Micro Aerial Vehicle Path Planning and Flight with a Multi-objective Genetic Algorithm

H. David Mathias and Vincent R. Ragusa
Department of Mathematics & Computer Science
Florida Southern College
Lakeland, Florida 33801
Email: hmathias@flsouthern.edu
Email: vragusa@mocs.flsouthern.edu

***Abstract*—Due to its importance for robotics applications, robotic path planning has been extensively studied. Because optimal solutions can be computationally expensive, the need for good approximate solutions to such problems has led to the use of many techniques, including genetic algorithms. This paper proposes a genetic algorithm for offline path planning in a static but very general, continuous real-world environment that includes intermediate targets in addition to the final destination. The algorithm presented is distinct from others in several ways. First, it does not use crossover as this operator does not appear, in testing, to aid in efficiently finding a solution for most of the problem instances considered. Second, it uses mass extinction due to experimental evidence demonstrating its potential effectiveness for the path planning problem. Finally, the algorithm was designed for, and has been tested on, a physical micro aerial vehicle. It runs on a single-board computer mounted on the MAV, making the vehicle fully autonomous and demonstrating the viability of such a system in practice.**

***Keywords*—Multi-objective genetic algorithms; Evolutionary computation; Robotic path planning; Autonomous flight; Continuous environment; Pareto optimization; NSGA-II**

I. Introduction

Micro Aerial Vehicles (MAVs) have generated significant interest among computer science researchers. While this is no doubt due, in part, to the appeal of studying and controlling flying vehicles, the primary motivation is likely the broad array of interesting problems touching on a range of areas. Interesting research problems include cooperative behaviors and swarming, localization and mapping, learning, autonomous operation, path planning, and various problems that involve computer vision, such as tracking, inspection in inhospitable or remote environments, and search and rescue.

None of the problems listed are particular to MAVs. Some, such as path planning, have a long history and multiple applications in robotics and other areas. In robotic path planning, a robot must plan a path from a starting location to an ending location in the presence of obstacles. If the environment is known *a priori*, the planning is said to be *offline*. The environment is sometimes bounded, modeling the limited range and/or battery life of the robot. In addition to reaching the target location, the goal is to optimize some quantity, or objective, most often path length. In some instances of the problem, it may be desirable to attempt to optimize multiple objectives. Other possible objectives include minimizing the number of waypoints, minimizing the sum of the angles of the turns required, or maximizing the number of intermediate targets reached. Not surprisingly, multiobjective optimization significantly complicates the search for a solution, particularly when objectives conflict. As a simple example, consider a path consisting of a straight line from source to target that hits numerous obstacles. Clearly this is optimal with respect to path length but is suboptimal for obstacle collisions. Decreasing the number of obstacles hit necessarily increases path length.

A number of different techniques have been applied to path planning problems. These include heuristic search (such as A*) [11], bug algorithms [5], cell decomposition [7], potential fields [4], neural networks [12], and evolutionary algorithms [13]. This work uses a genetic algorithm because of the numerous operators and wide array of tunable parameters that allow attempts to customize and experiment with the approach to the problem.

An important distinction for the problem considered here is that the search space is continuous, whereas most results in this area are for discrete environments. The continuous space can be discretized, resulting in a graph, however, this imposes artificial constraints on the environment in which the system operates and results in a very large problem instance. Consider a modest environment of 500m x 500m. With a granularity of 1m, this results in a graph with 250000 vertices and roughly 1000000 edges, assuming that only rectilinear and diagonal edges are created. Finer granularity or a larger environment dramatically increase the size of the graph.

The algorithm presented runs immediately before flight on a single-board computer onboard a small micro aerial vehicle. In planned, adaptive versions of the system, it will also run during flight, perhaps repeatedly. Therefore, due to the comparatively limited power of the onboard computer, efficiency is an important practical consideration.

Genetic algorithms are algorithmic corollaries to Darwinian evolution. A population of candidate solutions, intially random, evolves through some combination of selection, mating and mutation operators. The chromosome – the genetic material being evolved – differs greatly with the problem being solved. For path planning, intermediate destinations, referred to as *waypoints*, can be used as genes, with the sequence of waypoints being visited constituting the chromosome. *Crossover*, in which genes from each of two parents are combined to produce a child, simulates mating while changes to a waypoint take place at random as a form of *mutation*.

978-1-5090-1121-6/16/$31.00

Each generation of the population creates a new generation of children via crossover and/or mutation. A fixed population size, n, is maintained via *selection*, a form of survival of the fittest. This may be as simple as completely replacing one generation with a new one or may involve a process that is considerably more complex.

Having a population of candidate solutions necessitates an ability to rank population members to facilitate selection and, ultimately, identification of the best member. When attempting to optimize a single objective, this task consists of simple scalar comparisons of that objective value. Working with multiple objectives requires a more sophisticated approach. A commonly used method relies on Pareto optimization, a system for resource allocation devised by economist Vilfredo Pareto that has been adapted for use in engineering disciplines. Multi-objective optimization is discussed in more detail in Section III.

This paper details a genetic algorithm for multi-objective offline path planning for a micro aerial vehicle. Nearly all results in this area limit implementations to a discrete environment and run only in a simulator. This system was implemented for and tested on a micro aerial vehicle in a highly unconstrained physical and, therefore, continuous, environment that also includes intermediate target points. The algorithm runs on a single-board computer onboard the vehicle allowing completely autonomous flight. Further, the algorithm presented uses a mass extinction operator to periodically thin the gene pool. The literature does not appear to include another path planning genetic algorithm that uses extinction. The remainder of the paper outlines the problem addressed and describes the algorithm and the vehicle. Later sections discuss results and conclusions as well as directions for future research.

II. Related Work

Evolutionary methods have been applied to robotic path planning for more than two decades [27] [22]. Many results have extended the early work in various ways. Among these are Hermanu *et al.* [15], Sedighi *et al.* [28], and Ahmed and Deb [3] who explore chromosome representations specific to path planning. Jun and Qingbao [18] examine heuristic population seeding.

Unusual in the literature, Burchardt and Salomon [6] implement their algorithm on physical robots. The purpose of their system is to enable robots to play soccer; the ultimate goal to enable a robotic team to compete against human teams by 2050. Zheng, Ding, Zhou and Li [32] created a system for planning paths for multiple, cooperating aerial vehicles. However, their impressive algorithm runs only in a simulator. Siddiqi, Shriraishi and Sait [29] have addressed the need for efficiency in algorithms, genetic or otherwise, for path planning in embedded systems, an important consideration given that embeddable computers cannot compete with larger computers with respect to memory and computing power.

Hasircioglu *et al.* [14] consider a problem similar to that examined in this work: offline path planning in a continuous environment. While there are similarities in the genetic encoding used, there are also important differences between this work and theirs. Their system operates in a 3D environment but is run only on a simulator. For candidate evaluation, they use a scalar objective function. Additionally, there are differences in the genetic operators employed.

In 1997, Xiao, Michalewicz, Zhang and Trojanowski [30] outlined an adaptive approach to path finding using a genetic algorithm. Their two-stage approach, consisting of finding an initial, possibly imperfect, path which is refined during the mission, is likely more relevant today than when originally published due to the small size and light weight of hardware now available. Multi-stage, adaptive navigation is a topic for continued work as an extension of the current system.

The work on multi-objective optimization in genetic algorithms by Deb, Pratap, Agarwal and Meyarivan [8] is among the most seminal in the field. Their algorithm, known as NSGA-II, and in particular, their method of non-dominated sorting to create a partial order on the population and help ensure genetic diversity, informed design decisions made in the work presented here. A later result, NSGA-III, more effectively handles larger numbers of objectives [9] [16].

Applying the NSGA-II framework to the problem of path planning, Ahmed and Deb [3] examine the impact of chromosome representation. While they experiment with a binary encoded chromosome, their best results are achieved using integer valued chromosomes. They also explore the effects of changing crossover and mutation rates on the results obtained.

Recently, Mathias and Ragusa [23] undertook a study of the utility of the very commonly used crossover operator, and the much less commonly used mass extinction operator, on genetic path planning. They showed that mass extinction has a favorable impact in some cases, sometimes significantly so. Crossover, on the other hand, was largely negative and, therefore, is not implemented in the algorithm presented here.

III. The Genetic Algorithm

The environment in which the algorithm operates is continuous, a departure from most results in the literature. In fact, the algorithm operates in a representation of a physical environment with few constraints. The starting point and ending point can exist anywhere in the environment and the combination of obstacles and goal points can force the vehicle to fly in any direction. Additionally, problem instances can require circuits by co-locating the start and end points and introducing intermediate goals. This is a considerably more general environment than that in most previous results. Thus far, only two-dimensional space is used. Though the MAV operates in 3D, 2D space is easily modeled by flying missions at a fixed altitude.

An instance of the path planning problem considered here consists of a starting location, s; a destination location, d; a (possibly empty) list of obstacles, *obs*; and a (possibly empty) list of intermediate targets, *goals*. s and d are represented by GPS coordinates. Obstacles are circular or polygonal. If an obstacle is circular, it is represented by the GPS coordinates of it's center and a radius in meters. If polygonal, it is represented by a list of vertices, each of which is a GPS location. Intermediate targets do not appear in path planning results in the genetic algorithms literature. They are locations to which the vehicle must fly, prior to the destination, to successfully complete it's mission. These may represent locations near RF-enabled sensors from which the vehicle must take a reading or

locations from which a photograph must be taken, for example. Each element of *goals* is a circle represented in the same format as a circular obstacle. Several of the inputs on which the algorithm was tested are shown in Figure 2.

A solution to the problem consists of a sequence, $S = \langle w_0, \ldots, w_n \rangle$ of waypoints, where $w_0 = s$, $w_n = d$. Each waypoint w_i is represented as an ordered pair, (lat, lon), consisting of the latitude and longitude of the location. A solution is considered *feasible* if the path defined by S avoids all obstacles in *obs* and hits all targets in *goals*.

A. Chromosome Representation

There are two aspects to chromosome representation: the information being stored in the genetic material and the encoding used to store it. The information encoded in the chromosome is highly dependent on the environment and the specifics of the problem variant. In highly discretized environments, the genes often consist of row and column numbers and, perhaps, a direction of travel indicator [15] [28]. In some cases, positions are indicated as offsets rather than as absolute coordinates [3]. Continuous environments do not appear frequently in the literature. Exceptions are the work of Zheng *et al.* [32] and Hasircioglu *et al.* [14], in which the chromosomes consist of points in three dimensional space. In the former result, the chromosome also contains validity indicators for each point.

In many problem domains, the chromosome is encoded as a binary string. This allows a variety of crossover options as well as mutations as fine as flipping a single bit. Such an encoding is not, however, well-suited to some path planning problems. Consider a continuous environment with geo-fencing. Flipping an arbitrary bit in the binary encoding of one coordinate of a point encoded in the chromosome may result in a point that is well outside the environment (or, for that matter, the hemisphere). To avoid such problems, a floating point encoding is useful.

The chromosome used in this work was designed to accommodate the real-world constraints imposed by the physical system. The algorithm operates on an actual vehicle, with limited flight time, in a continuous environment. Thus, the chromosome is a real-valued vector consisting of waypoint sequence S as described above. Operators are applied to the latitude and longitude values of these waypoints, allowing some degree of control over the resulting changes. For example, the magnitude of a move mutation is limited to reflect physical reality.

B. Multi-objective Optimization

The optimization objectives used in this work are: path length, number of obstacles hit, number of goals hit and smoothness. Smoothness, defined as $\sum_{i=0}^{n-1} \angle w_i w_{i+1}$, is the sum of the angles between pairs of consecutive waypoints. A smoother path is desirable since sharp turns require more time and, possibly, more maneuvering for certain types of vehicles. The goal of the algorithm is to minimize all objectives. To achieve this for the number of goals hit, for which higher values are preferred, the algorithm negates that quantity so that it too can be minimized. Number of waypoints was considered as another objective to be minimized, however, it provided no benefit, likely subsumed by other objectives. Though not an objective in the genetic algorithm, number of waypoints is tracked and used as a tie-breaker when ranking population members.

As is discussed in Section I, while use of multiple objectives makes it possible to better guide the search for a solution, it complicates the process of doing so. One approach to multiple objectives is to define a function $f : [o_1, \ldots, o_k] \rightarrow \mathbb{R}$. In other words, f is a polynomial over the objective variables, mapping them to a real valued result. This has the advantage of making it easy to compare members of the population since each is represented by a single real value. However, finding effective values for the coefficients (and possibly exponents) in f is challenging.

Konak, Coit, and Smith [19] provide a tutorial on popular approaches to multiobjective optimization including a pair of well-known algorithms: NSGA-II [8], and MOEA/D [31]. In MOEA/D, the multiobjective problem is decomposed into a set of scalar optimization problems that are solved individually but simultaneously. In NSGA-II, the objectives are not separated but rather are dealt with in the aggregate. Both algorithms leverage the concept of *Pareto optimization*. In Pareto optimization, as used in genetic algorithms, the result is a partially ordered population of candidate solutions. This work follows the model of NSGA-II.

Let O be the set of optimization objectives. One candidate solution X *dominates* another candidate solution Y if $\forall\ o \in O,\ X[o] \leq Y[o] \wedge \exists\ o\ s.t.\ X[o] < Y[o]$ [8]. A solution that cannot be dominated is known as *Pareto optimal*. The set of all Pareto optimal solutions defines the *Pareto front*. It is likely that the algorithm will not find a Pareto optimal solution. Thus, possibly all of the candiate solutions are approximations. The idea is to assign each candidate solution to one of some number of *domination fronts* in which each candidate in front i is dominated by at least one candidate in front $i - 1$. Thus, the fronts impose a partial order on the candidates. Front 0 contains all non-dominated candidates. In a search for an optimal solution, front 0 constitutes the best found approximation of the Pareto front.

Selection of a best member of the population at the end of a run is performed autonomously by the algorithm. To make this identification, a radix sort is performed on front 0. The order of sort keys, from least significant to most significant, is: number of waypoints, smoothness, path length, obstacles hit, and goals hit. This order is motivated by the requirement to execute only those flight plans that are feasible, that is that hit no obstacles and hit all targets. Among feasible solutions, those with the shortest path lengths are chosen. If no feasible solution is found, the run is considered to have failed and is retried.

C. Population

The initial population, generation 0, consists of n randomly generated members. The chromosome for each member contains between 1 and 5 randomly generated waypoints. Each of these waypoints must be within some maximum distance of starting point s. As in any genetic algorithm, this initial population consists, with very high probability, of members that do not offer viable solutions to the problem. This initial

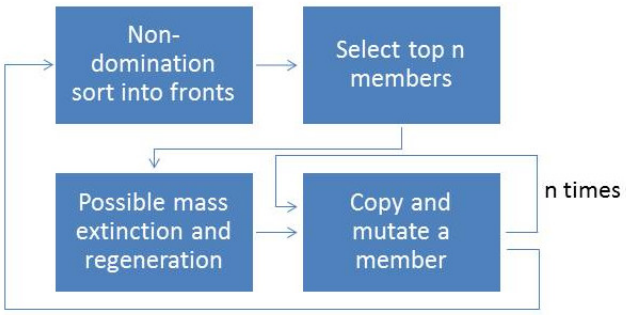

Fig. 1: Main loop of the genetic algorithm.

population is sorted into domination fronts as described in Section III-B.

An arbitrary iteration of the main loop of the algorithm proceeds as follows. Let generation g_j be the parent population, p_p. Initially, a child population p_c is created such that $|p_c| = |p_p| = n$. p_p and p_c are then merged into a combined population p with size $2n$. p is then resorted into domination fronts. Within a front, a niching measure [26] known as *crowding distance* [8] can be used to impose a partial order on the members in that front. The crowding distance of a member is a measure of the dissimilarity of that member to other members of the population. Greater values are preferred to help ensure diversity in the gene pool. For selection, let k be the largest index such that $\sum_{i=0}^{k} |f_i| = y \leq n$ where each f_i is a domination front. Then g_{j+1} consists of all members from f_0 through f_k and $n - y$ members from f_{k+1} chosen according to their crowding distances.

D. The Next Generation

Child creation occurs via mutation of a member of p_p. To select a member to mutate, the algorithm uses a tournament with *tournament pressure* equal 4. That is, it chooses four members of p_p and finds the "best" via a series of comparisons. Let member m_a be in front f_a and member m_b be in front f_b. m_a is better than m_b if $a < b$. If $a = b$, crowding distance is used to choose the winner. When the best member of the 4 has been chosen, it is copied and mutated. Both members are kept and will be subject to selection as described above.

Four mutation operators are implemented in the genetic algorithm: *add, delete, swap,* and *move*. When a member is chosen for mutation, exactly one of the operators is applied. The probabilities of the operators are 0.1, 0.05, 0.1 and 0.75, respectively. While *add* inserts a new waypoint into S, the other operators each alter an existing waypoint. Distinguished waypoints w_0 and w_n cannot be mutated.

The *delete*, *swap*, and *add* operators are easily explained. *delete* removes randomly selected w_i from S for some $0 < i < n$. *swap* randomly selects a pair of adjacent waypoints w_i and w_{i+1}, $0 < i < i + 1 < n$, and exchanges their positions in S. The *add* operator works by randomly selecting a pair of adjacent waypoints w_i and w_{i+1}, $0 \leq i < n$. It then creates a new waypoint, w_k, at the midpoint of the path segment between them. The *move* operator is then applied to w_k.

The *move* operator is somewhat more complex. A waypoint, w_i, $0 < i < n$, is chosen at random. The latitude and longitude values for w_i are changed independently according to a Gaussian distribution with mean 0 and standard deviation σ. The value for σ is selected by fair coin flip from among two candidates, *small_move* and *big_move*. *small_move* is fixed at 0.00002 degrees, approximately 2 meters. The value of *big_move* depends on the problem instance. For instances with small obstacles, it is 0.0002 degrees ($\sim$ 20 m) and for instances with large obstacles, it is 0.00067 degrees ($\sim$ 69 m).

E. Mass Extinction

Though the concept of mass extinction exists in the genetic algorithms literature [17] [20], it is not frequently used and has not been adopted for path planning. A mass extinction event eliminates some large fraction of the population. The intended goal of these events is to jump start evolution when the process stalls, to get out of evolutionary dead ends and avoid local minima. Lehman & Miikkulainen [20] note that in biological evolution, extinction events can result in accelerated evolution. *Elitism*, with respect to extinction events, means that some number of the best members of the population are exempted from deletion. The work of Mathias & Ragusa [23], while not conclusive, establishes that extinction implementations that incorporate a degree of elitism show promise for improving the number of generations required to achieve an acceptable solution.

Mass extinction is implemented as follows. Extinction events occur at random with fixed probability $p = 0.15$ every 50 generations. Eighty percent of the population is eliminated but the top three members are spared. Regeneration is a two step process. First, ten randomly generated members are added to the population to introduce new genetic material. Then, the balance of regeneration is achieved by copying and mutating existing members of the population.

F. Computational Complexity

The run-time of this genetic algorithm is dominated by the non-domination sort into fronts. For number of objectives m and population size n, the complexity is in $O(mn^2)$ for each generation. Thus for number of generations g, the total run-time is in $O(gmn^2)$. The linear dependence on number of generations and quadratic dependence on population size suggests use of longer runs with smaller populations, provided, of course, that such runs produce solutions of similar or better utility.

A *steady-state* genetic algorithm is one in which a non-domination sort is performed each time a member is added to the population. If performed naively, this obviously results in a large increase in the time requirement of the algorithm. However, Li, Deb, Zhang and Kwong [21] have shown that it is possible to selectively sort only those fronts affected by addition of a member. Despite possible ripple effects resulting from newly dominated members moving to another front, the run-time of their Efficient Non-domination Level Update (ENLU) approach is $O(gmn^{3/2})$, improving on that of NSGA-II. This approach has not yet been incorporated into the algorithm we present.

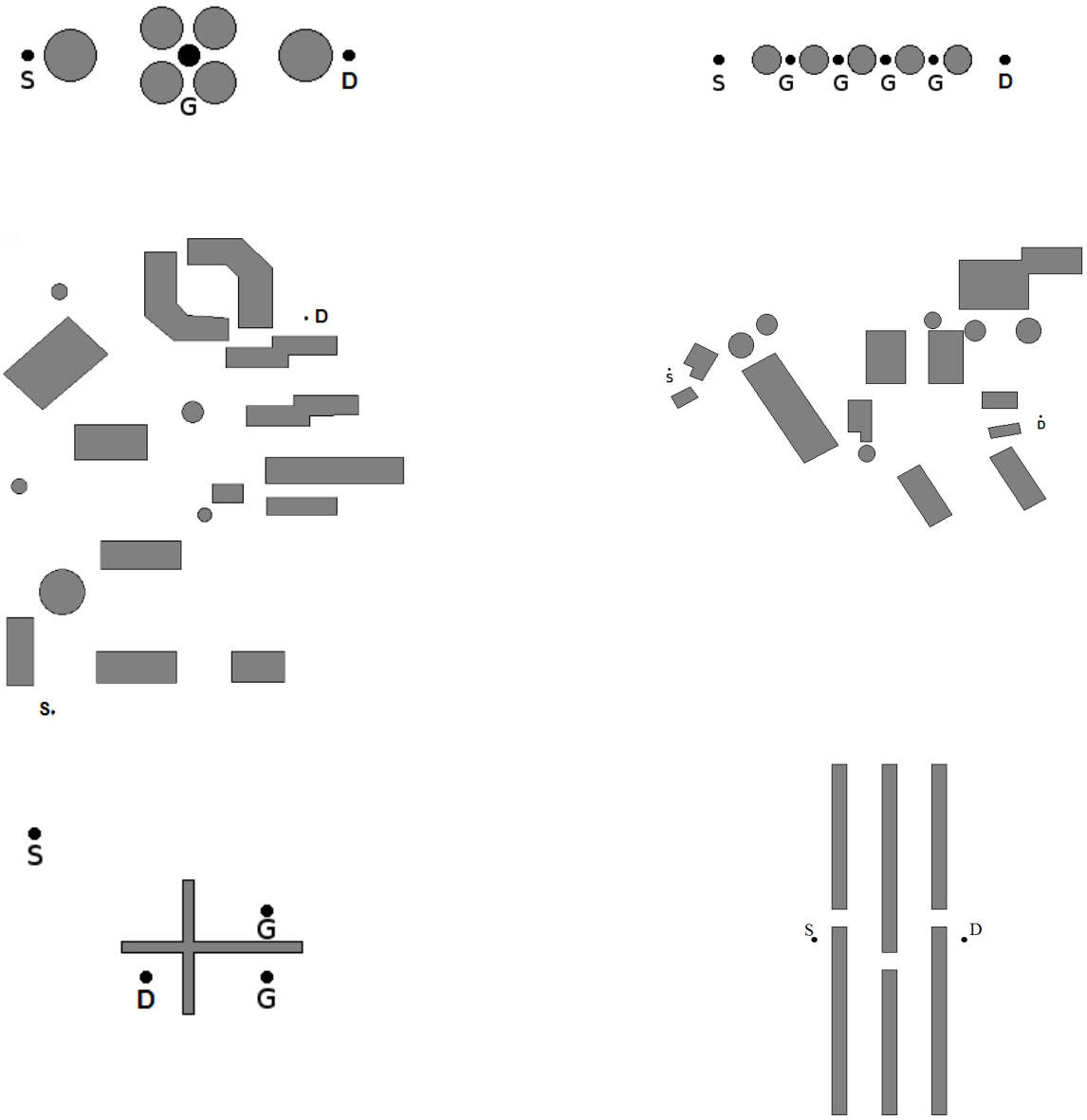

Fig. 2: Test inputs one through six, shown in row-major order. While additional test cases exist, extensive testing was conducted on those seen here. Each input shows starting point S, destination D, intermediate goals G, if any (all in black), and obstacles (in gray). Inputs three and four represent buildings on two college campuses with Euclidean distances from S to D of 305 meters and 360 meters, respectively. The other inputs are fabrications. Except for input 5, each of these has a Euclidean distance of approximately 62 meters, a consequence of the field on which most test flights are conducted.

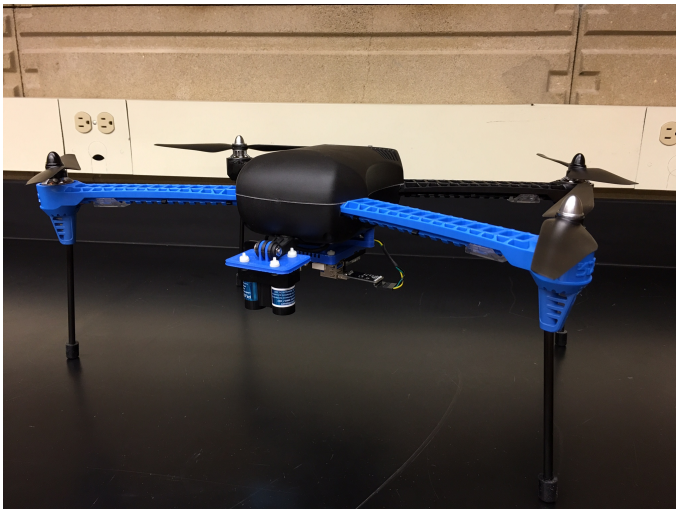

Fig. 3: The 3DR Iris+ with Odroid XU4 mounted below. In front is a LidarLite v2 laser range finder. The cable on the right is a serial to TTL connection between the XU4 and the Pixhawk which is inside the vehicle body.

IV. The Micro Aerial Vehicle

Though a number of researchers have examined evolutionary approaches to path planning for micro aerial vehicles, it does not appear that the algorithms have been tested in-flight. Flight is significant because the physical system and environment introduce real-world constraints that inform the design process. This section introduces the vehicle and accompanying hardware/software systems used.

The Iris+, introduced by 3DRobotics in Fall 2014, is a 550mm class (measured motor-to-motor) ready-to-fly quadcopter [1]. It is powered by a 5100mAh battery driving 4 950kV motors through a 4-in-1 electronic speed controller. It includes a Pixhawk flight control board (described below), an internal uBlox GPS with integrated 3-axis compass for navigation, and a 915MHz radio (or 433MHz, depending on country) for communication with a ground control station. Also included is a full-featured remote control transmitter with a preconfigured receiver onboard the drone. The Iris+ has an advertised flight time of 16-22 minutes.

Control of the Iris+ is facilitated via the Pixhawk flight control board [24]. Developed by Meier *et al.*, the Pixhawk is open source and very widely adopted. It is suitable for use in copters, planes and ground-based rovers. Connectivity options are numerous, allowing for use of a wide array of external devices such as GPS, range finders and companion computers. Internally, the unit integrates a three-axis accelerometer, a three-axis gyroscope, a three-axis compass and a barometer. These sensors allow for determination of motion, orientation, heading and relative altitude, respectively. Communication with the Pixhawk is facilitated by MAVLink [25], a message-based communication protocol.

MAVLink provides messages that facilitate navigation of the vehicle by allowing direct control of position, attitude and velocity. It does not currently provide direct manipulation of roll, pitch and yaw. Vehicle parameter values can be accessed and changed and vehicle status can be checked as well. The protocol is extensible, allowing users to define new messages for their purposes.

Firmware for the Pixhawk is the product of an open source project known as APM: Copter [10]. The experiments described below were run using version 3.3.2. Version 3.3 introduced a new, sophisticated extended Kalman filter for integrating GPS and inertial navigation systems for better vehicle control and stability. A recent version also introduced automatic tuning of PID parameters, relieving users of an often tedious chore.

In Spring 2015, 3DR introduced DroneKit-Python, a Python API for developing MAV applications [2]. DroneKit frees researchers and developers from many of the low-level aspects of MAVLink, providing a high-level interface to the protocol. DroneKit-Python provides primitives for connecting to, monitoring and controlling a vehicle. It also allows construction and sending of raw MAVLink messages from within client code.

Client applications to control the vehicle do not run on the Pixhawk but on a companion computer. The most straightforward implementation uses a ground-based laptop, communicating with the vehicle via a radio link. However, simplicity comes with a cost: the radio connection is low bandwidth and prone to noise. The low speed connection, 57600 baud, makes this option unsuitable for applications that require image processing, or other communication intensive computation, while noise can be problematic for mission-critical control messages.

A better alternative is to incorporate a single-board computer into the vehicle. The serial communication channel increases bandwidth to 1.5M baud and eliminates noise. And, of course, the MAV is fully autonomous during flight, a desirable feature in applications in which communication might be difficult. Thus, the vehicle has been fitted with an Odroid XU4, an SBC with a powerful heterogeneous 2.0 GHz octocore ARM processor and 2 GB of RAM. A 32GB eMMC 5.0 module serves as a solid state drive. The XU4 runs Ubuntu 15.04. Whlie this setup is not as computationally powerful as a laptop, improved communication speed and reliability make the tradeoff desirable.

V. Flying a Mission

Missions for the vehicle proceed as follows and as illustrated in Figure 4. First, the MAV and the onboard computer are powered. After booting, the XU4 creates a WiFi access point. A ground-based laptop uses this network to connect to the XU4 via ssh to enable launching the software. This connection also allows the vehicle to provide updates as it progresses through the mission. The software first establishes a connection between the onboard XU4 and the Pixhawk flight control board. If successful, a series of pre-arm checks are performed. These include acquiring a GPS signal which requires a minimum of six satellites. With the GPS operational, the vehicle's location is aquired and stored as the starting point for path planning.

With the starting position established and the destination known *a priori*, the program invokes the genetic algorithm to

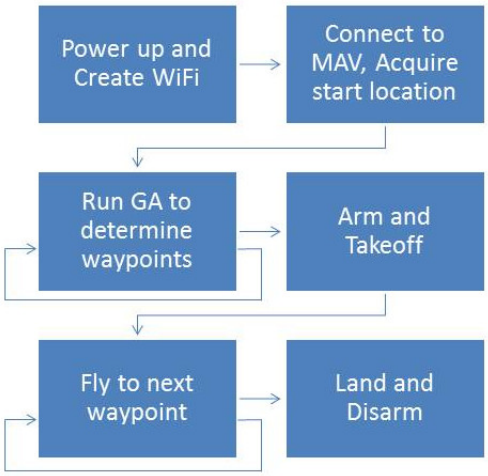

Fig. 4: Abstraction of the entire system. The genetic algorithm is run up to three times. If a feasible path is not found in those runs, the program terminates. If a feasible path is found, the waypoints are flown in order until the last waypoint, which is the destination, is reached.

find a path. Because the vehicle can fly only feasible paths, if a non-feasible path is returned, it is discarded and the GA is invoked again. If no feasible solution is found by the third run, the program disconnects from the vehicle and terminates. If a path is found, the vehicle is armed and takes off to a predefined altitude. This value is currently set at 5 meters. The vehicle then flies the sequence of waypoints in turn, ending at the destination location. Airspeed is set at 2 m/s. These altitude and airspeed values have been established for maximum safety. The software then instructs the vehicle to land and disarm before terminating the connection. Currently, most flights last only a few minutes or less. Flight length will increase as testing continues.

Figure 5 shows the flight code function that controls flying to a waypoint. The function invokes the DroneKit-Python function `simple_goto` which flies the drone from its current location on a heading toward a GPS location provided as a parameter. The heading is updated regularly along the way. Determining when the waypoint has been reached is done in client code, as seen in the figure. The API also provides functions for connecting to the vehicle, arming, taking off, directly controlling velocity (in all three axes) and heading, and landing.

VI. Results

Results in the context of this work are relative to the two stages of a mission. The first stage, the genetic algorithm, is evaluated based on its success in efficiently finding a viable solution to the path planning problem. The flight system, the second stage of the process, is evaluated on its effectiveness in completing the mission – even an optimal path is useless if the MAV doesn't successfully fly the missionl.

In testing, the genetic algorithm performs well. For each of the test inputs, a best-found path length was determined by way

TABLE I: Mean number of generations for each of the three measures described above, for six inputs. Extinction probability is 0.15.

Input	Feasible	10%	5%
1	20	112	170
2	61	188	234
3	537	560	572
4	22	42	135
5	45	75	109
6	99	352	674

of numerous runs of the genetic algorithm isolated from the flight platform. In the subsequent algorithm evaluation stage, three metrics were tracked: mean number of generations to a feasible solution, mean number of generations to a solution within 10% of best-found, and mean number of generations to a solution within 5% of best-found. These tests were run in isolation, independent of the flight control code and the simulator or MAV.

The measured values varied greatly with the complexity of the input. A simple, symmetric input, see the first input in Figure 2, was typically solved very quickly, while more complex inputs, such as those representing sections of two college campuses, required greater numbers of generations, see the third and fourth inputs in Figure 2. Each input was run 40 times with a population size of 100. The mass extinction probability is 0.15 and there is no crossover. This stage of testing was run on a Dell Latitude laptop with Core i7 processor. Typical runtimes are approximately 10 seconds/100 generations. The data values appear in Table I.

Figure 6 provides a time lapse sequence of a run of the genetic algorithm for input 4 and population size of 30. While a feasible solution is found very quickly, more generations are required to reduce the path length to near optimal.

After establishing the correctness of the genetic algorithm but prior to running it on the MAV, validity of the flight control software was established via extensive testing on a simulator. This was a crucial step in the development process as a number

```
def fly_waypoint(self, waypt, index):
    print "Flying to waypoint %s..." % index
    # create the waypoint in global_relative_frame
    dest_point = LocationGlobalRelative(waypt[0], waypt[1], self.alt_0)
    # fly to the waypoint at specified velocity
    self.vehicle.simple_goto(dest_point, groundspeed=self.v_base)

    # calculate current location and distance to waypoint so
    # that we can determine when we have gotten close enough
    current_loc = self.vehicle.location.global_relative_frame
    dist_lat, dist_lon, dist = self.distance_meters(current_loc, dest_point)
    while dist > self.dist_threshold:
        time.sleep(0.25)
        current_loc = self.vehicle.location.global_relative_frame
        dist_lat, dist_lon, dist = self.distance_meters(current_loc, dest_point)

    print "    At waypoint %s..." % index
```

Fig. 5: Code to fly to a waypoint given by GPS coordinates. DroneKit-Python provides the simple_goto function. The surrounding code sets up the call to simple_goto and determines when the specified location has been reached.

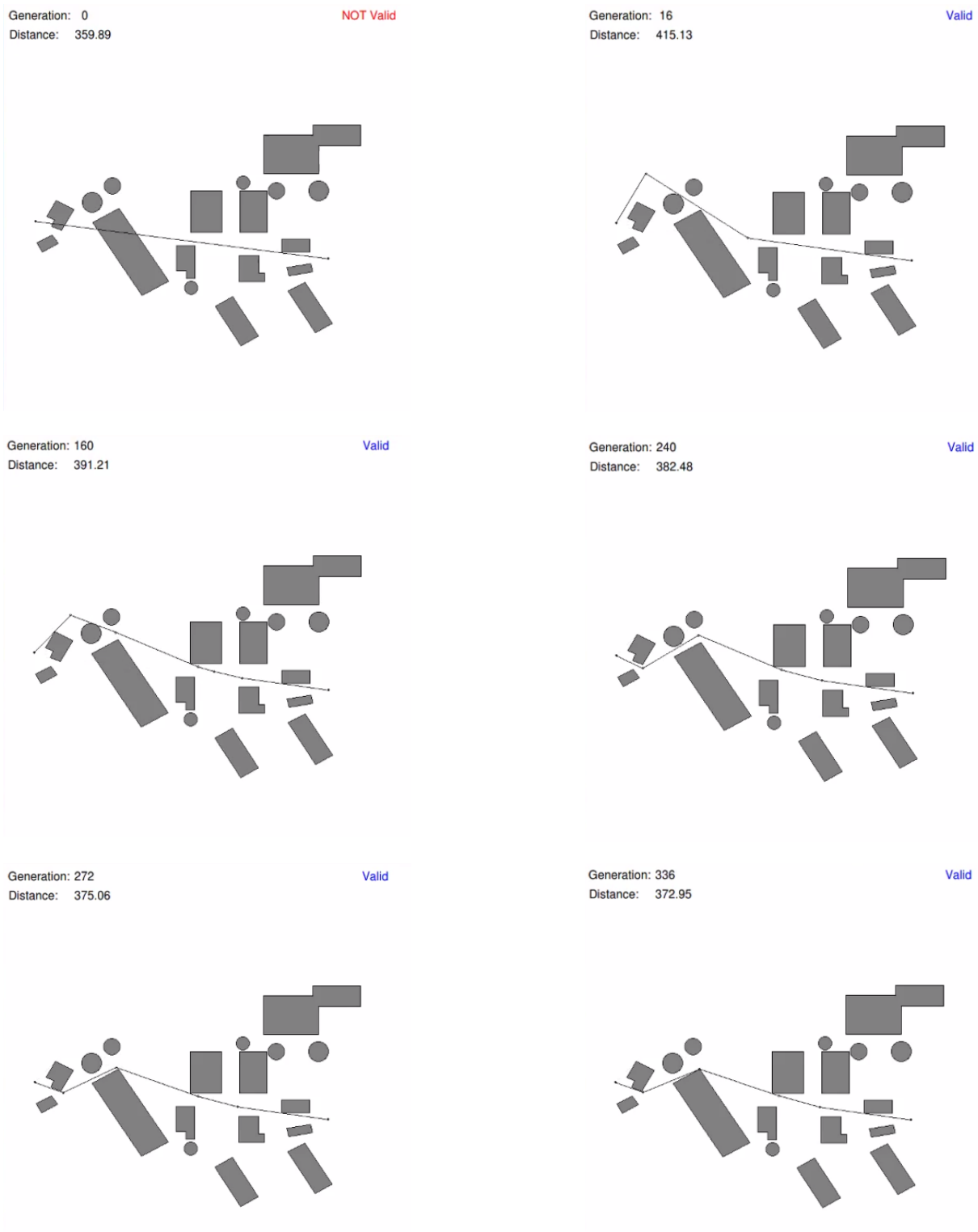

Fig. 6: A time lapse of a run on Input 4. The first image shows a straight line from source to destination. By generation 16, there is a feasible solution. This solution is refined until, by generation 240, a significantly different solution appears. This solution is incrementally refined at generation 272 and finally, by generation 336, little further refinement is possible.

of bugs were discovered without damaging or losing a drone.

Success of the flight platform is determined according to multiple criteria. The Odroid XU4 incorporated in the vehicle is slower than a modern laptop. This makes algorithmic efficiency and tuning particularly important. In practice, hardware speed has been sufficient. Taking advantage of multiple cores on the XU4, child evaluation is performed by four processes. A separate graphics process provides a graphical display of the genetic algorithm's progress.

To reduce run-times on this platform, during initial testing the population size was limited to 50. A run of 500 generations requires roughly 35 - 60 seconds, depending on complexity of the input. Recent testing using a smaller population, such as 30 members, and 1000 generations has demonstrated more reliable results. As suggested by the analysis in Section III-F the run-time is slightly improved as well. An improvement in both solution quality and run-time is compelling.

Recall that a solution to an instance of the path planning problem is a sequence of waypoints defining an obstacle-free path from the source to the destination. The second measure of the system's success is how accurately the vehicle is able to execute the mission. This is dependent on several factors: appropriate instruction set in the API, stability of the vehicle, and accuracy of the GPS unit. The Dronekit-Python API provided all necessary functionality. It did not constitute a limiting factor. Similarly, the Iris+ is very stable and responsive in flight. It proved to be a very capable flight platform.

GPS accuracy is defined by DOP, *dilution of precision*. The quantity of interest to us is HDOP, DOP in the horizontal plane. In testing, the uBlox GPS unit in the Iris+ typically achieves HDOP values of 1.7 meters, quite good for a consumer grade device. This degree of positional error will require that the algorithm treat the MAV as having a radius at least as large as the HDOP to reduce the probability of striking an obstacle. As a safety precaution, test flights have thus far been performed only with virtual obstacles. That is, flights are in an open field with obstacle positions marked by flags or poles but actual obstacles absent. Because of this, vehicle radius is not yet implemented.

VII. Conclusion

The need to solve path planning problems arises in many domains. Genetic algorithms have been widely applied to various versions of that problem. The system presented in this paper is the first use of a genetic algorithm for path planning to be incorporated into a live aerial vehicle. Building on previous work [23], a genetic algorithm has been specialized for the particular variant of path planning addressed. This includes eliminating crossover and integrating mass extinction.

There are opportunities for future research in several areas. Mass extinction shows promise for path planning. As implemented in the algorithm presented here, it improves mean number of generations to find a solution within 5% of optimal for some inputs tested. However, it seems likely that improvements are possible to the parameters determining when and how often extinction events occur and in the method of population regeneration after extinction.

Another area for future exploration, one that is currently being investigated, is adaptive path planning. In the work presented here, the environment is static and known *a priori*. Of course, in reality environments are rarely static. Moving objects, growing trees, cranes, new buildings all present the possibility of collision. To reduce the danger, sensors are being incorporated in the vehicle to allow detection of these hazards. Once the obstacles are known, a modified genetic algorithm can attempt to avoid them.

Finally, the system described here is currently being incorporated into a ground-based rover. This offers the benefit of greatly increased battery life and, therefore, mission time. In addition, movement on the ground poses additional challenges in the form of additional obstacles, such as stairs and hedges, that are inconsequential to a MAV. A non-technical benefit is that for rovers there is no concern about impending Federal Aviation Administration rules.

Acknowledgments

The authors would like to thank Florida Southern College for financial and other support for this work, and 3DR for providing educational pricing and support of open source projects important to research, commercial, and hobbyist projects for MAVs.

References

[1] 3D Robotics, Iris+, 2014, 3drobotics.com/iris-plus, retrieved December 13, 2015.

[2] 3D Robotics, DroneKit-Python Documentation, 2015, python.dronekit.io, retrieved December 13, 2015.

[3] F. Ahmed and K. Deb, Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms. Technical Report 2011013, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology Kanpur, 2011.

[4] K. Al-Sultan and M. Aliyu, A new potential field-based algorithm for path planning. Journal of Intelligent & Robotic Systems, 17(3), 265-282, 2010.

[5] N. Buniyamin, W. Wan Ngah, N. Shariff, and Z. Mohammad, A simple local path planning algorithm for autonomous mobile robots. International Journal fo Systems Applications, Engineering & Development, 5(2), 151-159, 2011.

[6] H. Burchardt and R. Salomon, Implementation of path planning using genetic algorithms on mobile robots. in 2006 IEEE Congress on Evolutionary Computing, 1831-1836.

[7] H. Choset and P. Pignon, Coverage path planning: the boustrophedon decomposition. in 1997 International Conference on Field and Service Robotics.

[8] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2), 182-197, 2002.

[9] K. Deb and H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, Part I: Solving problems with box constraints. IEEE Transactions on Evolutionary Computing 18(4), 577-601, 2014.

[10] Dronecode, APM: Copter, 2015, copter.arudpilot.com, retrieved December 13, 2015.

[11] D. Ferguson, M. Likhachev, and A. Stentz, A guide to heuristic-based path planning. in 2005 International Conference on Automated Planning and Scheduling.

[12] R. Glasius, A. Komoda, and S. Gielen, Neural network dynamics for path planning and obstacle avoidance. Neural Networks, 8(1), 125-133, 1995.

[13] D. Goldberg, Genetic Algorithms for Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[14] I. Hasircioglu, H. Topcuoglu, and M. Ermis, 3-D path planning for the navigation of unmanned aerial vehicles by using evolutionary algorithms. in 2008 ACM Genetic and Evolutionary Computation Conference, 1499-1506.

[15] A. Hermanu, T. Manikas, K. Ashenayi, and R. Wainwright, Autonomous robot navigation using a genetic algorithm with an efficient genotype structure. Intelligent Engineering Systems Through Artificial Neural Networks: Smart Engineering Systems Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life. ASME Press, 2004.

[16] H. Jain and K. Deb, An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, Part II: Handling constraints and extending to an adaptive approach. IEEE Transactions on Evolutionary Computing 18(4), 602-622, 2014.

[17] B. Jaworski, L. Kuczkowski, R. Smierzchalski, and P. Kolendo, Extinction event concepts for the evolutionary algorithms. Przeglad Elektrotechniczny (Electrical Review), 88(10b), 2012.

[18] H. Jun and Z. Qingbao, Multi-objective mobile robot path planning based on improved genetic algorithm. in 2010 IEEE International Conference on Intelligent Computation Technology and Automation, 752-756.

[19] A. Konak, D. Coit and A. Smith, Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering & System Safety 91, 992-1007, 2006.

[20] J. Lehman and R. Miikkulainen, Extinction events can accelerate evolution. PLoS ONE 10(8), August 2015.

[21] K. Li, K. Deb, Q. Zhang, and S. Kwong, Efficient non-domination level update approach for steady-state evolutionary multiobjective optimization. Technical Report 2014014, Computational Optimization and Innovation (COIN) Laboratory, Michigan State University, 2014.

[22] H.-S. Lin, J. Xiao, and Z. Michalewicz, Evolutionary navigator for a mobile robot. in Proceedings of the 1994 IEEE International Conference on Evolutionary Computation, 2199-2204.

[23] D. Mathias and V. Ragusa, On the utility of crossover and mass extinction in a genetic algorithm for pathfinding. To appear in Proceedings of the 2016 IEEE World Congress on Evolutionary Computation.

[24] L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Pollefeys, PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision. Autonomous Robots, 33, 21-39, 2012.

[25] L. Meier, MAVLink Common Message Set, 2015, pixhawk.ethz.ch/mavlink, retrieved December 13, 2015.

[26] O. Mengshoel and D. Goldberg, The crowding approach to niching in genetic algorithms. Evolutionary Computation, 16(3), 315-354, 2008.

[27] W. Page, J. McDonnell, and B. Anderson, An evolutionary programming approach to multidimensional path planning. in Proceedings of the First Annual Conference on Evolutionary Programming, 1992, 63-70.

[28] K. Sedighi, K. Ashenayi, T. Manikas, R. Wainwright, and H-M. Tai, Autonomous local path planning for a mobile robot using a genetic algorithm. Proceedings of the 2004 IEEE Congress on Evolutionary Computation, 1338-1345.

[29] U. Siddiqi, Y. Shriraishi, and S. Sait, Memory-efficient genetic algorithm for path optimization in embedded systems. IPSJ Transactions on Mathematical Modeling and Its Applications, 6(1), 2013.

[30] J. Xiao, Z. Michalewicz, L. Zhang, and K. Trojanowski, Adaptive evolutionary planner/navigator for mobile robots. IEEE Transactions on Evolutionary Computation, 1(1), 1997.

[31] Q. Zhang and H. Li, A multiobjective evolutionary algorithm based on decomposition. IEEE Transactions on Evolutionary Computation, 11(6), 2007.

[32] C. Zheng, M. Ding, C. Zhou, and L. Li, Coevolving and cooperating path planner for multiple unmanned air vehicles. Engineering Applications of Artificial Intelligence, 17, 887-896, 2004.