An On-Board Off-Board Framework for Online Replanning: Applied to UAVs in Urban Environments

Timothy Darrah[1], Jeremy Frank[2], Marcos Quiñones-Grueiro[1], Gautam Biswas[1]
[1]*Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA*
[2]*Intelligent Systems Division, NASA Ames Research Center, Mountain View, CA, USA*
{*timothy.s.darrah, marcos.quinones.grueiro, gautam.biswas*}*@vanderbilt.edu, jeremy.d.frank@nasa.gov*

Keywords: Genetic Algorithm, Agent-Based Replanning, Unmanned Aerial Vehicles (UAV), Cyber Physical Systems (CPS)

Abstract: Autonomous systems are being used in a multitude of areas at an increasing rate and require a high level of adaptivity and intelligence to operate safely, especially under faulty conditions. This paper introduces a novel genetic algorithm tailored for UAV trajectory replanning, with an improved execution time via search space reduction based on the operating conditions of the UAV and its remaining mission. A unique characteristic of the replanning agent is its fast-start and adaptive properties, pre-seeding candidates with partial solutions and dynamically tuning elitism, crossover, and mutation rates in correspondence to the average fitness and diversity of the population. A population restart mechanism and early stopping mechanism are evaluated as well to assess their effect on solution quality and runtime. Previous work on genetic algorithms for UAV replanning were conducted with short trajectories in a small state space. Our UAV operates in a 56,000 square meter simulated urban environment, with static obstacles and a total of 53 possible waypoints. The agent increases the safety and reliability of UAV autonomy when operating under faulty conditions and when replanning is required.

1 INTRODUCTION

Unmanned Aerial Vehicles (UAVs) are rapidly becoming an indispensable part of a variety of industries with both commercial and military applications. Ensuring their robust and reliable performance while operating with a multitude of uncertain factors is of critical importance. These factors include environmental conditions such as wind, internal conditions such as degradation, and unanticipated factors such as abrupt faults. This necessitates a high level of autonomy required for replanning in conditions where the search space is large, the safety constraints are tight, and time is of the essence.

Replanning under faulty conditions is a fundamental aspect of safe and reliable UAV operations and comes into play when unforeseen disturbances or faults occur during a flight mission. The primary aim of fault-aware replanning is to amend the current flight plan in order to ensure that the system is still able to operate and achieve some of its goals while finishing safely. This involves real-time adjustments to the flight path and waypoints, influenced by the dynamic feedback of the UAV's state and the environment. A necessary precursor is fault detection and isolation to identify when a fault occurs in real time as well as the fault magnitude. The literature is rich with techniques for fault detection and diagnosis methods, which can be classified as data driven (hypothesis testing, machine learning), model based (filtering, residuals), or hybrid approaches (filtering + hypothesis testing). The primary purpose of this work is to discuss the replanning algorithm used in such cases, and for a further discussion on fault management the reader is directed to (Hu et al., 2020), (Venkatasubramanian et al., 2003), and (Badihi et al., 2022) for a comprehensive review.

The replanning problem is formulated as a modified *Orienteering Problem* (OP), which is a type of constraint satisfaction problem that seeks to find a cycle in on the input graph which maximizes reward while not violating any constraints. The modification is that we are not finding a cycle, since replanning is triggered from an arbitrary point that is different than the goal location. A *Fast-Start Adaptive Genetic Algorithm* (GA) is implemented to solve this replanning problem, with 53 total waypoints over 56,000 square meters in an urban environment with static ob-

stacles (see Figures 4 and 6). The novel contribution is twofold: first, the on-board off-board agent-based replanning architecture allows for compute-intensive operations to be executed in an environment with a sufficiently large amount of resources and minimal delay in communication (2 seconds round trip); and second, key modifications to the GA implementation speed up computation time and generate better solutions with a higher reward to execution time ratio when compared to the vanilla GA.

1.1 Paper Organization

The rest of the paper is organized as follows. A brief literature review is presented in Section 2. The replanning approach is presented in Section 3. The GA is presented in Section 4. The experimental approach is discussed in Section 5. The results and discussion are given in Section 6, followed by the conclusion and future work in Section 7.

2 BACKGROUND

Planning is a crucial aspect of autonomous cyber physical systems that allow them to operate and perform their function with minimal human interaction. *Replanning* involves generating new plans in response to changes in the environment, changes in the system, or changes in the system's goals, and is necessary when operating in dynamic and stochastic environments. Replanning methods aim to ensure the system is adaptable, robust, and efficient by continuously updating plans or trajectories to achieve desired objectives. Virtually all planning algorithms can be modified to be a replanning algorithm, but this poses additional constraints and considerations that will be discussed. In the field of robotics (mobile robots, robotic arms, and multi-robot systems), planning studies have explored various techniques, including potential fields and model-predictive control (Li et al., 2021), and hierarchical task planning (Ryu, 2020) to enable robots to plan their actions. In the domain of UAVs, planning and is vital for tasks such as surveillance, reconnaissance, and package delivery in dynamic environments. Research works focus on developing real-time planning algorithms that consider factors like obstacle avoidance (Yang et al., 2022) or energy efficiency (Ahmed et al., 2016).

Online replanning assumes that an agent is initially following a predefined plan and due to new information must change the plan (Bonet and Geffner, 2011). With an updated knowledge base, a new plan can be computed that would allow it to achieve a subset of its objectives (Komarnitsky and Shani, 2016).

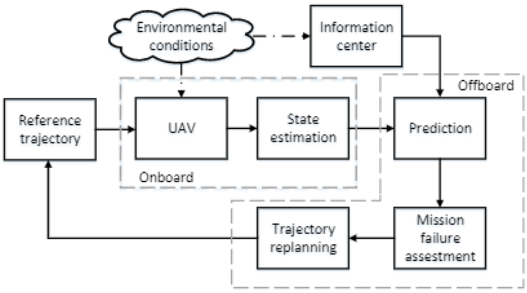

Figure 1: Replanning Approach using Off-board and Onboard Processing (Quiñones-Grueiro et al., 2021)

In the domain of CPS, an unviolable objective will always be to maintain safe operations and minimize risk of failure. Risk is therefore a probability function of the system's *State Of Health* (SOH) and performance constraints. A vast majority of the algorithms to solve these types of problems are well known, but their implementation in these systems is not a trivial task. In a static environment, everything can be computed offline without worry of processor limitations or computational complexity. In a dynamic environment, a complex system such as a UAV must repeatedly make these calculations and trigger a replanning mode if it detects a failure along its current trajectory. In In (Quiñones-Grueiro et al., 2021), the authors present a multi-objective cost function to assess risk of collision based on wind conditions and the current flight plan. A path search algorithm was utilized offboard to generate a new trajectory and then communicated to the UAV. The replanning approach is depicted in Figure 1.

2.1 Genetic Algorithms

GAs are heuristic-based optimization algorithms inspired by the principles of natural evolution and genetics, which include selection, crossover, and mutation (Goldberg, 1989). There are many variations to GAs applied to a wide variety of problems, and for an in-depth review the reader is directed to the works by (Chahar et al., 2021). The GA process begins with a randomly initialized population, where each individual represents a potential solution to the given problem, in this case, it is a search optimization problem. In each generation, individuals are evaluated based on a fitness function, which quantifies the quality of their solutions with respect to some value or reward in the problem domain. Next, selection operates to choose parents based on fitness, and then the chosen individuals produce offspring through crossover and mutation operations. Crossover combines the parts of the parent solutions into a new solution, and mutation in-

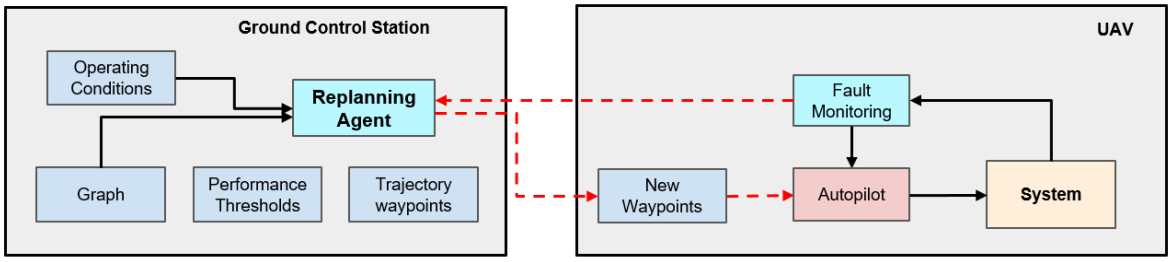

Figure 2: Replanning framework. The GCS uploads the trajectory and performance thresholds to the UAV prior to departure. Online, the UAV continually monitors for faults and estimates RUL. When it determines it cannot finish the original trajectory it requests a new plan from the ground station.

troduces small random changes into the offspring of the parents afterwards. Elitism is used to retain high-performing solutions from generation to generation.

The initial population is a crucial element, as it aids in reducing the time required for the GA to produce an optimal solution (Khaji and Mohammadi, 2014). In addition, initialization can significantly impact the quality of the final solution. In (Da Silva Arantes et al., 2015), the authors implemented a greedy heuristic in the population initialization function for a fixed-wing UAV tasked with finding an optimal landing spot and found that it safely landed the UAV 97% of the time, compared to 88% of the time without the greedy heuristic. In (Gyenes, Zoltán and Bölöni, Ladislau and Szádeczky-Kardoss, Emese Gincsainé, 2023), the authors presented a GA for online obstacle avoidance with sub-second execution times. However, the size of the search space was under 100 square meters, which renders the approach not applicable on larger scales.

A hybrid GA for path planning of a UAV combined with ray casting for obstacle avoidance is detailed in (de Moura Souza and Toledo, 2020). Their approach was implemented on a Raspberry Pi with a maximum timeout of 180 seconds. Unfortunately, 180 seconds of runtime is not feasible for online use. It is important to utilize domain knowledge in this regard when designing an algorithm to be used online. Generating the initial population informed by the goals and constraints of the problem is the first step to developing a rapidly converging GA in time-critical applications, such as replanning.

3 REPLANNING

The high level goal of replanning is to generate new mission parameters that allow the UAV to achieve as much of its original objectives as possible and return to base safely. Time is of the essence with complex systems in safety critical environments and mission designers need to know how the algorithms behave on

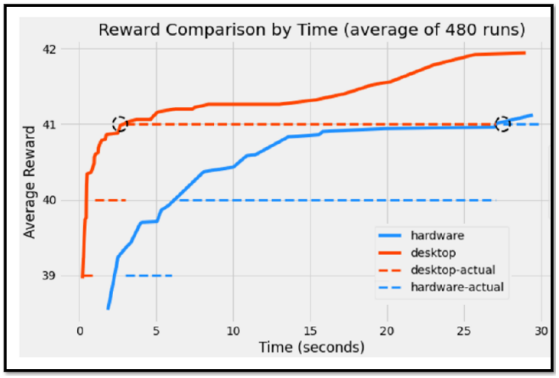

Figure 3: Reward comparison between the desktop computer (orange) and embedded hardware (blue) shown for up to 30 seconds of run time using the data from Example Scenario D (see Figure 4). Each reward point represents an additional waypoint added to the new trajectory, which represents an increase in utilization while still returning to base safely.

the target embedded hardware, as opposed to the computing platforms which they are developed on. This is exemplified in Figure 3, which shows the running time comparison for one replanning scenario (see Figure 4) when ran on the embedded hardware (discussed further in Section 5.2) compared to the desktop computer. This gives us the running time vs solution quality tradeoff, something that must be accounted for when desiging safety related software architecture.

Theoretically, replanning is no easier than planning (Nebel and Koehler, 1993), and in fact replanning itself *is* a planning problem. As this is the case, the modified OP (a ”planning” problem) is a suitable representation to frame the replanning problem, albeit with the slight modification that we are not looking for a path on the graph instead of a cycle. We leave the kinematics to a separate solver and focus on replanning the waypoints in a given trajectory.

The replanning framework is presented in Figure 1, consisting of the off-board replanning agent on the GCS, a pre-departure component (L), and an online component (R). During pre-departure the mission

plan is generated on the GCS and is loaded onto the embedded hardware with the performance thresholds. The RUL and auxiliary model that estimates flight time and power consumption are also loaded onto the embedded hardware. Since we are moving the replanning agent to the GCS one my ask why not also move the estimation models as well. The reasoning is that transmitting data consumes power and if we are constantly transmitting input data for the models then the battery will drain faster. Moreover, the RUL and auxiliary model execute in under a second - and moving these models to the GCS would increase the time to action by 200%. For these reasons, the models remain onboard, and the replanner is moved to the GCS.

During the online phase, the Health-Awareness architecture performs monitoring and estimation tasks. The distance, flight time, a power consumption estimates for each trajectory are known ahead of time. Every 10 seconds the onboard models estimate the RUL, remaining flight time, and remaining power. If at anytime either of these values are less than the original trajectory estimates or if the FDII module detects a fault, the UAV enters a hover mode and replanning agent on the GCS is activated. The GCS receives the UAVs current position and remaining distance, flight time, and power consumption estimates. It performs replanning and then sends the new trajectory back to the UAV, which then resumes flight.

3.1 Reward Function

The reward function of the replanner is such that original waypoints have a reward value of 5 units, and alternate waypoints have a reward value of 1 unit. In the example depicted in Figures 3 and 4, the theoretical maximum reward possible is 43 units, or 8 waypoints from the original trajectory (out of 9), and 3 additional waypoints. However, only 1 out of 5 trials was successful with this solution, whereas 5 out of 5 trials with one less alternate waypoint was successful (reward of 42). The UAV returned to base safely with an average of 7 seconds before failure is reached. Any mission designer will say this is to close for comfort.

A solution of 41 reward units, equivalent to 8 out of the original 9 waypoints and 1 additional waypoint, offers a buffer of approximately 28 seconds of flight time. Looking at Figure 3 again, we can see that the embedded hardware took 27.5 seconds to reach a solution of this quality, while the desktop computer took only 2.5 seconds. Now consider that we are able to communicate with the UAV and the round-trip communication time is 2 seconds. This means that we can move the replanner to the ground control station (GCS) and generate and execute a new plan in under 5 seconds.

The *value* of a waypoint w_i is the ratio of its reward to its distance to the original trajectory, $\mathbf{T}$, calculated for each waypoint as

$$values \leftarrow \frac{w^R_{1:|\mathbf{W}'|}}{dist(\mathbf{T}_{1:|\mathbf{W}'|})}, \forall\, w \in \mathbf{W}',$$

which is the first step of the population initialization function presented in Section 4.1.

3.2 Problem Formulation

The modified OP can be formulated as 4-tuple of elements, $(G, r, v_0, \mathbf{T})$ where $G = (\mathbf{W}, \mathbf{R})$ is an undirected graph with waypoints, $\mathbf{W} = [w_1, w_2, ...w_n]$, representing locations. Each edge in $\mathbf{R}$ is a tuple, $r_{ij} = (d_{ij}, t_{ij}, p_{ij}) \forall\, (w_i, w_j) \in \mathbf{W}$, which represents the segment distance, flight time, and power consumption, i.e. resources consumed between each pair of waypoints. Each waypoint has an associated reward, denoted by w^R. The start and goal location are the same for the original plan, $w_s = w_g$, but are different for the case of replanning, in which case the start location is the location at which the fault is detected, when the UAV enters hover mode to conduct replanning. The resource constraint that must not be violated, $r_{max} = (d_{max}, t_{max}, p_{max})$, is a tuple of max distance, max flight time, and max power consumption. Finally we have the original trajectory, $\mathbf{T}$, which is represented by a series of intermediate path points from waypoint to waypoint, beginning with the starting waypoint and ending at the goal waypoint. The path points can be anywhere from 5 meters to 40 meters apart based on the geometry of the trajectory (straight sections have path points that are further apart). This defines a graph-based optimization problem that we apply a modified GA to, discussed next.

4 FAST-START ADAPTIVE GENETIC ALGORITHM

The GA algorithm comprises several helper functions that initialize the population, calculate target fitness, calculate candidate fitness, calculate population diversity, perform the crossover and mutation operations, and calculate the adaptive rates for crossover, mutation, and elitism. Some functions are described in greater detail than others due to basic operations common to all GAs, and the GA itself is given in Algorithm 4. The proposed GA incorporates five modifications to the vanilla version, which are:

(A) Discard waypoints beyond a certain distance from the original trajectory to reduce the size of the search space;

(B) Initialize each candidate with a randomly assigned partial solution of the original trajectory to jump-start solution quality;

(C) Incorporate adaptive rates for elitism, crossover, and mutation, based on population performance and diversity to improve solution convergence;

(D) An early stopping mechanism to halt the algorithm when the solution quality plateaus; and

(E) A mechanism for population restart to reinitialize the population when solution quality does not improve.

We will return to this list of modifications in Sections 5 and 6. The algorithms are now detailed in the following subsections.

4.1 Population Initialization

The population initialization function, given in Algorithm 1, is where the *fast-start* property of the algorithm is contained. The algorithm prioritizes high-reward, close to the original trajectory waypoints via the *value function*. Furthermore, candidate solution is initialized by sampling from the remaining waypoints in the trajectory. To define what we mean by "close to the original path" we introduce a cutoff value, c, for which we use to consider waypoints to begin with. This allows us to restrict the search space and speed up the execution time of the algorithm. Formally, this is

$$W' = w \in \mathbf{W} : \exists\, D(w,p) \leq c, \forall\, p \in \mathbf{T}, \forall\, w \in \mathbf{W},$$

where W' is the reduced set of waypoints; w are individual waypoints in the set of all waypoints $\mathbf{W}'$; p are intermediate points along the trajectory $\mathbf{T}$; $D(w,p)$ is the distance between a waypoint w and an intermediate path point p; and c is the distance cutoff with which to ignore or consider waypoints. Moving forward, $\mathbf{W}'$ will denote the reduced set of waypoints used by the replanning agent.

The algorithm takes as input the population size, the graph, $\mathbf{G}$, the starting waypoint, w_s, the goal waypoint, w_g, the original trajectory, $\mathbf{T}$, and the resource threshold values, r_{max}. The *values*, line 2, is the value function used to guide the initial population generation process presented in Section 3.1. In line 3 the population is initialized to an empty set and in line 4 the population outerloop begins. A candidate, C, is initialized in line 5 by selecting a random number of

Algorithm 1: Generate initial population for GA

```
1  Function
     initialize_population(population-size,
     G(W', R), T, w_s, w_g, r_max)):
2      values ← w^R_{1:|W'|} / dist(T_{1:|W'|}), ∀ w ∈ W'
3      population ← ∅
4      for i = 1 ... population-size do
5          C ~ U(W, [0, |W'| − 2])
6          r ← (0, 0, 0)
7          while r < r_max do
8              E(w) ← values(w) / Σ values
9              wp ~ (waypoints, E(w))
10             waypoints ← waypoints − wp
11             r‘ ← r
12             if —C— ≡ 0 then
13                 r‘+ = R(w_s, wp)
14             end
15             else
16                 r‘+ = R(C_{−1}, wp)
17             end
18             if r‘ + R(wp, w_g) ≤ r_max then
19                 C ← C ∪ wp
20                 r ← r‘
21             end
22         end
23         population ← population ∪ C
24     end
25     return population
```

waypoints from the remaining waypoints in the original trajectory, while maintaining the initial ordering. The accumulated resources are initialized in line 6.

The inner loop starts on line 7, which builds an individual candidate solution. Lines 8-10 select a waypoint via weighted sampling from the *values* (the value function, line 2). Line 11 assigns the current resource utilization to r, which on the first iteration will be 0. Lines 12-17 accumulate these values based on if its a new candidate, line 13, or line 14 if not. Then in line 18, the resource thresholds, r_{max}, are checked, and the current waypoint is added to the candidate if the check is satisfied in line 19. The total resource utilization is updated in line 20. In line 23 the candidate is added to the population.

4.2 Helper Functions

The next function, *Calculate Target Fitness*, shown in Algorithm 2, calculates the target fitness that the average fitness of the population should exceed. This value is to allow the parameters of the algorithm to

dynamically change in real time based on the fitness of the overall population. One approach would be to calculate the reward per unit distance based on the total reward available in the search space and the max distance. However, this assumes an even distribution of rewards and travel segment distances, and there are many cases where this approach will yield a poor estimate.

Algorithm 2: Calculate target fitness

```
1  Function calc_target_fitness(G(W', R), w_s, w_g, r_max):
2      waypoints ← sort(W', values(W'))
3      f ← 0
4      r ← (0,0,0)
5      for each wp ∈ waypoints do
6          r‘ ← R(w_s, wp) + R(wp, w_g)
7          if r + r‘ ≤ r_max then
8              f ← f + w^V
9              r ← r + r‘
10         end
11         else
12             break
13         end
14     end
15     return f
```

A better estimation of the target fitness would account for the fact that some waypoints might provide more reward per unit distance than others. An improved approach involves sorting the waypoints based on their reward-to-distance ratio, using the distances from the starting waypoint and the goal waypoint. This is the approach given in Algorithm 2.

The waypoints are sorted according to their *values* in line 2. In line 3 fitness, *f*, is initialized, and in line 4 the resources, *r*, are initialized. The main loop begins on line 5, which iterates through the sorted list of waypoints and gets the distance, time, and power for the selected waypoint in line 6. Line 7 checks the thresholds if the thresholds are not violated then the fitness and and threshold values are accumulated in lines 8 and 9. The algorithm terminates when a resource threshold exceed the max allowable value, and the target fitness is returned in line 15.

Algorithm 3 performs the *Mutation* operation, and takes as input a candidate, C, the list of waypoints (excluding the start and goal waypoints, $\mathbf{W}'$, the mutation rate, m_{rate}, the resource thresholds, r_{max}, and the resource matrix, $\mathbf{R}$. Mutation operates on individual waypoints, or *genes*, within the candidate solution, using waypoints from the candidate, $w \in C$ or waypoints in the set of waypoints, $w \in \mathbf{W}'$. If a random

Algorithm 3: Mutation function

```
1  Function mutate(C, W', m_rate, r_max, R):
2      if random() ≤ m_rate then
3          op ← random(add, subtract, replace)
4          if op = add then
5              C_new ← C + random(w, w ∈ W', w ∉ C)
6          else if op = subtract then
7              C_new ← C − random(w, w ∈ C)
8          else if op = replace then
9              idx ← random(|C|)
10             C_new ← C
11             C_new^idx ← random(w, w ∈ W', w ∉ C)
12         if Σ_{w_i,w_j ∈ C_new} R(w_i, w_j) ≤ r_max then
13             return C_new
14         end
15     end
16     return C
```

number is less than the mutation rate then a mutation operation is randomly selected (lines 2 and 3).

To ensure maximal entropy in the resultant population, multiple mutation operations are implemented. The *addition* operation adds a waypoint to the candidate, C_{new} (line 5); the *subtraction* operation removes a waypoint from the candidate, C_{new} (line 7); and the *replace* operation replaces a waypoint in C_{new} (lines 9-11). *Transpose* is a common mutation operation that is discarded here because it breaks the ordering property that we do not want to violate. If the total resource consumption is less than the resource thresholds then the new candidate is returned (lines 12-13). If the mutation operation doesn't take place or if the resource thresholds are exceeded then the original candidate is returned (line 16).

The *Adaptive Rate* function (not depicted) calculates rates for crossover, mutation, and elitism based on the algorithms performance. The algorithms performance is characterized by the average population fitness, the population diversity, and the number of iterations since last improvement. It takes as input the average population fitness, $\bar{f}$; the target fitness, f^*; the population diversity, δ; the GA parameters, θ (x_{rate} is the crossover rate, m_{rate} is the mutation rate, e_{rate} is the elitism rate, α is the crossover and mutation step size, and β is the elitism step size); and the number of iterations since last improvement, i.

If the algorithm hasn't reached or exceeded the target fitness, has low diversity, or isn't improving, the crossover and mutation rates increase while the elitism rate decreases. If the target fitness has been

reached and the algorithm is improving, the crossover and mutations rates decrease while the elitism rate increases. If the target fitness has been reached but the algorithm isn't improving, then the elitism and crossover rates tend towards their initial value while the mutation rate increases. This is because both crossover and elitism utilize information within the population, but if the population is diverse but not producing better solutions, then we want to impart new candidates into the population using mutation instead.

4.3 The Genetic Algorithm

The proposed GA, given in Algorithm 4, uses the above functions and implements the *Fast-Start Adaptive Genetic Algorithm for Constrained Routing with Rewards*. Lines 1-7 initialize the best candidate, C^*, the fitness values, F, the iterations for improvement and the improvement counter, I & i, the *reset* counter, the target fitness values, f^*, and the population, $\mathbf{P}$. The main loop begins on line 7 at iterates for θ_N generations. The fitness for each candidate in the population is calculated in line 8, and the average fitness, $\bar{f}$, is calculated in line 9. The population diversity, δ, is calculated in line 10, and line 11 contains the adaptive check. If the number of iterations for improvement have passed without improvement, or if the population diversity is poor (values ¿ .5 are poor) and the average fitness is less than the target fitness, then a new population is initialized in line 12, and the reset counter is incremented in line 13. The only stopping condition outside of loop termination is in line 14, where the algorithm terminates if the population has been reinitialized 3 times. Otherwise, in line 16 the crossover, mutation, and elite rates, $\theta_{x_{rate}}$, $\theta_{m_{rate}}$, $\theta_{e_{rate}}$, respectively, are calculated.

Elitism is performed in line 18 by sorting the population based on fitness, and selecting the top N candidates, where $N = \theta_{e_{rate}}\ x\ |\mathbf{P}|$, the elitism rate multiplied by the length of the population. Next, in lines 19-23 the crossover operation is performed. First, a weighted sampling operation selects two random candidates, C_a & C_b, weighted by the population fitness. Then, the two candidates are crossed over and added to the offspring set in line 22. Line 24 performs mutation, mutating all candidates in the offspring set, whereby some candidates are returned unaltered based on the mutation rate, $\theta_{m_{rate}}$. The existing population is updated in line 25 with the elites and offspring, and in 26 the best potential candidate, $\hat{C}$ is taken. Line 27 performs two checks to determine the best candidate, C^*.

If the fitness of the potential candidate is equal to the fitness of best candidate, C^*, and the resource consumption of the potential candidate is less than the resource consumption of the best candidate, then the best candidate is replaced. Second, if the fitness of the potential candidate is greater than the fitness of the best candidate and the resource consumption of the

Algorithm 4: Adaptive Genetic Algorithm for Constrained Path Planning with Rewards

Input: The graph, $\mathbf{G}$, and GA parameters, θ
Output: C^*, the best candidate

1 $C^* \leftarrow \emptyset$
2 $F \leftarrow \emptyset$
3 $I, i \leftarrow \theta_I, 1$
4 $reset \leftarrow 0$
5 $f^* \leftarrow calculate\_target\_fitness(\mathbf{G})$
6 $\mathbf{P} \leftarrow generate\_initial\_population(\theta_{size}, \mathbf{G})$
7 **for** $j = 1 : \theta_N$ **do**
8 $\quad F \leftarrow calculate\_fitness(C, \mathbf{G}),\ \forall_C \in \mathbf{P}$
9 $\quad \bar{f} \leftarrow \frac{\sum F}{|\mathbf{P}|}$
10 $\quad \delta \leftarrow calculate\_diversity(\mathbf{P})$
11 $\quad$ **if** $i \mod I \equiv 0\ or\ (\delta > .75\ and\ \bar{f} < f^*)$ **then**
12 $\quad\quad \mathbf{P} \leftarrow generate\_initial\_population(\theta_{size}, \mathbf{G})$
13 $\quad\quad reset \leftarrow reset + 1$
14 $\quad\quad$ **if** $reset \equiv \theta_{reset}$ **then** break
15 $\quad$ **else**
16 $\quad\quad \theta_{x_{rate}}, \theta_{m_{rate}}, \theta_{e_{rate}} \leftarrow calculate\_dynamic\_rates(\bar{f}, f^*, \delta, \theta, i)$
17 $\quad$ **end**
18 $\quad elites \leftarrow sort(\mathbf{P}, F)_{1:\theta_{e_{rate}}\ x\ |\mathbf{P}|}$
19 $\quad offspring \leftarrow \emptyset$
20 $\quad$ **for** $n = 1 : |\mathbf{P}| - |elites|$ **do**
21 $\quad\quad C_a, C_b \sim (\mathbf{P}, F)$
22 $\quad\quad offspring \leftarrow offspring \cup crossover(C_a, C_b, \theta_{x_{rate}}, \mathbf{G})$
23 $\quad$ **end**
24 $\quad offspring \leftarrow mutate(C, \mathbf{G}, \theta_{m_{rate}})\ \forall_C \in offspring$
25 $\quad \mathbf{P} \leftarrow elites \cup offspring$
26 $\quad \hat{C} \leftarrow max(\mathbf{P}, F)$
27 $\quad$ **if** $(F(\hat{C}) \equiv F(C^*)\ and\ \mathbf{R}(\hat{C}) < \mathbf{R}(C^*))\ or\ ((F(\hat{C}) > F(C^*)\ and\ \mathbf{R}(\hat{C}) \leq r_{max})$ **then**
28 $\quad\quad C^* \leftarrow \hat{C}$
29 $\quad\quad i \leftarrow 1$
30 $\quad\quad reset \leftarrow 0$
31 $\quad$ **else**
32 $\quad\quad i \leftarrow i + 1$
33 $\quad$ **end**
34 **end**
35 **return** C^*

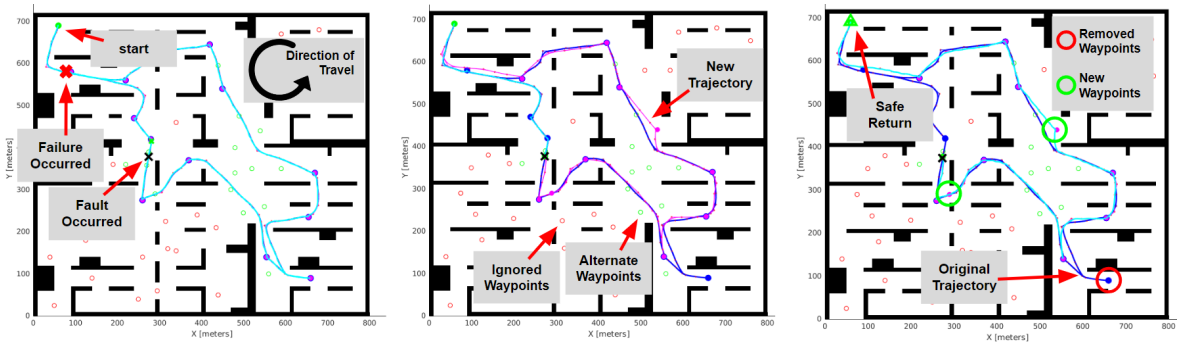

Figure 4: Replanning experimental process overview. Left: The UAV does not enter replanning mode and fails shortly before reaching base. Middle: The replanner is activated and selects a new trajectory using waypoints near the existing flight path. Right: The UAV executes the new trajectory and returns to base safely. The replanner discared one waypoint from the original trajectory, and added two additional waypoints. Fault onset time was at 207 seconds.

potential candidate is less than the resource thresholds, then the best candidate is replaced. If either check is satisfied, then the best candidate is updated in line 28, and the improvement and reset counters are reset in lines 29 and 30. Otherwise, the improvement counter increments in line 32. The algorithm returns the best candidate in line 35.

5 EXPERIMENTS

Several experiments were conducted to demonstrate the GA algorithm for replanning after the onset and detection of a fault. Three scenarios are presented in this work for discussion, with the scenario in Figure 4 depicting the overall experimental process. The experiments are set up as follows: First, the selected UAVs have completed between 40 and 80 flights, and the selected trajectories have an estimated flight time of 19-22 minutes. Possible faults include battery cell loss, parasitic load, and speed loss. Fault onset times randomly occur between 50 and 400 seconds into flight. Wind force magnitude is randomly set between 1.5 and 3.5 Newtons. The trajectory, UAV, fault time, fault type, and fault magnitude are determined randomly.

Figure 4 depicts a typical replanning scenario (with the proposed replanner), which will now be described moving from left to right. The left plot shows the UAV fly the original trajectory without replanning. The UAV starts in the upper left corner (green dot), and moves counter clockwise. There are a total of 12 waypoints (blue dots), and the fault (battery cell loss) occurs shortly after reaching the 3rd waypoint (black X) - 207 seconds into the flight. Without replanning, the UAV is able to reach all 12 waypoints but fails shortly before returning to base (red X).

The middle plot of Figure 4 depicts alternate waypoints that are ignored by the replanner (red markers), waypoints that are considered by the replanner (green markers) (i.e. $\mathbf{W}'$), and the new trajectory (purple line). The *distance-to-path* is a parameter of the replanner which is used to delineate waypoints that are considered and waypoints that are ignored. In this case, that value was set at 60 meters. The right plot shows that UAV flew the new trajectory and reached base safely (green triangle). We can also see that an original waypoint was removed (red circle), and two alternate waypoints were added (green circles).

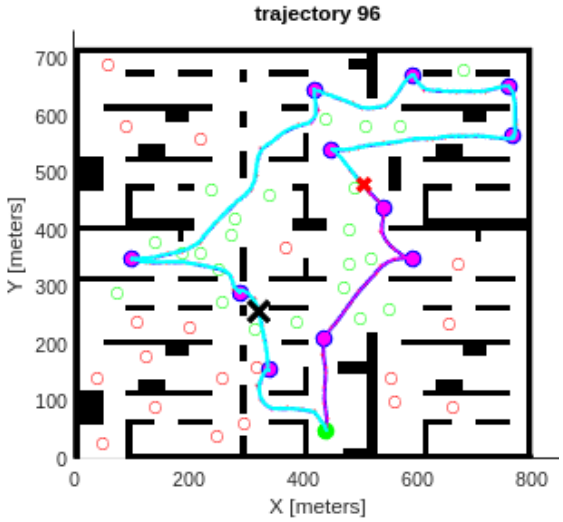

Figure 5: Scenario for GA comparison with a fault onset time of 135 seconds and a fault type of battery cell loss. The UAV travels clockwise and the fault occurs at the black X, with failure occurring at the red X.

5.1 GA Comparison

To demonstrate the effectiveness of the modified GA, we compare it to the vanilla GA and highlight both so-

lution quality and runtime. The vanilla GA considers all waypoints in the search, does not initialize candidates with partial solutions, does not have adaptive elitism, crossover, and mutation rates, and does not have a population restart or early stopping function. The scenario used for this study is shown in Figure 6. We run the GA 30 times for each trial. The trials consist of the vanilla GA; the GA with each modification A-B-C-D-E individually; the GA with A-B modifications; the GA with A-B-C modifications; the GA with A-B-C-D modifications, and finally the GA with each modification A-B-C-D-E combined. The results are discussed in Section 6.

The GA parameters for the replanning experiment are given below in Table 1.

Parameter	Symbol	value
number of generations	θ_N	100
population size	θ_{size}	1000
improvement iterations	θ_I	25
resets to halt	θ_{reset}	3
crossover rate	$\theta_{x_{rate}}$	1.0
mutation rate	$\theta_{m_{rate}}$	.3
elitism rate	$\theta_{e_{rate}}$	.3
rate increment 1	θ_α	.05
rate increment 2	θ_β	.02

Table 1: GA search parameters.

5.2 GA Performance Analysis

The GA was tested on the *Ground Control Station* (GCS) as well as the embedded hardware to understand how the algorithm performs on different computing platforms. The GCS is equipped with A 24 core Ryzen Threadripper 3960X CPU at 3.8 GHz with a clock speed of 1.78GHz. The embedded hardware used for the experiments is a Unibab Spacecloud Q7[1] with a dual-core AMD G-series SOC CPU operating at 1.8GHz. These compute modules are radiation tolerant and used for space applications.

A performance test of the GA was conducted to determine the relationship between the population size, number of generations, solution quality, and runtime on both platforms. This was initially discussed above with a runtime versus reward comparison plot between the hardware and desktop computer in Figure 3. The test will now be described, with the above example scenario in Figure 4 as context. In the above scenario, the original waypoint had 12 waypoints total, and 9 remaining waypoints after the failure occured. Of the remaining original waypoints, the replanning agent is allowed to select at maximum 8, or 1 less than the remaining total. The replanning agent was executed a total of 10 times for each population size and number of generation combinations taken from the following:

$$pop.\_size \leftarrow [100, 250, 500, 1000, 1500, 2000]$$

$$n\_gen. \leftarrow [25, 50, 75, 100, 125, 150, 175, 200]$$

This results in a total of 480 runs of the GA, executed for the above scenario on both the embedded hardware and the GCS. The results are discussed next.

6 RESULTS & DISCUSSION

We first discuss the GA ablation study, followed by a discussion of the GA performance on embedded hardware (on-board) versus the GCS (off-board). The results of the ablation study are shown in Table 2. What we see is that the vanilla GA (trial 1) performs the worst in terms of reward, and 2 out of 30 trials did not return a solution. We also see that the population restart mechanism (trials 6 and 10) had the highest time cost, but did not have the highest reward improvement. Trial 7, the GA with the 'fast-start' modifications (A-B), had the best time-to-reward ratio. Adding the 'adaptive' property, trial 8 (A-B-C), increase the reward achieved at the cost of execution time. The execution time was then reduced in trial 9 (A-B-C-D), with a slight reduction in reward as well. There was no benefit in terms of reward or execution time with incorporating population restart in trial 10 (A-B-C-D-E).

ID	Method	Time	Reward	Notes
1	Vanilla	5.26s	31.81	2*
2	A	5.13s	34.41	1*
3	B	5.33s	37.62	–
4	C	8.26s	35.16	1*
5	D	4.14s	31.91	2*
6	E	10.1s	33.77	–
7	AB	5.35s	38.14	–
8	ABC	7.69s	38.55	–
9	ABCD	6.57s	38.27	–
10	ABCDE	8.62s	37.95	–

Table 2: GA ablation experiment results.
*number of times a solution was not found

[1] https://unibap.com/wp-content/uploads/2020/03/1004001-unibap-information-sheet-on-unibap-e2000e2100-modules.pdf

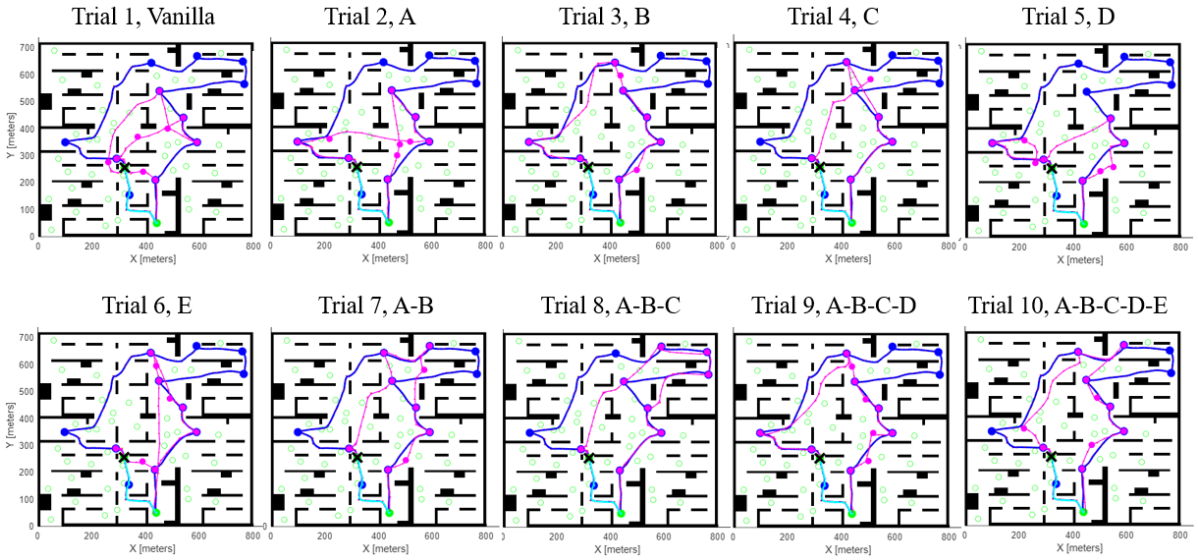

Figure 6: Example trajectories generated by the replanning agent with different versions of the GA. Each Trial maps to the ID field of Table 2.

6.1 On-Board Off-Board Analysis

Figures 8 and 9 each show a 4-D surface plot for the improved GA algorithm using data from the scenario depicted in Figure 4. Figure 8 is for the desktop test, and Figure 9 is the hardware test. The Z-axis represents running time, the Y-axis represents population size, the X-axis represents number of generations, and the colorbar represents reward. This plot allows us to see all four variables at once, and find the optimal manifold for which to choose our parameters from. The biggest different between the desktop and embedded hardware is in the running time, which is expected.

An additional factor that affects the performance of the algorithm is the distance threshold with which to either consider or discard alternate waypoints. More waypoints are considered for larger distances, and the resulting population contains more solutions with trajectories that may be sub-optimal, which affects convergence time. Therefore, it is recommended to keep the the distance cutoff threshold as small as possible.

The key takeaway from this experiment is that online replanning with time constraints and large search spaces is still a challenge that we face. Advances in communication technologies have made it possible to move the replanning agent off-board while still ensuring *in-time* solutions when accounting for the communication overhead, which is roughly 2 seconds round trip for a UAV in an urban environment, when the GCS is co-located in the area of operations.

6.2 Solution Quality as a Function of Runtime

Figure 7 depicts the health-aware replanning solution on the left with a population size of 500 and 100 generations, and on the right with a population size of 2000 and 200 generations. When comparing the left and right solutions, we can see that both solutions discarded the same 3 waypoints from the original trajectory. However, when the running time constraint is ignored (right), 2 additional waypoints are added to the trajectory. This results in a reward of 38 vs 36 for this particular example. The trade-off is that for two extra waypoints an additional 19 seconds of runtime is required.

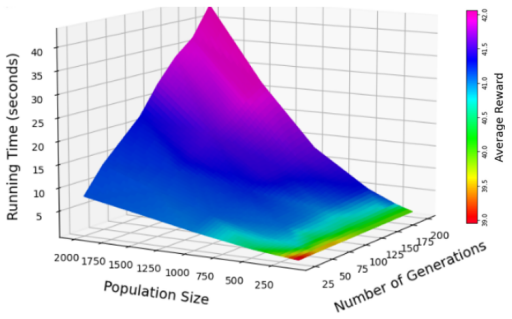

Figure 8: GA performance plots on desktop computer.

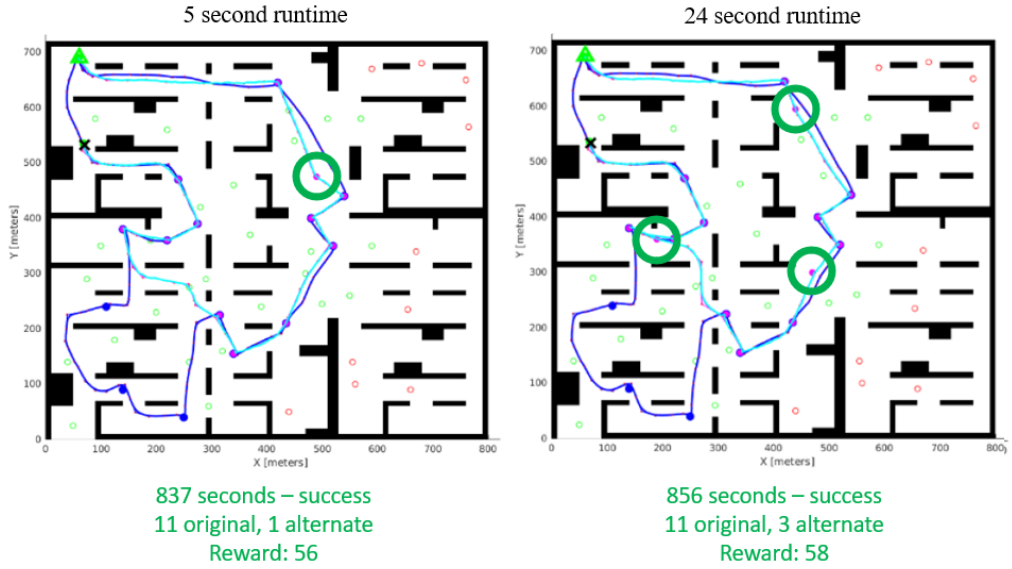

Figure 7: Left: replanning solution when controlling for execution time ($<$ 5 seconds $\rightarrow$ population size = 500, generations = 100) results in a reward of 36. Right: replanning solution when runtime is not a factor (population size = 2000, generations = 200) results in a reward of 38.

n_gen	pop_size	avg_time	avg_reward
200	250	1.75	41.06
100	500	2.89	41.01
150	500	3.71	41.10
175	500	4.14	41.22
200	500	4.46	41.29

Table 3: Top GA performance results on desktop with reward greater than or equal to 41 and runtime less than 5 seconds.

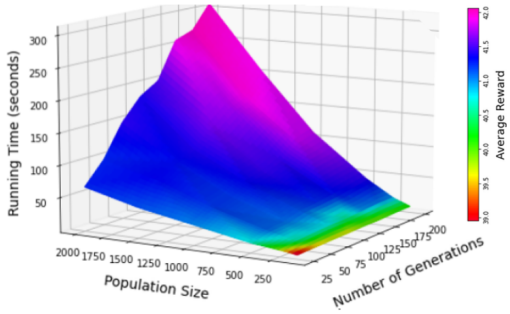

Figure 9: GA performance plots on the Unibap Qseven.

n_gen	pop_size	avg_time	avg_reward
100	500	27.11	41.01
125	500	29.46	41.12
150	500	32.45	41.20
175	500	37.58	41.30
200	500	42.17	41.36

Table 4: Top GA performance results on embedded hardware with reward greater than or equal to 41.

7 CONCLUSION

This research introduces on-board off-board framework for UAV replanning under time and resource constraints. A novel genetic algorithm tailored to the online constrained path planning with rewards problem is presented. Five different modifications to the vanilla GA were presented with a discussion and ablation study. These include include a ”fast start” population initialization technique, where candidate solutions are seeded with parts of the original trajectory, and waypoints beyond a threshold distance from the original path are not considered. The algorithm adaptively responds to the population performance and diversity by modifying its crossover, mutation, and elitism rates to either encourage exploration or exploitation as necessary. A population restart mechanism is also incorporated to rejuvenate the search when required. The results show that the fast start property (methods A-B) contributes the most achieving high rewards with relatively short execution time, while adding the adaptive property (A-B-C) increases the reward achieved at a cost of increasing the execution time. The execution time can then be reduced wile maintaining a higher level of reward by adding in an early stopping mechanism (A-B-C-D). Adding in the population restart mechanism (A-B-C-D) had a negative effect on both reward and execution time.

UAV resilience lies in effective replanning to complete as much of the original mission goals as possible in a safe manner. The replanning process is complicated by the need for time-constrained, on-the-fly

computations. The goal of replanning is twofold: to maximize rewards by flying through as many attainable waypoints as possible, with a preference for waypoints on the initial trajectory; and to ensure the UAV does not violate any of its resource thresholds. However, inaccuracies in resource estimates can lead the UAV to mistakenly believe it possesses more flight time, culminating in mission failure before returning to base.

Immediate future work in this direction would be to incorporate a continuous planning architecture, as opposed to a one-time, reactive approach. Instead of reactive adjustments post-failure, since we do not have computing constraints by moving the replanning agent off-board, we could continuously plan, sending only the next one or two waypoints at a time to the UAV. We could also continuously keep a tree of several alternate plans so that when the UAV requires a new plan it can be immediately uploaded. This agent, equipped with contingency blueprints for varying levels of potential failure severities, would usher in a paradigm of anticipatory path optimization.

ACKNOWLEDGEMENTS

This work is supported in part by NASA OSTEM Fellowship 20-154.

REFERENCES

Ahmed, S., Mohamed, A., Harras, K., Kholief, M., and Mesbah, S. (2016). Energy efficient path planning techniques for uav-based systems with space discretization. In *2016 IEEE Wireless Communications and Networking Conference*.

Badihi, H., Zhang, Y., Jiang, B., Pillay, P., and Rakheja, S. (2022). A comprehensive review on signal-based and model-based condition monitoring of wind turbines: Fault diagnosis and lifetime prognosis. *Proceedings of the IEEE*, 110.

Bonet, B. and Geffner, H. (2011). Planning under partial observability by classical replanning: Theory and experiments. In *IJCAI*.

Chahar, V., Katoch, S., and Chauhan, S. (2021). A review on genetic algorithm: Past, present, and future. *Multimedia Tools and Applications*, 80.

Da Silva Arantes, J., Da Silva Arantes, M., Toledo, C. F. M., and Williams, B. C. (2015). A multi-population genetic algorithm for uav path re-planning under critical situation. In *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*.

de Moura Souza, G. and Toledo, C. F. M. (2020). Genetic algorithm applied in uav's path planning. In *2020 IEEE Congress on Evolutionary Computation (CEC)*.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

Gyenes, Zoltán and Bölöni, Ladislau and Szádeczky-Kardoss, Emese Gincsainé (2023). Can genetic algorithms be used for real-time obstacle avoidance for lidar-equipped mobile robots? *Sensors*.

Hu, X., Zhang, K., Liu, K., Lin, X., Dey, S., and Onori, S. (2020). Advanced fault diagnosis for lithium-ion battery systems: A review of fault mechanisms, fault features, and diagnosis procedures. *IEEE Industrial Electronics Magazine*, 14.

Khaji, E. and Mohammadi, A. S. (2014). A heuristic method to generate better initial population for evolutionary methods. *CoRR*.

Komarnitsky, R. and Shani, G. (2016). Computing contingent plans using online replanning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*.

Li, J., Sun, J., Liu, L., and Xu, J. (2021). Model predictive control for the tracking of autonomous mobile robot combined with a local path planning. *Measurement and Control*, 54(9-10):1319–1325.

Nebel, B. and Koehler, J. (1993). Plan modification versus plan generation: A complexity-theoretic perspective. *13th International Joint Conference on Artifical Intelligence*.

Quiñones-Grueiro, M., Biswas, G., Ahmed, I., Darrah, T., and Kulkarni, C. (2021). Online decision making and path planning framework for safe operation of unmanned aerial vehicles in urban scenarios. *Aerospace Journal (to appear)*.

Ryu, H. (2020). Hierarchical path-planning for mobile robots using a skeletonization-informed rapidly exploring random tree*. *Applied Sciences*, 10.

Venkatasubramanian, V., Rengaswamy, R., Yin, K., and Kavuri, S. N. (2003). A review of process fault detection and diagnosis: Part i: Quantitative model-based methods. *Computers & Chemical Engineering*, 27(3):293–311.

Yang, F., Fang, X., Gao, F., Zhou, X., Li, H., Jin, H., and Song, Y. (2022). Obstacle avoidance path planning for uav based on improved rrt algorithm. *Discrete Dynamics in Nature and Society*, 2022.