Game Theory-based Parameter Tuning for Energy-Efficient Path Planning on Modern UAVs

DIKSHA MOOLCHANDANI, School of Information Technology, IIT Delhi, India
KISHORE YADAV, Computer Science and Engg., IIT Delhi, India
GEESARA PRATHAP, Department of Computer Science, Innopolis University, Russia
ILYA AFANASYEV, Huawei Technologies Co. Ltd., St. Petersburg, Russia
ANSHUL KUMAR, Computer Science and Engg., IIT Delhi, India
MANUEL MAZZARA, Department of Computer Science, Innopolis University, Russia
SMRUTI R. SARANGI, Computer Science and Engg. (joint appt. with Electrical Engg.), IIT Delhi, India

Present-day path planning algorithms for UAVs rely on various parameters that need to be tuned at runtime to be able to plan the best possible route. For example, for a sampling-based algorithm, the number of samples plays a crucial role. The dimension of the space that is being searched to plan the path, the minimum distance for extending a path in a direction, and the minimum distance that the drone should maintain with respect to obstacles while traversing the planned path are all important variables. Along with this, we have a choice of vision algorithms, their parameters, and platforms.

Finding a suitable configuration for all these parameters at runtime is very challenging because we need to solve a complicated optimization problem, and that too within tens of milliseconds. The area of theoretical exploration of the optimization problems that arise in such settings is dominated by traditional approaches that use regular nonlinear optimization techniques often enhanced with AI-based techniques such as genetic algorithms. These techniques are sadly rather slow, have convergence issues, and are typically not suitable for use at runtime. In this paper, we leverage recent and promising research results that propose to solve complex optimization problems by converting them into approximately equivalent game-theoretic problems. The computed equilibrium strategies can then be mapped to the optimal values of the tunable parameters. With simulation studies in virtual worlds, we show that our solutions are $5-21\%$ better than those produced by traditional methods, and our approach is $10\times$ faster.

CCS Concepts: • **Computing methodologies** → **Evolutionary robotics**; *Modeling methodologies.*

Additional Key Words and Phrases: game theory, parameter tuning, path planning, power model, UAVs

ACM Reference Format:
Diksha Moolchandani, Kishore Yadav, Geesara Prathap, Ilya Afanasyev, Anshul Kumar, Manuel Mazzara, and Smruti R. Sarangi. 2022. Game Theory-based Parameter Tuning for Energy-Efficient Path Planning on Modern UAVs.

1 INTRODUCTION

The [1] global commercial market for UAVs has seen a continued upsurge for the past decade due to their growing demand for a multitude of applications such as surveillance, search and rescue, aerial photography, smart agriculture, surveying, and construction. The UAV market that was worth

[1]This paper is an extension of our conference paper "Game Theory-based Parameter Tuning for Path Planning of UAVs" published in VLSI Design 2021 [28]

Authors' addresses: Diksha Moolchandani, diksha.moolchandani@cse.iitd.ac.in, School of Information Technology, IIT Delhi, India; Kishore Yadav, Computer Science and Engg., IIT Delhi, India; Geesara Prathap, Department of Computer Science, Innopolis University, Russia; Ilya Afanasyev, Huawei Technologies Co. Ltd., St. Petersburg, Russia; Anshul Kumar, Computer Science and Engg., IIT Delhi, India; Manuel Mazzara, Department of Computer Science, Innopolis University, Russia; Smruti R. Sarangi, Computer Science and Engg. (joint appt. with Electrical Engg.), IIT Delhi, India.

.

$20.8 billion USD in 2021 is expected to reach $501.4 billion USD by 2028 with a CAGR (Compound Annual Growth Rate) of 57.5% [34]. According to the U.S. Federal Aviation Administration (FAA), the number of UAVs registered as of 2019 was 1.1 million and this number was expected to exceed 4 million by 2021 [14].

Unfortunately, the software and computing aspects of a UAV have not been given adequate importance in the literature. For example, consider path planning algorithms for UAVs. In general, path planning is the most time-consuming step in a RRT* based path planner [20]. Boroujerdian et al. [4] have shown that with the wrong choice of the algorithm or its parameters, it is possible to compute paths that take four times longer to traverse. This is a wastage of time as well as battery power. Though there have been a lot of advances in path planning algorithms, their behavior is mostly governed by a number of parameters that need to be set based on runtime conditions. Hence, tuning a path planning algorithm is disproportionately important in UAV design, especially when we need to codesign it along with the overall computer vision system. The current day autonomous systems are assisted by computer vision techniques that run simultaneously with these path planning algorithms or as standalone applications (e.g. drones for smart agriculture). Hence, it becomes necessary to consider both path planning and computer vision algorithms in unison to be able to model the effect of one on another. This kind of modeling leads to even more tunable knobs and hence a larger search space. Moreover, such a form of parameter tuning needs to be done at runtime to be able to dynamically optimize for energy and performance.

The state-of-the-art techniques [21, 37] use a combination of classical optimization and AI enhanced algorithms to find the right set of these parameters. Though the results obtained are optimal, there are several problems with these approaches [31]: ❶ the computational complexity increases with the number of parameters and their possible values, ❷ these techniques take a prohibitive amount of time to converge in case of high-dimensional non-convex problems, ❸ they are not necessarily globally optimal, and ❹ the presence of an exponential number of local minima leads to slow convergence. These problems render the classical optimization approaches unsuitable for real-time applications.

To solve such optimization problems, fast yet approximate solution techniques such as genetic algorithms, Coral Reefs Optimization and Particle Swarm Optimization (PSO) methods are used [37]. Even the PSO algorithm is reported to have some convergence issues [21]. Hence, recent papers on resource allocation have used AI algorithms such as the Coral Reefs algorithm aided with *game-theoretic techniques* to speed up convergence. Game theory is also increasingly being used as a standalone technique for quickly solving optimization problems by converting them to roughly equivalent games. For example, Javarone et al. [18] solved the traveling salesman problem using a public goods game, and Xu et al. [41] solved the electromagnetic buffer optimization problem using the Nash equilibrium solution of the corresponding game-theoretic formulation.

The basic idea is to convert optimization problems to game-theoretic problems and find the Nash Equilibrium, which can then be used to derive a near optimal solution for the original problem. To the best of our knowledge, we are the first to solve the parameter tuning problem for the path planning of drones using game theory. Given that the path planning step is the *most time-consuming step*, we need efficient methods to solve the parameter tuning problems in real-time. The parameters also include the choice of the computer vision system (both hardware and software). We propose a generic framework that can be augmented with a variety of other knobs as well.

The UAV parameter tuning problem is not a traditional game – there are no well-defined players and payoffs. We use recent results in this area [8, 9, 26]. We first formulate a traditional optimization problem that finds the most suitable configuration of all the parameters (from the constrained parameter search space) such that the energy (in hovering, traveling sub-optimal paths, and running the vision algorithm) is minimized. We convert this optimization problem to a game theory-based

problem by converting the parameters to players and the objective function is accommodated within the payoffs of all the players. A part of the payoffs is also directly obtained from various sensitivity analyses results. We perform real experiments with drones as well as exhaustive simulations in virtual worlds. We convincingly show that a modest optimization problem with six tunable parameters (characterize the path planning phase and the vision system), each having a finite feasible range, cannot be solved in real-time and often does not converge.

In the game-theoretic version, we need to find the Nash equilibrium values of the parameters. We observe that solving for the Nash Equilibrium is 10× faster as compared to solving the basic optimization problem. Moreover, our solutions using game theory are at par with those obtained by solving the old-school optimization problem. Given that solution techniques for complex optimization problems are also approximate, our solutions are either comparable (for parameter tuning of vision-assisted path planning) or $5-21\%$ better (for parameter tuning of path planning algorithm). This implies that we achieve smaller (or equal) path lengths and hover times using the game-theoretic framework.

The novel contributions of this paper are as follows.

(1) We provide a comprehensive empirical power model for UAVs that is easily generalizable to UAVs of different weights and use it to formulate an optimization problem that minimizes the overall energy consumption of the UAVs during their flight. This includes the energy consumed in hovering, traveling a distance, and running the computer vision algorithm.
(2) We propose a game-theoretic formulation corresponding to the optimization problem formed in the previous step, where the tunable parameters become the players and the objective function forms the basis of the payoff functions of these players. We define novel payoff functions to incorporate the selfish and altruistic objectives of the players. We also show that both are needed.
(3) We show that our game-theoretic approach is 10× faster and provides solutions that are either comparable or $5-21\%$ better than the best optimization based approaches for three different virtual worlds. Our solutions converge in all cases.
(4) We show that the execution time to solve a game theory problem on Beagleboard is $0.05s$ for a 5-player game, which is well within the real time constraints ($100ms$) for these systems. The execution time for solving the optimization problem is prohibitive, it takes $0.4-0.9s$ to solve using the IPOPT solver on Beagleboard.

We shall describe the relevant background in Section 2, describe related work in Section 3, develop the power model in Section 4, explain the experimental setup in Section 5, formulate an optimization problem in Section 6, and show the formulation of the game in Section 7. We shall discuss the results in Section 8, establish theoretical equivalence of optimization problem and game theory-based approach in Section 9, and finally conclude in Section 10.

2 BACKGROUND

2.1 Navigation in UAVs

The problem of navigation in UAVs has been solved by both classical computational geometric methods and end-to-end learning-based methods. However, for the case of UAVs, the classical methods are still the most popular choice [4, 5] because of their simplicity and deterministic approach. These methods follow three basic steps: ❶ **Perception:** building a 3D view of the surroundings, and extracting information in the form of an obstacle/occupancy map, ❷ **Planning:** using the information about the obstacles from the Perception step to create a collision-free path, and ❸ **Control:** sending the control commands to the UAV to follow the planned path. This is referred to as the Perception, Planning, and Control (PPC) paradigm.

We focus on the path planning step because it is the most time consuming step in the entire pipeline (roughly 65% [20]). Yang et al. [42] suggested that among all the path planning algorithms with bounded time complexity, sampling-based algorithms are fast, self-sufficient, and provide good solutions respecting real-time constraints. We choose the most popular sampling based algorithm, RRT*, for this work, which is known to provide near-optimal solutions and allows dynamic re-planning.

The **RRT*** algorithm builds a path from the source to the destination in the form of a tree. To grow the tree, the algorithm first *samples* the environment and chooses a random point (p_{rand}). Subsequently, the node in the tree, A, that is the closest to p_{rand} attempts to create a new node in the direction of p_{rand}. We assume a step size or *resolution* (R) in the algorithm that restricts the maximum distance (from A) at which the new node can be placed. Subsequently, a new node B is placed R units away from A in the direction of p_{rand}. Node B is added to the vertex set of the tree if the direct path from A to B is free of obstacles. After B is added to the vertex set, it needs to connect to some vertices via edges to become a part of the tree. Instead of directly creating an edge from A to B, we create edges using the notion of a *cost function*. Each vertex in the tree has an associated cost that quantifies the cost of reaching that vertex from the start (root) node. To connect B, all the nodes within a radius r are checked. If any node C from this neighborhood has a path to B that is of lower cost as compared to the cost of the path from A to B, then an edge is created between C and B in place of the edge between A and B. Needless to say, the edge between C and B should be free of obstacles. The minimum distance between an edge and an obstacle should be at least equal to the *obstacle avoidance distance* – this avoids collisions even if there is a slight amount of nondeterminism in the UAV's position.

The number of samples, the step-size, obstacle avoidance distance, and the dimensions of the environment form the tunable parameters of the RRT* algorithm, which determine the behavior of the algorithm. The obstacle density also plays an important role in deciding the length of the planned path and the time taken to plan the path. In this work, we perform experiments to identify this behavior and develop a game-theoretic framework to model this behavior at runtime.

2.2 Game Theory Preliminaries

In a game-theoretic system, there are multiple selfish yet rational players. Each player has a strategy, which it plays to maximize its chances of winning the game. The notion of winning the game is captured by the payoff or utility that a player derives by *playing* a certain strategy. Thus, for a combination of strategies across the players, each competing player obtains a payoff.

There is no notion of optimality here, because fundamentally the players are at odds with each other. Hence, we define the notion of a *Nash equilibrium* instead, where no player can increase its payoff by unilaterally changing its strategy (the rest of the strategies remaining the same). The notion of a *Nash equilibrium* is very useful in describing the results of games, and it is often possible to derive profound insights about the inherent trade-offs and feasible solutions. A Nash equilibrium is said to be *stable* if a small change in the strategy for any player makes it strictly worse off. The strategies should be independent, implying that the players can independently choose their strategies regardless of the strategy of other players.

2.3 Relating Optimization Problems to Game Theory

In general, optimization problems take an unpredictable amount of time to converge to a solution especially when the constraints are non-linear in nature. Most of the time, they get stuck in local minima or return infeasible solutions. Hence, instead of solving these problems exactly, they are typically solved using approximate optimization techniques. There are many such approximate techniques that are much faster than classic optimization techniques such as genetic algorithms,

particle swarm optimization, ant colony optimization, and the coral reefs algorithm. Game theory is a newly applied technique in this space that can be used to solve the optimization problem approximately and very quickly [37][2]. In many recent papers, game theory has been used to improve the convergence speed of the classical and approximate optimization techniques such as tuning the weighting parameters for the particle motion in a particle swarm optimization [21]. Xu et al. [41] used game theory for solving the electromagnetic buffer optimization problem. The optimization problem was converted to a game based on several sensitivity analyses experiments. To the best of our knowledge, we are not aware of any automatic algorithm that takes an arbitrary optimization problem and converts it to a game. However, a lot of optimization problems can be converted to a game by ingeniously creating players and their associated payoffs. The computed Nash equilibrium can be mapped to the solution of the original optimization problem. We follow a similar approach.

3 RELATED WORK

3.1 Parameter Tuning

There are multiple proposals that target the problem of parameter tuning for path planning algorithms by formulating an optimization problem that aims to optimize a cost metric such as the hover time, planning time, or path length.

Luo et al. [23] and Dunlap et al. [13] studied the effect of tuning the parameters that determine the path length. They showed that the relationship of the path length with these parameters has a very complicated form. Cano et al. [7] formulated the optimization problem as a cost minimization problem. The aim was to find a parameter combination that provides a valid trajectory using the path planning algorithm and minimizes the planning time. Since the time required for parameter exploration and tuning is large, they employed four intelligent search space exploration techniques based on random sampling, random forests, Bayesian Optimization, and AUC Bandit (AI algorithm). Similarly, Burger et al. [6] solved the optimization problem using the SMAC [17] tool. The tool internally uses random forests to explore the parameter space. Due to the time consuming nature of these techniques, they set a cut-off time for the exploration. In contrast, our game-theoretic formulation provides theoretical guarantees about the solution and is far more time-efficient.

3.2 Power Modeling

There can be two approaches for modeling the power consumption of drones: analytical or ML-based. For building an analytical model, an in-depth knowledge of the relationship of energy consumption with the vehicle dynamics is needed. Liu et al. [22] proposed one such model. For an ML-based approach, the basic idea is to collect the power consumption behavior of the drone for different types of motions along with different configurations of the kinematic parameters such as velocity, distance, height, acceleration, and deceleration. Subsequently, a regression model can be fitted to the collected data to obtain a power model. One of the most recent works by Prasetia et al. [33] proposed three regression-based energy models for horizontal motion, vertical motion, and hovering, respectively. Their energy models for the horizontal and vertical motion relied on the respective velocities and distances, while the energy model for the hovering motion relied on the hover duration. Their model is not generic as it is designed from the data collected on one drone, relies on a finite number of fixed mission commands for the separation of motions to three types: horizontal, vertical, and hovering, while in practice the drone can have complex motions too that are a combination of the basic motions. Additionally, there is no clear partition of the motion types in a practical random drone flight.

[2]Note that classical game theory was developed in the late 1940s

Another comprehensive power model was proposed by Abeywickrama et al. [2] that models the power for all the different types of drone motions such as take-off, hovering, horizontal, and vertical movements, impact of the payload, and the effect of wind. However, this model does not take into account the effect of different horizontal velocities, and does not generalize to drones of different weights. Franco et al. [12] proposed an energy model that models the energy for all the motions as captured by Abeywickrama et al. [2], with the exception of the take-off movement. However, the developed models did not take into account the effect of height in both the vertically upward and the hovering movements. Similar empirical models were proposed by Ji et al. [19] and Maekawa et al. [24], however they do not consider all the kinds of drone motions.

We, on the other hand, **develop a comprehensive model** for all the types of motions that are encountered in a drone flight and also take into account the effect of height, velocity, acceleration, and the weight of the drone. This makes our model far more generic and accurate.

4 MODELING THE ENERGY CONSUMPTION OF UAVS

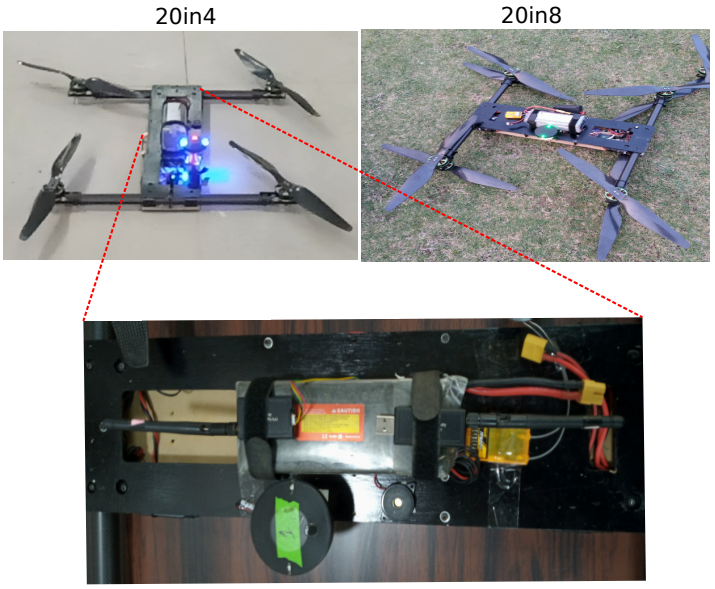

Fig. 1. Drones used in our experiments

4.1 Experimental Setup

We conducted experiments on three drones manufactured by BotLab Dynamics (two of them are shown in Figure 1): 20in4, 20in8, and 18in4 drones. The 18in4 drone consists of four 18-inch propellers, the 20in4 drone consists of four 20-inch propellers, while the 20in8 drone contains eight 20-inch propellers. Four propellers are mounted on one side of the motor and the other four are mounted on the opposite side (see Figure 1). The weights of the 18in4, 20in4 and 20in8 drones are 1.5 Kg, 1.5 Kg and 2.7 Kg without batteries, respectively. The weight of the battery is 1.2 Kg.

4.2 Data Collection

We use the power module circuit (as shown in Figure 2) consisting of voltage and current sensors to get the readings. The two ends of the circuit are connected to the battery and the UAV, respectively. The sensed data is read by the Pixhawk-2.4.8 flight controller via the power slot. The logged data

is then sent to the ground station via the telemetry module operating at 433 MHz. We use the ArduPilot firmware [38] and the Mission Planner software [39] to design different mission plans.

The collected data contains a lot of information corresponding to different components such as the battery, barometer, IMUs, and the position control data. We extract the voltage and current values from the log messages corresponding to the battery (BAT) module, the altitude (Z) from the log messages of the barometer (BARO) module, the horizontal distance, velocity and acceleration ($d_x, d_y, v_x, v_y, a_x, a_y$) from the log messages of the position control data (PSC) module. The sampling interval of the data logger is 100 ms. We calculate the instantaneous power using the standard equation $P = V \times I$.

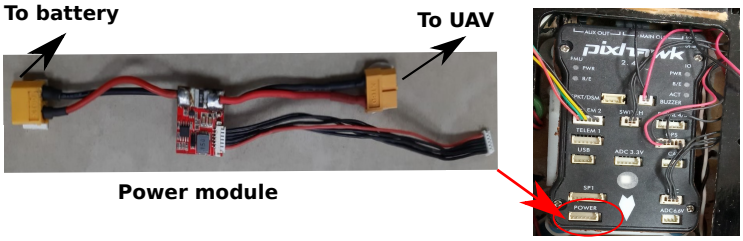

Fig. 2. Power module and Pixhawk flight controller

4.3 Data Pre-processing

We pre-process the data obtained from the data logger to get additional useful data such as the velocity in the z-direction, the total horizontal distance traveled, the total horizontal velocity, and the total acceleration in the horizontal direction. For calculating the velocity in the z-direction, we need to have a relationship of the distance traveled in the z-direction with time t. Based on the collected data from BARO, we first use a linear regression model to fit a curve that models the variation of altitude (Z) with time (t). Subsequently, we differentiate the equation of the curve to get the instantaneous velocity (v_z).

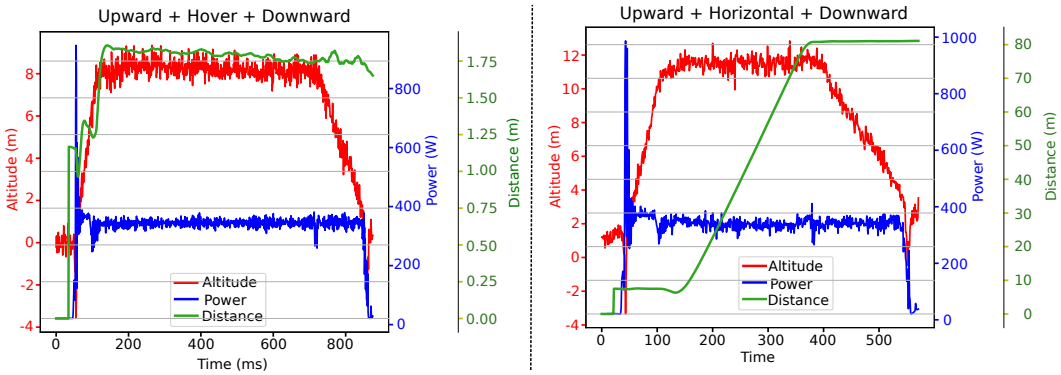

Fig. 3. Phases in a flight of the $20in4$ drone

We consider four standard types of motions that can occur in any flight after the drone has taken off: vertically upward, vertically downward, horizontal, and hovering (staying at the same place). We divide the entire flight into phases where each phase is identified by the type of motion in that phase. An example of the four phases in a flight is shown in Figure 3. In each phase, we collect multiple readings spread over the entire duration of the phase.

4.4 Quantification and Modeling

We experiment with two kinds of drone flights: controlled and random. A controlled flight is the one in which the height, distance, velocity, and/ or acceleration are restricted, while a random flight is not pre-planned or controlled. The controlled and random flights are shown in Figures 4 and 5, respectively. From nearly 39 controlled flights and 13 random flights for the two drones (20*in*4 and 20*in*8), we extract the phases and their characteristics (power, velocity, distance, and acceleration). Due to control over the quantities in controlled flights, the phases are easily separable and each phase consists of multiple continuous readings. They are mostly stable with minor fluctuations in power, velocity, and acceleration. Hence, we consider the average values of these quantities for each phase. In the case of a random flight, the separation between the phases is not clear. However, if the duration under consideration is small enough such that the readings do not change significantly, it can be considered to represent one particular phase, that is, it behaves in a controlled manner for small durations. For these small durations, we extract the average values of the desired quantities. There is a good match between the parameters extracted for both the flights (power and velocity).

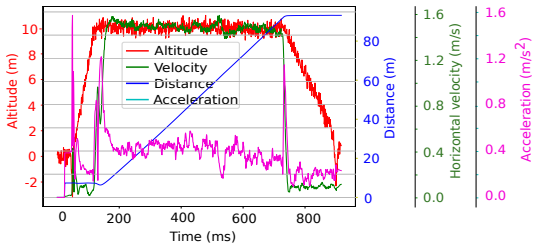

Fig. 4. Controlled flight

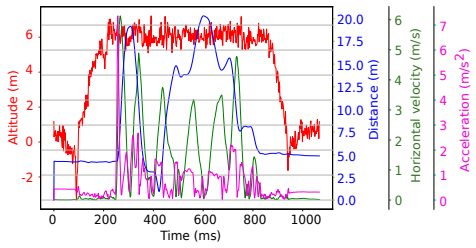

Fig. 5. Random flight

4.4.1 Power Model for Hovering. For estimating the power consumption of the drone during hovering, we considered all the relevant parameters such as the altitude of hovering and the hovering duration. We perform a sensitivity analysis with both of these parameters, by varying one at a time and keeping the other constant. The drones were made to hover at different heights (10, 20, and 40 m) for different durations (60, 120, and 180 seconds). We controlled the highest altitude in the controlled experiments. We never performed experiments for heights beyond 40 m due to legal restrictions. We did not observe any significant effect of the hovering duration on the power drawn and hence it does not appear in the power model (see Equation 1).

Figure 6 shows the variation of hovering power with respect to the hovering altitude (the scattered values are the real measurements and the lines are the best fit obtained). We observe that the hovering power consumed by the 20*in*8 drone is roughly 1.8× the power consumed by the 20*in*4 drone. This is correlated with their weights (3.9 Kg and 2.7 Kg, respectively, including the weight of battery). The power is mostly constant with respect to the hovering altitude with minimal variations limited to 20 W for both the drones. These variations in power occur mainly because the drone tries to stabilize itself to maintain a pre-specified position for the entire hover duration. We present a general formulation of power consumed by our drones of different weights in Equation 1. Here, W_d is the weight of the drone, h is the hovering altitude, E_{hover} is the energy consumed, T_{hover} is the hover duration, and P_{hover} is the power consumed by the drone. The fitted curve estimates the power with an accuracy of 98.4%. The accuracy can be improved further if a single drone is considered as is done in the

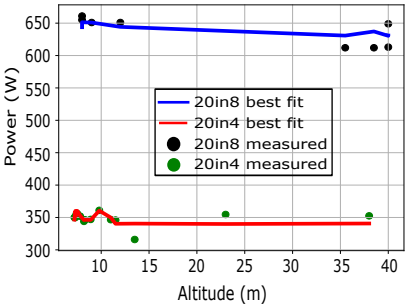

Fig. 6. Hovering power

related work on power modeling for drones, however our aim is to provide a power model that generalizes to different drones having different weights. We checked for the generalizability of the model for the $18in4$ drone.

$$\begin{aligned} P_{hover} &= 247.9 \times W_d - 0.6 \times h - 315.74 \\ E_{hover} &= P_{hover} * HT \end{aligned} \tag{1}$$

Theoretically, the hover power should not depend on the hovering altitude. The standard equation for hovering power is given by Equation 2. However, in our experiments we observe that the hovering power reduces to some extent as the altitude increases. This is primarily due to other unavoidable environmental factors involved in the measurements such as minor changes in the air pressure. Abeywickrama et al. [2] have observed similar variations with the altitude.

$$P_{hover} = \sqrt{\frac{(m * g)^3}{2 * \rho * n * \pi * r^2}} \tag{2}$$

4.4.2 Power Model for Vertically Upward Motion. For modeling the power consumption during a vertically upward motion, the ideal approach [2, 35, 39] is to make the drone climb upwards at a constant velocity till it reaches a certain height. We extract the average vertical velocity and the distance traveled for the upward motion from the random and controlled flights as explained in Section 4.4. Subsequently, these values along with the weights of the drones are used as features to get the power model for the vertically upward motion.

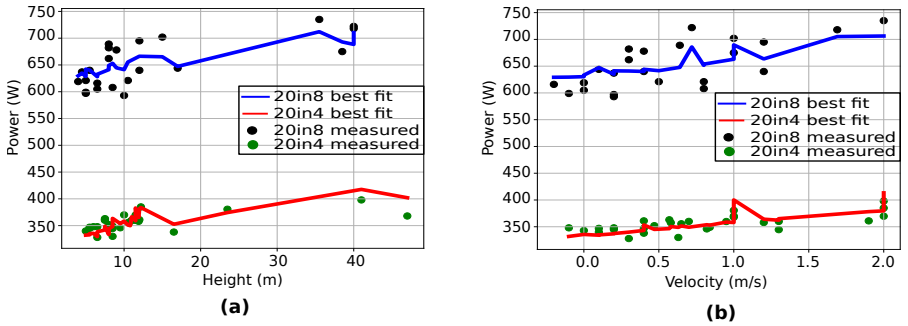

Fig. 7. Power consumption for vertically upward motion of the $20in4$ and $20in8$ drones

Figure 7 plots the overall measured power consumption during the upward motion on the y-axis and the height and vertical velocity on the x-axes. Note that we plot the overall power, and thus the power on the y-axis is a result of all environmental conditions, the current vertical velocity and the vertical distance traveled. Nevertheless, we plot the overall power with each parameter separately so as to get a rough idea of the effect of the parameter on the power. Hence, the fit will not be exactly accurate when we plot the overall power with respect to only one parameter as is visible from Figure 7 (because the other parameter varies). Nevertheless, we observe that the power for the upward motion increases as a result of increasing the vertical distance and velocity.

Upon quantifying the effect of both the parameters on the overall power in Equation 3, we observe that the effect of the vertical velocity on power is much more as compared to the effect of the vertical distance. In Equation 3, W_d is the weight of the drone, h is the total vertical distance to travel, and v_z is the velocity in the vertical direction. This equation gives the best fit for the measured power values with an accuracy (mean absolute percentage error = 2%) of 98%. Though we were able to get a similar accuracy using polynomial features of degree 2, we chose the linear

model due to its simplicity. Here, E_{climb} is the energy consumed, T_{climb} is the time taken, and P_{climb} is the power of the drone in the vertically upward motion (assumed to be continuous from the last hover position).

$$\begin{aligned} P_{climb} &= 248.5 \times W_d + 1.22 \times h + 15.34 \times v_z - 338.7 \\ E_{climb} &= P_{climb} * T_{climb} \end{aligned} \tag{3}$$

4.4.3 Power Model for Vertically Downward Motion. For vertically downward motion, we perform similar experiments as we did for the case of the vertically upward motion. Here, we start by considering two quantities that can have an effect on the power: vertical velocity, and the vertical distance. However, we do not observe any variation in power with the vertical velocity because during vertically downward motion or during landing, the drone automatically switches to a lower and safer constant velocity value. Hence, this parameter is not important. Thus, we show the effect of height (vertical distance traveled) on the power consumption.

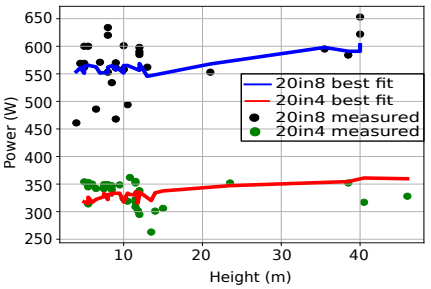

Fig. 8. Power consumption for vertically downward motion of the 20*in*4 and 20*in*8 drones

Figure 8 shows the effect of height (as we have defined) on the power consumption. The power consumption increases with an increase in the height because at higher heights, the potential energy is more and hence more work is done by the drone to fly downwards at constant velocity. We observe that the power consumption in the downward motion is less than that in the upward motion for the same height owing to gravity. Equation 4 fits the measured power data with an accuracy of 96%. Here, h is the vertical distance traveled, E_{fall} is the energy consumed, T_{fall} is the time taken, and P_{fall} is the power of the drone in the vertically downward motion.

$$\begin{aligned} P_{fall} &= 195.786 \times W_d + 0.84 \times h - 207.5 \\ E_{fall} &= P_{fall} * T_{fall} \end{aligned} \tag{4}$$

4.4.4 Power Model for Horizontal Motion. We perform experiments to measure the power consumption of the drone when it is moving horizontally. We consider two primary factors that decide the power consumption during the horizontal motion: the horizontal velocity and the vertical height. A pure horizontal motion requires the drone to maintain a constant height while flying horizontally. The power increases slightly with the height. However, the effect of height is less pronounced and is more or less negligible.

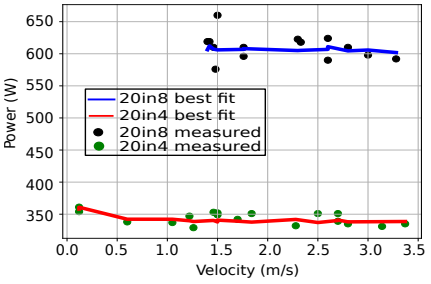

Fig. 9. Power consumption for the horizontal motion of the 20*in*4 and 20*in*8 drones

Figure 9 shows the variation in power with respect to the horizontal velocity. We observe that the effect of horizontal velocity is similar to that observed in reference [35] where the power decreases with the velocity initially and then increases suddenly. There is no clear reason for this behavior. One possible reason is that at lower velocities, the drone stabilizes more frequently and hence leads to a larger power consumption. This is mainly because the time taken to travel a particular distance is more with lower velocities as compared to the corresponding time taken with higher velocities.

Note that we do not experiment with very high velocities (beyond 6 m/s), hence we do not see the inflection point. The fitted curve is given by Equation 5 that has an accuracy of 98.2%. Here v_h is the horizontal velocity, E_{hor} is the energy consumed, T_{hor} is the time taken, and P_{hor} is the power of the drone in the horizontal motion.

$$\begin{aligned} P_{hor} &= 225 \times W_d - 3.93 \times v_h - 257 \\ E_{hor} &= P_{hor} * T_{hor} \end{aligned} \tag{5}$$

4.4.5 Summary. Our power model is similar in structure to other models proposed in prior work [2, 35, 39]. Note that our model is much more comprehensive in terms of the scenarios and parameters that have been considered to model different types of motion. Moreover, we also consider random flights unlike the related work that considers the clean separation of phases. Hence, our experiments model the real-world flights more accurately. In general, for drones there is a need to create a bespoke model for a given family of drones because most models do not generalize very well. We did a limited study of generalizability: we collected most of our data for the $20in4$ and $20in8$ drones. The resultant model estimates the power consumption of the $18in4$ drone very well. We thus are confident that our model will hold for other drones in the same range of weights.

4.5 Accuracy Comparison of Different Power Models

Figure 10 compares the mean absolute percentage error of different ML models for estimating the power consumption of the drones. We observe that SVR has the highest error for all the types of motion. Decision tree, random forest and gradient boosting regressors perform nearly equivalently to the linear regressor, with linear regression having an error within 5% for all the types of motions. We **choose** linear regression because of its higher accuracy and simplicity over other competing models.

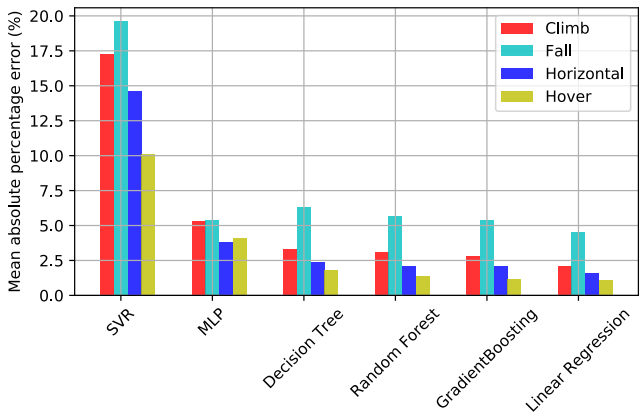

Fig. 10. Mean absolute error (%) of ML models for estimating the drone power

5 EXPERIMENTAL SETUP

5.1 Overview

In this work, we aim to identify the suitable configuration of the tunable knobs involved in UAV navigation such that the total energy consumption is minimized. We primarily consider the tunable parameters of the RRT* path planning algorithm as the knobs (see Section 2.1) because these parameters play a role in deciding the flight statistics such as the hover time and the path length. We also consider the different platforms for running vision algorithms such as CPUs, GPUs, and accelerators as the tunable knobs. Based on the performance and power requirements of the system and these platforms, a suitable platform is chosen. Note that such a heterogeneous system is currently the state-of-the-art for such systems. The Nvidia AGX Xavier board is one such example.

We need to formulate a joint optimization problem that finds the optimal values of the tunable parameters of RRT* and the *optimal platform* for the vision algorithm. The joint optimization objective is to minimize the total energy consumed, which includes the energy consumed in hovering, traveling a certain path length, and running a vision algorithm. To formulate the optimization problem for the RRT* path planning step, we model the tunable parameters as the variables. We

then need to find a relationship between the parameters of RRT* and the flight statistics that directly impact the energy consumption (hover time and path length). Due to the complex relationship between the parameters (see Table 1) and the uncertain nature of sampling-based algorithms such as RRT*, we need to collect a huge amount of data for multiple environments to formulate a *master equation* for the optimization problem. This process is extremely time-consuming if done in a real outdoor setting, thus we perform exhaustive simulations using an open-source robotics simulator, Gazebo, for multiple configurations and virtual worlds.

The optimization problem that we create has complex, non-convex, nonlinear constraints and it thus takes a long time to solve (if at all it converges). Hence, to solve it in real-time, we map it to an approximately equivalent game-theoretic framework where the independent parameters (see Table 1) are the players and the dependent parameters (flight statistics such as hover time and path length) are used to formulate the payoffs of the players (explained in detail in Section 7). We, in effect, perform sensitivity analyses of the dependent parameters with respect to the independent parameters. Once the payoffs of the players are formulated using the sensitivity results, we use Gambit-v15.1.1 [25] along with its Python API to calculate the Nash equilibria of the game. This is then mapped to the *optimal solution*.

5.2 Setup for Sensitivity Analyses

Table 1. Tunable parameters of the RRT* algorithm

Parameter	Description	Range
Dimension size (dim)	Dimension of the search space for RRT* path planning (physical 3D dimensions)	$5\times5\times5-40\times 40\times 40$ m^3
#samples (sam)	Number of random samples for RRT*	$100-2000$
Step size (res) of RRT*	Minimum step length that can be taken in the direction of the chosen random sample	$0.01-10$ m
Obstacle avoidance distance (obs_av)	Minimum distance that should be maintained between the nearest obstacle and the calculated path	$0.1-0.5$ m
These values are obtained as feasible ranges from the experiments. Feasible values are those that do not degrade the accuracy significantly.		

We use an NVIDIA Xavier board (state-of-the-art board for autonomous vehicles such as UAVs) for the sensitivity analyses. It consists of 8 ARMv8.2 cores having a frequency of 2.26 GHz, main memory of 16 GB, 8 MB L2 cache, 4 MB L3 cache, and a 512-core NVIDIA Volta GPU with 64 Tensor cores. We emulate the path planning algorithm on the NVIDIA Xavier board to get the sensitivity results. The board runs the Robot Operating System (ROS). ROS allows running multiple concurrent processes, also called nodes. These nodes pass data between each other using non-blocking FIFO queues [4]. ROS uses the publish-subscribe model where some nodes publish messages while other nodes receive the messages by subscribing to the publishing nodes. The messages can belong to some specific categories, also called *ROS topics*. For the sensitivity analyses, we kept the simulation environment the same. We primarily relied on pre-recorded messages for simulation that were stored in a *ROS bag* file during the outdoor flight of the UAV. *ROS bag* files help us realize a deterministic simulation by replaying the pre-recorded messages.

5.3 Gazebo and Rviz for UAV Simulation

To collect the data for formulating the optimization problem, we simulate the virtual world and the UAV in Gazebo-9. Gazebo takes in a *world* file that specifies the environment along with a map of the obstacles, and the UAV's specifications. After the planning step, the control commands are sent to the Gazebo engine to make the UAV move in the desired direction. We use Rviz-v1.13.13 for visualizing the surroundings in the form of an occupancy map. Rviz is a ROS graphical interface that allows us to visualize the position, orientation of the UAV, and the locations of the obstacles.

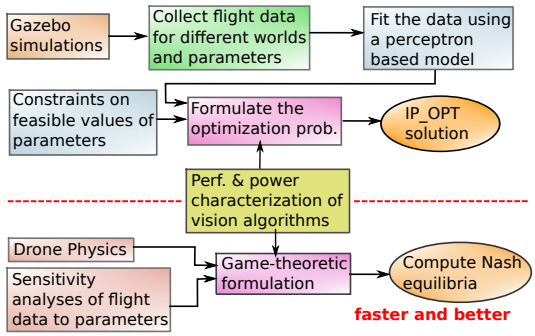

Fig. 11. Overview of our approach

5.4 Creation of Virtual Worlds

For creating the virtual Gazebo worlds, we have used standard tools [1, 30] from the literature. Oleynikova et al. [30] generate random sylvan worlds with trees of varying heights scattered over a given area. Abbyasov et al. [1] generate virtual worlds by taking in any 2D image and producing a 3D model of it.

Table 2. Baseline system

Parameter	Type/Value
CPU	2 x Intel Xeon Gold 5118 (Skylake)
# of cores	24 physical
Frequency	2.3 GHz
Main memory	128 GB
GPU	NVIDIA Tesla T4 (Turing)
CUDA cores	2560
Tensor cores	320

5.5 IPOPT and AMPL

In order to solve the nonlinear optimization problem formulated using the data collected from Gazebo, we used a nonlinear solver, *IPOPT-v3.12.13* [40]. We model the optimization problem using two ways: an algebraic modeling language called *AMPL* and Python (using the Gekko [3] package). Figure 11 shows an overview of the steps involved in the formulation of the optimization problem and the game-theoretic approach. These are discussed in detail in Sections 6 and 7.

Table 3. Details of the system (source [27])

Parameter	Big core	Small core
Issue Width	4	2
Pipeline Type	Out-of-order	In-order
Frequency	3.1 GHz	1.55 GHz
L1 cache latency	4 cyc.	2cyc.
L2 cache latency	20 cyc.	

5.6 Setup for Measuring the Power and Performance of Vision Algorithms

We collected the power and performance numbers of the vision algorithms on three types of execution units: Intel Xeon-like big core, Intel Atom-like small core, and Tesla T4 GPU. The configuration of the GPU is given in Table 2. The configurations of the big core and the small core are given in Table 3. The big and the small cores are simulated using the Tejas architectural simulator [36] that is very well calibrated with native hardware. The weights of the big core, small core and GPU are 768 gm, 390 gm, and 500 gm, respectively [5, 11, 16]. We consider these weights in the power calculation using the power model derived in Section 4. We considered the traditional vision workloads inspired from the MEVBench vision benchmark suite [10]. Table 4 provides a brief description of the workloads used in this study, and the power and performance of these workloads on different platforms.

6 FORMULATION OF THE OPTIMIZATION PROBLEM

As explained in Section 5.1, we need to perform exhaustive simulations in Gazebo to collect multiple data points corresponding to different parameter configurations and different virtual

Table 4. Benchmarks (derived from MEVBench)

Bench.	Description	Big Core		Small Core		GPU T4	
		Time (s)	Power (W)	Time (s)	Power (W)	Time (s)	Power (W)
Sift	Feature detection and extraction algorithm. Finds features that are invariant to scale, lighting, viewpoint, and orientation	0.092	26	0.88	4.1	0.03	26.1
Surf	Feature extraction algorithm similar to SIFT but is much faster, suitable for embedded systems	0.062	31	0.53	4.5	0.04	27
Fast	Extracts the features of the image corners	0.02	26.1	0.14	4.5	0.03	26
Orb	Uses *Fast* to detect the corners and BRIEF to extract the features corresponding to these corners	0.02	28.7	0.18	4.7	0.04	26.6
HoG	Uses gradients in an orientation of the image to describe the features.	0.2	20	0.71	4.3	0.03	26.6
SVM	Classifies the features on the basis of a support vector	0.04	25.8	0.4	4.7	0.05	25.1
KNN	Uses the nearest neighbor algorithm to classify the features	0.05	27	0.6	4.44	0.03	27

worlds. Subsequently, we use a novel ML-inspired technique to fit a curve on these data points. The equation for the curve is then used to formulate the constraints of the optimization problem. The objective of the formulation is to minimize the total energy consumption including the *energy consumed in traveling from a source to a destination* and the *energy consumed by the computer vision algorithm*. The energy consumption in going from the source to the destination can be written as the sum of the energy consumed in hovering at a location and the energy consumed in traveling horizontally from the source to the destination. The hovering energy is proportional to the hover time (HT) as shown in Equation 1, where HT signifies the time elapsed before a decision is made by the planner. The UAV hovers at the current position during this time and is not doing any useful work. The energy for the horizontal motion is proportional to the path length (PL) as shown in Equation 5. In order to collect the flight data (HT, and PL) from the Gazebo simulator, we simulate the path planning algorithm for different parameter configurations and collect the corresponding HT and PL for different virtual worlds. We collect HT and PL instead of hovering energy and the energy consumed in the flight because the simulator is not equipped with its own energy model and the energies are proportional to HT and PL, respectively, as shown in Section 4. We use the setup as described in Section 5 to collect the power and performance characteristics of the vision algorithms on different platforms.

6.1 Collection of Data Points

The idea is to collect the flight data (HT and PL) for multiple virtual worlds from Gazebo for varying configurations of the tunable parameters (shown in Table 1) of the RRT* path planning algorithm. We performed simulations for all possible configurations of the independent parameters in their feasible ranges. We collected 300 data points for each virtual world. A data point is an n-tuple of the parameter configuration (res, sam, dim, obs_av), input environment (obs_den), HT, and PL (parameters explained in Section 2.1). The input environment is captured in terms of the obstacle

density (*obs_den*) of the virtual world. For each simulation, the start and the end points were kept the same so that the effect of changing the configurations can be accurately captured. To tackle the uncertainty in the experiments, we find 10 paths and hover time values for every configuration and use the average of these values as the final path length and hover time of the data point.

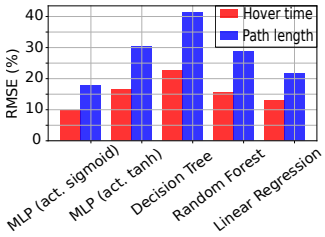

Fig. 12. RMSE comparison of different models for World 1

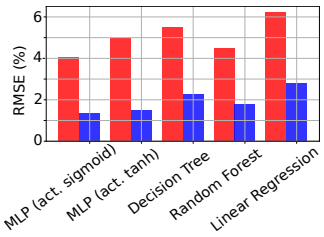

Fig. 13. RMSE comparison of different models for World 2

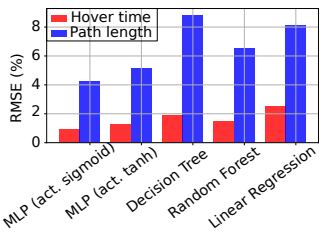

Fig. 14. RMSE comparison of different models for World 3

6.2 Curve Fitting: Hover Time and Path Length

The configuration parameters and the obstacle density together form the feature vectors of these datapoints. To get the exact dependence of the flight data (HT and PL) on the feature vector, we performed curve fitting. The curve fitting problem takes the collected data points as its input and provides a trained predictor model as the output. The equation for this model is the fitted curve that provides a relationship of the flight data with the configuration parameters and the input environment. Here the idea is to consider 80% of the collected data points and fit the curve using these points. We use the remaining 20% of the data points as the test points to test the accuracy of the fitted curve. We use the root mean square error (RMSE) metric to quantify the error of prediction.

All the data points are normalized using the min-max scaling technique (APIs present in Scikit-learn [32]). Since two values (HT and PL) need to be predicted for each feature vector, this is a multi-output regression problem. Figures 12, 13, and 14 show the comparison of five different learning/regression techniques for three different virtual worlds. Our aim is to *derive* a mathematical expression that relates the output to the inputs for the best learning technique. We achieve the lowest root mean square error (RMSE) using the multi-layer perceptron (MLP) algorithm with a Sigmoid-based activation function and 2 hidden layers, each having 10 neurons. It captures nonlinear dependences well and fortunately, it is possible to represent its action quite easily in mathematical terms as we shall see next.

6.3 Formulation of the Optimization Problem

$$\begin{aligned} minimize \quad FlightEnergy &= E_{hover} + E_{hor} + E_{vision}/fps \\ &= P_{hover} \times HT + P_{hor} \times FT + P_{vision} \times (HT + FT)/fps \\ &= P_{hover} \times HT + P_{hor} \times PL/v_h + P_{vision} \times (HT + PL/v_h)/fps \end{aligned} \tag{6}$$

As explained in Section 5.1, we solve the joint optimization problem, where the objective is to minimize the total energy consumed, which includes the energy consumed in hovering, traveling a certain path length, and running a vision algorithm as shown in Equation 6. The energy is then written as a product of the corresponding power (obtained from the power model) and time. Here, HT is the hover time, FT is the flight time to travel the distance from the source to the destination, PL is the path length, P_{vision} is the power consumed by the vision algorithm, and fps is the frame rate. We scale the energy consumption of the vision algorithm by the frame rate because given the choice of platforms with varying energy and execution time requirements, we need to choose

the one that does the maximum work for the same energy. If such a scaling factor is not used, the choice will always be in favor of the most energy-efficient core, that is, the small core, which is a biased result. Hence, we scale it by the frame rate. This ensures that for any vision algorithm, the platform that processes maximum frames with the least energy consumption will be chosen. Note that we have three different platforms and thus have three different values of P_{vision} and fps, where only one is ultimately chosen; the implemented optimization problem captures this fact with integer constraints. This has not been shown for the sake of ease of readability.

$$\begin{aligned}
\text{s.t.} \quad h1[i] &= \sum_{j=1}^{5} w_1^{\intercal}[i,j] * v[j] + b1[i], \forall i \in [1,10] \\
h1o[i] &= 1/(1+e^{-h1[i]}), \forall i \in [1,10] \\
h2[i] &= \sum_{j=1}^{10} w_2^{\intercal}[i,j] * h1o[j] + b2[i], \forall i \in [1,10] \\
h2o[i] &= 1/(1+e^{-h2[i]}), \forall i \in [1,10] \\
h3[i] &= \sum_{j=1}^{10} w_3^{\intercal}[i,j] * h2o[j] + b3[i], \forall i \in [1,2]
\end{aligned} \tag{7}$$

$$\begin{aligned}
h3[1] - \xi_h \leq & HT \leq h3[1] + \xi_h \\
h3[2] - \xi_p \leq & PL \leq h3[2] + \xi_p \\
1 \leq & HT \leq FT/2 \\
dist(start, dest.) \leq & PL \leq 2 * dist(start, dest.) \\
0.01 \leq & v[1] \leq 10 \\
100 \leq & v[2] \leq 5000 \\
dist(start, dest.) \leq & v[3] \leq 5 * dist(start, dest.) \\
0.1 \leq & v[4] \leq 0.5
\end{aligned} \tag{8}$$

Note that the horizontal motion is usually composed of three kinds of motions [39]: acceleration, flight at constant velocity, and deceleration. Ideally, FT should have been replaced by the time taken in these three phases as opposed to considering the time taken in only the constant velocity phase. We do not consider the acceleration and the deceleration phases because those are necessary phases to reach a constant velocity and to come back to a zero velocity, respectively. Thus, there is no scope of energy minimization/reduction in these phases and hence we consider only the constant velocity phase. The curve derived from the MLP formulation provides the equations for the constraints on the tunable parameters of the RRT* path planning algorithm as shown in Equation 7.

Here $\langle h1, h2, h3, h1o, h2o \rangle$ are the neurons in the hidden layers of the MLP. We use two hidden layers and correspondingly three weight matrices $\langle w_1, w_2, w_3 \rangle$ and three bias vectors $\langle b1, b2, b3 \rangle$ from the input layer to $h1$, $h1o$ to $h2$, and $h2o$ to $h3$, respectively, where $h3$ is error-corrected to give the outputs HT and PL. The input to the MLP is the feature vector v that captures the configuration parameters and the input environment. Here, $v = \langle res, sam, dim, obs\_av, obs\_den \rangle$.

In Equation 7, the first constraint for the first hidden layer ($h1$) is the sum of the product of the weights (w_1) learned from curve fitting and the feature vector (v). A bias term ($b1$) is also added to each neuron ($h1[i]$) of the hidden layer ($h1$). The second equation for $h1o$ introduces a nonlinearity in $h1$ using the Sigmoid activation function. The equations for the second hidden layer ($h2$) are obtained similarly.

The output of the second hidden layer is multiplied by its corresponding weights to generate the constraints on HT and PL. The equations for HT and PL capture the final regression output along with the error margins: ξ_h and ξ_p (see Equation 8). The equations containing the Sigmoid activation are expanded using the Taylor Series expansion of e^{-y}, which is $1 - \frac{y}{1!} + \frac{y^2}{2!} - \frac{y^3}{3!}$...; hence, the formulated optimization problem can have an approximate polynomial form (see Equations 6, 7, and 8). The next two constraints on HT and PL ensure a good quality of the solution. Equation 8 shows the constraints on the feasible regions of all the configuration parameters: res ($v[1]$), sam ($v[2]$), dim ($v[3]$), and obs_av ($v[4]$). These are obtained from experimental observations. We did not find it useful to impose a constraint on the obstacle density ($v[5]$) because it is an environmental parameter and can significantly vary based on the ambient. $dist(start, dest.)$ is the Euclidean distance between the start and the end points.

Additionally, we introduce a new constraint as shown in Equation 9. This constraint provides a power cap to the objective function. Here, P_{drone} is the power cap for the $20in4$ and $20in8$ drones. We obtain approximate values of the power cap for the two drones from our power modeling experiments. P_{drone} is set such that it provides a feasible solution for all the vision algorithms. For some vision algorithms, a tighter constraint on the power envelope is sufficient, while for others such as SURF, a relaxed power constraint is necessary to provide feasible solutions of the optimization problem. We apply this constraint only for experimentation to get an idea of the variation of the flight statistics with P_{drone}.

$$P_{hover} \times HT + P_{hor} \times PL/v_h + P_{vision} \times (HT + PL/v_h)/fps \leq P_{drone} \times (HT + PL/v_h) \tag{9}$$

6.4 Sensitivity to the Power Envelope (P_{drone})

Figures 15 and 16 show the sensitivity of the path length to the power cap (P_{drone}) for the $20in4$ and $20in8$ drones, respectively. The platform is the big core for this experiment. In this experiment, we perform a sensitivity analysis of the path length with the power constraint (see Equation 9). We start with the constraint that provides a feasible solution for all the vision algorithms and keep on relaxing it. Note that since this is a minimization problem, a less than constraint should not have an effect on the optimal solution. However, we see some variation in the path length (and hence the chosen configuration) as P_{drone} is increased. We observe that as P_{drone} is increased from 560 W to 1500 W or 800 W to 2000 W, the optimal configuration and hence the path length changes. This is mainly because the search space increases with increasing P_{drone}. We observe fluctuating results because of the non-convex search space and the tendency of solvers to get stuck in local minima.

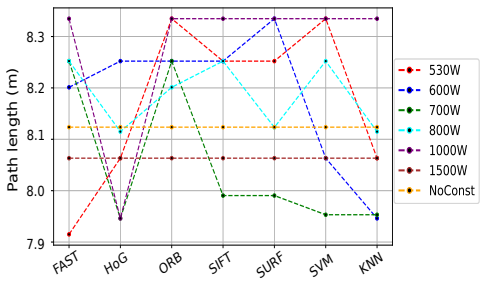

Fig. 15. Sensitivity of path length to P_{drone} for the $20in4$ drone

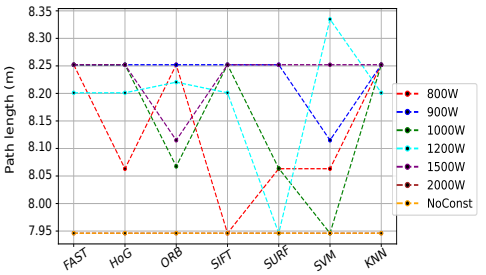

Fig. 16. Sensitivity of path length to P_{drone} for the $20in8$ drone

7 GAME THEORY

We show in Section 8 that solving the formulated optimization problem using a nonlinear solver takes a prohibitive amount of time and sometimes even does not converge to a solution. Thus, we propose to develop games where the tunable parameters of the RRT* path planning algorithm are the players. We divide the parameters into *dependent* and *independent* parameters. The independent parameters (shown in Table 1) are the players while the dependent parameters are used in conjunction with the independent parameters to formulate the payoffs of these players. This is because the energy consumption of the drones is proportional to these dependent parameters (HT and PL). Hence, if these parameters are not controlled judiciously, the energy consumption will be high. Additionally, we take into account the energy consumption of the vision algorithm in formulating the payoffs of these players. We do not consider the type of platform (for the vision algorithm) as a player because it is unrelated to the flight statistics and hence would have acted as an indifferent player. However, we realize that the energy consumption of the vision algorithm on a resource will play an important role in determining the overall energy budget and hence it should be a part of the payoff equations.

Our approach is to make the payoff of the players a function of two objectives: *altruistic* and *selfish*. The **altruistic objective** of the players is to minimize the hover energy, the energy consumed in traversing the path (of length *PL*), and the energy consumption of the vision algorithm. It is negative in nature. The **selfish objective** of the players is proportional to the values of their individual parameters. Thus, the payoffs of the players is a combination of maximization of the selfish objective and minimization of the overall energy, given the map (specifically, obstacle density) of the environment.

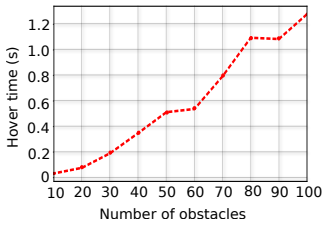

Fig. 17. No. of obstacles v/s hover time

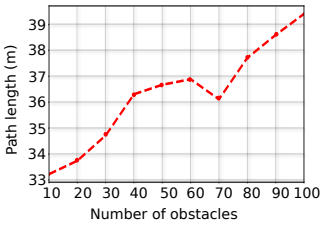

Fig. 18. No. of obstacles v/s path length

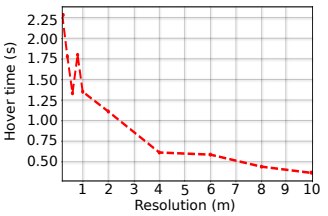

Fig. 19. Resolution of the path v/s hover time

7.1 Sensitivity Analyses

We performed the sensitivity analyses of the UAV flight (HT and PL) with respect to the parameters of the path planning algorithm on an NVIDIA Xavier board (see Section 5 for the setup and details of the experiments). We make the following relevant observations. Figure 17 shows the relationship of the worst-case hover time with the number of obstacles present in the search space. The worst-case hover time is when the UAV has to do re-planning at every obstacle. With an increase in the number of obstacles, the congestion in the search space increases, thereby leading to a higher hover time.

Figure 18 shows the relationship of the length of the path with the number of obstacles, while the source and destination for all the experimental points are kept the same. As the number of obstacles increases, the number of free spaces to form a collision-free path reduces. Thus, the length of the path increases because the planner has to take many detours.

If the resolution (step-size) of the path increases, the UAV takes larger steps in the direction of the destination, thereby reducing the number of collision checks and replanning steps. Hence, the hover time reduces. Figure 19 shows the relationship of the resolution with the hover time of the UAV.

7.2 Game Setup

We developed a game with five players: number of samples (*sam*), obstacle avoidance distance (*obs_av*), dimension of the search space (*dim*), resolution of the path (*res*), and the obstacle density (*obs_den*). Here, the first four players are the tunable parameters of the path planning algorithm and the obstacle density is the input to capture the nature of the world through which the navigation is to be done. A higher obstacle density tries to reduce the payoff of the players by increasing the hover time and the path length as observed in Figures 17 and 18. This in turn will negatively affect the energy consumption of the vision algorithm, which will scale proportionally to the power consumption of the algorithm on the underlying platform. However, a higher frame rate in this scenario will find a shorter collision-free path. Hence, the payoff of the players will have the energy consumption of the vision algorithm per unit frame capture rate as one of the terms with a negative sign. Moreover, we have three types of platforms for the vision algorithms as explained in Section 5.6. These platforms have varying performance and power requirements. While some of them are high performing but energy-inefficient, others are not real-time but highly energy efficient. Hence, scaling the energy consumption of the vision algorithm (E_{vision}) by the frame rate (fps) will provide a fair decision in terms of the platform.

The complexity of the RRT* algorithm is directly proportional to the number of samples. Thus, the decision time and hence the hover time increases with an increase in the number of samples. The *sam* player would want to increase the number of samples to make a better decision, however it wants to minimize the wasted energy. In this case, the wasted energy is equal to the hovering energy, which is equal to the product of the hovering power and the hover time [12] (see Equation 1). The payoff of the *sam* player is shown in Equation 10.

$$\begin{aligned} Payoff\_sam &= \alpha * sam - E_{hover} - \theta * obs\_den - \beta * E_{vision}/fps \\ &= \alpha * sam - P_{hover} * HT - \theta * obs\_den - \beta * P_{vision} * (HT + PL/v_h)/fps \end{aligned} \tag{10}$$

The *obs_av* player wants that the UAV should fly at a distance from the obstacles. Thus, it would want to increase this distance; this would lead to an increase in the time for path planning and increased path length. Moreover, there will be a nonlinear relationship with the path length owing to the uncertainty in the sampling-based path planning algorithms and the vision algorithm will require more energy. Hence, the payoff is captured in Equation 11, where E_{pl} is the energy spent in covering the path. It is equal to E_{hor}, which is proportional to the horizontal velocity v_h and path length PL as shown in Equation 5.

$$\begin{aligned} Payoff\_obs =& \alpha' * obs\_av - \gamma * PL^{\lambda} - \\ & E_{pl} - E_{hover} - \theta * obs\_den - \beta' * E_{vision}/fps \\ =& \alpha' * obs\_av - \gamma * PL^{\lambda} - \\ & P_{hor} * PL/v_h - P_{hover} * HT - \theta * obs\_den - \beta' * P_{vision} * (HT + PL/v_h)/fps \end{aligned} \tag{11}$$

The relationship of the hover time and resolution of the path planning step is accurately captured in Figure 19. The hover time is a hyperbolic function of the resolution of the path. Thus, the *res* player would want to decrease the resolution, still observe all the obstacles and form a collision-free path, however the hovering energy would increase. Hence, the payoff can be captured using Equation 12.

$$\begin{aligned} Payoff\_res &= \alpha''/res - E_{hover} - \theta * obs\_den - \beta'' * E_{vision}/fps \\ &= \alpha''/res - P_{hover} * HT - \theta * obs\_den - \beta'' * P_{vision} * (HT + PL/v_h)/fps \end{aligned} \tag{12}$$

As the dimension increases, the sample density reduces. This reduces the number of samples to choose from for the next nearest node. Thus, the time spent in planning and hence hovering increases. Due to the dispersed samples, the path length also increases. The relationship of the dimension with the path length is nonlinear, however the exact relationship is hard to deduce. The *dim* player would want to do the planning for a larger dimension, however it needs to minimize the wasted energy as a result of increased hovering and increase in the path length (E_{pl}). Equation 13 captures the payoff for the *dim* player.

$$\begin{aligned} Payoff\_dim &= \alpha''' * dim - \gamma' * PL^{\lambda'} - \\ &\quad E_{pl} - E_{hover} - \theta * obs\_den - \beta''' * E_{vision}/fps \\ &= \alpha''' * dim - \gamma' * PL^{\lambda'} - \\ &\quad P_{hor} * PL/v_h - P_{hover} * HT - \theta * obs\_den - \beta''' * P_{vision} * (HT + PL/v_h)/fps \end{aligned} \tag{13}$$

All the constants – the αs, γs, βs, and θs – are the hyper-parameters of the game. We are using the constant λs in the equations involving PL because the exact function is not known. We however experiment with different values of λs and choose an appropriate value as we show next.

7.3 Choosing the Hyper-parameters

We experimented with multiple combinations of hyper-parameter values and found the corresponding Nash equilibria (NE). It was observed that for all these experiments, the obtained NE were non-trivial – at least one player played its non-trivial strategy. A trivial Nash equilibrium refers to an equilibrium point where all the players play their dominant strategy (extrema in the range). A non-trivial Nash equilibrium refers to a point where the players need to adjust their strategies to non-dominant strategies based on the strategies played by the other players. We prefer the hyper-parameter ranges for which the obtained NE parameter configuration is such that the overall energy gets minimized. We found that the best ranges for the hyper-parameters are αs $\in [0.1, 0.5]$, γs $\in [0.5, 1]$, θs $\in [0.5, 1]$, λs $\in [1, 1.5]$, and βs $\in [5, 10]$. We found that if αs are less than γs and θs, the above observation holds. Hence, the exact values of the hyper-parameters are not important. For the final experiments, we set the hyper-parameters to the mid-points of their respective ranges.

In order to map the continuous range of the feasible values of the tunable parameters to a finite number of distinct strategies of these parameters (parameters become the players), we discretize the feasible range. After the Nash equilibrium is found, we get a unique strategy for all the players. The granularity of discretization is found empirically with practical considerations in mind. We found that discretizing *sam* at the granularity of 250 samples, *dim* at $5 \times 5 \times 5\,\mathrm{m}^3$, *res* at 0.2 m, and *obs_av* at 0.1 m gives the best results both in terms of the NE solution and the time taken to solve the game.

8 RESULTS

We show two categories of results: standalone parameter tuning of RRT* and parameter tuning of RRT* with computer vision algorithm. In the first type, we do not consider any vision algorithm that is executing alongside the path planning algorithm. In the second case, we have a vision-assisted drone and hence a vision algorithm runs alongside the path planning algorithm in an always-on mode. We run both the IP-OPT solver and the Gambit solver on a BeagleBone Black Board. It consists of an ARM Cortex-A8 processor clocked at 1 GHz, 4GB eMMC on-board flash storage, and 512 MB DDR3 RAM.

We first show the comparison of the two approaches for parameter tuning of RRT* path planning algorithm in Section 8.1. We also show the comparison of the path lengths and hover times obtained using the game theory-based configurations with the random configurations in Section 8.1.1. Next,

we compare the execution times of the optimization solver and the game theory solver on the BeagleBone Black board in Section 8.2. We then show the path lengths and hover times obtained by using game theory to jointly tune the parameters of RRT* and vision algorithm in Section 8.3. Subsequently, we show the comparison of the two approaches for jointly tuning the parameters of RRT* and vision algorithm in Section 8.4.

8.1 Comparison of Optimization-based and Game Theory-based Approaches for RRT*

We compare the hover time and the path length obtained using the optimization-based approach and the game-theoretic approach for three different virtual worlds. Figures 20, 21, and 22 show the occupancy map of the three worlds used in the experiments. Table 5 shows the comparison of the hover time and the path length obtained using the two approaches.

Table 5. Comparison of optimization-based and game theory-based approaches

Environment	Optimization		Game Theory	
	HT (sec)	PL (m)	HT (sec)	PL (m)
World 1	0.72	6.3	0.56	5.8
World 2	0.4	5.7	0.29	5.24
World 3	0.5	8.3	0.4	7.9

We observe from Table 5 that our game-theoretic approach provides solutions that are 5-21% better than the best optimization-based approach. In terms of the time taken to calculate the optimal solution, it takes 10 ms to compute the Nash Equilibrium using the Gambit solver. On the contrary, the IPOPT solver does not converge to a solution in finite time. The numbers reported in Table 5 correspond to the results obtained by simplifying the search space of the optimization problem. For the cases where IPOPT converges with the actual or reduced search space, it still takes 0.1 s to reach the solution, which is 10× slower than the time taken by Gambit. We also observe that the values of the hover time and the path length for World 3 are more than that for World 2. Both the worlds correspond to a forest, however the tree density is half in World 2 (0.1 $trees/m^2$) as compared to World 3 (0.2 $trees/m^2$) as shown in Figures 21 and 22. A higher obstacle density leads to higher congestion and hence the hover time increases, which is expected.

Table 6. Comparison of planning time (sec) for random and best configurations

Players	Random	Best	Improvement(%)
sam, res	4.9	2.08	57.55
sam, dim	6.15	4.71	23.4
res, dim	4.98	2.64	46.9
res, obs_av	2.3	1.2	47.8
sam, dim, res	1.9	1.6	15.78
start: (0, 0, 2); dest.: (5.5, -2.0, 1.0)			

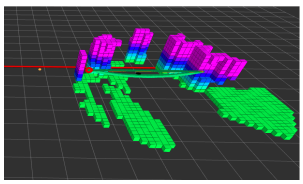

Fig. 20. 3D Occupancy map and the planned path for World 1

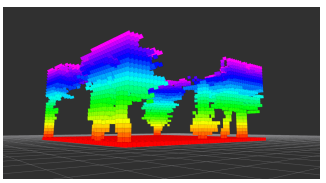

Fig. 21. 3D Occupancy map for World 2 (tree density 0.1 $trees/m^2$)

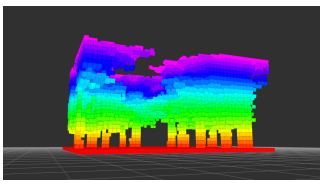

Fig. 22. 3D Occupancy map for World 3 (tree density 0.2 $trees/m^2$)

8.1.1 Comparison of Game Theory-based and Random Configurations for RRT.* In this section, we show the results for World 1 (see Figure 20) by considering two to three parameters as the players while the other parameters are assigned random values within their feasible ranges. Table 6 shows a comparison of the planning time for the random and the best parameter configurations of these players. The best configuration is the one that is provided by the game theory-based approach. We observe a 15-57% improvement in the planning time.

8.2 Performance Comparison of Solvers on BeagleBone Black for RRT*+Vision

In the literature, it has been reported that with an increase in the number of players or strategies, the time taken to calculate Nash Equilibria using the Gambit solver becomes prohibitive [25]. Nevertheless, this is not a problem for us because in the path planning algorithms, the number of parameters and their ranges (strategies) are relatively small and finite [1, 7]. Thus, our solution is scalable for a wide range of path planning algorithms for drone-based settings. We have a lot of leeway.

We compare the time taken to solve an optimization problem using the IPOPT solver and the Gambit game theory solver. We observe that the time taken to solve the optimization problem using IPOPT is between $0.4 - 0.9$ s. The time taken to solve the game theory problem is $0.05 - 0.07$ s. For the 2-player, 3-player, 4-player, and 5-player games, the approximate execution times are $0.01 - 0.03$ s, $0.02 - 0.035$ s, $0.04 - 0.06$ s, and $0.055 - 0.07$ s, respectively. The corresponding execution times for the IPOPT solver and the Gambit solver are an order of magnitude less on the desktop machine (Intel Core i7 CPU @ 1.9 GHz); nevertheless the trends remain the same.

8.3 Results from the Game Theory-based Approach for RRT*+Vision

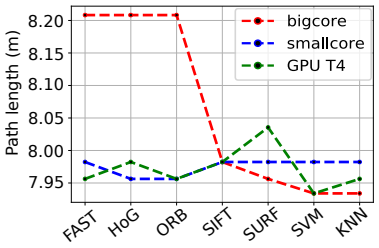

Fig. 23. Path length for the 20*in*4 drone

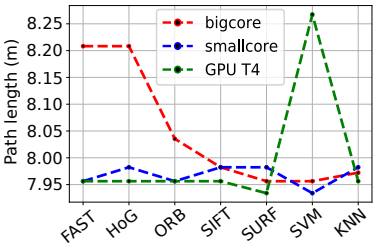

Fig. 24. Path length for the 20*in*8 drone

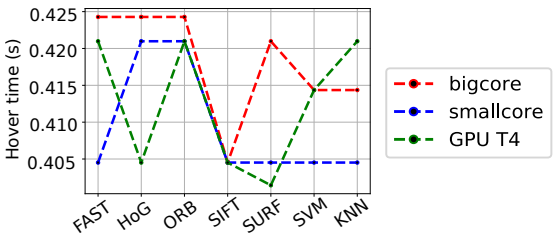

Fig. 25. Hover time for the 20*in*4 drone

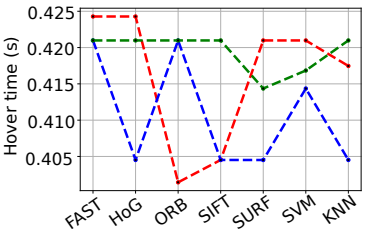

Fig. 26. Hover time for the 20*in*8 drone

Figures 23 and 24 show the path lengths corresponding to the optimal parameters *chosen* by the Gambit game theory solver for the 20*in*4 and 20*in*8 drones, respectively. Similarly, Figures 25 and 26 show the hover times corresponding to the optimal parameters chosen by the Gambit game theory solver for the 20*in*4 and 20*in*8 drones, respectively. We observe that the results are mixed for both the drones. All the three platforms achieve the best path length and hover time for at least one vision algorithm. However, a platform achieving a smaller hover time and path length does not mean that the corresponding platform is optimal and is *finally chosen*. The payoffs of the players contain many other variables, hover time and path length are just two of them. We shall see the results corresponding to the chosen platform in the next section.

8.4 Comparison of Game Theory and Optimization-based Approaches for RRT*+Vision: Gambit vs IPOPT

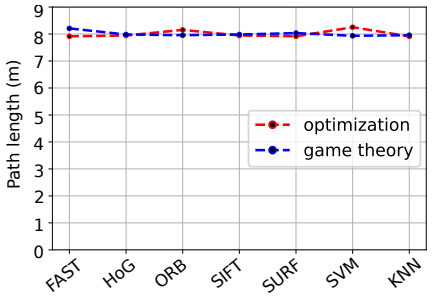

Fig. 27. Path lengths for the 20*in*4 drone

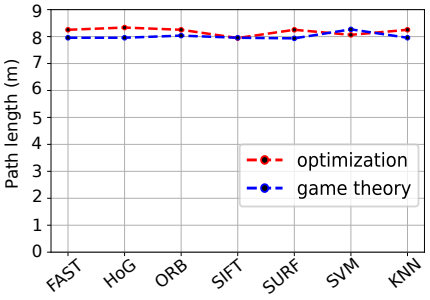

Fig. 28. Path lengths for the 20*in*8 drone

Figures 27 and 28 show the path lengths calculated using the optimization-based approach and the game theory-based approach for the 20*in*4 and 20*in*8 drones, respectively. The results are mixed. For nearly half of the vision algorithms, the optimization-based approach provides better path length while for the other half, the reverse is true (the difference is minimal). The maximum difference in the path lengths obtained by the optimal configurations found using the optimization-based approach and game theory-based approach is 0.35 m and 0.45 m for the 20*in*4 and 20*in*8 drones, respectively. The baseline path length for the corresponding scenario is 8.1 m. Hence, the two approaches are comparable in terms of finding the optimal solution, albeit the game theory-based approach is 10× faster.

Table 7 shows the chosen platform for each type of vision algorithm. We observe that even if a platform minimizes the path length and hover time; it is not necessary that it is ultimately *chosen*. This is because our optimization objective and payoffs are a combination of many factors where path length and hover time are just two of the many factors. Since there is sufficient variance in the power and performance of these algorithms on different platforms, this gets translated to varying path lengths and hover times. We can say that our formulation is fair and is not biased towards the platform with the lowest energy or the highest performance and is also not biased towards a parameter configuration with the lowest path length or the lowest hover time. Our payoffs favor those platforms and parameters for which the total energy gets minimized. It relies heavily on the characteristics of the algorithm on the three platforms, and the characteristics of the platform itself, which are captured in our payoffs. For the lowest energy point, the path length and the hover time can either be the smallest or anywhere in between. This is natural because their relationship is not linear. We also observe that for some algorithms, two platforms are very similar and optimal. For example, for *Fast*, BC+SC is the chosen platform for the 20*in*8 drone. This implies that both the platforms provide comparable payoffs and hence a mixed strategy can be played, where any of the two platforms can be chosen.

Table 7. Best platform

Algorithm	Optimization		Game	
	20*in*4	20*in*8	20*in*4	20*in*8
Fast	BC	SC	BC	BC+SC
HoG	GPU	GPU	GPU	GPU
Orb	SC	BC	SC+BC	BC
Sift	GPU	GPU	GPU	GPU
Surf	GPU	GPU	GPU	GPU
SVM	GPU	GPU	BC	BC+GPU
KNN	GPU	GPU	GPU	GPU
BC → big core, SC → small core				

9 THEORETICAL EQUIVALENCE OF GAME THEORY AND OPTIMIZATION PROBLEM

Until now we provided a lot of empirical justification of the equivalence of the game theoretic formulation and the optimization problem. We now provide formal guarantees using the well-established theory of potential functions. It is already well established that for non-cooperative games, there always exists a mixed strategy Nash equilibrium [29]. The theory of potential functions

states that if we can design a specially defined *potential function* that follows some properties corresponding to the game, then the potential function will be maximized when the game attains a pure Nash equilibrium [15].

For all the functions, we evaluate the difference in payoffs and potential function if we vary the strategy of just one parameter/player keeping the strategies of the rest of the parameters/players the same. Let us refer to these differences as ΔP and $\Delta\phi$, respectively. Based on the relationship of ΔP and Δphi, there can be five types of potential games [15]. Let us list a few prominent ones. If $\Delta P = \Delta phi$, then we have an exact potential game. If $\Delta P = w_i \Delta\phi$, then we have a weighted potential game. Here, w_i is a weight associated with the i^{th} parameter/player (one that is being varied). For our formulation, the most relevant was the best-response potential game, which states that the strategy of the i^{th} parameter/player that maximizes the corresponding payoff also maximizes the potential (rest remaining the same). Regardless of the type of the potential function, the basic guarantees are the same. Hence, it makes sense to define the potential function as a simple linear function of the objective that needs to be optimized.

In our case, we found that if the potential function is defined as the negative of the energy spent during a flight, then our game becomes a potential game. This formulation of the potential function is just the negative of the optimization objective that we use in Section 6. In order to identify the class that our potential function falls in, we performed analyses with the data collected from Gazebo simulations. We found that the game is a best-response potential game, implying that the strategy of a player that maximizes its payoff also maximizes the potential function, keeping the strategies of other players constant.

We plot the relationship between different parameters such as the #obstacles and resolution with the hover time and path length in Figures 17, 18, and 19. The hover time and path length determine the overall energy. It is important to note that the relations are mostly piece-wise linear. This can also be argued from the point of view of basic physics. For instance, if we increase the number of obstacles, the path length will keep increasing (in a mostly piece-wise linear fashion). This is because the number of free spaces to form a collision-free path will reduce and hence the planner has to take many detours, resulting in a longer path length. If we have K obstacles, we expect to add an additive constant $O(K)$ (order of K) to our path length. Similarly, the congestion in the search space increases with an increase in the number of obstacles leading to a higher hover time. The worst case hover time would be when the planner has to do re-planning at every obstacle. We will seldom have points of inversion, where more is better. Even if we do have such points, they will mostly be localized noise. For other parameters such as the number of samples or the dimensions, the relationships are mostly similar (sometimes they are more complex). For the resolution, there is an inverse relationship. Given this property, the weighted potential functions seem to be the right choice. However, we observed that the weights do not remain the same because the payoffs depend on strategies of the players too, which in turn affect the hover time and/or path length. This implies that with a change in the strategy of the player, either hover time or path length or both can change. However, the potential is always a function of both the hover time and the path length. Hence, we used the best-response potential function where the same parameter value maximizes both the potential function and payoff. The reason it works in this case is because all our payoffs are set as linear combinations of the parameter value and other objectives that it is linearly related to in its neighborhood such as the overall energy or path length. Because of this simple combination, the payoff increases or decreases monotonically with the potential function, that is, the overall energy. This is exactly whey the best-response potential method works – both are maximized at the same value.

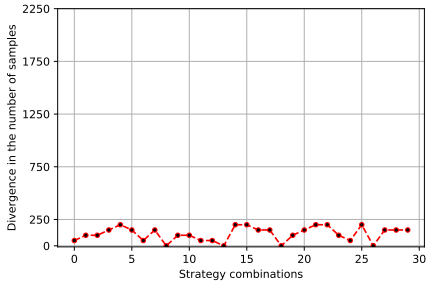

Fig. 29. Divergence in the strategy of the *sam* player

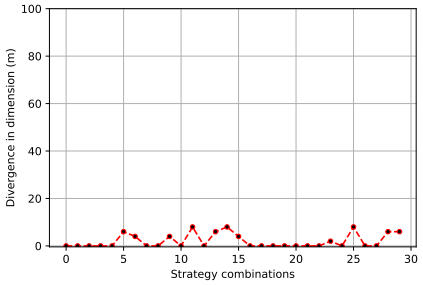

Fig. 30. Divergence in the strategy of the *dim* player

Let us now conduct an experimental investigation to exhaustively verify this fact. We shall show some of the representative results. Figures 29, 30, 31, and 32 show the divergence of the strategies that maximize the potential function and payoff of the *sam*, *dim*, *res*, and *obs* players, respectively. Here, *divergence* is defined as the absolute difference between the strategies that maximize the potential function and the payoff of the corresponding player, respectively. The divergence is defined as $Divergence_i = \arg\max_{p_i} P_i(p_{-i}, p_i) - \arg\max_{p_i} \phi(p_{-i}, p_i)$. Formally, $Divergence_i$ is the divergence of player i, P_i is its payoff, and ϕ is the potential function. p_{-i} denotes the strategies of all the players except player i, and p_i is the strategy of player i.

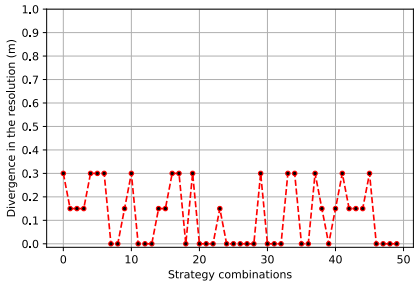

Fig. 31. Divergence in the strategy of the *res* player

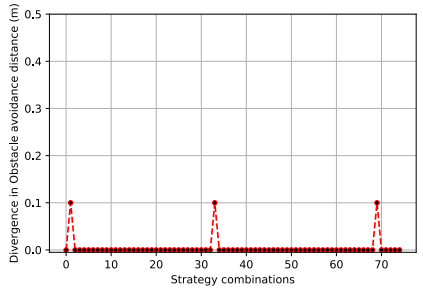

Fig. 32. Divergence in the strategy of the *obs* player

In all the figures, the maximum range of the y-axis is the maximum feasible value that can be taken by the respective divergence. Hence, we should interpret the plots as a set of relative values. We observe that for the *dim* and *obs* players, the divergence is mostly zero, indicating that it is the best-response potential game. For the other two players, the divergence is still relatively a very small quantity. Hence, our formulated game is an approximate best-response potential game, where the potential function is derived from the optimization objective. Given that we are modeling a practical system, we cannot expect ideal conditions to prevail. Hence, within these realistic limitations, the two formulations are equivalent.

10 CONCLUSION

In this paper, we propose a very novel solution to the parameter tuning problem for path planning algorithms used in UAVs. A path planning algorithm has many tunable parameters that play a significant role in deciding the overall length of the planned path. Additionally, several computer vision algorithms run on drones that are application specific, and sometimes assist in navigation.

Hence, the overall parameter space is huge, comprising the parameters of the path planning algorithms, vision algorithms, and choice of hardware (if there is any). Instead of solving this problem using traditional AI-enhanced optimization-based approaches, we take a diametrically different approach; we propose to convert a regular nonlinear optimization problem to a *game*, and quickly find the set of equilibrium strategies for the game. This gave us a 10× speedup without compromising on the quality of the solution. This work has broad implications beyond the scope of the current problem. This is the first paper to provide a novel, formal methodology to quickly solve complex joint optimization problems that usually arise in cyber-physical systems by converting them to equivalent non-traditional games. This approach has a lot of scope across all cyberphysical system domains.

The future work involves obtaining a better power model using the real-world drone data for many different environments and drone types. In a UAV pipeline, there are multiple tasks that execute concurrently. This work can be further extended to tune the parameters of the UAV pipeline that includes a choice of multiple algorithms and platforms for each of these concurrent tasks. Another useful direction could be to incorporate the dynamically changing obstacle density (in case of moving obstacles) in the formulation.

REFERENCES

[1] Bulat Abbyasov, Roman Lavrenov, Aufar Zakiev, Konstantin Yakovlev, Mikhail Svinin, and Evgeni Magid. 2020. Automatic tool for gazebo world construction: from a grayscale image to a 3d solid model. In *ICRA*. IEEE, 7226–7232.

[2] Hasini Viranga Abeywickrama, Beeshanga Abewardana Jayawickrama, Ying He, and Eryk Dutkiewicz. 2018. Comprehensive energy consumption model for unmanned aerial vehicles, based on empirical studies of battery performance. *IEEE Access* 6 (2018), 58383–58394.

[3] Logan DR Beal, Daniel C Hill, R Abraham Martin, and John D Hedengren. 2018. Gekko optimization suite. *Processes* 6, 8 (2018), 106.

[4] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Wenzhi Cui, Aleksandra Faust, and Vijay Reddi. 2018. Mavbench: Micro aerial vehicle benchmarking. In *MICRO*. IEEE, 894–907.

[5] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Bardienus Pieter Duisterhof, Brian Plancher, Kayvan Mansoorshahi, Marcelino Almeida, Wenzhi Cui, Aleksandra Faust, and Vijay Janapa Reddi. 2019. The Role of Compute in Autonomous Aerial Vehicles. *arXiv preprint arXiv:1906.10513* (2019).

[6] Ruben Burger, Mukunda Bharatheesha, Marc van Eert, and Robert Babuška. 2017. Automated tuning and configuration of path planning algorithms. In *ICRA*. IEEE, 4371–4376.

[7] José Cano, Yiming Yang, Bruno Bodin, Vijay Nagarajan, and Michael O'Boyle. 2018. Automatic Parameter Tuning of Motion Planning Algorithms. In *IROS*. IEEE, 8103–8109.

[8] Daizhan Cheng and Zequn Liu. 2020. Optimization via game theoretic control. *National Science Review* (2020).

[9] Sayak Ray Chowdhury. 2015. *A Game Theoretic Approach to Robust Optimization.* Ph.D. Dissertation. Indian Institute of Science Bangalore.

[10] Jason Clemons, Haishan Zhu, Silvio Savarese, and Todd Austin. 2011. MEVBench: A mobile computer vision benchmarking suite. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 91–102.

[11] Nvidia Corp. [n.d.]. *NVIDIA TURING GPU ARCHITECTURE.*

[12] Carmelo Di Franco and Giorgio Buttazzo. 2015. Energy-aware coverage path planning of UAVs. In *ICARSC*. IEEE, 111–117.

[13] Damion D Dunlap, Charmane V Caldwell, Emmanuel G Collins, et al. 2011. Motion planning for mobile robots via sampling-based model predictive optimization. *Recent advances in mobile robotics* 1 (2011).

[14] FAA. [n.d.]. *Fact Sheet – FAA Forecast–Fiscals Years 2016-37.*

[15] David González-Sánchez and Onésimo Hernández-Lerma. 2016. A survey of static and dynamic potential games. *Science China Mathematics* 59, 11 (2016), 2075–2102.

[16] Ramyad Hadidi, Bahar Asgari, Sam Jijina, Adriana Amyette, Nima Shoghi, and Hyesoon Kim. 2021. Quantifying the design-space tradeoffs in autonomous drones. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 661–673.

[17] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *LION*. Springer, 507–523.

[18] Marco Alberto Javarone. 2017. Solving optimization problems by the public goods game. *The European Physical Journal B* 90, 9 (2017), 1–7.

[19] Yao Ji, Chao Dong, Xiaojun Zhu, and Qihui Wu. 2019. Fair-energy trajectory planning for multi-target positioning based on cooperative unmanned aerial vehicles. *IEEE Access* 8 (2019), 9782–9795.
[20] Geesara Kulathunga, Dmitry Devitt, Roman Fedorenko, Sergei Savin, and Alexandr Klimchik. 2020. Path planning followed by kinodynamic smoothing for multirotor aerial vehicles (mavs). In *2020 International Conference Nonlinearity, Information and Robotics (NIR)*. IEEE, 1–7.
[21] Cédric Leboucher, Hyo-Sang Shin, Rachid Chelouah, Stéphane Le Ménec, Patrick Siarry, Mathias Formoso, Antonios Tsourdos, and Alexandre Kotenkoff. 2018. An enhanced particle swarm optimization method integrated with evolutionary game theory. *IEEE Transactions on Games* 10, 2 (2018), 221–230.
[22] Zhilong Liu, Raja Sengupta, and Alex Kurzhanskiy. 2017. A power consumption model for multi-rotor small unmanned aircraft systems. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 310–315.
[23] Jingru Luo and Kris Hauser. 2014. An empirical study of optimal motion planning. In *IROS*. IEEE, 1761–1768.
[24] Kotaro Maekawa, Shunsuke Negoro, Ittetsu Taniguchi, and Hiroyuki Tomiyama. 2017. Power measurement and modeling of quadcopters on horizontal flight. In *2017 Fifth International Symposium on Computing and Networking (CANDAR)*. IEEE, 326–329.
[25] McKelvey, Richard D., McLennan, Andrew M., and Theodore L. Turocy. [n.d.]. *Gambit: Software Tools for Game Theory, Version 15.1.1*.
[26] Rui Meng, Ye Ye, and Neng-gang Xie. 2010. Multi-objective optimization design methods based on game theory. In *WCICA*. IEEE, 2220–2227.
[27] Diksha Moolchandani, Anshul Kumar, José F Martínez, and Smruti R Sarangi. 2020. VisSched: An Auction-Based Scheduler for Vision Workloads on Heterogeneous Processors. *IEEE TCAD* 39, 11 (2020), 4252–4265.
[28] Diksha Moolchandani, Geesara Prathap, Ilya Afanasyev, Anshul Kumar, Manuel Mazzara, and Smruti R Sarangi. 2021. Game Theory-based Parameter-Tuning for Path Planning of UAVs. In *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*. IEEE, 187–192.
[29] John F Nash Jr. 1950. Equilibrium points in n-person games. *Proceedings of the national academy of sciences* 36, 1 (1950), 48–49.
[30] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. 2016. Continuous-time trajectory optimization for online UAV replanning. In *IROS*. IEEE, 5332–5339.
[31] Sanaa Oulaourf, Abdelfatteh Haidine, and Hassan Ouahmane. 2016. Review on using game theory in resource allocation for LTE/LTE-advanced. In *2016 International Conference on Advanced Communication Systems and Information Security (ACOSIS)*. IEEE, 1–7.
[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *JMLR* 12 (2011), 2825–2830.
[33] Alex S Prasetia, Rong-Jong Wai, Yi-Lun Wen, and Yu-Kai Wang. 2019. Mission-based energy consumption prediction of multirotor uav. *IEEE Access* 7 (2019), 33055–33063.
[34] Grand View Research. [n.d.]. *Commercial Drone Market Size, Share, and Trends Analysis*.
[35] Hazem Sallouha, Mohammad Mahdi Azari, and Sofie Pollin. 2018. Energy-constrained UAV trajectory design for ground node localization. In *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 1–7.
[36] Smruti R Sarangi, Rajshekar Kalayappan, Prathmesh Kallurkar, Seep Goel, and Eldhose Peter. 2015. Tejas: A java based versatile micro-architectural simulator. In *PATMOS*. IEEE.
[37] Mohammad Karim Sohrabi and Hossein Azgomi. 2020. A survey on the combined use of optimization methods and game theory. *Archives of Computational Methods in Engineering* 27, 1 (2020), 59–80.
[38] ArduPilot Dev Team. [n.d.]. Ardu Copter, Copter Home. http://ardupilot.org/copter/. Accessed: 2021-06-15.
[39] ArduPilot Dev Team. [n.d.]. Mission Planner, Mission Planner Home. http://ardupilot.org/planner/index.html. Accessed: 2021-06-15.
[40] Andreas Wächter and Lorenz T Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106, 1 (2006), 25–57.
[41] Fengjie Xu, Guolai Yang, Zixuan Li, Liqun Wang, and Quanzhao Sun. 2021. Electromagnetic buffer optimization based on Nash game. *Acta Mechanica Sinica* (2021), 1.
[42] Liang Yang, Juntong Qi, Jizhong Xiao, and Xia Yong. 2014. A literature review of UAV 3D path planning. In *WCICA*. IEEE, 2376–2381.