Real-time Collision Avoidance and Path Optimizer for Semi-autonomous UAVs.

A.F Hawary[a], N.A Razak[b]

School of Aerospace Engineering, Engineering Campus, Universiti Sains Malaysia, 14300, Nibong Tebal, Penang, Malaysia

Corresponding author: [a]aefaizul@usm.my

Abstract. Whilst UAV offers a potentially cheaper and more localized observation platform than current satellite or land-based approaches, it requires an advance path planner to reveal its true potential, particularly in real-time missions. Manual control by human will have limited line-of-sights and prone to errors due to careless and fatigue. A good alternative solution is to equip the UAV with semi-autonomous capabilities that able to navigate via a pre-planned route in real-time fashion. In this paper, we propose an easy-and-practical path optimizer based on the classical Travelling Salesman Problem and adopts a brute force search method to re-optimize the route in the event of collisions using range finder sensor. The former utilizes a Simple Genetic Algorithm and the latter uses Nearest Neighbour algorithm. Both algorithms are combined to optimize the route and avoid collision at once. Although many researchers proposed various path planning algorithms, we find that it is difficult to integrate on a basic UAV model and often lacks of real-time collision detection optimizer. Therefore, we explore a practical benefit from this approach using on-board Arduino and Ardupilot controllers by manually emulating the motion of an actual UAV model prior to test on the flying site. The result showed that the range finder sensor provides a real-time data to the algorithm to find a collision-free path and eventually optimized the route successfully.

1. Introduction

The path planner is considered a key element of the unmanned aerial vehicle (UAV) navigation system. Basically, its task is to compute the optimal path from a start point to an end point. Unlike the trajectories for commercial airlines, UAVs trajectories are constantly changing depending on the terrain and conditions prevailing at the time of their flight. A simple trajectory optimization for typical UAVs is to find a shortest path using search algorithms as in [1], though other objectives, e.g., distance travelled, the average altitude, the fuel consumptions are equally important but not covered within the scope of this work. As an example, the authors of [2] proposed the use of a genetic algorithm (GA) to improve the TSP exploration and better avoid local minima when searching for an optimal path. The authors of [3] also use a SGA, but the collision is not included.

In this paper, we study the effectiveness and the practicality of UAV path planner and collision detection using an on-board controller for fixed-wing UAVs. Essentially, we combine brute force and non-deterministic algorithms to develop an operational path planner modified from [4], and the avoidance based on [5] but we use a laser range finder instead. We present two important contributions. First, we propose a simple integration between Ardupilot (APM) and Arduino Mega, which mutually work to optimize the route as well as avoiding obstacles. This allows us to use a generic optimization algorithm (without major modification) as the search algorithm. In our case, we use the Nearest Neighbour (NN) and the Simple Genetic Algorithm (SGA), but these could be replaced by other algorithms. Second, we present a technique to combine both the SGA and the NN while minimizing the communication between the processes in order to achieve a near real-time performance. Both algorithms have been widely used for robot path planning. However, to our knowledge, there exists no rigorous comparison between the two algorithms when applied to this particular problem. The results we present in this paper provide a clear insight how to develop a simple algorithm for UAV path planning in real complex environments. The remainder of this paper is organized into sections. Section 2 provides the details of the UAV specification. We present in Section 3 the methodology to combine and integrate the algorithms. And finally we include our result and conclusion in Sections 4 and 5, respectively.

2. UAV Specification

This study utilizes a fixed-wing model aircraft (model X-UAV Sky surfer X8) with 1.4 m wing span, 0.95 m length and weighs about 0.6 kg. The body is made of Expanded PolyOlefin (EPO) foam and powered by a brush-less motor. The flap mechanisms are activated by four servos using an Ardupilot (APM) as an on-board flight controller. The built-in proportional-integral-derivative (PID) controller uses various input parameters (e.g. airspeeds, roll rate, yaw rate, etc.) through a 6 degree-of-freedoms (6-DOF) inertial measurement unit (IMU) to achieve in-flight stability. A Global Positioning System (GPS) unit is also installed to provide its position (latitude, longitude) during navigation. In addition, a separate Arduino Mega is installed as a path-planner and collision module where the algorithm proposes in this paper executes. A picture of the UAV is shown in Figure 1.

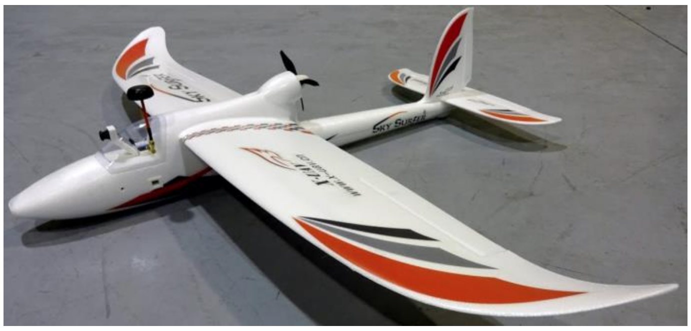

Figure 1. X-UAV Sky surfer X8 model aircraft.

This UAV features a typical 4-channel setup with a single propeller where its stability depends on the deflection angle of the control surfaces such as aileron, rudder, and elevator as shown in Figure 2 below.

3. Methodology

It is worth mentioning here that, the UAV can fly remotely with good stability using APM controller even without a path planner. Although APM has its own mission planner, it does not have the optimizing and re-routing capabilities. Hence, we propose a separate module to handle the path-planning and collision avoidance algorithm to speed up the computation. In general, the UAV route is made from a series of waypoints called target points (TPs). We define that TP is a set of points where the UAV has to visit or reach in order to cover certain areas. TPs can be a random or specified waypoints defined by users within the area of interest. Then, the path optimizer would compute the best route that covers all TPs in a way that it minimizes the distance and the number of crossing paths. In real-time UAV mission, the pre-planned path is considered valid, unless there is a potential collision that requires real time re-routing. In order to do this, the optimizer and the obstacle avoidance module has to work together to optimize a return journey that begins from a specific point. Therefore, we relate this problem to a classical Travelling Salesman Problem (TSP) [6] with a slightly modified algorithm to embed collision avoiding capabilities.

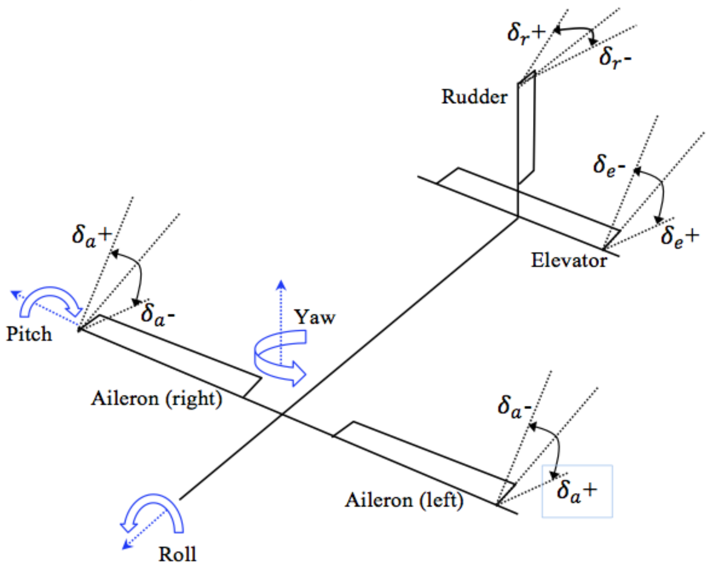

Figure 2. Control surface specification ($\delta_a = \pm 40°, \delta_e = \pm 30°, \delta_r = \pm 30°$).

3.1. Travelling Salesman Problem

TSP simply rules the salesman to plan a return journey through all cities so that it makes the most profit without visiting any cities twice. The profit can be any functions that contribute to his profit, e.g., distance, flight time. From graph theory, TSP can be described as follows:

TSP = *(G, f, t)*

where, $G = (V, E)$ is a complete graph, f is a function $V \times V \rightarrow Z$, and $t \in Z$, G is a graph that contains a travelling salesman tour with cost that does not exceed t in case t is restricted.

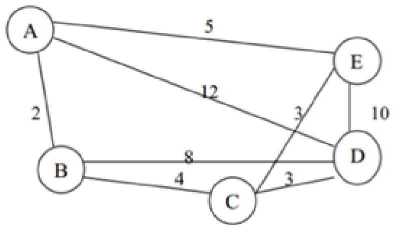

Figure 3. A simple TSP routing problem.

Figure 3 illustrates how a shortest return path from point A is obtained using TSP. Let us route Path1 from A, B, C, D, E and A and route Path2 from A, B, C, E, D and A. Computing the length of Path1 results in 24 units and Path2 has 31 units respectively. Obviously, Path2 is favourable due to shorter in length. Solving this kind of problems over hundreds of points are complex; therefore, a systematic algorithm is needed. Mathematically, TSP generalizes the question of Hamiltonian circuit in a graph as shown in Eq.(1).

$$F = min\sum_{i=1}^{n-1} dist\left(x_i, x_{i+1}\right) + dist\left(x_{n-1}, x_1\right) \tag{1}$$

Where, x is the city and i =1,2,3,…n represents the number of cities. There are numerous methods available to solve TSP. Among those methods, we favour a brute force method due to its simplicity and fast i.e., Nearest Neighbour. In addition, we combine the brute force with a meta-heuristic evolutionary algorithm SGA proposed in [7] by adopting Lin-Kernighan tour.

3.2. Routing Algorithm

SGA belongs to the larger class of evolutionary algorithms (EAs), which generate solutions using techniques inspired by the evolution theory. Its heuristic search mimics the process of natural evolution through selection, crossover and mutation within the *gene* in the chromosome. This method provides a wider search spectrum to locate the global solutions in optimization problem. Due to its nondeterministic nature, SGA lacks of real-time capability and time consuming. This is critical in the operation that requires instant decisions such as disturbance, collision or lost tracked from its original route. Moreover, the delay could prolong the UAV in an 'unguided state' before the algorithm produces a new solution. Therefore, we explore a simple brute force method using NN to take over the 'unguided state' and only kick in when the path collides with obstacles. As opposed to SGA, NN is rather simple in both constructions and operations where it always favours the nearest point. Thus, the decision can compute almost instantly and simple to process on any on-board controllers.

3.3. Collision Avoidance

Collision avoidance measures time of flight between the UAV and the obstacles to obtain the distance if the beams collide with an object. Using the speed of light as a constant, the laser rangefinder (LIDAR) sensor calculates the duration between the time a pulse is sent and the time it is received and then determines the distance to the object. In this project, a tiny rangefinder Lite 3 from Roboshop.com is used as it provides reliable and extremely precise measurements with great repeatable accuracy and short response time up to 500 measurements per second and can measure up to 40 m measuring range. Moreover, the communication to the module is simplified via I2C protocol and highly compatible with Arduino Mega. Basically, the distance, D between the two points is given by Eq.(2);

$$D = \frac{ct}{2} \tag{2}$$

where c is the speed of light $(3 \times 10^8\ m/s)$ in the atmosphere and t is the amount of time for the rount-trip between two points given by Eq.(3).

$$t = \frac{\phi}{\omega} \tag{3}$$

where ϕ is the phase delay made by the light travelling and ω is the angular frequency of the optical wave.
Substituting Eq.(2) into Eq.(3) yields,

$$D = \frac{1}{2}ct = \frac{1}{2}c\frac{\phi}{\omega} = \frac{c}{4\pi f}(N\pi + \Delta\phi) = \frac{\lambda}{4}(N + \Delta N) \tag{4}$$

where, λ is the wave length of $\frac{c}{f}$, $\Delta\phi$ is the part of phase delay that does not fulfil π, N is the integer number of wave half-cycles of the round trip and ΔN the remaining fraction part.
The collision mechanisms built on a single laser rangefinder which is pointing to five different directions by rotating (motorized) the sensors as shown in Figure 4 below.
The pointing directions of the LIDAR are prefixed with an *RF* for rangefinder followed a specific direction, i.e., head, up, down, left and right. The state of each direction of the rangefinders is used to control the deflection of rolling angles (Aileron), yawing angles (rudder) and pitching angles (elevator). This setup provides narrow angle collision detection within 20 m at nominal glider cruising speed of 10 m/s. That means; the controller has two

seconds to compute a new avoiding route before collision with approximately 1000 readings. Depending on the state of the sensors, a decision is based on a look-up table to command a necessary control strategy to the APM for every condition as shown in Table 1.

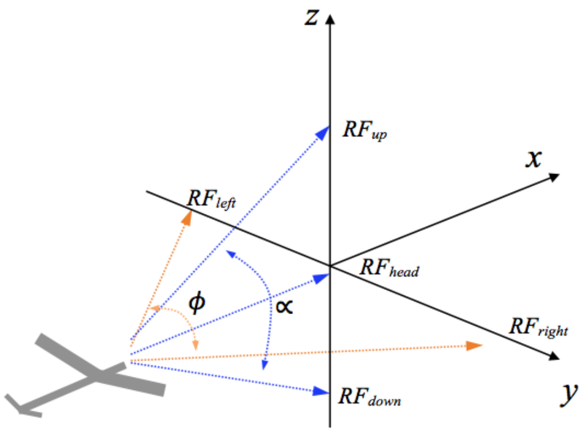

Figure 4. LIDAR setup based on motorized pointing direction.

Table 1. A look-up decision table for collision avoidance.

Case	Rangefinder Status (1 = block, 0 = clear)					Decision	Surface control			
	RF_{left}	RF_{right}	RF_{up}	RF_{down}	RF_{head}		Aileron (δ_a)	Rudder (δ_r)	Elevator(δ_e)	Throttle (%)
1	0	0	0	0	0	Proceed	0	0	0	0
2	1	1	1	0	0	Proceed	0	0	0	0
3	0	1	1	1	1	Left turn	-20	0	-10	+5
4	0	0	1	1	1	Left turn	-20	0	-10	+5
5	0	0	0	1	1	Left turn	-20	0	-10	+5
6	0	0	0	0	1	Left turn	-20	0	-10	+5
7	1	0	1	1	1	Right turn	+20	0	-10	+5
8	1	0	0	1	1	Right turn	+20	0	-10	+5
9	1	0	1	0	1	Right turn	+20	0	-10	+5
10	1	1	1	1	1	Sharp Climb	0	0	-30	+20
11	1	1	1	0	1	Sharp Climb	0	0	-30	+20
12	1	1	0	1	1	Climb	0	0	-15	+10
13	1	1	0	0	1	Climb	0	0	-15	+10

The signals of the control surfaces are operating based on pulsing and monitor. The pulse's interval is between 1 to 2 milliseconds. Note that, regardless of the state of other sensors, for as long as the RF_{head} is 0 or unblocked, the decision is remained. Otherwise, the decision depends on the positioning of the obstacles detected by the other sensors. As for the case of 10-13, the UAV has to climb up or sharply climb, but it remains on the same original route with elevated trajectories. For the case of 3, 7, 8 and 9, the decision is chosen based on the direction which is clear from obstacles. However, for the case of 4, 5 and 6 where both left and right sensors are clear the decision, whether to take left turn or right turn are not defined. For this case, a simple rule is applied considering the distance of the next point to the original point on the route which is taken care by NN algorithm and then re-route by SGA. In order to do this, NN will search and calculate the nearest point during collision and assign a set of temporary

TPs to navigate to a new TP. While NN diverting the route through TP, S will take its current position and calculate for a new optimal route and then terminate NN routine. This process repeat if the UAV faces the obstacles.

3.4. Algorithm Configuration

Within SGA, the selection of the parents is based on Roulette Wheel Selection (RWS) and a single-point crossover and flip-bit mutation. We set the probability of crossover P_c and P_m to be 0.8 and 0.01 respectively. The population size is set to 10 and max generation to 10. As for the NN, there is no additional setting other than computing the nearest trajectory point. The combined algorithm for both methods is shown in the algorithm listing below.

```
//Genetic Algorithm main routine
Initialization : Assign SGA parameters, such as $p_c$, $p_m$, popsize, max_generation, home, etc.
Generate chromosome of [popsize],
   Assign [nearest point] as a starting point
  Permutate randomly
  Evaluate their fitness values.
  gen = 0.
Main()
While generation < max_generation.
     Select individuals [popsize] from current gen using RWS to generate the mating pool.
     While population < popsize
          Select two individuals from the mating pool randomly
          Perform single-point crossover with probability $p_c$,
          Perform bit-flip mutation for every gene of the offspring with probability $p_m$.
          Then insert the mutant into the new population.
          Evaluate the fitness value for every new individual in the new population.
          Replace the current population with the new population. Generation = generation+1.
Submit the fittest individual as the results of GA to APM controller to execute.
End
//Nearest Neighbour Interrupt routine
Read sensor states
  Refer decision look-up table
  Determine the nearest point
  Assign temporary target point
  Send command to APM to navigate to a temporary target point
  Assign [nearest point] to GA as the next point
Return
```

Ideally, the original solution produced by SGA algorithm alone would continue valid for as long as the route does not interfere with any obstacles.

4. Simulations

A simple functional test performed to compare the solution of three different algorithms namely Random, NN and SGA (based on Lin-Kernighan Tour) and is comparable with Concorde TSP Solver. The test involves 100 random points within a defined space. The simulation result for every algorithm as shown in Figure 5 below.

All algorithms have been run using 100 random points shown in Figure 5(a). Obviously, the solution from Random algorithm is not appropriate for flight path planning (too many crossing paths). A slightly better solution produced by NN algorithm is shown in Figure 5(c) with several zigzag paths. As discussed earlier, NN is good in determining the nearest point quickly without having to rely on expensive processing time and yet the route is acceptable. Interestingly, the solution in Figure 5(d) has shown a smooth path without any path-crossing, and it is solved using SGA with slightly longer computing time with significant improvement on the length. So, SGA could produce a smooth-and-shorter path, but it takes slightly long as compare to the NN. Therefore, we take advantage of the instantaneous decision by NN and the solution optimality from SGA to provide the best from both worlds to embed a simple-and-practical algorithm for UAV.

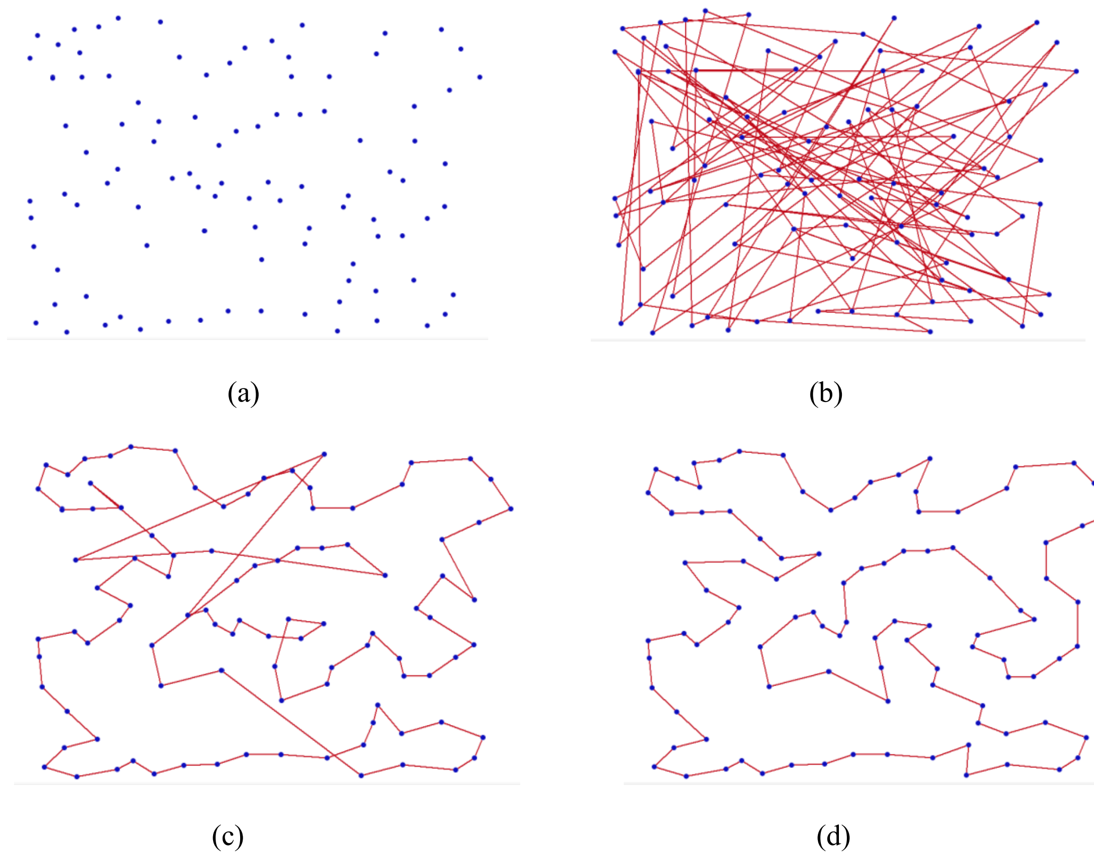

Figure 5. (a) 100 random points, (b) A solution using Random algorithm (t = 0.57s , l =5283), (c) A solution using Nearest Neighbor (t = 0.75s, l =962), (d) A solution using Genetic Algorithm (t = 0.91s, l =767).

5. Result and Discussion

The algorithm was tested with a simple obstacle starting from a point called Home. If no obstacle detected along the route, it would follow the pre-planned path which begins from 'Home' all the way to 1, 2, 3…11 and returns to 'Home' as shown in Figure 6(a). Ideally, the UAV would follow the pre-planned route from the beginning till the end. However, if the pre-planned route intersects with an obstacle on the heading direction (RF_{head}=1), the flight path is diverted from Home to point 11 through two temporary points derived from NN algorithm that seeks a nearest point and eventually diverts its journey to point 11 instead and continue to 10, 9…2, 1 and return Home (Figure 6(b)). This is because at the point of collision, NN sees that TP 11 is nearer that TP 1 hence the decision is to turn right instead. It is observed that while NN is calculating a nearest point, SGA runs its permutation and rearranges the way point from 1, 2, 3…10 and 11 previously to a new way point beginning from 11, 10, 10…2, 1 and returns Home. Since NN algorithm runs in the interrupt mode, it provides a sufficient time for the on-board processor to compute S hence reduces the 'unguided state' delay.

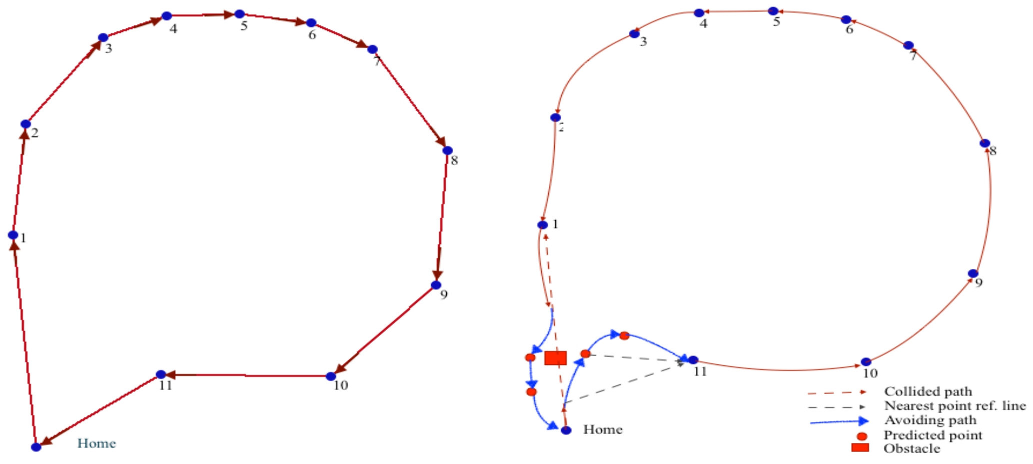

Figure 6. (a) Pre-planned route without obstacles. (b) Simulated real-time route with an obstacle.

Figure 6 shows the combined approaches used here could provide a simple real-time path planning and obstacle avoidance solution for the UAV researchers and hobbies out there. Since the scope of this work is to simulate the behavior of the UAV along the route, we only show a typical motion by moving the UAV manually. However, the position tracking data is not presented al it can be easily extracted from a GPS module. Moreover, due to the lack of the actual and real-time flight testing, the detail response of the control surfaces is not presented here. We reserve the detail observation of the flight performance during the real-time performance test.

6. Conclusion

In conclusion, we observe that the result from this study and the initial experimental provide a practical solution to embed the complete algorithm into the Arduino Mega and APM. Not only, it provides a simple solution; it is also affordable to be implemented especially for the beginner with minimal efforts. Hence, for the scope of this paper, we consider the objective of this study is achieved.

References

[1] X. Z. Gao, Z. X. Hou, X. F. Zhu, J. T. Zhang, and X. Q. Chen, “The shortest path planning for manoeuvres of UAV,” *Acta Polytech. Hungarica*, vol. 10, no. 1, pp. 221–239, 2013.

[2] Y. Yu, Y. Chen, and T. Li, “A New Design of Genetic Algorithm for Solving TSP,” in *Computational Sciences and Optimization (CSO), 2011 Fourth International Joint Conference on*, 2011, pp. 309–313.

[3] J. Y. Potvin, “Genetic algorithms for the traveling salesman problem,” *Ann. Oper. Res.*, vol. 63, no. 3, pp. 337–370, 1996.

[4] J. Tisdale, Z. K. Z. Kim, and J. Hedrick, “Autonomous UAV path planning and estimation,” *IEEE Robot. Autom. Mag.*, vol. 16, no. 2, pp. 35–42, 2009.

[5] N. Gageik, P. Benz, and S. Montenegro, “Obstacle detection and collision avoidance for a UAV with complementary low-cost sensors,” *IEEE Access*, vol. 3, pp. 599–609, 2015.

[6] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, “The Traveling Salesman Problem: A Computational Study,” *Princet. Univ. Press*, p. 593, 2006.

[7] Y. Deng, Y. Liu, and D. Zhou, “An Improved Genetic Algorithm with Initial Population Strategy for Symmetric TSP,” *Math. Probl. Eng.*, vol. 2015, 2015.