Module 5B - Game of Drones: Optimization of Agricultural UAV Swarms

Omar Betancourt, Payton Goodrich, Emre Mengi

June 11, 2023

Contents

Objectives:
The objective of this module is to provide basic modeling and simulation techniques for systems of multiple interacting Unmanned Aerial Vehicles (UAVs), so called "swarms", for applications in mapping. Also, this modules illustrates the application of basic machine-learning algorithms to optimize their information gathering.
Prerequisite Knowledge: N/A
Prerequisite Modules: 1A - Calculus, 1B - Linear Algebra, 1D - Differential Equations, 2C - Particle Dynamics, 3C - Generic Time Stepping, 4A - Genetic Algorithms, 4B - Gradient-based Optimization
Difficulty: Hard
Summary:
In this module, you will learn basic modeling and simulation techniques for systems of multiple interacting Unmanned Aerial Vehicles (UAVs), so called "swarms", for applications in mapping. Also, the paper illustrates the application of basic machine-learning algorithms to optimize their information gathering. Numerical examples are provided to illustrate the concepts.

1 Theory

1.1 Introduction

The advances in Unmanned Aerial Vehicles (UAV) technologies have the potential to revolutionize rapid mapping capabilities of complex environments to directly benefit society. Furthermore, advances in manufacturing, three dimensional (3D) printing, embedded microscale and nanoscale electronics, Lidar (Light Detection and Ranging), hyperspectral cameras and associated technologies have made the production of sophisticated multi-UAV systems, so-called "swarms", for mapping very economical. One key enabling technology for such systems is rapid, adaptive, path planning for such systems. Accordingly, the objective of this work is to develop relatively simple mechanistic models and numerical solution strategies for the direct simulation of path planning of swarms, which can be achieved with relatively standard laptop-level computing equipment for deployed use in the field. The models developed here are simple enough to be computed several thousand times per hour, thus enabling machine-learning algorithms to optimize the multi-UAV system performance.

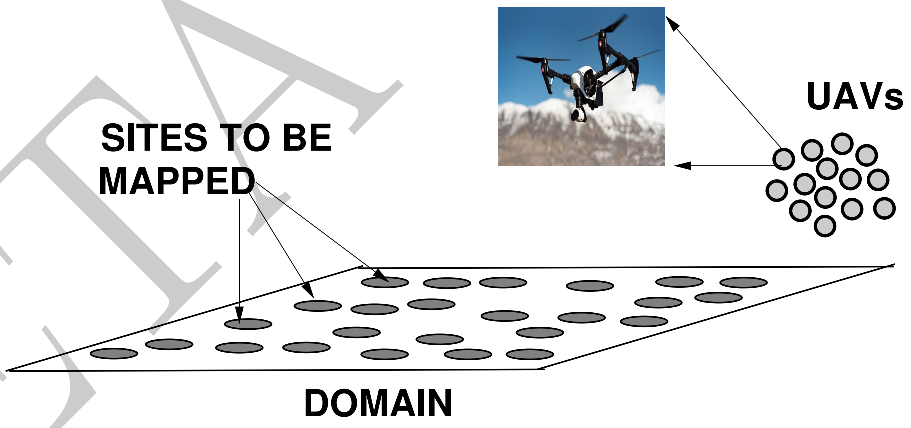


Figure 1.1: A domain with sites of interest to be mapped.

1.2 A brief history of UAVs/drones

Following a review found in Zohdi [1], in 2019, the use of UAVs range from (a) hobbyists, (b) mid-sized military and commercial applications and (c) large-scale military vehicles and (d) stealth combat vehicles. Issues include (1) structural design, (2) power supply, (3) onboard computing, (4) sensing devices, (5) generated acoustics, (6) data acquisition software, (7) control algorithms, (8) communications and (9) autonomy.

Research on UAVs started in the early 1900s. Initially, most of the research was geared towards military applications. This research accelerated moderately during World War II, to train antiaircraft gunners and to fly attack missions. However, with the exception of the V-2 (Vergeltungswaffe) rocket system program in Germany, they were primarily “toy” airplanes. It was not until the 1960’s, during the cold war, when the US was involved in a variety of military conflicts and the US Air Force was concerned about losing pilots over hostile territory, that UAV research started to grow rapidly. As of 2012, the US Air Force operated approximately 7,500 UAVs. As of 2017, nearly every industrialized country has a growing UAV manufacturing base.

Due to a steady increase in inexpensive Unmanned Aerial Vehicle (UAV) and camera technology, there are a wide variety of non-military applications, such as world-wide anti-poaching and anti-whaling efforts. Furthermore, for example in oil and gas exploration, UAVs have been used for geophysical mapping, in particular geomagnetic surveys, where measurements of the Earth’s varying magnetic field strength are used to calculate the nature of the underlying magnetic rock structure, in order to locate mineral deposits. Because of the huge expanse associated with oil and gas pipelines, monitoring activity can be enhanced and accelerated by deployment of UAVS. In the field of archaeology, drones are used to accelerate surveying to protect sites from looters. Another obvious application is cargo transport, which has been promoted by Amazon, DHL, Google, etc. The use of UAVs in agriculture is also obvious for crop dusting and crop health monitoring.

However, the overarching goal of all of these applications is the real-time mapping of large areas, such as those struck by after a multi-location disaster, such as an earthquake, fire, tsunami, etc., by multiple UAV’s; so-called “swarms”. Because of the complex multifaceted infrastructure that needs to be mapped (roads, bridges, pipelines, power grid and water) after a disaster, there exists the need for different mapping strategies (Figure 1.1). Such sectors need to be mapped with different technologies (infrared, RF, optical, microwave, etc). Small UAVs are usually battery powered (Figure 1.1), thus they have limited range and their paths must be planned carefully to conserve power. Specifically, the objective of this work is to provide an introduction to the basic modeling and simulation techniques for multiple interacting UAVs for a target audience of young scientists. Specifically, simultaneous advances in inexpensive UAVs, computational modeling techniques, camera and sensor technologies have made rapid post-disaster mapping a potential reality. One motivator for this research is the monitoring/maintenance of ultra-large facilities, such as industrial-scale solar farms. However, an over-arching key motivator that is driving multiple-UAV system development is large-area mapping for disaster mitigation and management, for example, the devastating 2017 Sonoma/Napa fires and the 2018 Paradise fires in California. In terms of disaster response, integration of UAVs, advanced sensing, communications and rapid simulation techniques with fire-fighting to develop large-scale rapid-response systems for emergency fire control, management and risk assessment is of high interest. A key objective is to develop paradigms that provide accurate, real-time feedback to deployed firefighters. A core issue across all domains of application is the ability of a system to adapt to rapid changes in the environment and system capabilities by autonomously modifying tasks and relationships with other agents, and then to apply various data-collection techniques. Oftentimes, autonomous capabilities will be necessary for continued operations due to the large distances, resulting communications delay, and lack of 24/7 connectivity to the systems (a function of limited ground stations in the tracking, telemetry, and commanding network). These factors can be incompatible with human reaction/decision times.

Simultaneous advances in inexpensive UAVs, computational modeling techniques, camera and sensor technologies have made rapid pre- and post-disaster mapping of complex terrain a reality. Agent-based paradigms for simulation of coupled complex systems have become powerful predictive tools. Because different infrastructures have different grids and different quantities to be mapped, the optimal path for a set of released swarms will vary over the same terrain. It is relatively easy to develop so-called agent-based models for a team of swarm-members (UAVs) intending to map large areas with various optimality conditions: minimum time, minimum energy usage, optical sensing, infrared sensing, acoustic sensing, water spillage sensing, etc. Although UAV’s in themselves are of interest, the long-term goal is coordination of large-numbers of them, so-called “swarms”, which is the focus of this work. At the end of this work, ancillary technologies, such as UAV integration and Lidar are also discussed, with an eye towards systems which blend artificial intelligence, machine learning, and software analytics with data to create living digital computer models that can update and change in tandem with their physical counterparts.

Remark: It is important to note that new FAA regulations require eligible owners to register their UAV’s prior to flight. For owners less than 13 years old, a parent or other responsible person must file an FAA registration form, and the UAV's must have an FAA-issued registration number. In June 2016, the FAA announced regulations for commercial operation of small UAVs, those between 0.55 and 55 pounds (about 0.250- 25 kg), including payload, which require the onsite presence of licensed Remote Pilot in Command (above 16 years of age). Because of the growing use of UAVs, privacy concerns have mounted, and led to property owners shooting down such vehicles which enter their airspace. Shooting down a drone is illegal, since the debris can harm people below. Such vehicles are shot down at a rate of once a month in the US (in 2016). In Zohdi[2] the dynamical response of a quadcopter to a series of random external impulses, such as from shotgun pellets, was formulated using a Discrete Element Method (DEM), and allowed one to compute trajectories and the distribution of the debris field. This is potentially useful in settling disputes over location of drones at the time of shooting. This topic is outside of the scope of the current work, however, we refer interested readers to Zohdi[2].

1.3 Modeling and rapid simulation of swarms

The origins of swarm modeling has origins in the description of biological groups (flocks of birds, schools of fish, crowds of human beings, etc.) responses to to predators or prey (Breder (1952 [3]). We focus on decentralized paradigms where there is no leader, making the overall system less vulnerable. Early approaches that rely on decentralized organization can be found in Beni [10], Brooks [11], Dudek et al [12], Cao et al [13], Liu and Passino [14] and Turpin et al [15]. Usual models incorporate a tradeoff between long-range interaction and short-range repulsion between individuals, dependent on the relative distance between individuals (see Gazi and Passino [4], Bender and Fenton [5] or Kennedy and Eberhart [6]). The most basic model is to treat each individual as a point mass (Zohdi [7]), which we adopt here, and to allow the system to evolve, based on Newtonian mechanics, using a combination of short-range and long-range interaction forces (Gazi and Passino [4], Bender and Fenton [5], Kennedy and Eberhart [6] and Zohdi [7, 8, 9]).[1]

Remark: For some creatures, the "visual field" of individuals may play a significant role, while if the agents are robots or UAVs, the communication can be electronic. However, in some systems, agents interact with a specific set of other agents, *regardless* of whether they are far away (Feder [20]). This appears to be the case for Starlings (Sturnus vulgaris). In Ballerini et al [21], the authors concluded, that such birds communicate with a certain number of birds surrounding it and that that interactions are governed by topological distance and not metric distance. Interested readers are referred to Ballerini et al [21].

1.3.1 Notation

Throughout the analysis, the objects are assumed to be small enough to be considered (idealized) as point-masses and that the effects of their rotation with respect to their mass center is considered unimportant to their overall motion. Boldface symbols imply vectors or tensors. *A fixed Cartesian coordinate system will be used throughout this work.* The unit vectors for such a system are given by the mutually orthogonal triad of unit vectors $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$. We denote the position of a point (swarm) in space by the vector $\boldsymbol{r}$. In fixed Cartesian coordinates we have

$$\boldsymbol{r} = r_1\boldsymbol{e}_1 + r_2\boldsymbol{e}_2 + r_3\boldsymbol{e}_3, \tag{1.1}$$

and for the velocity we have

$$\boldsymbol{v} = \dot{\boldsymbol{r}} = \dot{r}_1\boldsymbol{e}_1 + \dot{r}_2\boldsymbol{e}_2 + \dot{r}_3\boldsymbol{e}_3, \tag{1.2}$$

and acceleration we have

$$\boldsymbol{a} = \ddot{\boldsymbol{r}} = \ddot{r}_1\boldsymbol{e}_1 + \ddot{r}_2\boldsymbol{e}_2 + \ddot{r}_3\boldsymbol{e}_3. \tag{1.3}$$

[1]There are other modeling paradigms, for example mimicing ant colonies (Bonabeau et al., [16]) which exhibit foraging-type behavior and trail-laying-trail-following mechanisms for finding food sources (see Kennedy and Eberhart [6] and Bonabeau et. al [16], Dorigo et. al, [17], Bonabeau et. al [16], Bonabeau and Meyer [18] and Fiorelli et al [19]).

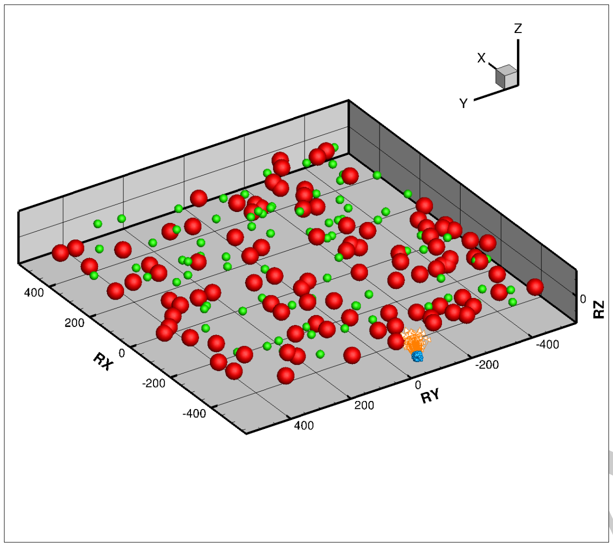


Figure 1.2: Model problem consisting of targets (red), obstacles (green), distributed randomly in a $(\pm 500, \pm 500, \pm 10)$ meter domain and swarm-members (blue, distributed initially in a $(-500, 0, \pm 10)$ domain).

1.3.2 Construction of a swarm

In the analysis to follow, we treat the swarm members as point masses, i. e. we ignore their dimensions (Figure 1.2). For each swarm member (N_s in total) the equations of motion are

$$m_i \dot{\boldsymbol{v}}_i = m_i \ddot{\boldsymbol{r}}_i = \boldsymbol{\Psi}_i^{tot} = \mathcal{F}(\boldsymbol{N}_i^{mt}, \boldsymbol{N}_i^{mo}, \boldsymbol{N}_i^{mm}) \tag{1.4}$$

where $\boldsymbol{\Psi}_i^{tot}$ represents the total forces acting on a swarm member i, $\boldsymbol{N}_i^{mt}$ represents the interaction between swarm member i and targets to be mapped, $\boldsymbol{N}_i^{mo}$ represents the interaction between swarm member i and obstacles and $\boldsymbol{N}_i^{mm}$ represents the interaction between swarm member i and other members. In order to illustrate the overall computational framework, we focus on a model problem having a sufficiently large parameter set which allows for complex dynamics. The parameters optimized to drive the system towards desired behavior via a Machine-Learning algorithm. This approach can be used on a variety of models.

1.4 Characterization of interaction

1.4.1 Member-target interaction

Consider member-target interaction

$$||\boldsymbol{r}_i - \boldsymbol{T}_j|| = \left((r_{i1} - T_{j1})^2 + (r_{i2} - T_{j2})^2 + (r_{i3} - T_{j3})^2\right)^{1/2} \overset{\text{def}}{=} d_{ij}^{mt}, \tag{1.5}$$

where $\boldsymbol{T}_j$ is the position vector to target j and the direction to each target is

$$\boldsymbol{n}_{i\to j} = \frac{\boldsymbol{T}_j - \boldsymbol{r}_i}{||\boldsymbol{r}_i - \boldsymbol{T}_j||}. \tag{1.6}$$

For each swarm-member (i), we compute a weighted direction to each target

$$\hat{\boldsymbol{n}}_{i\to j} = (w_{t1} e^{-a_1 d_{ij}^{mt}} - w_{t2} e^{-a_2 d_{ij}^{mt}}) \boldsymbol{n}_{i\to j}, \tag{1.7}$$

where the w_{ti}s are weights reflecting the importance of the target, a_i are decay parameters, which is summed (and normalized later in the analysis) to give an overall direction to move towards

$$\boldsymbol{N}_i^{mt} = \sum_{j=1}^{N_t} \hat{\boldsymbol{n}}_{i\to j}. \tag{1.8}$$

1.4.2 Member-obstacle interaction

Now consider member-obstacle interaction

$$||\boldsymbol{r}_i - \boldsymbol{O}_j|| = \left((r_{i1} - O_{j1})^2 + (r_{i2} - O_{j2})^2 + (r_{i2} - O_{j2})^2\right)^{1/2} \overset{\text{def}}{=} d_{ij}^{mo}, \tag{1.9}$$

where $\boldsymbol{O}_j$ is the position vector to obstacle j and the direction to each obstacle is

$$\boldsymbol{N}_{i\to j} = \frac{\boldsymbol{O}_j - \boldsymbol{r}_i}{||\boldsymbol{r}_i - \boldsymbol{O}_j||}. \tag{1.10}$$

For each swarm-member (i), we compute a weighted direction to each obstacle

$$\hat{\boldsymbol{n}}_{i\to j} = (w_{o1}e^{-b_1 d_{ij}^{mo}} - w_{o2}e^{-b_2 d_{ij}^{mo}})\boldsymbol{n}_{i\to j}, \tag{1.11}$$

where the w_{oi}s are weights reflecting the importance of the obstacle, b_i are decay parameters, which is summed (and normalized later in the analysis) to give an overall direction to move towards

$$\boldsymbol{N}_i^{mo} = \sum_{j=1}^{N_o} \hat{\boldsymbol{n}}_{i\to j}. \tag{1.12}$$

1.4.3 Member-member interaction

Now consider member(i)-member(j) interaction

$$||\boldsymbol{r}_i - \boldsymbol{r}_j|| = \left((r_{i1} - r_{j1})^2 + (r_{i2} - r_{j2})^2 + (r_{i3} - r_{j3})^2\right)^{1/2} \overset{\text{def}}{=} d_{ij}^{mm}, \tag{1.13}$$

and the direction to each swarm-member

$$\boldsymbol{n}_{i\to j} = \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{||\boldsymbol{r}_i - \boldsymbol{r}_j||}. \tag{1.14}$$

For each swarm-member (i), we compute a weighted direction to each swarm-member

$$\hat{\boldsymbol{n}}_{i\to j} = (w_{m1}e^{-c_1 d_{ij}^{mm}} - w_{m2}e^{-c_2 d_{ij}^{mm}})\boldsymbol{n}_{i\to j}, \tag{1.15}$$

where the w_{mi}s are weights reflecting the importance of the members, c_i are decay parameters, which is summed (and normalized later in the analysis) to give an overall direction to move towards

$$\boldsymbol{N}_i^{mm} = \sum_{j=1}^{N_m} \hat{\boldsymbol{n}}_{i\to j}. \tag{1.16}$$

1.4.4 Summation of interactions

We now aggregate the contributions by weighting their overall importance with weights for swarm-member/target interaction, W_{mt}, swarm-member/obstacle interaction, W_{mo} and swarm-member/swarm-member interaction, W_{mm}:[2]

$$\boldsymbol{N}_i^{tot} = W_{mt}\boldsymbol{N}_i^{mt} + W_{mo}\boldsymbol{N}_i^{mo} + W_{mm}\boldsymbol{N}_i^{mm}, \tag{1.17}$$

normalize the result

$$\boldsymbol{n}_i^* = \frac{\boldsymbol{N}_i^{tot}}{||\boldsymbol{N}_i^{tot}||}. \tag{1.18}$$

The forces are then constructed by multiplying the thrust force available by the UAV propulsion system, F_i, by the overall normal direction

[2]The parameters in the model will be optimized shortly.

$$\boldsymbol{\Psi}_i^{tot} = F_i \boldsymbol{n}_i^*. \tag{1.19}$$

We then integrate the equations of motion:

$$\boldsymbol{m}_i \dot{\boldsymbol{v}}_i = \boldsymbol{\Psi}_i^{tot}, \tag{1.20}$$

yielding

$$\boldsymbol{v}_i(t+\Delta t) = \boldsymbol{v}_i(t) + \frac{\Delta t}{m_i}\boldsymbol{\Psi}_i^{tot}(t) \tag{1.21}$$

and

$$\boldsymbol{r}_i(t+\Delta t) = \boldsymbol{r}_i(t) + \Delta t \boldsymbol{v}_i(t). \tag{1.22}$$

Note that if

$$||\boldsymbol{v}_i(t+\Delta t)|| > v_{max}, \tag{1.23}$$

then we define $\boldsymbol{v}_i^{old}(t+\Delta t) = \boldsymbol{v}_i(t+\Delta t)$ and the velocity is rescaled

$$\boldsymbol{v}_i^{new}(t+\Delta t) = v_{max}\frac{\boldsymbol{v}_i^{old}(t+\Delta t)}{||\boldsymbol{v}_i^{old}(t+\Delta t)||}, \tag{1.24}$$

with $\boldsymbol{v}_i(t+\Delta t) = \boldsymbol{v}_i^{new}(t+\Delta t)$. We then determine if any targets have been mapped by checking the distance between swarm-members and targets

$$||\boldsymbol{r}_i - \boldsymbol{T}_j|| \leq Tolerance. \tag{1.25}$$

For any $\boldsymbol{T}_j$, if any swarm-member has satisfied the criteria, then take $\boldsymbol{T}_j$ out of the system for the next time step so that no swarm-member wastes resources by attempting to map $\boldsymbol{T}_j$. Similarly, if $||\boldsymbol{r}_i - \boldsymbol{O}_j|| \leq Tolerance$, then $\boldsymbol{r}_i$ is "immobilized". This stops the *ith* swarm member from contributing further to the mapping. The entire process is then repeated for the next time step.

1.5 An algorithm for mapping of a region

To "map" a region, consider the following algorithm:

1. Initialize the locations of the targets to be mapped: $\boldsymbol{T}_i = (T_x, T_y, T_z)_i$, i=1, 2,...$N_T$=targets.
2. Initialize the locations of the obstacles to be mapped: $\boldsymbol{O}_i = (O_x, O_y, O_z)_i$, i=1, 2,...$N_O$=obstacles.
3. Initialize the locations of the swarm-members (UAVs): $\boldsymbol{r}_i = (r_x, r_y, r_z)_i$, i=1, 2,...$N_s$=swarm-members.
4. For each swarm-member (i), determine the distance and directed normal to each target, obstacle and other swarm-members.
5. For each swarm-member (i), determine interaction functions $\boldsymbol{N}_i^{mt}$, $\boldsymbol{N}_i^{mo}$, $\boldsymbol{N}_i^{mt}$ and $\boldsymbol{n}_i^*$.
6. For each swarm-member (i), determine force acting upon it, $\boldsymbol{\Psi}_i^{tot} = F_i\boldsymbol{n}_i^*$.
7. For each swarm-member (i), integrate the equations of motion (checking constraints) to produce $\boldsymbol{v}_i(t+\Delta t)$ and $\boldsymbol{r}_i(t+\Delta t)$.
8. Determine if any targets have been mapped by checking the distance between swarm-members and targets

$$||\boldsymbol{r}_i - \boldsymbol{T}_j|| \leq Tolerance. \tag{1.26}$$

For any $\boldsymbol{T}_j$, if any swarm-member has satisfied the this criteria, then take $\boldsymbol{T}_j$ out of the system for the next time step so that no swarm-member wastes resources by attempting to map $\boldsymbol{T}_j$. As indicated before, if $||\boldsymbol{r}_i - \boldsymbol{O}_j|| \leq Tolerance$, then $\boldsymbol{r}_i$ is "immobilized".

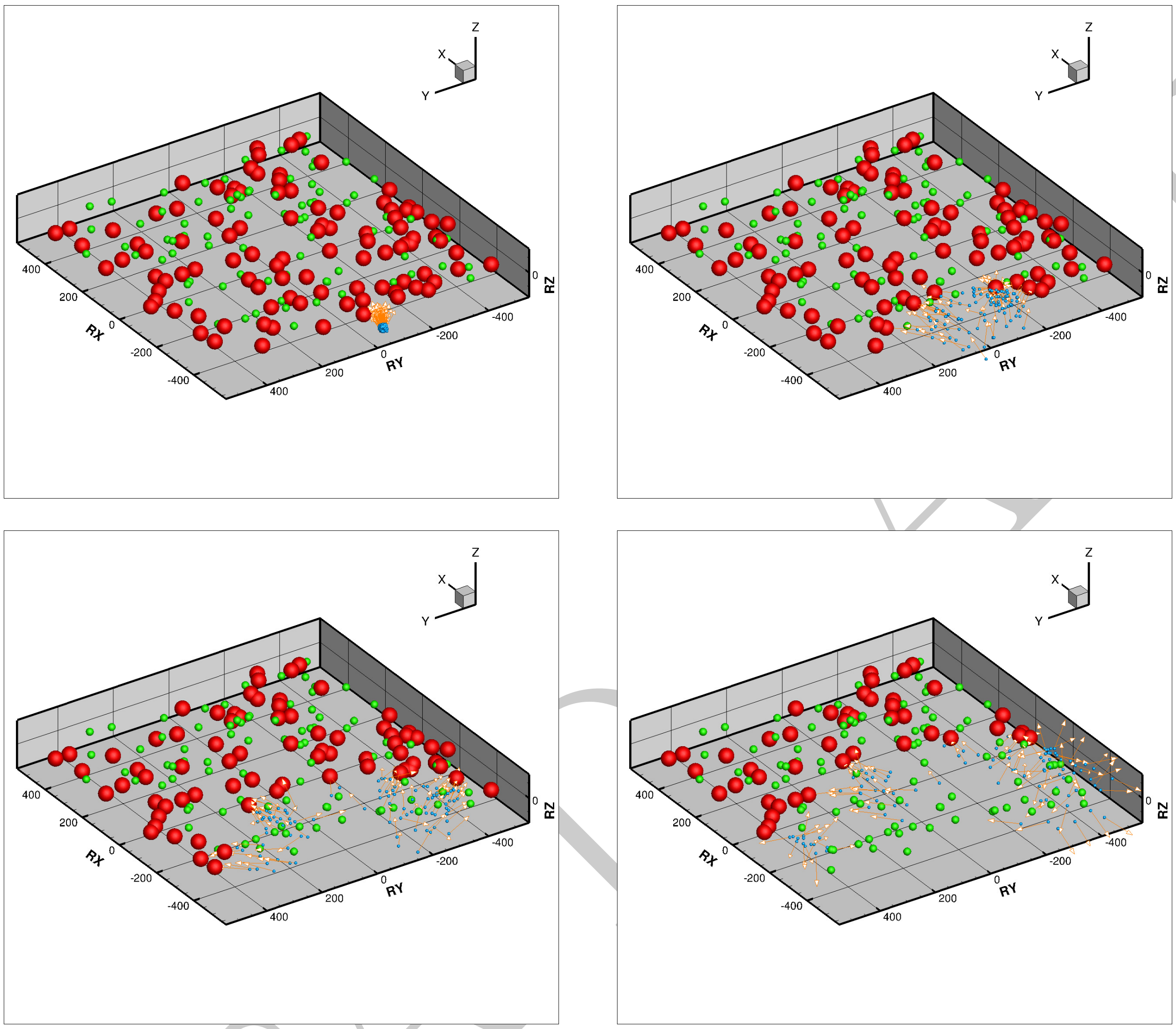


Figure 1.3: From left to right and top to bottom: sequences of mapping the model problem (green=obstacle, red=unmapped target, blue=swarm-member) The vectors on the swarm-members are the velocities.

9. The entire process is then repeated for the next time step.

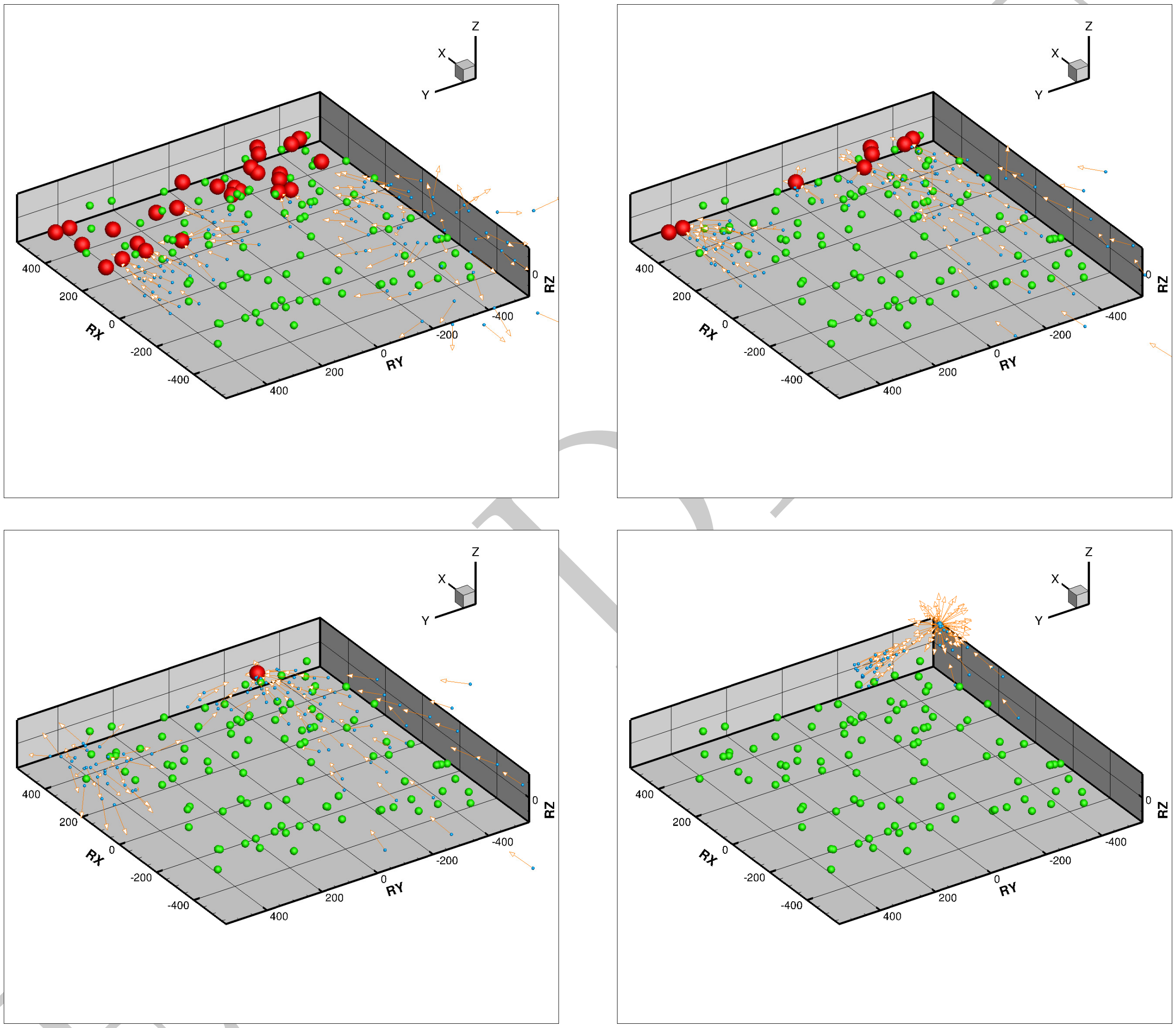


Figure 1.4: From left to right and top to bottom: sequences of mapping the model problem (green=obstacle, red=unmapped target, blue=swarm-member) The vectors on the swarm-members are the velocities.

2 Example

2.1 Preliminary numerical example

As a preliminary example, consider the following parameters:

- Mass = 10 kg,
- 100 swarm-members,
- 100 targets,
- 100 obstacles,
- $T = 30$ seconds,
- $\Delta t = 0.001$ seconds,
- Initial swarm velocity, $\boldsymbol{v}_i(t = 0) = \mathbf{0}$ m/s,
- Initial swarm domain (10,10,10) meters,
- Thrust force available by the UAV propulsion system, $F_i = 10^6$ Nt,
- Domain to be mapped (500,500,10) meters,
- Maximum velocity swarm-member $v_{max} = 100$ m/s.

The "design" vector of system parameter inputs

$$\boldsymbol{\Lambda}^i \overset{\text{def}}{=} \{\Lambda_1, \Lambda_2 ... \Lambda_N\} = \{W_{mt}, W_{mo}, W_{mm}, w_{t1}, w_{t2}, w_{o1}, w_{o2}, w_{m1}, w_{m2}, a_1, a_2, b_1, b_2, c_1, c_2\} \tag{2.1}$$

was given by a randomly generated vector

$$\boldsymbol{\Lambda}^i \overset{\text{def}}{=} \{7.96, 7.24, 8.97, 0.259, 0.587, 0.593, 0.831, 0.284, 0.845, 0.136, 0.529, 0.999, 0.764, 0.894, 0.636\}, \tag{2.2}$$

in the following intervals:

- Overall weights: $0 \leq W_{mt}, W_{mo}, W_{mm} \leq 10$,
- Target weights: $0 \leq w_{t1}, w_{t2} \leq 1$,
- Obstacle weights: $0 \leq w_{o1}, w_{o2} \leq 1$,
- Member weights: $0 \leq w_{m1}, w_{m2} \leq 1$ and
- Decay coefficients: $0 \leq a_1, a_2 \leq 1$, $0 \leq b_1, b_2 \leq 1$, $0 \leq c_1, c_2 \leq 1$.

We allowed a long enough time to map the whole domain (30 seconds). The results are shown in Figures 1.3 and 1.4. The sequences of mapping the model problem show initially green (unmapped) targets, which are marked as blue after they are mapped. The vectors on the swarm-members represent the velocities. The algorithm is quite adept in picking up missed targets by successive sweeps. We note that as the targets get mapped, they are dropped from the system, and the swarm-members naturally aggregate to the targets that are remaining. The decisions needed to deploy and operate such systems optimally, require guidance from real-time modeling and simulation that the preceeding model can provide, *if the parameters are known,* which is the topic of the next section.

2.2 Machine-learning for rapid UAV path planning

There are many parameters in the system, warranting the use a Machine Learning Algorithm. Here we follow Zohdi [27, 30, 31] in order to optimize behavior by minimizing a cost function. For example, let us consider minimizing the following cost function

$$\Pi(\boldsymbol{\Lambda}) = (w_1 A + w_2) T_m, \tag{2.3}$$

where A represents the percentage of targets remaining to mapped, $0 \leq T_m \leq 1$ represents the percentage time (normalized by the maximum simulation time) for this event to occur. In other words, the system is being driven to the parameters generating the best case scenario. The design vector of system parameters is:

$$\boldsymbol{\Lambda} = \{\Lambda_1, \Lambda_2 ... \Lambda_N\} = \{W_{mt}, W_{mo}, W_{mm}, w_{t1}, w_{t2}, w_{o1}, w_{o2}, w_{m1}, w_{m2}, a_1, a_2, b_1, b_2, c_1, c_2\}. \tag{2.4}$$

2.2.1 Algorithmic specifics

Following Zohdi [27, 30, 31], the algorithm is as follows:

- **STEP 1:** Randomly generate a population of S starting genetic strings, $\boldsymbol{\Lambda}^i, (i = 1, 2, 3, ..., S)$:
  $\boldsymbol{\Lambda}^i \stackrel{\text{def}}{=} \{\Lambda_1, \Lambda_2 ... \Lambda_N\} = \{W_{mt}, W_{mo}, W_{mm}, w_{t1}, w_{t2}, w_{o1}, w_{o2}, w_{m1}, w_{m2}, a_1, a_2, b_1, b_2, c_1, c_2\}$.
- **STEP 2:** Compute fitness of each string $\Pi(\boldsymbol{\Lambda}^i)$, (i=1, ..., S)
- **STEP 3:** Rank genetic strings: $\boldsymbol{\Lambda}^i$, (i=1, ..., S)
- **STEP 4:** Mate nearest pairs and produce two offspring, (i=1, ..., S)
  $\boldsymbol{\lambda}^i \stackrel{\text{def}}{=} \Phi^{(I)}\boldsymbol{\Lambda}^i + (1 - \Phi^{(I)})\boldsymbol{\Lambda}^{i+1}, \quad \boldsymbol{\lambda}^{i+1} \stackrel{\text{def}}{=} \Phi^{(II)}\boldsymbol{\Lambda}^i + (1 - \Phi^{(II)})\boldsymbol{\Lambda}^{i+1}$
- **STEP 5:** Eliminate the bottom $M < S$ strings and keep top $K < N$ parents and top K offspring (K offspring+K parents+M=S)
- **STEP 6:** Repeat STEPS 1-6 with top gene pool (K offspring and K parents), plus M new, randomly generated, strings
- **NOTE:** $\Phi^{(I)}$ and $\Phi^{(II)}$ are random numbers, such that $0 \leq \Phi^{(I)} \leq 1$, $0 \leq \Phi^{(II)} \leq 1$, which are different for each component of each genetic string
- **OPTION:** Rescale and restart search around best performing parameter set every few generations
- **REMARK:** The system parameter search is conducted within the constrained ranges of $\Lambda_1^{(-)} \leq \Lambda_1 \leq \Lambda_1^{(+)}$, $\Lambda_2^{(-)} \leq \Lambda_2 \leq \Lambda_2^{(+)}$ and $\Lambda_3^{(-)} \leq \Lambda_3 \leq \Lambda_3^{(+)}$, etc., etc. These upper and lower limits would, in general, be dictated by what is physically feasible.

2.3 Model problem

We applied the Machine Learning Algorithm and reduced the allowable simulation time from $T = 30$ seconds (as in the previous example) to $T = 10$ seconds, in order to make it difficult find parameter sets that deliver optimal performance. Shown are the best performing gene (design parameter set, in ***red***) as a function of successive generations, as well as the average performance of the population of the top four genes (designs, in ***green***). The design parameters were optimized over the following intervals:

- Overall weights: $0 \leq W_{mt}, W_{mo}, W_{mm} \leq 10$,
- Target weights: $0 \leq w_{t1}, w_{t2} \leq 1$,
- Obstacle weights: $0 \leq w_{o1}, w_{o2} \leq 1$,

- Member weights: $0 \leq w_{m1}, w_{m2} \leq 1$ and
- Decay coefficients: $0 \leq a_1, a_2 \leq 1$, $0 \leq b_1, b_2 \leq 1$, $0 \leq c_1, c_2 \leq 1$.

We used the following MLA settings:

- Population size per generation: 20,
- Number of parents to keep in each generation: 4,
- Number of children created in each generation: 4,
- Number of completely new genes created in each generation: 12,
- Number of generations for readaptation around a new search interval:10,
- The cost function weights in Equation 2.3, $w_1 = 1$ and $w_2 = 0.0001$ and
- Number of generations: 120.

The algorithm was automatically reset every *10 generations*. The entire 120 generation simulation, with 20 genes per evaluation (2400 total designs) took on the order of 4 minutes on a laptop, *making it ideal as a design tool.* Figure 2.1 (average of all genes performance and top gene performance) and Table 3.1 (values of the gene components) illustrate the results. The MLA/GA is able to home in of a variety of possible designs, including the one corresponding to the original set of parameters that generated the observations and alternatives that achieve virtually the same results. This allows system designers to more flexibility in parameter selection. We note that, for a given set of parameters, a complete simulation takes on the order of 0.1 seconds, thus over 36,000 parameter sets can be evaluated in an hour, *without even exploiting the inherent parallelism of the MLA.*

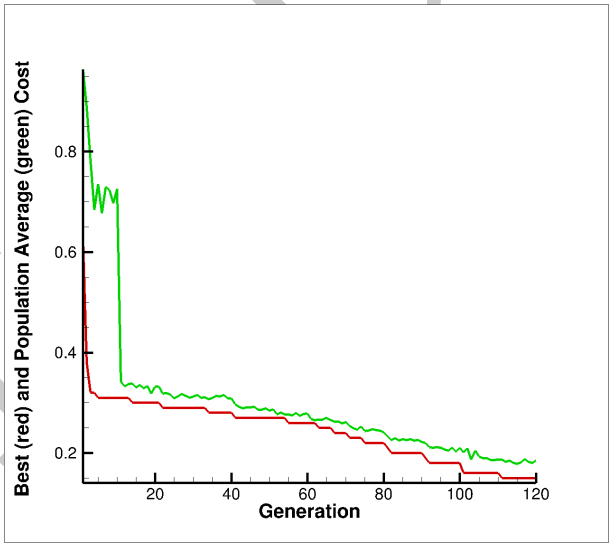


Figure 2.1: Machine learning output, generation after generation-the reduction of the cost function (Π) for the 15 parameter set is shown. This cost function represents the percentage of unmapped sites in the zone of interest. Shown are the best performing gene (***red***) as a function of successive generations, as well as the average performance of the population of genes (***green***).

Remark 1: If one wishes to have more detailed descriptions beyond a point mass model (for example a quadcopter), one must augment the balance of linear momentum ($\boldsymbol{G}_{cm,i}$)

$$\dot{\boldsymbol{G}}_{cm,i} = m_i \ddot{\boldsymbol{r}}_{cm,i} = \boldsymbol{\Psi}_i^{tot}, \tag{2.5}$$

#	Λ_1	Λ_2	Λ_3	Λ_4	Λ_5	Λ_6	Λ_7	Λ_8	Λ_9	Λ_{10}	Λ_{11}	Λ_{12}	Λ_{13}	Λ_{14}	Λ_{15}	Π
1	2.10	2.01	4.51	0.355	0.848	0.343	1.07	0.521	0.94	0.014	0.220	1.00	0.866	0.423	0.074	0.17
2	2.35	2.41	3.34	0.429	0.527	0.489	1.06	0.522	0.74	0.017	0.144	0.72	0.595	0.472	0.076	0.18
3	1.46	2.00	3.01	0.299	0.593	0.347	0.88	0.361	1.13	0.014	0.175	0.76	0.663	0.376	0.110	0.18
4	2.17	2.45	2.84	0.475	0.816	0.356	1.05	0.402	1.13	0.013	0.170	1.00	0.717	0.432	0.110	0.18
5	1.48	1.93	3.09	0.466	0.517	0.352	1.04	0.481	1.10	0.016	0.191	0.75	0.777	0.456	0.086	0.18
6	1.78	2.79	3.67	0.404	0.656	0.466	1.18	0.415	1.04	0.019	0.188	0.96	0.715	0.393	0.103	0.19
7	2.11	2.99	4.34	0.412	0.752	0.446	1.05	0.537	1.20	0.016	0.233	0.72	0.607	0.423	0.097	0.19
8	1.48	1.96	3.80	0.426	0.729	0.364	1.15	0.441	0.81	0.016	0.187	0.99	0.578	0.415	0.097	0.19
9	2.16	2.75	3.45	0.464	0.724	0.368	0.94	0.498	0.79	0.012	0.225	1.06	0.783	0.443	0.108	0.19
10	1.80	2.88	4.03	0.403	0.806	0.347	1.18	0.381	0.87	0.017	0.229	0.74	0.905	0.481	0.078	0.19

Table 2.1: The top 10 system parameter performers.

with a balance of angular momentum ($\boldsymbol{H}_{cm,i}$), given by

$$\dot{\boldsymbol{H}}_{cm,i} = \frac{d(\overline{\mathcal{I}}_i \cdot {}_i)}{dt} = \boldsymbol{M}^{tot}_{cm,i}, \tag{2.6}$$

where $\boldsymbol{M}^{tot}_{cm,i}$ is the total external moment about the center of mass, $\overline{\mathcal{I}}_i$ is the mass moment of inertia and ${}_i$ is the angular velocity. There are a number of numerical methods that are capable handling complex interaction of multiple vehicles, for example see Zohdi[2]. Another issue that was not taken into account are the details on the actuation and motor control that appear in the models as simply "attraction" and "repulsion". For example, for detailed modeling of the dynamics and control of UAVs we refer the reader to Mueller and D'Andrea [22, 23], Mueller et al. [24], Hehn et al [25], Houska et al [26] and Zohdi [2].
Remark 2: In many applications, the computed positions, velocities and accelerations of the members of a swarm, for example people or vehicles, must be translated into realizable movement. Furthermore, the communication latency and information exchange poses a significant technological hurdle. In practice, further sophistication, i.e. constraints on movement and communication, must be embedded into the computational model for the application at hand. However, the fundamental computational philosophy and modeling strategy should remain relatively unchanged.
Remark 3: One could reformulate the cost function to minimized energy usage, incorporating the range and performance of actual UAV's (see Taglibue etc al [28] and Holda et al [29].

2.4 Summary and extensions

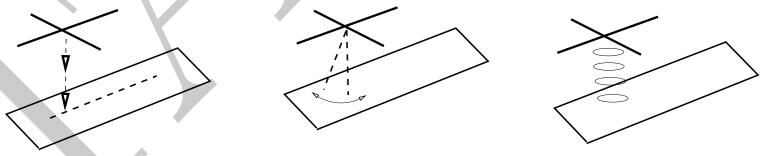


Figure 2.2: Various modes of Lidar scanning by an aircraft: linear, angular sweeps and flash-type.

A key for multi-UAV technology to flourish are efficient mapping algorithms. In this regard, agent-based algorithms are a viable approach. Agent-based paradigms for simulation of coupled complex systems have become powerful predictive tools. One of the main proposed uses of multi-UAV systems has been the deployment to help fight fires. Thus, in closing, we highlight some ongoing technological issues a that are being pursued in order make such systems a robust reality. In this regard, it is important to fully embrace mobile computing, rapid telecommunication and hyperspectral sensing in harsh environments from high- and/or lower-altitude UAVs, and ground vehicles with "humans in the loop". An important part of autonomy is energy autonomy, which is especially crucial in the performance of aerial vehicles and all deployed devices. The ability of such systems to adapt their behavior in response to energy consumption will allow them to perform longer. Algorithmic approaches to this problem are of particular interest, as they may often be retrofitted onto existing platforms or added to new platforms at negligible per-unit cost. For example, UAVs equipped with cameras can collect videos or images for pre-emptive strategic analysis. Since the swarm would include UAVs to enable visible inspection with the cameras, one can implement a deep learning approach for image/video pattern/feature recognition. In this regard, remote sensing has become an integral part of UAV-based imaging systems. In particular, Lidar (light detection and ranging) and time-of-flight signal processing have become key tools. Lidar usually uses light in the high-frequency ultraviolet, visible and near infrared spectrum (Ring [43], Cracknell and Hayes [44], Goyer and Watson [45], Medina et al [46] and Trickey et al [47]). It is classified as a "time-of-flight" type technology, utilizing a pulse of light and the time of travel to determine the relative distance of an object. Over the last 20 years, these devices have steadily improved and have become quite lightweight [48-54]. There are a variety of time-of-flight technologies that have been developed, primarily for military reasons, of which Radar, Sonar and Lidar are prime examples. The various types range from (1) conventional radar, (2) laser/radar altimeters, (3) ultrasound/sonar/seismograms, (4) radiometers and photometers-which measure emitted radiation, (5) hyperspectral cameras, where each pixel has a full spectrum and (6) geodetic-gravity detection, etc. For example, from satellites, the spatial resolution is on the order of pixel-sizes of 1-1000 meters using infrared wavelengths of 700-2100 nanometers. Hyperion-type cameras have even a broader range, 400-2500 nanometers with 200 bands(channels) and 100 nanometers per band. For example, thermographic/infrared cameras, form a heat-zone image (700nm-14000nm), however, the focusing lens cannot be glass, and are typically made of germanium or sapphire. These devices are fragile and require coatings, making them expensive. There are two main thermographic camera types: (a) cameras using cooled infrared detectors, which need specialized semiconductors, and have a relatively high resolution and (b) cameras using uncooled detectors, sensors and thermo-electronic resistance, which have relatively lower resolution. Furthermore, the initial image is monochrome, and must be color-mapped. Additionally, there are a variety of "corrective" measures (post-processors), such as (1) radiometric corrections, which correct the illumination for material properties, (2) topographic corrections, which correct the reflectivity due to shade, sunniness, etc. and (3) atmospheric corrections, which correct for atmospheric haze. There is a wide range of satellites available, such as Landsat, Nimbus (Weather), Radarsat, UARS (Civil, Research and Military), etc., which utilize these technologies. Typically, Lidar will employ thousands of narrow pulses per second to scan a domain, which can be time-consuming (Figure 2.2). Accordingly, wide-area flash pulse Lidar has started to become popular. In addition to their speed, flash-type cameras/scanners have some advantages because:

- The systems are simple, since they do not have moving parts associated with a scanner, and can thus be made very compact.
- The systems measure the entire surface in a single pulse, hence they are fast and can be used in real time and
- The systems do not require sophisticated post-processing units and are therefore inexpensive.

However, the greatest problems arise from multiple reflections of a pulse from a nonconvex surface, which can ruin time-of-flight calculations and other subsequent post processing. This is a subject of current investigation by the author, and we refer the reader to Zohdi [55] for more details.

3 Assignment

In this project, you will use machine learning / a genetic algorithm to find optimal control parameters for a swarm of autonomous vehicles tasked with mapping a set of targets quickly while avoiding collisions with obstacles and other vehicles. This problem could represent a real-world scenario like inspecting a disaster zone or construction site.

Problem 1: Theory-Based Exercises

Answer the following questions *prior* to coding the assignment to better understand the background physics and mathematics that govern the given models. You **may** solve these problems by hand **and/or** using computational tools such as `Python` etc. Please include all handwritten work and code used to solve each problem.

The template code to complete the project is given here on `GitHub`.

Problem 1.1

Analytically solve for the magnitude of the maximum possible velocity of the agents, in terms of *airspeed.* **Note that airspeed is the difference between ground speed and wind speed: $\boldsymbol{v}_{AS} = (\boldsymbol{v}_i - \boldsymbol{v}_a)$.**

Problem 1.2

Write down the Forward Euler equation for time discretization. Explain all the terms.

Problem 1.3

In one sentence, what would happen if any of the a, b, or c design variables became negative?

Problem 2: Coding Exercises

Use the given python notebook template to complete the following coding exercises.

Problem 2.1

Define the constants used in the simulation.

Problem 2.2

Run the genetic algorithm.

Problem 3: Analyzing Your Results

Answer the following questions about the code you created.

Problem 3.1

Report your best-performing 4 designs in a table similar to the following, but replacing $_i$ with the design variables specific to this project. Use `pandas DataFrame` to generate the table in cell `Problem 3.3`.

DESIGN	1	2	3	*etc*	Π
1					
2					
3					
4					

Table 3.1: The top 4 system parameter performers.

Problem 3.2

Provide a convergence plot showing the total cost of the best design, the mean of all parent designs, and the mean of the overall population for each generation. Do so by running cell `Problem 3.4/3.5`. Discuss any important observations.

Problem 3.3

Provide a plot showing the individual performance components (i.e., plot M^*, T^*, and L^*), for the overall best performer. Do so by running cell `Problem 3.4/3.5`. Discuss any important observations.

Table 3.2: Dynamics and Integration Parameters

Symbol	Type	Units	Value	Description
A_i	scalar	m^2	1	agent characteristic area
$C_{d,i}$	scalar	none	.25	agent coefficient of drag
m_i	scalar	kg	10	agent mass
$F_{p,i}$	scalar	N	200	Prop. force mag.
$\boldsymbol{v}_a$	3×1 vector	m/s	$\mathbf{0}$	Air velocity
ρ_a	scalar	kg/m^3	1.225	Air density
Δt	scalar	s	.2	Time step size
t_f	scalar	s	60	Maximum task time

Table 3.3: Objects and Interactions Parameters

Symbol	Type	Units	Value	Description
`agent_sight`	scalar	m	5	Target mapping distance
`crash_range`	scalar	m	2	agent collision distance
N_m	scalar	none	15	Number of initial agents
N_o	scalar	none	25	Number of obstacles
N_t	scalar	none	100	Number of initial targets

Table 3.4: Genetic Algorithm Parameters

Symbol	Type	Units	Value	Description
`children`	scalar	none	6	Strings generated by breeding
`parents`	scalar	none	6	Surviving strings for breeding
`S`	scalar	none	20	Designs per generation
`G`	scalar	none	100	Total generations
w_1	scalar	none	70	Weight of mapping in net cost
w_2	scalar	none	10	Weight of time usage in net cost
w_3	scalar	none	20	Weight of agent losses in net cost
`SB`	15×2	none	[0, 2]	Search bound interval for the drone path optimization

4 Solution

Problem 1: Theory-Based Exercises

Problem 1.1

The maximum possible velocity of the agents is limited by the drag force acting on the body:

$$\underbrace{\boldsymbol{\Psi}_i^{tot}}_{\text{net force}} = \underbrace{\boldsymbol{F}_{p,i}}_{\text{prop. force}} + \underbrace{\boldsymbol{F}_{d,i}}_{\text{drag force}} = 0 \tag{4.1}$$

$$\underbrace{\boldsymbol{F}_{d,i}}_{\text{drag force}} = - \underbrace{\boldsymbol{F}_{p,i}}_{\text{prop. force}} \tag{4.2}$$

$$\tfrac{1}{2}\rho_a C_{d,i} A_i \|\boldsymbol{v}_a - \boldsymbol{v}_i\|(\boldsymbol{v}_a - \boldsymbol{v}_i) = -\boldsymbol{F}_{p,i} \tag{4.3}$$

$$\|\boldsymbol{v}_i - \boldsymbol{v}_a\|(\boldsymbol{v}_i - \boldsymbol{v}_a) = \frac{2\boldsymbol{F}_{p,i}}{\rho_a C_{d,i} A_i} \tag{4.4}$$

$$\|\boldsymbol{v}_{AS}\|\boldsymbol{v}_{AS} = \frac{2\boldsymbol{F}_{p,i}}{\rho_a C_{d,i} A_i} \tag{4.5}$$

The magnitude of the max velocity is:

$$\|\boldsymbol{v}_{AS}\|^2 = \frac{2\|\boldsymbol{F}_{p,i}\|}{\rho_a C_{d,i} A_i} \tag{4.6}$$

$$\|\boldsymbol{v}_{AS}\| = \sqrt{\frac{2\|\boldsymbol{F}_{p,i}\|}{\rho_a C_{d,i} A_i}} \tag{4.7}$$

Problem 1.2

The forward Euler discretization for position and velocity terms are as follows:

$$\boldsymbol{v}_i(t+\Delta t) \doteq \boldsymbol{v}_i(t) + \boldsymbol{a}_i(t)\Delta t = \boldsymbol{v}_i(t) + \boldsymbol{\Psi}_i^{tot}(t)\tfrac{\Delta t}{m_i} \tag{4.8}$$

$$\boldsymbol{r}_i(t+\Delta t) \doteq \boldsymbol{r}_i(t) + \boldsymbol{v}_i(t)\Delta t \tag{4.9}$$

Problem 1.3

If any of the a, b, or c design variables became negative, the agents will interact with far away targets/agents/obstacles, rather than close ones.

Problem 2: Coding Exercises

Problem 2.1

Define the constants used in the simulation. Use the cell `Problem 2.1` in `Drones.ipynb` as your template.

Solution Cell Block:

```
# ################################### Problem 2.7
    ####################################################

## System Parameters
Nm = 15 # number of initial agents
No = 25 # number of obstacles
Nt = 100 # number of targets to map

locx = 100 # x bound of target/obstacle region
locy = 100 # y bound of target/obstacle region
locz = 10 # z bound of target/obstacle region

## Domain Parameters
xmax = 150 # x bound of domain
ymax = 150 # y bound of domain
zmax = 60 # z bound of domain


droneConsts = {

## System Parameters
'Nm' : Nm, # number of initial agents
'No' : No, # number of obstacles
'Nt' : Nt, # number of targets to map

## Physical Parameters
'Ai' : 1, # agent characteristic area (m^2)
'Cdi' : 0.25, # agent coefficient of drag
'mi' : 10, # agent mass (kg)
'va' : [-0.2, 0.2, 0.5], # Air velocity (m/s)
'ra' : 1.225, # Air Density (kg/m^3)
'Fp' : 200, # Propolsion force magnitude (N)

## Time Stepping Parameters
'dt' : 0.2, # time step size (s)
'tf' : 60, # Maximium task time (s)

## Object Interaction Parameters
'agent_sight' : 5, # maximum target mapping distance
'crash_range' : 2, # agent collision distance

'w1' : 70, # Weight of mapping in net cost
'w2' : 10, # Weight of time usage in net cost
'w3' : 20, # Weight of agent losses in net cost

## Domain Parameters
'xmax' : xmax, # x bound of domain
'ymax' : ymax, # y bound of domain
'zmax' : zmax, # z bound of domain

# Initial Obstacle Positions (m)
'obs' : np.hstack((uniform.rvs(-locx,2*locx,size = (No,1)),(uniform.rvs(-locy,2*locy,size =
    (No,1))),
                                                       (uniform.rvs(-locz,2*locz,size = (No
    ,1))))),

# Initial Target Positions (m)
'tar' : np.hstack((uniform.rvs(-locx,2*locx,size = (Nt,1)),(uniform.rvs(-locy,2*locy,size =
    (Nt,1))),
                                                       (uniform.rvs(-locz,2*locz,size = (Nt
    ,1))))),

# Initial Drone Positions (m)
'pos' : np.array([(xmax - 0.05*xmax)*np.ones(Nm), np.linspace(-ymax + 0.05*ymax, ymax -
    0.05*ymax, Nm),
          np.zeros(Nm)]).T,

'vel' : np.zeros([Nm,3]), # Initial Agent velocities (m/s)

}

# ## Genetic Algorithm Parameters
K = 6 # Strings generated by breeding
P = 6 # Surviving strings for breeding
S = 20 # Design strings per generation
G = 100 # Total Generations
SB = np.hstack((np.zeros((15,1)),2*np.ones((15,1)))) # Same search bounds for all design strings
```

Problem 2.2

Run the genetic algorithm.

Solution Cell Block:

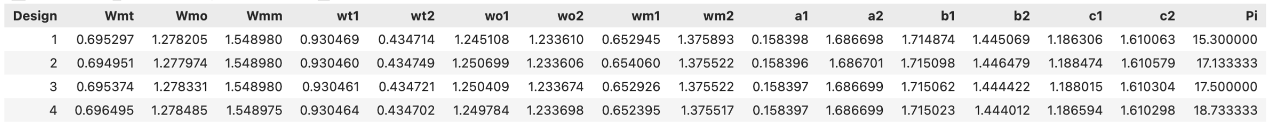
```
def myGA(S,G,P,SB,Func,consts):

    # Get number of design variables from size of search bound array
    dv = SB.shape[0]

    # set number of kids (K) equal to number of parents
    K = P

    # Initialize all variables to be saved
    Min = np.zeros(G) # Minimum cost for each generation
    PAve = np.zeros(G) # Parent average for each generation
    Ave = np.zeros(G) # Total population average for each generation
    lamHist = np.zeros((G,dv))  # history of best design string per generation


    Pi = np.zeros(S) # All costs in an individual generation

    # Initialize Lam array which will contain all of the design strings for all design variables
    Lam = np.zeros((S,dv))

    # Generate initial random population by looping through the number of design variables and building out the Lam array
    for i in range(dv):
        Lam[:,i] = uniform.rvs(SB[i,0],SB[i,1]-SB[i,0],size = S)

    # In first generation, calculate cost for all strings.
    # After, only calculate new strings since fitness for top P parents are already calculated
    # Initialize an index parameter to 0 to track which design string to start evaluating costs from
    start = 0

    for i in range(G): # Loop through generations

        # Calculate fitness of unknown design string costs
        for j in range(start,S): # Evaluate fitness of strings

            # Plug in design string control variables and array of function constants
            output = Func(Lam[j,:], consts) # Outputs dict of function outputs

            Pi[j] = output['Pi'] # Extract cost from dict of outputs and assign it to cost array


        # Sort cost and design strings based on performance
        ind = np.argsort(Pi)
        Pi = np.sort(Pi)
        Lam = Lam[ind,:]

```

```python
        # Save best design string for current generation
        lamHist[i,:] = Lam[0,:]

        # Generate offspring radnom parameters and indices for vectorized offspring calculation
        phi = np.random.rand(K,SB.shape[0]) # Generate random weights for offspring
        ind1 = range(0,K,2) # First set of children based on even numbered parameters
        ind2 = range(1,K,2) # Second set of children based on odd numbered parameters

        Parents = Lam[0:P,:] # Top P performing parents
        Children1 = phi[ind1,:]*Lam[ind1,:] + (1-phi[ind1,:])*Lam[ind2,:] # First set of children
        Children2 = phi[ind2,:]*Lam[ind2,:] + (1-phi[ind2,:])*Lam[ind1,:] # Second set of children

        # Initialize newPopulation array which will have S-P-K new random strings for all design variables
        newPop = np.zeros((S-P-K,dv))

        # Generate S - P - K new random strings by looping through the number of design variables and building out the new Population array
        for j in range(dv):
            newPop[:,j] = uniform.rvs(SB[j,0],SB[j,1]-SB[j,0],size = S-P-K)

         # Vertically stack parents, children, and new strings to use in next generation
        Lam = np.vstack((Parents, Children1, Children2, newPop))

        # Save minimum, parent average, and population average cost values for plotting
        Min[i] = Pi[0]
        PAve[i] = np.mean(Pi[0:P])
        Ave[i] = np.mean(Pi)

        # Update index parameter to P such that only new string cost values are calculated
        start = P

        # Print miminum value of cost for debugging (should monotonically decrease over generations)
        print("Best cost for generation " + str(i+1) + ": " + str(Min[i]))

    bestLam = Lam[0,:] # Extract best design string parameters afer all generations are run

    return(lamHist, Lam, bestLam, Pi, Min, PAve, Ave)
```

Problem 3: Analyzing Your Results

Problem 3.1

Example output table

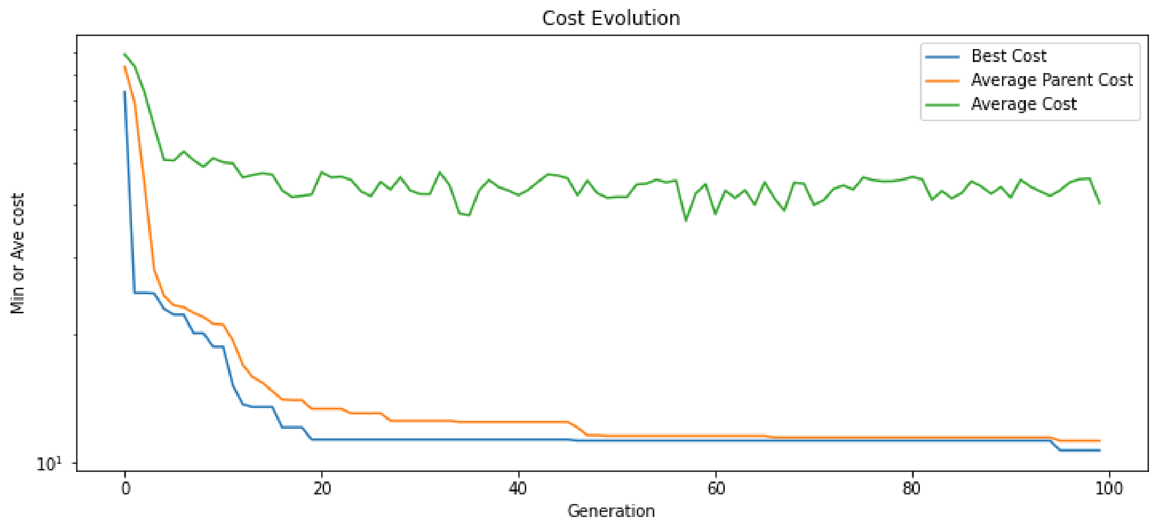


Design	Wmt	Wmo	Wmm	wt1	wt2	wo1	wo2	wm1	wm2	a1	a2	b1	b2	c1	c2	Pi
1	0.695297	1.278205	1.548980	0.930469	0.434714	1.245108	1.233610	0.652945	1.375893	0.158398	1.686698	1.714874	1.445069	1.186306	1.610063	15.300000
2	0.694951	1.277974	1.548980	0.930460	0.434749	1.250699	1.233606	0.654060	1.375522	0.158396	1.686701	1.715098	1.446479	1.188474	1.610579	17.133333
3	0.695374	1.278331	1.548980	0.930461	0.434721	1.250409	1.233674	0.652926	1.375522	0.158397	1.686699	1.715062	1.444422	1.188015	1.610304	17.500000
4	0.696495	1.278485	1.548975	0.930464	0.434702	1.249784	1.233698	0.652395	1.375517	0.158397	1.686699	1.715023	1.444012	1.186594	1.610298	18.733333

Problem 3.2

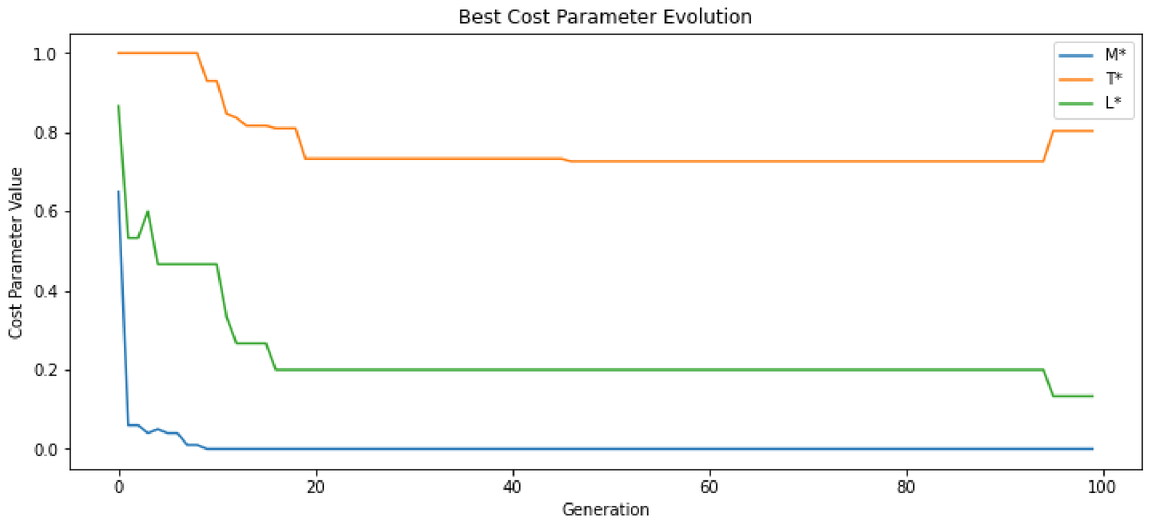
Example solution below

The convergence plot showing the total cost of the best design, the mean of all parent designs, and the mean of the overall population for each generation is as follows:



Problem 3.3

Plots showing the individual performance components (i.e., plot M^*, T^*, and L^*), for the overall best performer:



Findings:

- Best cost is monotonously decreasing
- Cost associated with mapping the target is minimized the most, while crashed agents is secondary and total time used is of tertiary importance, which is in line with the weights used in the total cost function
- GA effectively finds the best solution around G=20
- (*Accept answers that are in line with the specific plots generated per student basis*)

5 References

1. Zohdi, T. I. (2018). Multiple UAVs for Mapping: a review of basic modeling, simulation and applications. Annual Review of Environment and Resources. https://doi.org/10.1146/annurev-environ-102017-025912

2. Zohdi, T. I. (2017). On the dynamics and breakup of quadcopters using a discrete element method framework. *Computer Methods in Applied Mechanics and Engineering.* Volume 327, pp 503-521.

3. Breder, C. M. 1954. Equations descriptive of fish schools and other animal aggregations. *Ecology.* **35**. no. 3, pp 361-370.

4. Gazi, V. and Passino, K. M. 2002 . Stability analysis of swarms. *Proceedings of the American Control Conference. Anchorage, AK May 8-10.*

5. Bender, J. and Fenton, R. 1970. On the flow capacity of automated highways. *Transport Science.* **4**, pp. 52-63.

6. Kennedy, J. & Eberhart, R. 2001. *Swarm Intelligence.* Morgan Kaufmann Publishers.

7. Zohdi, T. I. (2003). Computational design of swarms. *The International Journal of Numerical Methods in Engineering.* **57**, 2205-2219.

8. Zohdi, T. I. (2009) Mechanistic modeling of swarms. *Computer Methods in Applied Mechanics and Engineering.* Volume 198, Issues 21-26, Pages 2039-2051.

9. Zohdi, T. I. (2017) An agent-based computational framework for simulation of competing hostile planet-wide populations. Computer Methods in Applied Mechanics and Engineering. doi:10.1016/j.cma.2016.04.032

10. Beni, G. 1988. The concept of cellular robotic system. In *IEEE International Symposium on Intelligent Control*, pages 57-62.

11. Brooks, R. A. 1991. Intelligence without reason. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595.

12. Dudek, G., Jenkin, M., Milios, E. , and Wilkes, D. 1996. A taxonomy for multi-agent robotics. *Autonomous Robots*, 3:375–397.

13. Cao, Y. U., Fukunaga, A. S., and Kahng, A.. 1997. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–27.

14. Liu, Y. and Passino, K. M. 2000. Swarm intelligence: Literature overview. Technical report, Ohio State University.

15. Turpin, M., Michael, N. and Kumar, V. (2014) Capt: Concurrent assignment and planning of trajectories for multiple robots," in International Journal of Robotics Research, Jan. 2014. [https://journals.sagepub.com/doi/1

16. Bonabeau, E., Dorigo, M. and Theraulaz, G. 1999. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York.

17. Dorigo, M., Maniezzo, V., and Colorni, A. 1996. Ant system: optimization by a colony of cooperating agents. *Systems, Man and Cybernetics, Part B, IEEE Transactions on*, 26(1):29–41.

18. Bonabeau, E. and Meyer, C. 2001. Swarm intelligence: A whole new way to think about business. *Harvard Business Review*, 79(5):106–114.

19. Fiorelli, E., Leonard, N. E., Bhatta, P., Paley, D., Bachmayer, R., and Fratantoni, D. M. 2004. Multi-auv control and adaptive sampling in monterey bay. In *Autonomous Underwater Vehicles, 2004 IEEE/OES*, pages 134–147.

20. Feder, T. 2007. Statistical Physics is for the Birds. *Physics Today* p28-29.

21. Ballerini, M., Cabibbo, N., Candelier, R., Cavagna, A., Cisbani, E., Giardina, I., Lecomte, V., Orlandi, A., Parisi, G., Procaccini, A., Viale, M. and Zdravkovic, V. 2008. Interaction ruling animal collective behavior depends on topological rather than metric distance: evidence from a field study. PNAS, vol. 105, no. 4, 1232-1237

22. Mueller, M. W. and D'Andrea, R. (2014) Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. *IEEE International Conference on Robotics and Automation (ICRA), 2014.*

23. Mueller, M. W. and D'Andrea, R. (2015) Relaxed hover solutions for multicopters: application to algorithmic redundancy and novel vehicles. *International Journal of Robotics Research* Volume 35, Number 8, 873-889.

24. Mueller, M. W., Hehn, M. and D'Andrea, R. (2015) A computationally efficient motion primitive for quadrocopter trajectory generation, IEEE Transactions on Robotics, Volume 31, no.8, pages 1294-1310.

25. Hehn, M., Ritz, R., & D Andrea, R. (2012) Performance benchmarking of quadrotor systems using time-optimal control. *Autonomous Robots* Volume 33(1-2), 69-88.

26. Houska, B., Ferreau, H., and Diehl, M. (2011) ACADO Toolkit: An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods* Volume 32, Number 3, 298-312.

27. Zohdi, T. I. (2003). Genetic design of solids possessing a random-particulate microstructure. *Philosophical Transactions of the Royal Society: Mathematical, Physical and Engineering Sciences.* Vol: 361, No: 1806, 1021-1043.

28. Tagliabue, A., Wu, X., and Mueller, M. W.(2018). Model-free Online Motion Adaptation for Optimal Range and Endurance of Multicopters, in IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2019

29. Holda, C., Ghalamchi, B., and M. W. Mueller, Tilting multicopter rotors for increased power efficiency and yaw authority, in International Conference on Unmanned Aerial Systems (ICUAS), IEEE, 2018, pp. 143–148.

30. Zohdi, T. I. (2017). Dynamic thermomechanical modeling and simulation of the design of rapid free-form 3D printing processes with evolutionary machine learning. Computer Methods in Applied Mechanics and Engineering. https://doi.org/10.1016/j.cma.2017.11.030

31. Zohdi, T. I. (2018). Electrodynamic machine-learning-enhanced fault-tolerance of robotic free-form printing of complex mixtures. Computational Mechanics. https://doi.org/10.1007/s00466-018-1629-y

32. Holland, J. H. 1975. *Adaptation in natural & artificial systems.* Ann Arbor, Mich. University of Michigan Press.

33. Goldberg, D. E. 1989. *Genetic algorithms in search, optimization & machine learning.* Addison-Wesley.

34. Davis, L. 1991. *Handbook of Genetic Algorithms.* Thompson Computer Press.

35. Onwubiko, C. 2000 *Introduction to engineering design optimization.* Prentice Hall.

36. Goldberg, D. E. & Deb, K. 2000. Special issue on Genetic Algorithms. *Computer Methods in Applied Mechanics & Engineering.* 186 (2-4) 121-124.

37. Lagaros, N., Papadrakakis, M. and Kokossalakis, G. (2002). Structural optimization using evolutionary algorithms. *Computers & Structures.* 80, 571-589.

38. Papadrakakis, M., Lagaros, N., Thierauf, G. and Cai, J. (1998a). Advanced solution methods in structural optimisation using evolution strategies, *Engineering Computational Journal.* **15** (1), 12-34.

39. Papadrakakis, M., Lagaros, N. and Tsompanakis, Y. (1998b). Structural optimization using evolution strategies and neutral networks, *Computer Methods in Applied Mechanics and Engineering.* **156**, (1), 309-335.

40. Papadrakakis, M., Lagaros, N. and Tsompanakis, Y. (1999a). Optimization of large-scale 3D trusses using Evolution Strategies and Neural Networks. *Int. J. Space Structures.* **14** (3), 211-223.

41. Papadrakakis, M., Tsompanakis, J. and Lagaros, N. (1999b). Structural shape optimisation using evolution strategies. *Eng. Optimization.* **31**, 515-540.

42. Goldberg, D. E. & Deb, K. 2000. Special issue on Genetic Algorithms. *Computer Methods in Applied Mechanics & Engineering.* 186 (2-4) 121-124.

43. Ring, J. (1963) The Laser in Astronomy. p. 672-3, New Scientist.

44. Cracknell, A. P. and Hayes, L. (2007). Introduction to Remote Sensing (2 ed.). London: Taylor and Francis. ISBN 0-8493-9255-1. OCLC 70765252.

45. Goyer, G. G. and Watson, R. (1963). The Laser and its Application to Meteorology. Bulletin of the American Meteorological Society. 44 (9): 564–575 [568].

46. Medina, A., Gaya, F., and Pozo, F. (2006). Compact laser radar and three-dimensional camera. 23. J. Opt. Soc. Am. A. pp. 800–805

47. Trickey, E. Church, P. and Cao, X. (2013) Characterization of the OPAL obscurant penetrating LiDAR in various degraded visual environments Proc. SPIE 8737, Degraded Visual Environments: Enhanced, Synthetic, and External Vision Solutions 2013, 87370E (16 May 2013); doi: 10.1117/12.2015259

48. Hansard, M., Lee, S., Choi, O.& Horaud, R. (2012). Time-of-flight cameras: Principles, Methods and Applications. SpringerBriefs in Computer Science. doi:10.1007/978-1-4471-4658-2. ISBN 978-1-4471-4657-5.

49. Schuon, S., Theobalt, C., Davis, J. & Thrun, S. (2008). High-quality scanning using time-of-flight depth superresolution. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008. Institute of Electrical and Electronics Engineers. pp. 1–7.

50. Gokturk, S. B., Yalcin, H. & Bamji, C. (2005). A Time-Of-Flight Depth Sensor - System Description, Issues and Solutions. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2004. Institute of Electrical and Electronics Engineers: 35–45. doi:10.1109/CVPR.2004.291.

51. ASC's 3D Flash LIDAR camera selected for OSIRIS-REx asteroid mission. NASASpaceFlight.com. 2012-05-13.

52. Aue, Jan; Langer, Dirk; Muller-Bessler, Bernhard; Huhnke, Burkhard (2011-06-09). Efficient segmentation of 3D LIDAR point clouds handling partial occlusion. 2011 IEEE Intelligent Vehicles Symposium (IV). Baden-Baden, Germany: IEEE. doi:10.1109/ivs.2011.5940442. ISBN 978-1-4577-0890-9.

53. Hsu, S., Acharya, S., Rafii, A. & New, R. (2006). Performance of a Time-of-Flight Range Camera for Intelligent Vehicle Safety Applications. Advanced Microsystems for Automotive Applications 2006. VDI-Buch. Springer: 205–219. doi:10.1007/3-540-33410-6-16. ISBN 978-3-540-33410-1. Archived from the original (pdf) on 2006-12-06. Retrieved 2018-06-25.

54. Elkhalili, O., Schrey, O. M., Ulfig, W. Brockherde, W. & Hosticka B. J. (2006), A 64x8 pixel 3-D CMOS time-of flight image sensor for car safety applications. European Solid State Circuits Conference 2006, pp. 568–571, doi:10.1109/ESSCIR.2006.307488, ISBN 978-1-4244-0302-8, retrieved 2010-03-05

55. Zohdi, T. I. (2019). Rapid simulation-based uncertainty quantification of flash-type time-of-flight and Lidar-based body-scanning processes. Computer Methods in Applied Mechanics and Engineering. $https://doi.org/10.1016/j.cma.2019.03.056$