementations of gene fitness functions, crossover functions, and mutation functions will affect the performance of the algorithm. The algorithm generates an evolution process based on operations like mutation, crossover and reproduction. The result is an iterative process where successive populations are generated until an optimal solution is obtained [18].The major advantage of using evolutionary algorithms is that the time of computation is not directly dependent on the number of constraints.

M. Radmanesh et al., 2018 [9] and M. Radmanesh et al., 2017 [10] provide advanced mathematical solutions for path planning, using Bayesian frameworks and partial differential equations respectively, in uncertain, hostile environments with multiple drones.
Mohammadreza Radmanesh et al., 2018 [8] is a comparative study of many of the algorithms mentioned above.This paper tests the performance of each algorithm by comparing the computational time and solution optimality, and also tests each algorithm with variations in the availability of global and local obstacle information.
Most recent works in the field of autonomous drones make use of the GNSS (Global Navigation Satellite System) and a compass as the main navigational tools in the drone. [19] uses an object detection module that helps in identifying obstacles in the video stream by using a combination of MobileNet and the Single Shot Detector (SSD) framework for a fast and efficient deep learning-based implementation of object detection. The autopilot is implemented using Erle-Brain 3 which consists of a Linux based embedded system and an autopilot shield on which the entire system is designed. (Robot Operating System) ROS and Autopilot Software are installed on the system. GPS is used to get information on the current position of the drone. Precise positional data is necessary to have a smooth navigational performance. In order to acquire this precise data, a large amount of satellite data needs to be collected which is why the use of GNSS and compass that can connect to many variations of satellites is necessary. The autopilot shield used in the drone consists of sensors and other essential components for flying a drone.

Wyder PM et al., 2019 [20] is also a noteworthy work which focuses on Unmanned Aerial vehicles that autonomously detect, hunt and take down other small aerial vehicles in a GPS-denied environment. With the increasing threat of misuse of drones for privacy violations and military activities, a drone system to report and neutralize these threats is necessary. The system proposed in this work detects, tracks and follows another drone within its sensor range using a machine learning model that is pre-trained. The target drone is detected and tracked using the input images captured by the drone camera. The output bounding box location and size of the detected drone are sent to the navigation-control system where the co-ordinates of the target are calculated with reference to the drone using algorithms such as regional convolutional neural network.

Islam et al., 2019 [21] proposes a novel mission-oriented path planning algorithm for multiple Unmanned Aerial Vehicles. In this algorithm, each drone takes autonomous decisions to calculate its flight path towards the target mission area while avoiding collisions with stationary and mobile obstacles. One potential applications of this algorithm would be to have a group of drones collectively cover an evolving forest fire and thus provide firefighters with a virtual reality model of the area. The main distinction with other algorithms is that the target destination for each drone is not fixed prior to take off and the drones position themselves such that they can collectively cover a time-varying mission area. The algorithm was formulated based on Reinforcement Learning (RL) with a new method to accommodate continuous state space for adjacent locations.

III. Proposed System 1

The Algorithm takes any number of way-points and any number of cylindrical obstacles. The cylindrical obstacles are no-fly zones where the drone cannot enter, these cylinders can be used to encompass tall towers or the locations of other drones so that the drone does not crash into them. The algorithm starts from the first way-point and if the height is sufficient it passes over the obstacles to get to the next way-point. If the obstacle height is significantly higher, then the cylinders are considered as circles from the top view and tangents are drawn to the closest circle blocking the way to the next way-point.

The system will be a goal-based, Utility-based agent which will have its goal to reach the next waypoint while choosing the path with the least number of obstacles and least distance. After a waypoint is captured the goal becomes the next waypoint. The environment is fully observable, partially cooperative multi- agent, deterministic, episodic, dynamic and continuous.

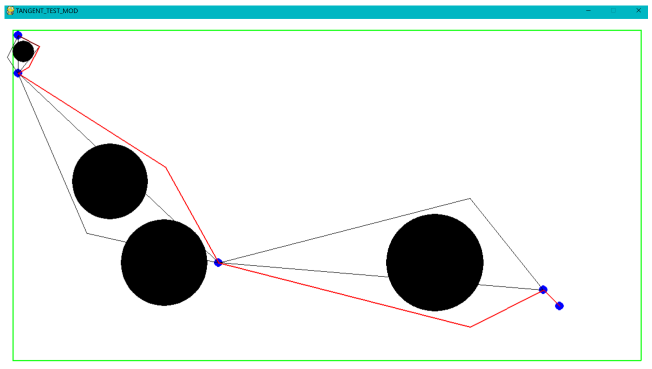


Figure. 2: Pygame simulation of the Tangent Algorithm.

The lines drawn in black in Fig.2 are all the unsuccessful attempts. First the agent tries to reach the target through a straight path, if an obstacle is detected two tangents are drawn to that obstacle and the best one is chosen.

This algorithm first selects the tangent which intersects the least number of obstacles. If the algorithm detects that both tangents have same number of obstacles then decision is made by drawing two straight lines from the end of the tangents to the destination using the two current tangents and checking their obstacles, if there is still a discrepancy then decision is made on the basis of the distance between the end point of tangent and next way-point.

A gap is maintained between the tangent and the obstacle. The tangent is extended beyond the point of contact with the circle to prevent the agent from recursively operating on the same circle as shown in Fig.3.
Once a tangent is chosen, the code recursively draws tangents on the obstacles lying in its path and after each tangent, again attempts to reach the way-point through a straight-line path. The tangent will be extended by a factor directly proportional to the radius of the obstacle and inversely proportional to the distance between the start point and the centre of the circle.

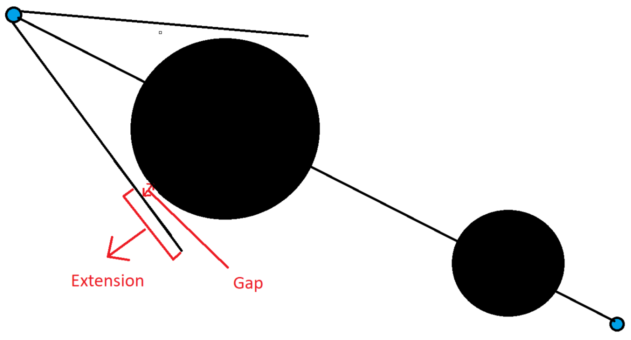


Figure. 3: Extension and Gap.

Between the first and second way-point (as can be seen in the fig.2) the agent has to abandon the shortest path because that would take the drone out of the boundary set for the flight.
After each iteration, an obstacle or way-point can be added or removed and the path will be recalculated. This can be used for moving obstacles. Other drones will constantly send telemetry to each other, based on the telemetry the agent can move the circle that represents the specific drone and recalculate its path.
So basically, each drone will be encompassed in a cylinder that other drones cannot enter. Stationary obstacles will also be modelled in the same way regardless of their shape.

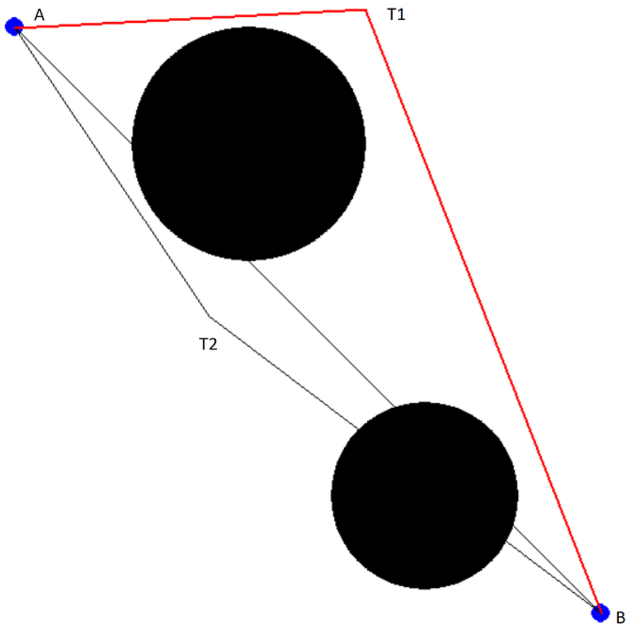


Figure. 4: Decision made based on three measures.

For example in Fig.4 :

1. Draw path AB. Not possible.
2. Draw tangents AT1 and AT2 on the nearest obstacle.
3. Check number of obstacles obstructing AT1 and AT2. Both are zero.
4. Next check number of obstacles obstructing T1B and T2B. T1B has zero and T2B has one. So choose T1 tangent.

5. Since the previous test passed, no need to check for distance between T1B and T2B.

Encountering less obstacles is given more priority than distance to maintain completeness. In the case that the way point is surrounded by obstacles and there is only one small gap, the drone needs to focus on getting out of the tight situation even if more distance is covered to get to destination.

IV. Proposed System 2

Alternately, we can ignore the three measures and use threads to test out every different possibility. Tangents will not be chosen based on any measures, on the contrary all the tangents will be expanded. Each tangent will call two more tangents and this process keeps on going until all the paths reach the destination or quit. Distance will be calculated for all path and the shortest path will be chosen. This method takes more time and processing power than the first method but it ensures an optimal solution. Also, even in the case where the obstacles touch each other and form a ring around the start point, only leaving a narrow gap, completeness is still ensured.

V. Algorithm

The algorithm for the first proposed system will input two input arrays: waypoint array with each element as a tuple of x and y coordinate and obstacle array with each element as a tuple of x,y coordinate and radius.

For each pair of consecutive way-points draw a rectangle with the two way-points as two diagonally opposite corners. Increase x-coordinate of 2 rightmost points by diameter of largest obstacle, Decrease x-coordinate of 2 leftmost points by diameter of largest obstacle, Increase y-coordinate of 2 topmost points by diameter of largest obstacle, Decrease y-coordinate of 2 bottom-most points by diameter of largest obstacle. Thus each consecutive pair of way-points will have its rectangle. Take each obstacle and if the end points of its two diameters which are parallel to x-axis and y-axis lie within the rectangle, then add it to the list.
If we have n way-points then list[n-1] will have each element as a list of coordinates of the obstacles within the the rectangles.
cor is an array containing the four corners of the boundary.

This algorithm is a two-dimensional simulation. It implements the concept of adding the gap and extension.But,the algorithm will not recursively reduce them if there is a collision because of excessive gap or extension. So we will choose the smallest possible values for the gap and extension from the start. The algorithm only takes in circular obstacles. But, regardless of shape any obstacle can be modeled as the smallest possible circle that encompasses the entire obstacle.

Algorithm 1 Path planning and obstacle avoidance using tangent algorithm

```
procedure COLLISION($st, ed$)
    Satisfy the line between st and ed with each circle in list[count] and find points of intersection.
    closest obs_no = circle with point of intersection closest to st point
    obs_count = Number of circles intersecting
    return closest obs_no , obs_count
procedure TANGENT($st, ed, ob$)
    Using point st draw two tangents on the obstacle[ob] ▷ Can use pole polar concept in conics to quickly satisfy the polar with the circle and get the 2 points
    Let the points of contact be s1 and s2.
    Add gap:
    x = obstacle[ob][0]                      ▷ x-coordinate of obstacle
    y = obstacle[ob][1]                      ▷ y-coordinate of obstacle
    sdx1 = x+1.1*(s1[0]-x)                   ▷ Equation 1
    sdx2 = x+1.1*(s2[0]-x)
    sdy1 = y+1.1*(s1[1]-y)
    sdy2 = y+1.1*(s2[1]-y)
    Add Extension:
    d1 = distance between st and (sdx1,sdy1)
    d2 = distance between st and (sdx2,sdy2)
    r = obstacle[ob][2]                      ▷ Radius of Obstacle
    t1[0] = st[0]+((r+d1)/d1)*(sdx1-st[0])   ▷ Equation 2
    t2[0] = st[0]+((r+d2)/d2)*(sdx2-st[0])
    t1[1] = st[1]+((r+d1)/d1)*(sdy1-st[1])
    t2[1] = st[1]+((r+d2)/d2)*(sdy2-st[1])
    return $t1, t2$
start=waypoint[0]
end=waypoint[1]
count = 0
procedure MAIN()
    while $start \neq waypoint[n]$ do
        draw straight line between start and end
        closest obs_no,obs_count=collision(start,end)
        if closest obs_count == 0 then
            set straight line as final path
            start = end
            end = next way-point in the list
            count++
            restart loop
        t1,t2 = tangent(start,end,obs_no.)
        out of boundary1 = boundary(start,t1,cor)
        obs_no1,obs_count1=collision(start,t1)
        out of boundary2 = boundary(start,t2,cor)
        obs_no2,obs_count2=collision(start,t1)
        if out of boundary1 == 1 then
            if obs_count2 == 0 then
                Set start to t2 as final path
                start=t2
                restart loop
            else
                end = t2
                restart loop
```

Algorithm 2 Path planning and obstacle avoidance using tangent algorithm (contd.)

```
procedure MAIN_CONTINUED()
    while previous while loop continued do
        Similarly, check is second tangent goes out of flight-boundary.
        if obs_count1 ¿ obs_count2 then
            if obs_count2 == 0 then
                set start to t2 as final path
                start=t2
                restart loop
            end=t2
            restart loop
        else if obs_count2 ¿ obs_count1 then
            if obs_count1 == 0 then
                set start to t1 as final path
                start=t1
                restart loop
            end=t1
            restart loop
        else
            closest obs_no3,obs_count3=collision(t1,end)
            closest obs_no4,obs_count4=collision(t2,end)
            if obs_count3 ¿ obs_count4 then
                if obs_count2 == 0 then
                    set start to t2 as final path
                    start=t2
                    restart loop
                end=t2
                restart loop
            else if obs_count4 ¿ obs_count3 then
                if obs_count1 == 0 then
                    set start to t1 as final path
                    start=t1
                    restart loop
                end=t1
                restart loop
            else
                Calculate distance between
                t1 and end = d1
                t2 and end = d2
                if d1¿d2 then
                    if obs_count2 == 0 then
                        set start to t2 as final path
                        start=t2
                        restart loop
                    end=t2
                    restart loop
                else if d2¿d1 then
                    if obs_count1 == 0 then
                        set start to t1 as final path
                        start=t1
                        restart loop
                    end=t1
                    restart loop
                else
                    Choose any Tangent
```

Extension and gap is added by creating vectors and extending them in the "Add gap" and "Add Extension" section of the first part of the algorithm.

Here, s1 and s2 are the points of contact of the tangent with the circular obstacle, (x,y) is the center of the obstacle and st is the start way-point from which the tangents are drawn.

(sdx1,sdy1) and (sdx2,sdy2) are vectors along the center of the obstacle and the points of contact but, are slightly larger than the radius, and thus give us the points with gap. In Equation 1, the vector sdx1 starts at x and extends in the direction (s1[0]-x) that is, towards point of contact. 1.1 is multiplied to the direction vector to give extension in that direction. The number after the decimal place can be increased or decreased to increase or decrease the gap.

Similarly in Equation 2, a vector is created between the start way-point and the point of contact with gap obtained in equation 1. The extension factor, 1+(r/d1) is a function of radius of obstacle and distance between start way-point and point of contact with gap. This function always returns the best extension factor as the larger the circle, the larger the extension required to cross it and if the way-point is very close to the circle that is, d1 is very small then tangent drawn will be small and will thus require a larger extension.

VI. Performance Measures

If we impose the condition that more than three obstacles do not touch or intersect in an arc-like fashion as shown in Fig.5 then the following measures can be calculated.
The black circles represent the obstacles and the red dots are the way-points. The start way-point is on the left (inner side of arc).
Completeness: The algorithm is complete. Since the obstacles have gaps between them (assumption, because two buildings or two drones flying will always have a gap between them. A tangent can be drawn to pass any obstacle to reach the goal node.

1. The tangent undergoes a fixed extension based on a function which is inversely proportional to the distance between the way-point and the centre of the obstacle and directly proportional to radius of the obstacle. This is done to add some distance between the end of the tangent and the obstacle.

2. Also a processor can make rounding errors during calculation and draw the tangent a little inside the obstacle which might lead to mistakes in the obstacle detection function. So while drawing a tangent we maintain a small gap between the tangent line and the obstacle to which it was drawn.

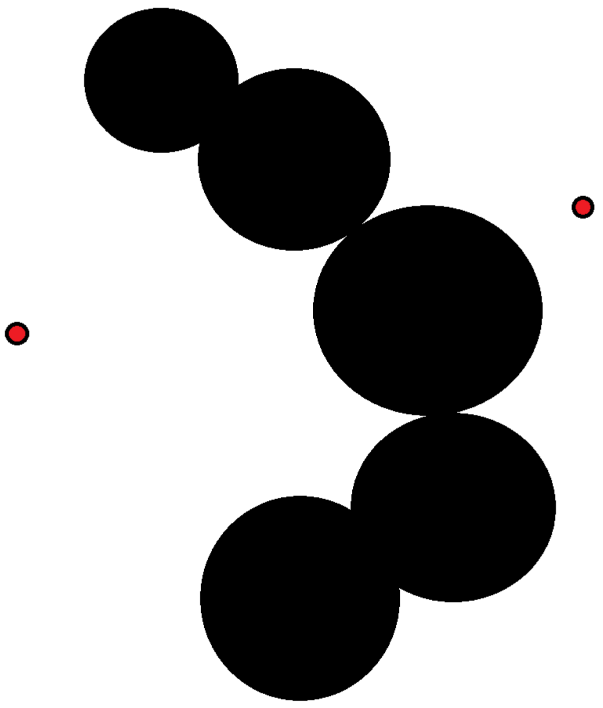


Figure. 5: **Case where first proposed system does not work.**

If the gap between two obstacles is too tight and small then functions 1 and 2 may cause problems in drawing tangents. Thus, the code will check if there is any collision in the extended length in [1] so that the extension can be reduced recursively until it fits. Also, the gap in [2] will be set to a very minute value.

Optimal-path: The algorithm picks the most optimal solution based on three performance measures:

1. Number of obstacles intersecting the tangent drawn from start point.
2. Number of obstacles intersecting the line between the end point of tangent and goal point that will be drawn using the tangent chosen in the previous step.
3. Distance between endpoint of tangent and goal point.

The measures are given in order of decreasing priority. Thus, decision is made not only based on the obstacle count of current tangent but also on the future consequences of choosing that tangent. The algorithm thus chooses the path with least obstacles. Thus, the algorithm may not give shortest path all the time. But, the priority of the three measures can be changed to generate the most suitable result. Therefore, sum of all three measures multiplied by their priority can be calculated for each tangent and decision can be made accordingly.

Time Complexity: The algorithm starts by drawing a straight line between the start way-point and the next way-point to be captured. If there are no obstacles detected between the two way-points then the straight line is chosen as the final path and we move to the next way-point. The time depends on the number of obstacles. To avoid checking each and every obstacle for an intersection we can create a rectangular area using the coordinates of every pair of consecutive way-points. The rectangle will be drawn by taking the two way-points as corners and while keeping the centre same, increasing all dimensions by twice the diameter of the largest obstacle. If an obstacle lies entirely or a part of it lies in the rectangle then it is added for consideration. This can be done during start up time that is when the unmanned vehicle is taking off. Thus, if there are no obstacles in the way no tangents need to be drawn. Only time required is for checking with each obstacle in the range if it intersects the straight-line path or not. Thus, if obstacles in range of the two way-points are ‘c’ in number then the time complexity will be O(c). The number of obstacles will be a constant because within the fixed rectangle there will only be space for few obstacles.

Considering the second case that is, when there are obstacles in the way. Consider that there are ‘x’ number of obstacles in the way. After a tangent is drawn to an obstacle, the end point of the obstacle is again considered as a start point and the algorithm loops again. Due to the extension and gap function most obstacles will be bypassed in a single tangent. Thus, for each tangent it will need to check for collision with all obstacles in range. Therefore, c*x. But there are 2 tangents for each obstacle. Therefore, 2*c*x. There is also the feature to check the obstacles in the future path that is between the end of each tangent and the goal way-point. Therefore, 4*c*x. Finally, one more check after no more obstacles are present. Therefore, the final complexity will be O(c*(4x+1)). Therefore, the best-case complexity is O(c) and the worst case could go up to O(c^2).

The assumption in the first part of this section was made because the first proposed system might get stuck in an infinite loop as shown in Fig.6.

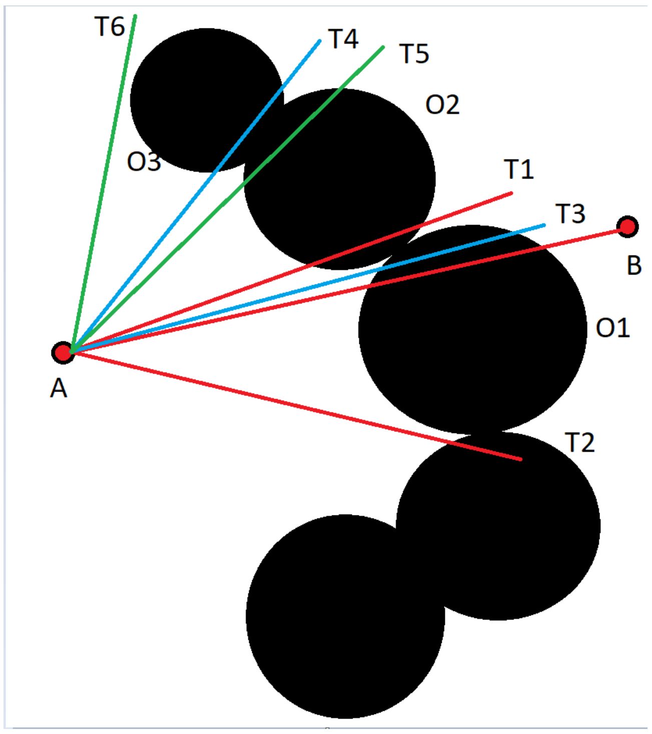


Figure. 6: **Algorithm gets stuck on obstacle 1 and 2.**

1. AB is drawn. Obstacles O1 blocks the path.
2. Draw tangents AT1 and AT2. Choose AT1 because T1 is closer to B.

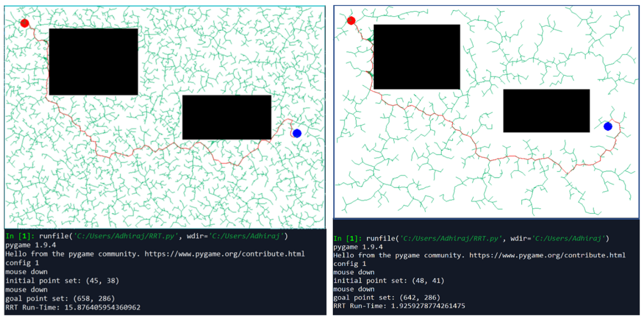
3. AT1 intersects O2. Draw tangents AT3 and AT4. Choose AT3 as T3 is closer to B. O1 is blocking AT3.

4. Again T1 and T2 are drawn and this loop repeats.

A possible solution for this problem is as follows:
An array that stores A,B, obstacles and the two tangent points for that obstacle(T1 and T2). If T1 is chosen set its flag to 1. This denotes that T1 has been chosen and if the algorithm comes back here when start and destination are same then we will choose the other tangent. If we come back to O1, reset T1 flag and set T2 flag. The algorithm will thus go from O1 to O2 and then back to O1 and then to the obstacle below and again back to O2 and then O3. AT5 will be chosen and the oscillation will continue until finally T6 will be chosen and a path can thus be created.
If we use threads (Proposed system 2) then each tangent is put into a new thread that is each tangent function will call two more functions for every obstacle in the path. Each new call can be put in a thread to process them in parallel. So, the problem of chain obstacles does not exist anymore.

The algorithm will be complete, since every possible tangent is expanded.Also, optimal path is obtained as every possible path is expanded. The total distance of each path can be calculated while the path is being created and these paths can then be compared to get the shortest one.The time complexity will be O(c^2) where 'c' is the number of obstacles in the region of the two obstacles.

Testing the Tangent algorithm against RRT:
We tested the pygame codes for Tangent algorithm and RRT in the IPython console and compared their run-time using the time package in python. Both algorithms were tested for two way-points and two obstacles.

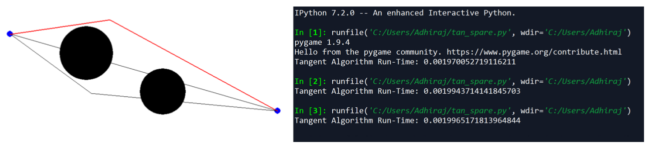


```
In [1]: runfile('C:/Users/Adhiraj/RRT.py', wdir='C:/Users/Adhiraj')
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
config 1
mouse down
initial point set: (45, 38)
mouse down
goal point set: (658, 286)
RRT Run-Time: 15.876405954360962
```

```
In [1]: runfile('C:/Users/Adhiraj/RRT.py', wdir='C:/Users/Adhiraj')
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
config 1
mouse down
initial point set: (48, 41)
mouse down
goal point set: (642, 286)
RRT Run-Time: 1.9259278774261475
```

Figure. 7: Run-times for RRT

RRT is a probability based algorithm. Sometimes, the algorithm gets lucky and a shorter tree is able to reach the next way-point faster. As such, the algorithm showed run-times ranging from 1.5 all the way up to 16 seconds when there were minor changes in the coordinates of the way-points.



```
IPython 7.2.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/Adhiraj/tan_spare.py', wdir='C:/Users/Adhiraj')
pygame 1.9.4
Hello from the pygame community. https://www.pygame.org/contribute.html
Tangent Algorithm Run-Time: 0.001970052719116211

In [2]: runfile('C:/Users/Adhiraj/tan_spare.py', wdir='C:/Users/Adhiraj')
Tangent Algorithm Run-Time: 0.0019943714141845703

In [3]: runfile('C:/Users/Adhiraj/tan_spare.py', wdir='C:/Users/Adhiraj')
Tangent Algorithm Run-Time: 0.001996517181396484 4
```

Figure. 8: Run-time for Tangent Algorithm

On the other hand, the Tangent Algorithm showed a constant time of around 0.0019 seconds or less even with major changes in the location of way-points or obstacles. This shows that the tangent algorithm is faster and more reliable. The constant run-times are due to the fact that the same tangent is most likely to be chosen if there are small changes in the coordinates of the way-points. All tangents are checked before one is chosen for the path, so even if a different tangent is chosen it does not effect the run-time.

The tangent algorithm will give even shorter run-times if there are no obstacles in the way because of its speculative approach of always trying the straight line path first. On the other hand, RRT still takes the same amount of time because it still generates its random tree and waits for a branch to reach the next way-point.

Similar comparison with algorithms like A* and other variations of A* and RRT like D*, IDA* and RRT* have been conducted. The Tangent algorithm displays exponentially faster times compared to all of these algorithms.

References

[1] E. Lalish and K. A. Morgansen, "Decentralized reactive collision avoidance for multivehicle systems," in Proceedings of the 47th IEEE Conference on Decision and Control, December 2008.

[2] L. Pallottino, V. G. Scordio, E. Frazzoli, and A. Bicchi, "Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems," in Proceedings of the 2006 IEEE International Conference on Robotics and Automation, May 2006, , pp. 2448-2453.

[3] Jiri Samek, D. Sislak, P. Volf, and M. Pechoucek, "Multi-party collision avoidance among unmanned aerial vehicles," Czech Technical University in Prague, Tech. Rep., 2007, In Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT '07). IEEE Computer Society, USA, pp. 403–406.

[4] J. Ruchti, R. Senkbeil, J. Carroll, J. Dickinson, J. Holt, and S. Biaz, "Uav collision avoidance using artificial potential fields," Auburn University, Tech. Rep. CSSE11-03, Aug. 2011, pp. 140-144.

[5] E. P. Dadios and O. A. M. Jr., "Cooperative mobile robots with obstacle and collision avoidance using fuzzy logic," in Proceedings of the 2002 IEEE International Symposium on Intelligent Control, October 2002, pp. 75-80.

[6] T. Crescenzi, A. Kaizer, and T. Young, "Collision avoidance in uavs using dynamic sparse a* ", Auburn University, Tech. Rep., 2011.

[7] D. Ferguson and A. Stentz, "Anytime rrts," in Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 2006, pp. 5369 - 5375.

[8] Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study by Mohammadreza Radmanesh, Manish Kumar, Paul H. Guentert, Mohammad Sarim, 2018, pp. 1-24.

[9] M. Radmanesh, M. Kumar, P. H. Guentert and M. Sarim, Grey Wolf optimization based sense and avoid algorithm in a Bayesian framework for multiple UAV path planning in an uncertain environment, Aerospace Science and Technology (2018), pp. 168–179.

[10] M. Radmanesh, P. H. Guentert, M. Kumar and K. Cohen, Analytical pde based trajectory planning for unmanned air vehicles in dynamic hostile environments, in American Control Conf. (ACC), 2017 (IEEE, 2017), pp. 4248–4253.

[11] C. M. Eaton, E. K. P. Chong and A. A. Maciejewski, Multiple-scenario unmanned aerial system control: A systems engineering approach and review of existing control methods, Aerospace (2016) .

[12] Optimal path planning for fixed-wing UAVs in 3D space, Nikhil Kumar Singha and Sikha Hotab, December, 2017, IIT Kharagpur, India.

[13] M. Sniedovich, Dynamic Programming: Foundations and Principles (CRC Press, Boca Raton, 2010).

[14] X. Tian, Y. Bar-Shalom and K. R. Pattipati, Multistep look-ahead policy for autonomous cooperative surveillance by UAVs in hostile environments, in Decision and Control, 2008. CDC 2008. 47th IEEE Conf. (IEEE, 2008), pp. 2438-2443.

[15] Y. Bj€ornsson, M. Enzenberger, R. C. Holte and J. Schaeffer, Fringe search: Beating A* at pathfinding on game maps, CIG, 5 (2005)

[16] J. Kober, J. A. Bagnell and J. Peters, Reinforcement learning in robotics: A survey, The International Journal of Robotics Research 32(11) (2013), pp. 1238-1274.

[17] A. Chaudhry, K. Misovec and R. D'Andrea, Low observability path planning for an unmanned air vehicle using mixed integer linear programming, in 43rd IEEE Conf. Decision and Control, 2004. CDC. Vol. 4 (IEEE, 2004), pp. 3823 - 3829.

[18] O. K. Sahingoz, Flyable path planning for a multi-uav system with genetic algorithms and bezier curves, in Unmanned Aircraft Systems (ICUAS), 2013 Int. Conf. (IEEE, 2013), pp. 41–48.

[19] Patrik, A., Utama, G., Gunawan, A.A.S. et al. GNSS-based navigation systems of autonomous drone for delivering items. J Big Data 6, 53 (2019), pp. 5-10

[20] Wyder PM, Chen Y-S, Lasrado AJ, Pelles RJ, Kwiatkowski R, Comas EOA, et al. (2019) Autonomous drone hunter operating by deep learning and all-onboard computations in GPS-denied environments.

[21] Shafkat Islam, Abolfazl Razi. (2019). A Path Planning Algorithm for Collective Monitoring Using Autonomous Drones. pp. 1-6.

Author Biographies

Adhiraj Shetty is a student pursuing his undergraduate degree in Computer Science and Engineering at Vellore Institute of Technology, Vellore in India, graduating in the year 2021. His research interest is in the field of Artificial Intelligence.

Annapurna Jonnalagadda is an Associate Professor at Vellore Institute of Technology, Vellore in India. Her research interests are Artificial Intelligence, Game theory and Social Network Analysis.

Aswani Kumar Cherukuri is a Professor at the School of Information Technology and Engineering, Vellore Institute of Technology, Vellore in India. His research interests are Information Security, Machine learning.