A Lightweight Drone Simulator

1st Evan Blake Putnam
Department of Computer Science
California State Polytechnic University, Humboldt
Arcata, California
ebp20@humboldt.edu

2nd Dylan Nihal Senarath
Department of Electrical and Computer Engineering
California State Polytechnic University, Pomona
Pomona, California
dnsenarath@cpp.edu

3rd Giselle Ramirez Urquijo
Department of Computer Science
California State Polytechnic University, Humboldt
Arcata, California
gu16@humboldt.edu

4th Cihan Tunc, 5th Renee Bryce
Department of Computer Science and Engineering
University of North Texas
Denton, Texas
{cihan.tunc,renee.bryce}@unt.edu

***Abstract*—Drones are valuable assets across industries for a variety of purposes, including but not limited to surveillance, transportation, delivery, smart agriculture, etc. However, many come with significant costs to purchase, configure, and deploy to study algorithms and how drones interact with each other. As a solution, drone simulators can provide a lower-cost option to investigate drones and related algorithms. Nevertheless, drone simulators on the market today tend to be computationally heavy, have a steep learning curve, and may be overwhelming for abstract problem-solving studies. There is a need for an inexpensive and easy testing alternative for researchers and for high-level algorithm development. In this paper, we present a lightweight and easy-to-use drone simulator that allows the definition of multiple drones with their sensors, actuators, speed, and battery limit as well as their communication among themselves. The simulator also allows the definition of the simulation world using small blocks that can be defined using different parameters. We presented two scenarios, such as package delivery and survey area coverage, to show the capabilities of the simulator. We are planning to share the simulator with the researchers.**

***Index Terms*—Drone, Unmanned Aerial Vehicle, UAV, Simulator, Package delivery**

I. Introduction

Although drones, or unmanned aerial vehicles (UAV), are conventionally thought of as tools for the military or toys for hobbyists, their usage extends far beyond as they have been considered for surveillance, transportation, delivery, disaster relief, search and rescue, firefighting, and smart agriculture [1]. Drones can outperform these traditional methods compared to other solutions such as helicopters because drones require little to no physical infrastructure, are able to traverse areas regardless of most terrain, are highly flexible, and can reduce costs [1], [2]. Furthermore, many modern sensors and actuators are small enough in size to be included in a drone. Thus, drones have been even used by major companies, such as Amazon Prime Air [3], (Alphabet-owned) Wing [4], and UPS Flight Forward [5] for air transportation. USA Federal Aviation Administration (FAA) shows that, as of now (July 2023), there exist 869,472 drones (348,057 commercial and 516,835 recreational drones) registered and 331,573 remote pilots certified in the USA (as of July 2023) [**?**]. Overall drone market was estimated to be around USD $27.4 billion in 2021 and is projected to reach USD $58.4 billion by 2026 [6].

As interest in drone technologies is growing fast, we need techniques to simulate drone operations because drone deployment in the real world can pose several safety challenges, especially in populated environments [7]. Additionally, it would be highly expensive and time-consuming to test a physical drone in the real world. These simulations can be stand-alone, approximate, and non-realistic or, on the opposite side, very accurate, as the simulated drone runs as a real one would with certain inputs and outputs [8]. In addition, these simulators should be capable of creating an environment to fly, allowing the use of sensors (e.g., cameras, lidars, GPS, microphones) and actuators (e.g., brushless or servo actuators), in which there are no unique simulators currently on the market that work for all these aims [9]. There exist multiple drone simulators such as XPlane [10], Flightgear [11], Gazebo [12], JMavSim [13], Microsoft Airsim [14], and UE4Sim [15] that have the capabilities of having many different physical representations to provide a realistic simulation. The greater majority of these simulators utilize a 3D engine, require expensive systems and expertise to set up, and can even have a licensing price. One widely adopted simulation software among them is Gazebo, as it provides a comprehensive framework to simulate drones and other 3D robotics environments with extensive features and flexibility. However, it also requires high computational complexity and has a very steep learning curve, which makes it impractical for abstract-level studies and experiments to create a drone-based environment. Furthermore, there are many aspects of research related to drones that simply do not require a 3D visualization, and a simple 2D model could provide a sufficient environment. We believe that there is a need for a more abstract, expandable, and easier-to-use drone simulator that still has many of the critical features and components contained within other simulators, but makes them more accessible to people/groups who do not have the experience with other simulators or do

not need all of the required features found in them. In this paper, we explain our solution to this problem, the creation of a lightweight drone simulator.

II. Related Work

Research for drones and their potential use cases rely on simulators and models created in order to experiment with lower costs, fewer resources, and even less time. Chowdhury et al. [16] and Rabta et al. [17] present models that focus on drones and some of the different aspects that can affect the usefulness of a drone in humanitarian efforts. Chowdhury et al. [16] presents a Continuous Approximation (CA) model for drones to transport emergency supplies in disaster situations, where decisions are represented as variables and hence CA can reduce the complexity of a model as a result. The affected region is first divided into sub-regions, where the CA model is used to determine system cost, ordering quantity, as well as the most optimal area to operate from. They are able to solve the issue with densely populated areas awaiting supplies as well as managing supply inventory in a way that minimizes cost. This model is useful for path planning dependent on unique situations but does not address energy issues that drones could possibly encounter in the field. Rabta et al. [17] presents an optimization model for drones in humanitarian situations that minimizes total travel distance to extend the operating distance of a drone. In this model, energy consumption is a function of payload and flight mode which considers the possibility of eliminating trips to base to recharge by having the drone instead stop at charging stations installed on the way to its destination. Eliminating the drone's trip to the base considerably increases its working distance and makes it more useful for supply drop-off missions. In addition to this, defined priority classes and priority rules that the drone follows were discussed that may also be added to further optimize the model.

Simulators currently available for public use offer similar features, with some more streamlined options for a specific purpose or product. UTSim [18] and Gazebo [12] are both flexible simulators that offer a multitude of options in order to test drone functionality and capabilities. UTSim is a user-friendly drone simulator built using the Unity Platform and its intended use is to simulate a variety of drones in environments that contain both static and moving objects. The simulation itself is highly customizable, with options available for manipulating the environment, as well as drones. By utilizing Unity as the simulator's base engine, UTSim is characterized as an easy-to-use and customizable simulation option for researchers. However because UTSim is using Unity as its engine, it runs the risk of having higher system requirements than some researchers may have access to. Gazebo is a popular, open-source robotics simulator that offers a variety of custom 3D landscapes. Gazebo's main purpose is to create realistic worlds that can be traversed by drones. This program allows you to build various models of drones, equipped with different arms and sensors that are then able to navigate through an environment. These models can be curated by researchers, which allows them to observe the drone-to-environment interactions in the simulation. Therefore, Gazebo comes with performance concerns due to system requirements.

Another simulation software popular in the industry is AirSim built using the Unreal Engine [19] that offers physical and visually realistic simulations for a variety of different autonomous vehicles including drones. It is open-source with its core components including its physics engine and various models for environments, drones, and sensors which all have independent utility from one another. AirSim aims to simulate scenarios as realistically as possible by using popular protocols such as MavLink to support its physics engine that operates at a high frequency.

FlyNetSim [7] and the simulator presented by Fernando et al. [20] are both examples of how simulators are developed to study a specific niche or product in the market. Fernando et al. [20] present a simulation developed in MATLAB Simulink specific to quad-rotor UAVs. The dynamics of the quad-rotor are modeled using Newton-Euler method which predicts the forces and torques the four propellers on the drone generate. Using the simulator built based on this model, various control algorithms can be created and tested. A scalable and flexible simulator, FlyNetSim, combines two open-source tools to observe and evaluate UAV swarms and how they operate. FlyNetSim interfaces the two open-source simulators, Network Simulator (NS-3) and ArduPilot, using a lightweight middleware layer to allow the simulator to analyze the large amount of UAVs necessary in swarm situations. Using this technology, FlyNetSim simulates the UAV network with considerably reduced system resource usage. Regardless of its ease of use, FlyNetSim limits itself by specializing in drone swarm scenarios whereas Fernando et al. simulator remains specific to quad-rotors.

III. Methodology

A. Overview and Assumptions

Our main assumption in this work is to create a lightweight and easy-to-use simulator that has the core components and modules comparable to the resource-heavy simulators currently being used in the world today with the intent to have an abstract way of defining the environment and drone behavior, as well as possible challenges, which can all be defined by the user. The following modules were used to create objects within our environment: Drones are capable of moving through a simulated 2D environment, sensing their surroundings, completing assigned tasks, and communicating with each other. We give a brief description of each of these modules to better understand how they work.

In our simulation, we define the smallest unit of time as a *'tick'* – the iteration of the main simulation loop. This limits the actions of individual components so that the proper order of operations is maintained chronologically.

B. Definitions

1) Drone: We define a drone as an autonomous vehicle capable of moving through a simulated 2D/3D environment,

sensing its surroundings, completing assigned tasks, and communicating with other drones. A physical drone consists of a number of components such as motors and propellers, a battery, a mission controller and planner (processors), some sensors (e.g., camera, GPS, and IMU), and actuators (e.g., servo actuators), the initial location, and speed. We define all these components in our simulation environment using a JSON file, which is a lightweight data interchange format. A sample definition is shown in Listing 1 where a drone is defined with an ID: 8, the battery is shown to have a maximum capacity of 250 mAh, and the drone consumes 10 mAh per tick. The drone is also defined to have a speed of 1 cell per tick with an initial position ("init_pos") of (2, 1, 0) in the simulation space. A position is defined by three integer values `X`, `Y`, and `Z`, where `X` is the lateral direction, `Y` is the longitudinal direction, and `Z` is the vertical direction. Please note that for the simplicity of the simulator, currently we have only implemented `X` and `Y`.

```
"drones":
[
    {
        "id":8,
        "battery_max": 250,
        "battery_move_cost": 10,
        "speed": 1,
        "init_pos": [2, 1, 0],
        "sensors" : {}
    }
]
```

Listing 1. Example of a section of a scenario JSON containing a drones list and one drone item.

When a drone is called in the simulation, it checks its current battery level to see if it is able to act or not, as well as if the network has any messages for the drone and processes them if any exist. Next, the drone decides how to act depending on its current state and whether the action for its assigned task is allowed by the simulation space. After the drone has acted, it sends a new network message containing information about itself, such as battery level, position, and its state. Finally, the drone passes information about itself to the logger about its state when it reaches the end of the tick.

a) Sensors: We define a sensor as an additional object as a part of the drone object. The main purpose of sensors within our drone is to provide the functionality to gather information on the environment based on the position of the drone and the attributes of its environment. By repeatedly sensing during every tick throughout the course of the simulation, we simulate a realistic environment that can be used by all drones for better route planning and task execution.

Each drone in a simulation can have zero or more sensors associated with it. We use the "direction" attribute to define direction relative to the drone, and "range" as the number of blocks around the drone that the sensor attribute can sense. Listing 2 shows an example of a section of a scenario JSON containing a sensors list with two sensors. The first sensor is a temperature sensor and has a range of 1 cell around the drone. The second sensor is a hypothetical sensor to detect color with a range of 5 cells in front of the drone to identify any possible light sources (including fire).

```
"sensors" :
[
    {"attr": "temperature",
     "direction": "NONE",
     "range":  1},

    {"attr": "color",
     "direction": "FORWARD",
     "range": 5}
]
```

Listing 2. Example of a section of a scenario JSON containing a sensors list and two sensor items.

b) Actuator: Actuators are defined with a similar structure to sensors and this function contains methods to grab, assign, and change a cell attribute when the drone at that position is needed for package delivery or other actions. Similarly to sensors, actuators also have "attr", "direction", and "range" sections that function in the same way. Actuators also have a "mode" which describes what actions they will be able to perform. A sample actuator is shown in Listing 3 where the actuator is defined to have a mode of "GRAB" to grab items and a direction of "BACKWARD" showing its location on the drone with a range of 1 cell.

```
"actuators":
[
    {"attr": "parcel",
     "mode": "GRAB",
     "direction":  "BACKWARD",
     "range": 1}
]
```

Listing 3. Example of a section of a scenario JSON containing an actuators list and one actuator item.

2) Simulation World: The simulation world is the representation of the real world in the simulation, which contains a 3D map (grid of cells) and interacts with it. For simplicity, currently, we are building a 2D map, but it can be extended to 3D. We define the smallest building block of a map as a *cell* to guide the drone (using it to show where the drone locates) and to contain the information about characteristics of that area such as possible obstacles (blocking the cell completely), weather information such as temperature, humidity, etc. Cell attributes are composed of a type and a value, where the type is a string and the value is no specified type. An example is "temperature" with a value of "72.0" (F°).

3) Drone Communication: In the real world, drones can communicate with each other to create a complete autonomy to coordinate their actions and avoid dangerous situations. Additionally, drones may have different levels of autonomy, where some drones may be controlled by a human operator and others may be fully autonomous. To simulate this, we model the communication between drones and also ground controllers using a network class that holds a list of messages with the capabilities of adding, removing, and searching for messages. We define messages as objects that contain a message type, sender device identifier, receiver device identifier, and contents, which is a dynamic list defined in message type (type and size). The network facilitates communication between devices (drones and ground controllers) in the simulation through a message

list, which holds and processes messages. These devices are able to able to search and filter for messages within the list that correlate to their specific device ID. In Listing 4, we provide a sample of communication of messages where there exist two drones and one ground controller. The drones are identified by their device ID of 1 and 2 and the ground controller is identified by 0. At the beginning of the simulation, the drones send messages for the path information and then their tasks are exchanged.

```
NET: 1: MessageType.INFO_UPDATE
NET: 1: MessageType.REQ_PATH
NET: 2: MessageType.INFO_UPDATE
NET: 2: MessageType.REQ_PATH
Controller: 0 ticking...
1.0909090909090908 : (0, 0, 0) (0, 0, 0) (5, 7, 0)
1.1111111111111112 : (2, 0, 0) (2, 0, 0) (7, 5, 0)
Drone: 1 ticking...
TASKING: IType.GOTO
Drone: 1 At: (0, 0, 0)
Drone: 2 ticking...
TASKING: IType.GOTO
Drone: 2 At: (2, 0, 0)
```

Listing 4. Sample communication details.

4) Tasks and Instructions: Drones operate either by directly communicating with the ground control stations (so that users can control) or by a list of instructions defined in their mission controller. To implement this behavior by creating tasks of the drones as a list of instructions defined by user. Some of the instructions currently implemented include, 'goto', 'fromto', 'wait', etc. We also included parameters such as execution time to show how many ticks are required to complete that task.

We define a task for a drone as an ordered list of instructions using a task identifier, a controller identifier, a drone identifier, and a boolean defining if it is tracked. The tracking boolean, if true, is used to demonstrate a task must be completed within its deadline and if that drone cannot accomplish that task in a timely manner, that task must be revoked and be assigned to a new drone. An example of such a case would be a drone encountering some interrupts causing too much battery use to complete the given task, the controller would then revoke the task reinserting it into the task queue.

C. Simulator Components

This section explains the individual modules used to create the simulator.

1) Configuration Parser: The configuration parser uses 3 main different predefined configuration files (as explained in Section III-B): environment, tasks, and drones. The environment configuration file contains the information necessary to create a simple grid-like simulation world (composed of cells). The drone configuration contains all the information for the individual drones in the simulation. Finally, the scheduler file includes a list of tasks that are distributed amongst all the drones in the simulator.

2) Simulation Engine: The simulation engine is responsible for the order of operations and mediating the interactions between objects contained for every tick (i.e., every single iteration in the main simulation loop defined as the smallest period of simulation time). Within the simulation, we created a tick function to execute functions (e.g., sensor and actuator functions) as needed in order. First, the scheduler function is executed, allowing tasks to be assigned to drones which provides them with their destinations and paths (i.e., which cells to visit). The drone is then ticked, which initializes its interaction within the simulation space, moving to the next iteration of its path array if available or following other logic depending on its state and task. At the end of the simulation tick, we can choose whether we want the simulation space to be displayed depending on the scenario settings hard-coded in the simulation module.

Another feature we included in our simulator is the "step mode" function which, if enabled, allows the user to either press the 'enter' keystroke to continue to the next tick of the simulation or type 'pause' which halts the simulation and allows user to select from a variety of commands, such as adding a new drone, changing a drone's target, etc. to alter variables manually or execute other tools. If the function is not enabled, the simulation will run through all iterations or ticks until the code is complete, without user input to step through each iteration.

When a drone travels from the source to the destination, the 'cost map' is calculated using Dijkstra's algorithm to find the shortest path cost. The path is initialized as an empty list which will store the positions that the drone will move through. We then create a 2D grid that represents the "cost" associated with a position. We initialize the costs for all points to -1 at the start, indicating that they are all unvisited cells. After creating a "point queue" list and setting it equal to the initial position, we can begin a loop that will continue until point_queue is empty or the destination is reached. For each of the surrounding cells around our currently examined position, we can check if the position is unvisited (still has a value of -1) or if the cost to reach that cell is less than the previous cost.

3) Scheduler: The scheduler is responsible for distributing tasks among the drones and ensuring those drones have all the resources (e.g., sensors, actuators, and battery) necessary to complete their given tasks; meanwhile, it also responds to requests for tasks, requests for paths, and drone updates, all of which are network messages. While an individual drone does not have a complete view of the space it is in, the scheduler keeps a record of every activity including what each drone has encountered and can also update its model of the simulation space based on drone updates.

The tick function of the controller works by using the 'get_messages' function from the network in order to load the list of open task messages into a local variable. We then iterate through each of the messages in the list and filter out the messages that are intended for that controller. We then process the remaining messages based on the message type and utilize various controller functions to do what is necessary for that task. For example, if the message queue has a message with message type 'REQ_TASK', the tick will filter this message out and call 'reply_req_task' which will process the message

and send the next network message for what it should do.

A sample is shown in Listing 5 where we define an individual instruction with an instruction type, priority, and parameters. The scheduler interacts with the network when it is ticked to process the message created by the instruction, such as 'GOTO', which sends a 'reply_task_message'. This back and forth between the network and controller continues as we tick the simulation, and thus the scheduler, until the messages associated with the task are fully processed. After the drones receive their path from the network, the simulation continues moving drones until the task is complete. It then closes the task and sends a 'GO HOME' command to return the drone to the base.

```
"controllers":
[
    {
    "tasks":
    [
        [ { "IType" : "GOTO", "Priority" : "LOW", "
    Position" : [ 5, 7, 0 ] } ],
        [ { "IType" : "GOTO", "Priority" : "LOW", "
    Position" : [ 7, 5, 0 ] } ]
    ]}
]
```

Listing 5. Example of a section of a scenario JSON containing one task and two task instructions.

D. Effector

The effector class was created to simulate rogue drone behavior for testing and realistic behavior purposes as in a real-world case, a drone can face indeterministic environmental changes (e.g., sudden weather change, sudden obstacles such as a bird) or cyberattacks and, thus may fail to complete the scheduled task. If the drone did not complete the task within the estimated amount of ticks the controller calculated, the drone's reliability and hence its trustability would be negatively affected.

In order the implement this behavior, we apply the following approach: After the initialization, the user can select from a list of simulation scenarios written in JSON to quickly and effectively change the simulation environment parameters. Once the user selects a scenario from the list, the simulation engine applies it to block cells so that the drone can not propagate, edit drone parameters, and fail to complete tasks.

After the initialization is complete, we begin operating through the simulation loop which is the actual driver of the program. The simulation is initialized with just 3 boolean values, representing the desire for logging simulation data, stepping through each tick, and seeing the visualized environment. The simulation engine ticks through iterations of the drones' positional movement, allowing us to see the updated environment after every step. After every tick, the simulation can accept either a pause or continue command. Pausing the simulation enters command mode where various options such as adding a new drone, adding a new blocked cell, exiting the simulation, etc. are available to choose from. Any updates to the simulation while in command mode are added and the simulation resumes with the updated environment when the "play" command is entered.

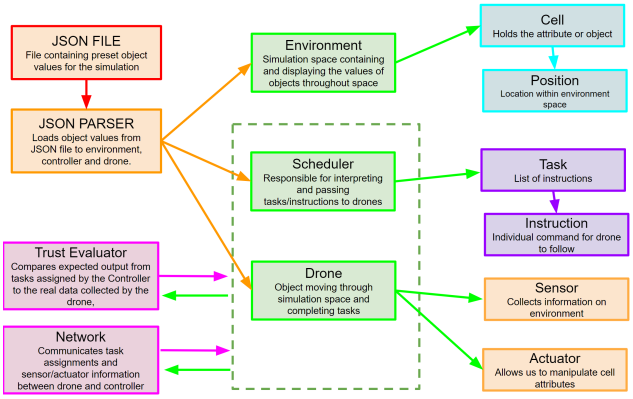

Fig. 1. Breakdown of connection between individual modules within the drone simulation.

E. Trust Score Evaluator

During the interactions of the drones with the simulated world using sensors and actuators, they can face various unpredicted events that can result in failure of a task, damage of drone(s), or even complete failure of drone(s). This would highly affect the trustability of the individual drones in a simulated environment similar to real-world scenarios. Therefore, to take these cases into account, we introduced a trust score evaluation to demonstrate the reliability of the individual drones based on their activities. At the beginning of a simulation, all drones are initialized with a Trust score of 1 to represent full trust, but throughout the simulation their trust scores are decremented due to factors such as pathing efficiency and remaining battery level. The updated scores can then be utilized by the scheduler to make informed decisions on task assignments. This means that if the scheduler has a new task to assign, it will take trust scores into account and avoid assigning high-priority tasks to a drone that has not been operating as expected. By eliminating faulty drones from task assignments, the overall performance of the simulation can be enhanced.

For the trust score evaluation, we use multiple formulations as follows. if the drone completed its task in the estimated amount of time calculated by the controller. As the ratio of estimated ticks to actual ticks gets larger, the trust score of the drone also increases with a max value of one, see Figure 2. The trust score is updated by multiplying the current score value by `Estimated Ticks/Actual Ticks` as shown in Equation 1.

$$Trust_{time} = CurrentTrust \times \frac{EstimatedTaskTime}{ActualTaskTime} \quad (1)$$

The simulator will trust a drone as long as it has 50% battery or more (with the assumption of the fact that a drone needs to leave from the source, finish a given task at the destination, and return back to the source). Once it reaches that predetermined threshold, the trust score begins to be decremented as shown in Figure 3 as shown in Equation 2.

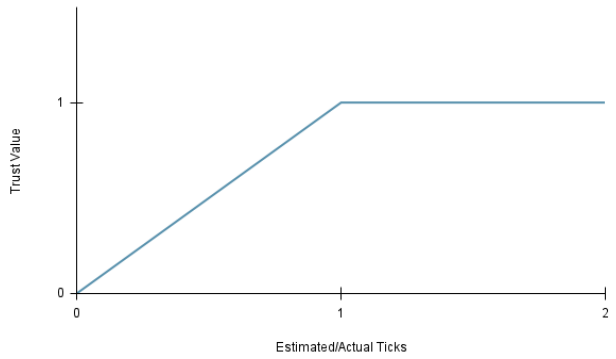

Fig. 2. Estimated/Actual Ticks vs Trust.

$$Trust_{battery} = CurrentTrust \times \frac{CurrentBatteryLevel}{MaxBattery \times 0.5} \quad (2)$$

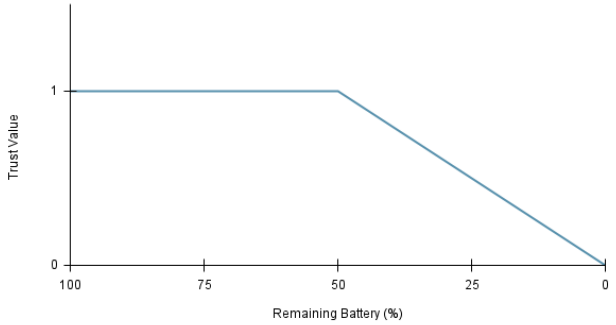

Fig. 3. Remaining Battery vs Trust.

We can introduce further trust evaluation formulations and then various techniques can be applied for a better scheduler decision, which can be calculated using Equation 3 where α and β can be decided based on the environment.

$$CurrentTrust = \alpha \times Trust_{time} + \beta \times Trust_{battery} \quad (3)$$

IV. Experimental Results

A. Package Delivery Scenario

The first scenario we present is package delivery (`package-delivery.json`), shown in Figure 4, which contains a simulation space with nine packages and locations for delivery for three drones. We also introduced three chargers next to the initial package locations so the drones do not run out of battery. We defined each package with cells containing an attribute called 'package'. The three drones have actuators in which the actuator attribute is the string 'package' and the mode is set to 'grab'. Figure 5 shows the steps a drone must take in order to complete an example task from the `package-delivery.json` scenario.

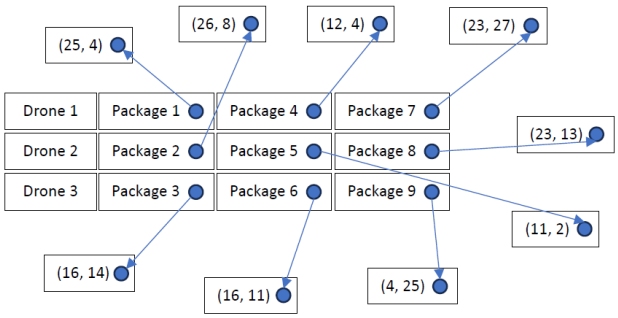

Fig. 4. Package destinations for package-delivery.json scenario.

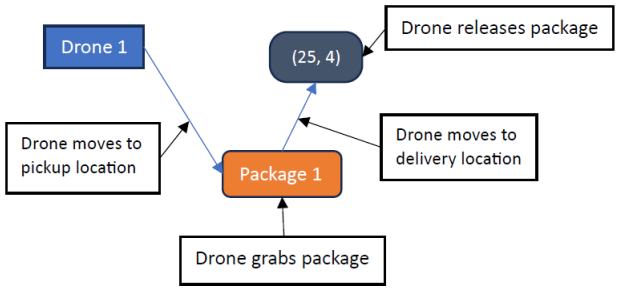

Fig. 5. Example of a singular task in the package-delivery.json scenario.

The execution of this scenario has no task interruptions (such as blocked cells, environmental hazards, or other conditions), which may impact a drone's ability to complete its tasks. The effector class is disabled to avoid incidents such as network disruptions or bad pathing. The expected result is that trust scores are 100% across all drones and tasks are completed as expected. This is done to verify the maximum trust score of 1 is achievable in optimal conditions.

We present the score of each drone during the execution in Listing 6 where we show the drone ID, their trust, and the estimated and actual ticks required to complete the tasks. The score (trust score) is 1.0 for each of them, which means it followed the expected path, took the estimated ticks, and had acceptable battery usage.

```
Drone_id: 3, Score: 1.0, Est: 20, Act: 20
Drone_id: 2, Score: 1.0, Est: 25, Act: 25
Drone_id: 1, Score: 1.0, Est: 27, Act: 27
Drone_id: 3, Score: 1.0, Est: 25, Act: 25
Drone_id: 2, Score: 1.0, Est: 30, Act: 30
Drone_id: 1, Score: 1.0, Est: 35, Act: 35
Drone_id: 2, Score: 1.0, Est: 30, Act: 30
Drone_id: 3, Score: 1.0, Est: 43, Act: 43
Drone_id: 1, Score: 1.0, Est: 36, Act: 36
```

Listing 6. Trust scores of each task displaying the drone identifer trust score as a percentage and the estimated and actual ticks the drone took to complete the given task.

The experimental run with blocked cells and the effector class partially active alters the resulting trust scores of the tasks completed. Within the experimental version of the scenario, there are fifteen added blocked cells that are not accessible to

drones. The effector class is minimally active, only extending the paths taken by drones so as to not interfere with any of the drone's ability to complete a given task. The scores and behaviors change as shown in Listing 7. We also demonstrate the environment using Figure 6 where drones are in the process of delivering packages. The integer values, e.g. '1' and '2', represent the device identifier of each drone. The character 'X' represents a blocked cell with obstacles that drone' can not enter. The character 'A' represents the final locations of the packages delivered. The character '·' represents an empty cell.

```
Drone_id: 3, Score: 0.909, Est: 20, Act: 22
Drone_id: 1, Score: 0.844, Est: 27, Act: 32
Drone_id: 2, Score: 0.625, Est: 25, Act: 40
Drone_id: 3, Score: 0.862, Est: 25, Act: 29
Drone_id: 1, Score: 0.733, Est: 33, Act: 45
Drone_id: 2, Score: 0.800, Est: 32, Act: 40
Drone_id: 3, Score: 0.915, Est: 43, Act: 47
Drone_id: 1, Score: 0.879, Est: 29, Act: 33
Drone_id: 2, Score: 0.810, Est: 34, Act: 42
```

Listing 7. Experimental run trust scores of each task displaying the drone identifer trust score as a percentage and the estimated and actual ticks the drone took to complete the given task.

```
*** Tick 134 ***
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . A . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . A . . . . . . . . . . . . A . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . X . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . X . . X . . . . . . . . . .
. . . . . . . . . . . . . . . . 1 . . X . . . . . . . . A . . .
. . . . . . . . . . . . . 2 . . . . . X . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . X . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . A . . . . . . . . . . . . .
. . . . . . . . . . . X . . . . . . . . . . . . . . X . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . .
. . . . . . . . . . . . X X X X X A . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . X . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . 3 . . A . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Fig. 6. Visualizer output from tick 134 of the package delivery scenario including blocked cells and effector active.

B. Covering a Survey Area

This scenario `survey-area.json` contains multiple cells with specific attributes that can be sensed by drones. The scenario's goal is to confirm that individual drones have the ability to properly sense their environment and send information via the network to the scheduler so that all drones have access to this newly sensed data. This is important because we depend on the sensors being accurate and reliable in order to detect obstacles and attributes that may affect their path. It is important to remember that in a real-life scenario, the drones may not have all necessary information about the environment prior to their run, so we may rely on the data collected by its sensors to update the interpreted environment.

We demonstrate the scenario using Figure 7, where we have a wall of cells with an attribute describing the current temperature. The drone in the scenario has been equipped with a temperature sensor. As the network receives messages about the newly sensed data from the drone, it will output the contents of those messages to the terminal window. We conduct a secondary check when sending an output message to the terminal, which ensures that a message has not already been sent. This will help to avoid duplicating messages sent to the network while the drone is making its return path.

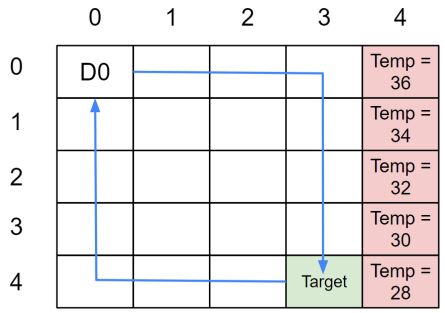

Fig. 7. Drone, D0, route while passing by a wall of cells with temperature attributes.

As we can see from the Listing 8, the drone was successfully able to sense its environment. As described in the methodology section for "Message" earlier, the messages have the following attributes: message type, sender id, receiver id, and a list of "contents". In this case, the sender ID is 1, as there exists only one drone. The message type is "cell attribute," which tells us that we have newly sensed data. From the 'content' attribute of the messages, we observe 3 pieces of information: the position of the sensed cell, the type of information that is held within it, as well as the value associated with that type of information. As expected, the first cell which is sensed within the environment is (4,0,0) which is the first block to appear on the right side of the drone while it is moving through its path. As the drone moves toward its target, it encounters the rest of the cells with attributes, first listing the reading at (4,1,1) all the way until it reaches its target and displays the final reading at (4,4,0). In this scenario, the drone was successfully able to read all 5 temperature values and send network messages for each one to allow public access to that information.

```
1, 0, MessageType.CELL_ATTR , content : (4, 0, 0) ,
    content : temperature , content : 36 ,
1, 0, MessageType.CELL_ATTR , content : (4, 1, 0) ,
    content : temperature , content : 34 ,
1, 0, MessageType.CELL_ATTR , content : (4, 2, 0) ,
    content : temperature , content : 32 ,
1, 0, MessageType.CELL_ATTR , content : (4, 3, 0) ,
    content : temperature , content : 30 ,
1, 0, MessageType.CELL_ATTR , content : (4, 4, 0) ,
    content : temperature , content : 28 ,
```

Listing 8. Output from terminal displaying the messages sent to the network due to the newly sensed cells.

V. Conclusion

As the drone market continues to grow, the need for research tools increases as well. Simulators are highly needed for drone operations because experimenting with a physical drone in the real world can be costly and unsafe. These simulators can have a wide variety of features depending on the need for the work. Most of the current simulators used in drone studies require powerful systems and expertise, which may be overwhelming for high-level decision-makers. Therefore, we need lightweight and expandable simulators that can be a help for high-level algorithm development and decision making. In this paper, we explain the necessary components and modules to create such a simulator. We have considered intricate details for components to work well together and created the lightweight tool. We plan to serve this simulator to the community as an open-source tool.

Acknowledgment

This research was supported by the National Science Foundation (NSF) Award Number: 2149969.

References

[1] E. Yağdereli, C. Gemci, and A. Z. Aktaş, "A study on cyber-security of autonomous and unmanned vehicles," *The Journal of Defense Modeling and Simulation*, vol. 12, no. 4, pp. 369–381, 2015.

[2] A. Nayyar, B.-L. Nguyen, and N. G. Nguyen, "The Internet of Drone Things (IoDT): future envision of smart drones," in *First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2019*. Springer, 2020, pp. 563–580.

[3] A. Staff, "Amazon Prime Air prepares for drone deliveries," Jun 2022. [Online]. Available: https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries

[4] "Wing." [Online]. Available: https://wing.com/(LastAccessed: July25,2023)

[5] "UPS Flight Forward adds innovative new aircraft, enhancing capabilities and network sustainability," March 2021. [Online]. Available: https://about.ups.com/us/en/newsroom/press-releases/innovation-driven/ups-flight-forward-adds-new-aircraft.html

[6] ReportLinker, "The global UAV market is estimated to be USD 27.4 billion in 2021 and is projected to reach USD 58.4 billion by 2026, at a CAGR of 16.4%," Jun 2021. [Online]. Available: https://www.globenewswire.com/news-release/2021/06/18/2249504/0/en/The-global-UAV-market-is-estimated-to-be-USD-27-4-billion-in-2021-and-is-projected-to-reach-USD-58-4-billion-by-2026-at-a-CAGR-of-16-4.html

[7] S. Baidya, Z. Shaikh, and M. Levorato, "FlyNetSim: An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. Montreal QC Canada: ACM, Oct 2018, p. 37–45. [Online]. Available: https://dl.acm.org/doi/10.1145/3242102.3242118

[8] F. D'Urso, C. Santoro, and F. F. Santoro, "An integrated framework for the realistic simulation of multi-UAV applications," *Computers & Electrical Engineering*, vol. 74, pp. 196–209, 2019.

[9] A. I. Hentati, L. Krichen, M. Fourati, and L. C. Fourati, "Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*. Limassol: IEEE, Jun 2018, p. 1495–1500. [Online]. Available: https://ieeexplore.ieee.org/document/8450505/

[10] "Most realistic flight simulator," 2023. [Online]. Available: https://www.x-plane.com/

[11] "FLIGHTGEAR FLIGHT SIMULATOR - sophisticated, professional, open-source," 2023. [Online]. Available: https://www.flightgear.org/

[12] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[13] "jMAVSim with SITL," February 2023. [Online]. Available: https://docs.px4.io/main/en/simulation/jmavsim.html

[14] "Microsoft AirSim," 2021. [Online]. Available: https://microsoft.github.io/AirSim/

[15] "UE4Sim: A Photo-Realistic Simulator for Computer Vision Applications," August 2023. [Online]. Available: https://repository.kaust.edu.sa/bitstream/handle/10754/626562/1708.05869v1.pdf;jsessionid=D54CAE107DB8D718AFD1832EAA73C86A?sequence=1

[16] S. Chowdhury, A. Emelogu, M. Marufuzzaman, S. G. Nurre, and L. Bian, "Drones for disaster response and relief operations: A continuous approximation model," *International Journal of Production Economics*, vol. 188, pp. 167–184, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925527317301172

[17] B. Rabta, C. Wankmüller, and G. Reiner, "A drone fleet model for last-mile distribution in disaster relief operations," *International Journal of Disaster Risk Reduction*, vol. 28, pp. 107–112, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212420918302000

[18] A. Al-Mousa, B. H. Sababha, N. Al-Madi, A. Barghouthi, and R. Younisse, "UTSim: A framework and simulator for UAV air traffic integration, control, and communication," *International Journal of Advanced Robotic Systems*, vol. 16, no. 5, p. 1729881419870937, 2019. [Online]. Available: https://doi.org/10.1177/1729881419870937

[19] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Field and Service Robotics*. Springer International Publishing, 2018, pp. 621–635.

[20] H. Fernando, A. De Silva, M. De Zoysa, K. Dilshan, and S. Munasinghe, "Modelling, simulation and implementation of a quadrotor uav," in *2013 IEEE 8th International Conference on Industrial and Information Systems*. IEEE, 2013, pp. 207–212.