Using Reinforcement Learning for Real-Time Trajectory Planning of Aerial Multi Agent Systems

Arya Kumar, Alan Zheng

Abstract
Control of aerial multi-agent systems, or drone swarms, is an open problem with many applications in rescue operations and defense. We find that Reinforcement Learning approaches, particularly Proximal Policy Optimization, are capable of real-time trajectory planning for these systems. We developed a real-time simulator based on the onboard firmware (CFsim), and applied existing control models, like Particle Field Controllers (PFCs). Our approach was able to extend PFCs to model an arbitrary number of constraints, such as battery life, time, and collision avoidance. We show that our approach is stable and performing it by providing a real-world demo using Crazyflie Unmanned Aerial Vehicles (UAVs or drones), and a Vicon Motion Capture System.

Keywords
Machine Learning — Reinforcement Learning — Trajectory Planning — Aerial Systems — Multi-agent Systems

Contents

1. Introduction

Across industries, there is a major push to transition from large, expensive, individual drones, to small drone swarms[1]. This is due to their high maneuverability, small size, low hardware costs, high fault tolerance, and emergent behavior[2]. Two major impediments to using UAVs concern path planning and autonomy. Solutions to problems that require perfect information about the entire environment are unrealistic for applications such as reconnaissance, combat, and search and rescue missions, where conditions constantly change on the fly[3], and controlling each drone individually would not be feasible[4]. However, examples from nature demonstrate that this autonomy should be achievable. Swarms of insects perform these desired behaviors with only limited knowledge of the world around them[5], while high density flocks of starlings outmaneuver hawks by keeping track of just seven of their neighbors[6].

Currently, the military has demonstrated swarms of UAV Perdix drones capable of "collective decision-making, adaptive formation flying, and self-healing." [7] However, none of this information or technology is publicly available. With perfect information, many efficient algorithms exist for computing good paths, using Ant colony based path planning [8] and Particle Swarm Optimization [9]. These algorithms work extremely well, and can even be programmed to make more or less aggressive maneuvers for greater efficiency [10]. For their light show of 1,218 Shooting Star drones at the 2018 Winter Olympics, Intel modeled the stadium in simulation, and pre-programmed the flight paths for each drone months in advance.[11]

Research has also begun for non-perfect information. These most commonly use reinforcement learning or genetic algorithms [12, 13] to solve problems, but results are still ongoing.

We attempt to address the hardest subset of these problems: imperfect information, real-time trajectory planning with a swarm of drones. We build a system that can run on-board existing industry drones, and can easily be extended to work with other systems and constraints.

2. Simulation Methods

The simulation involved three major components - a high fidelity realistic physics simulator, a stable control system[14], and a reinforcement learning architecture.

2.1 Physics Simulator

2.1.1 Requirements

Since we couldn't afford to break costly drones as they learn the basics of simple tasks, we wanted to train them first

within a physics simulator. We initially tried to use Gazebo 9, which is extremely high fidelity and CUDA compliant. However, it proved to be a problem because it was too slow, even when GPU-accelerated. Training was extremely slow, limiting the pace at which we could test and tune our models.

We turned towards creating our own simulator, CFFirm, one custom-made for our specific Crazyflie training and rendering purposes. After using Gazebo, we realized a couple things:

- The CrazyFlie firmware was already doing the position calculations
- There were SWIG libraries available to convert C to Python bindings
- By adding in Gaussian noise, the firmware could act as a suitable substitute for higher fidelity and accurate simulation.

2.1.2 Computation

Using SWIG libraries, we compiled portions of the Crazyflie firmware, such that we were able to query for future pose positions given a current pose and an action vector:

$$state_{future} \approx firmware(state_{current}, action) \quad (1)$$

By avoiding costly aerodynamics modeling and using the functions in the firmware, we created a significantly faster simulator:

Table 1. Comparison of Simulator Speeds

Simulator	Speed (compared to realtime)
Gazebo	0.47x
CFFirm	1937x

CFfirm was over *4000*x faster than Gazebo.

2.1.3 Visualization

The simulator also needed an appropriate visualizer - we turned to VisPy, a GPU accelerated high performance Python library.

We added on features as we needed them, and the final visualizer (Figure 1) was capable of:

1. Visualizing **points with various colors**, for A to B trajectories
2. Visualizing both **static and dynamic obstacles** with boxes of various colors
3. Manipulable **camera views**, to change the perspective in real-time.
4. **Breadcrumbs**, to model the path taken by the drones

2.2 RL Architecture

2.2.1 Algorithm choice

RL algorithms are computationally expensive[15]: our simulated drones have no labelled data and no ground truths

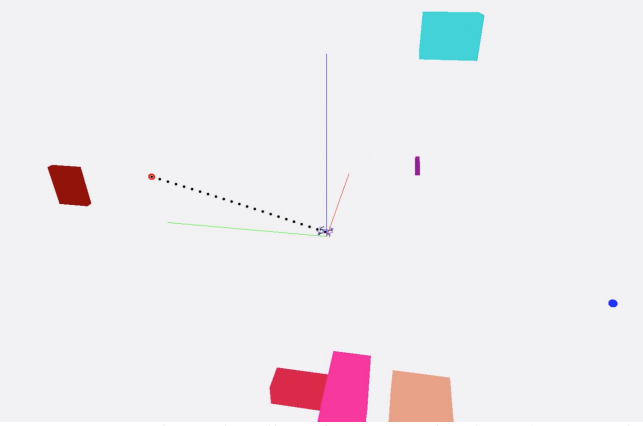

Figure 1. CFFirm Visualizer for DynamicObstacleEnv task

of the optimal way to avoid an obstacle or navigate to a given point. All they know is that they must travel efficiently as a swarm from point A to point B.

We settled on OpenAI's Proximal Policy Optimization (PPO) algorithm, which performs better than state-of-the-art approaches while being simpler to implement and tune. We find that PPO is an appropriate solution to our tasks, able to generate favorable trajectories in real time. Its rewards-based training makes it easy to add additional constraints when necessary. We address the rigidity of training RL models by providing a methodology for varying environments and creating effective reward functions. Each drone we train has limited knowledge about the surrounding environment; it only knows its global position and where the obstacles directly around it are. Because of this and the flexibility of training, the drones are able to navigate a variety of environments, even ones with dynamic states.

2.2.2 Environment Structure

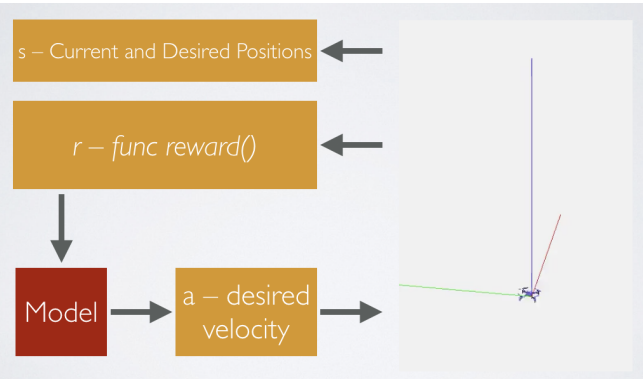

Figure 2. General learning structure for HoverEnv task

The overall learning model (Figure 2) takes in a observation state of two coordinates, or six numbers - the current position of the drone and its desired hover position. Next is the reward function, which is critical in reinforcement learning. Our model is trying to maximize a cumulative reward over some trajectory of actions:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t. \quad (2)$$

Time t goes to infinity since we want our model not to be fixed to a specific episode length. $\gamma \in (0,1)$ is our discount factor, which weights future rewards and more immediate rewards differently. It ensures that, under reasonable conditions, the infinite sum converges.

Our reward function is defined as a combination of three factors. The first two are the direction reward and the distance reward, which have to do with the vector between the current and desired points and helps the drone learn the general direction of the trajectory. The last one is the movement cost, which helps prevent oscillation upon arrival near the desired point. Based on the current state and reward, the model will generate a new action for the drone to take. Actions are in the form of a desired velocity vector. PID Controllers on board each drone convert from velocity vectors to motor speeds.

2.2.3 Implementation

For the actual implementation of the algorithm, we used the Stable-Baselines library, a fork of OpenAI Baselines that focuses on stability and performance. The implementation had a number of configuration options:

```
policy_kwargs = dict(act_fun=tf.nn.tanh, net_arch=[200, 200])

model = PPO2(
    MlpPolicy, env,
    gamma=0.99,
    n_steps=256,
    policy_kwargs=policy_kwargs,
    nminibatches=256,
    learning_rate=5e-4,
    cliprange=0.2,
    tensorboard_log='/home/um/tensorboards/',
    verbose=1
)
```

Figure 3. PPO configuration for HoverEnv task

As indicated by the *Tensorboard logs* keyword, it also integrated a number of debugging tools in the form of Tensorboards.

The library makes it easy to tune hyperparameters (Figure 3), and view the results on Tensorboards (Figure 4). These debugging tools proved invaluable when models would fail silently. Often by looking through the graphs and logs generated, we were able to determine that various parameters needed to be adjusted.

2.2.4 Training Goals

With the core structure in place, we began a list of environments that were progressively more complicated. We follow with a description of each environment, and some observations, if any.

Hover Beginning at a point near (0,0) on the ground plane ($z = 0$ m), hover the drone at a height of $z = 1$ m. To achieve stable convergence at the goal point, we built a reward function comprised of three parts. The model was rewarded for pointing in the right direction, minimizing distance to the goal, and minimizing extraneous movement.

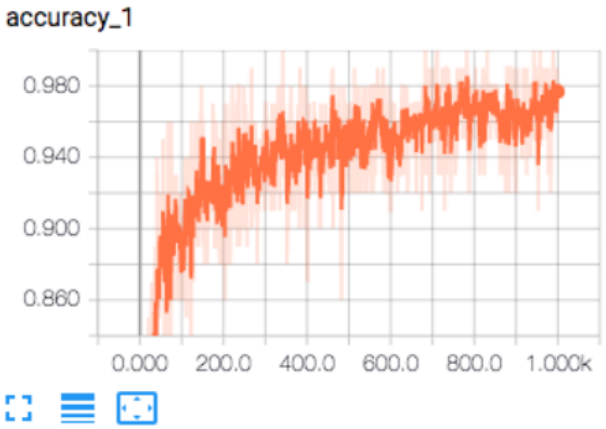

Figure 4. Sample Tensorboard accuracy graph

AtoB Travel from any arbitrary point A, to another point B. To avoid a similar "overfitting" of the reward function to the task, we explored alternatives. Eventually we settled on varied training as an alternative to a tuned reward function. This is consistent with the discoveries of Google Deepmind[16]

StaticObst + DynamicObst AtoB, but with obstacles in the way. We developed a method of modeling obstacles as positively charged conductors in a field, like a Potential Field Controller (PFC). By providing the model with a gradient and value of the field at the drone's location, the PFC was able to approximate an arbitrary number of obstacles of any shape and size. For the purpose of the simulation, we exclusively used cuboids aligned with the *xyz* axis.

StaticSwarm For our first foray into a multi-drone simulation, we initialized drones in random locations, and tasked them with converging on the origin without crashing into each other. This ended up being very similar to previous obstacle based tasks - we simply added a second layer of PFCs for other drones.

DynamicObstSwarm The AtoB task again, this time with a swarm of drones. This simplified to the question "can the model handle multiple PFCs at once?" We found that it could, when we increased the size of the network, up to 3 layers of 200 neurons, each using ReLU activation functions.

SwarmSwap A final task, to demonstrate the versatility of the model trained for DynamicObstSwarm. We instantiated two swarms at random points, and tasked them

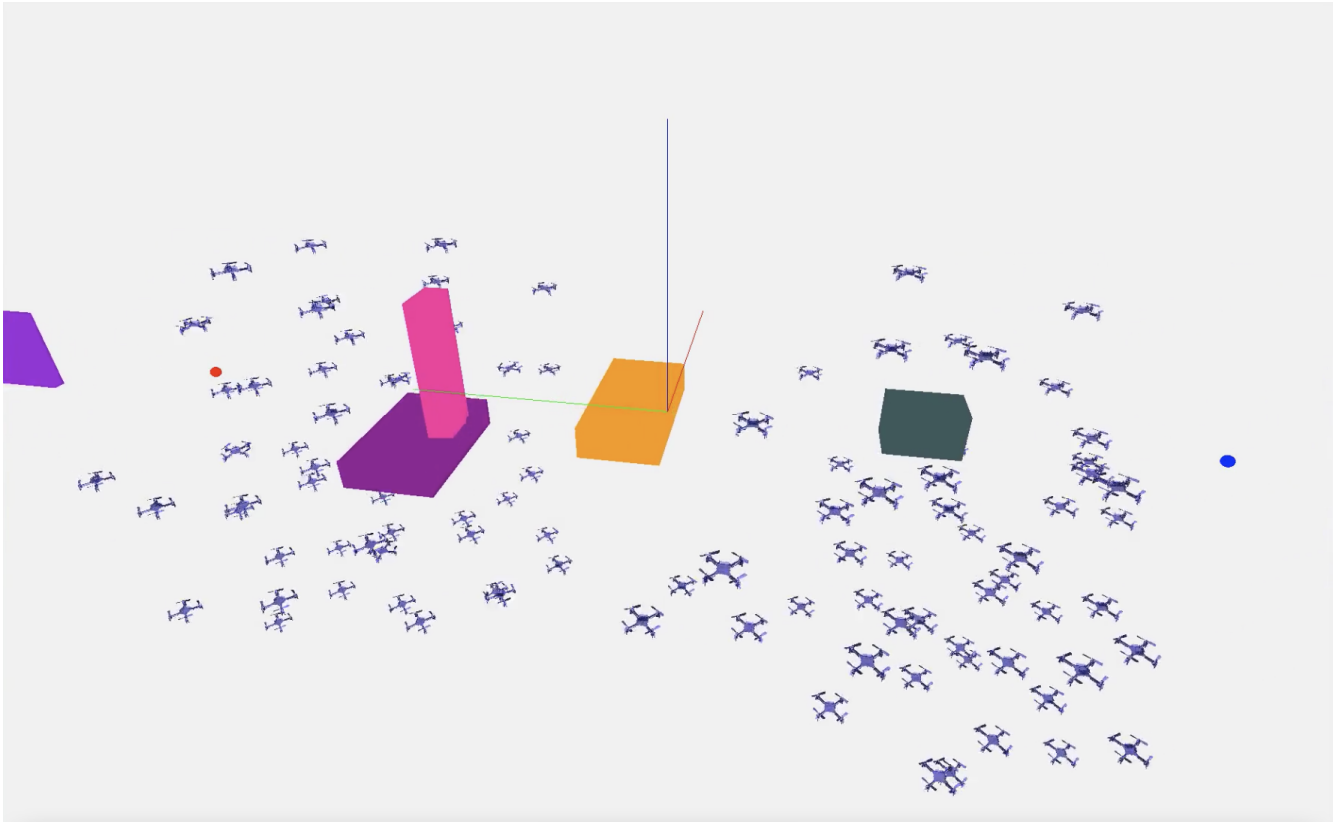

Figure 5. SwarmSwapEnv shortly after instantiation

with swapping locations without crashing into each other, or the obstacles in the way (Figure 5). Once we had trained a model capable of this task, we considered the simulation portion of the project complete.

3. Simulation Results and Discussion

We progressed through the simulations fairly consistently. The primary limitation with the model was that there was no sense of progression through time. We were unable to task a single drone to travel through a series of waypoints because the drone had trouble keeping track of the points that it had already visited. Usage of LSTMs instead of simple neural networks could help with this problem.

Another major observation was that varied training was a valid substitute for highly tuned reward functions. By working with a wide foray of starting positions, the model could generalize more effectively, and naturally seek optimums. This meant that we were able to drop portions of reward functions, such as the movement penalty from HoverEnv - given suitable training, models were able to converge on the optimal policy on their own.

In effect, the final policies were expansions upon their analogs in control theory, Potential Field Controllers. PFC's primary limitations are that simply moving in the direction of the gradient does not always yield an ideal path, and makes it challenging to model additional constraints. Through tailoring of the reward funciton to include additional incentives, like maximizing battery life, our approach makes it easy to expand upon controllers to tailor to situations.

4. Demonstration

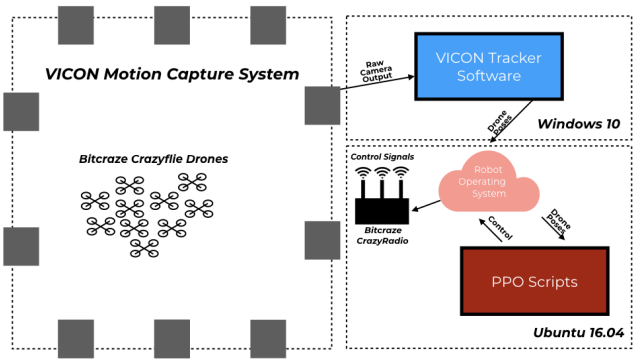

Figure 6. The Demonstration Setup

In the spirit of *facta non verba*, after training drones in CFFirm that could navigate all our designed environments reliably, we started working on using our scripts to fly real-life CrazyFlies. We used a Vicon motion capture system of 6 T40 and 4 T20 cameras to localize the drones.

Vicon Tracker software running onboard a Windows 10 machine is used to locate each drone by tracking the 4 reflec-

tive markers on it (Figure 7). The marker configuration was the same for each drone, and Extended Kalman Filters, or EKFs, are used to keep track of drones from frame to frame.

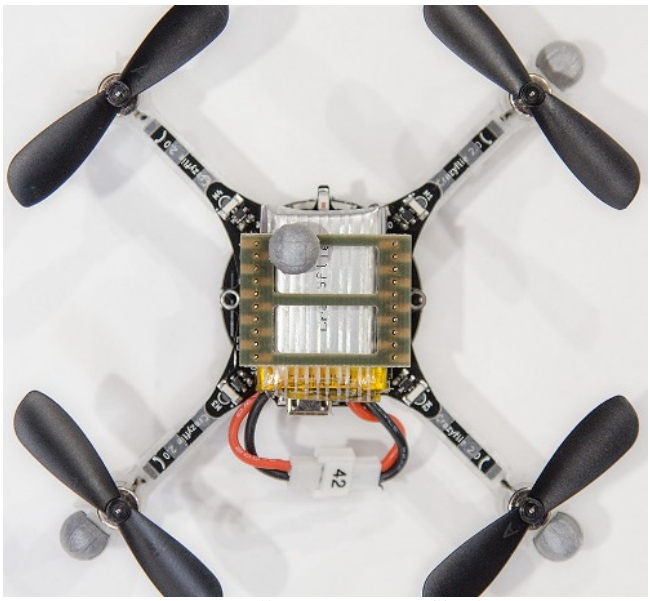

Figure 7. Fully set-up Crazyflie drone

We utilized the **crazyswarm** software created by USC's ACT lab, which allows for control of up to 50 crazyflies at once, using custom firmware that runs on board the drones. The software uses Robot Operating System, or ROS, to tie together pose updates from the Motion Capture system, and Planners, written in C or Python.

Once the planner, in this case our PPO Models, determines the correct course of action for each drone, information is sent through rostopics to the CrazyRadio antennae, which relay the information back to the drone through a 2MB/s 1.2GHz wireless connection.

With the full system in action, we were able to successfully run our models in real life, with minimal issues. We found that the varied training led to extremely stable models that successfully handled the inaccuracies in pose estimation from motion capture localization.

5. Conclusions and Open Questions

Our results are relatively unique in the field of Aerial Multi-Agent Systems. Most existing algorithms focus on perfect information approaches to downwash-aware controllers (Figure 9). Our approach is novel on that it focuses on being extendable, and is able to perform with a highly limited set of sensor information.

The architecture is highly versatile, and has proven stability in real world demonstration. In the future, we would attempt to connect our architecture to drones in the field. This research only considered localization errors by adding noise to the simulation. In practice, localization errors from SLAM, Optical flow, GPS and magnetization would have to be examined more carefully.

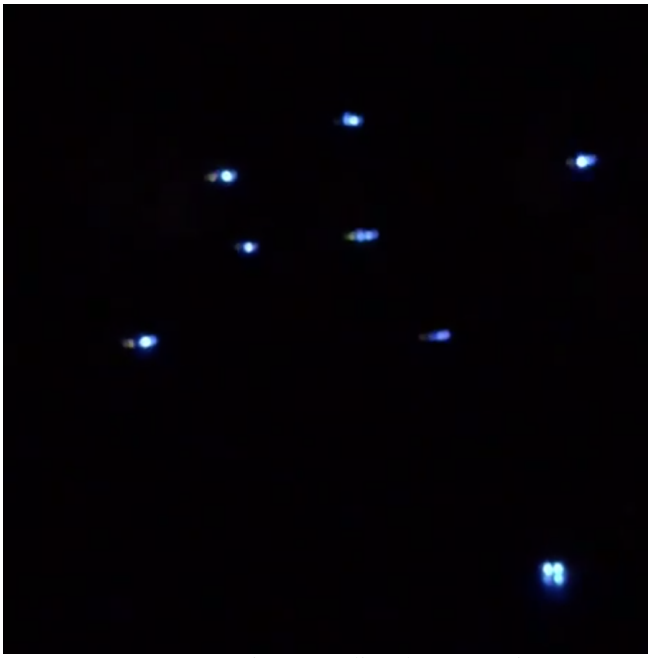

Figure 8. StaticSwarm live demonstration

Future research could aim to test the algorithm on drones that were viable for search and rescue, such as the Parrot Bebop 2.0. It could then be expanded with Computer Vision that could detect humans in rubble, and potentially deployed in disaster relief situations to aid rescuers in locating survivors after natural disasters.

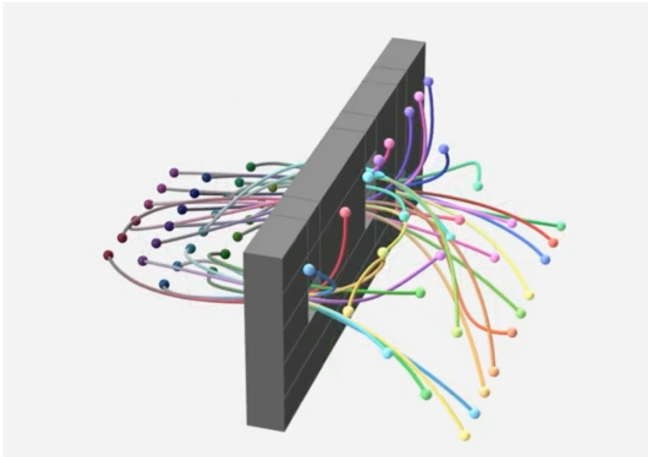

Figure 9. Example of perfect information flight path algorithms

6. Acknowledgments

We would first like to thank VICON for loaning us a free motion capture system for this research. We would also like to thank computer systems lab directors Peter Gabor, Patrick White and Shane Torbert for their guidance and oversight.

Finally, we'd like to thank our high school, Thomas Jefferson High School for Science and Technology, for providing us with the resources to make this project possible.

We would also like to acknowledge two part epoxy for being our savior when we discovered that super glue is not super.

References

[1] David Hambling. Drone swarms will change the face of modern warfare, Oct 2017.

[2] Dalal Prieditis. Smartswarms: Distributed uavs that think, Feb 2016.

[3] Fast lightweight autonomy (fla), Jan 2017.

[4] Suranga Hettiarachchi and William M. Spears. Distributed adaptive swarm for obstacle avoidance. *International Journal of Intelligent Computing and Cybernetics*, 2(4):644–671, 2009.

[5] Nicole Kobie. Drones inspired by insects could keep flying even when damaged, Jan 2017.

[6] Dario Floreano and Robert J. Wood. Science, technology and the future of small autonomous drones. *Nature*, 521(7553):460–466, 2015.

[7] Department of defense announces successful micro-drone demonstration. `www.defense.gov/News/News-Releases/News-Release-View/Article/1044811/department-of-defense-announces-successful-micro-drone-demonstration/`, 2017.

[8] Aakrati Agrawal, A.P. Sudheer, and S. Ashok. Ant colony based path planning for swarm robots. In *AIR '15 Proceedings of the 2015 Conference on Advances in Robotics*, number 61. Association for Computing Machinery (ACM), 2015.

[9] Xiaohui Hu. Particle swarm optimization. `www.swarmintelligence.org/`, 2006.

[10] Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. In *Robotics: Science and Systems VIII 28*, 2012.

[11] Intel drone light show breaks guinness world records title at olympic winter games pyeongchang 2018.

[12] Suranga Hettiarachchi and William M. Spears. Distributed adaptive swarm for obstacle avoidance. *International Journal of Intelligent Computing and Cybernetics*, 2(4):644–671, 2009.

[13] Chengyu Hu. Autonomous robot path planning based on swarm intelligence and stream functions. In *Evolvable Systems: From Biology to Hardware : 7th International Conference, ICES 2007, Wuhan, China, September 21-23, 2007 : Proceedings*, pages 277–284, 2007.

[14] Yi Wei, M. Brian Blake, and Gregory R. Madey. An operation-time simulation framework for uav swarm configuration and mission planning. *Procedia Computer Science*, 18:1949–1958, Jun 2013.

[15] Openai blog, Dec 2018.

[16] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017.