A Simulation of UAV Power Optimization via Reinforcement Learning
Amir Ehsan Niaraki Asli, Ali Jannesari
Department of Computer Science, Iowa State University, Ames, IA

Abstract
This paper demonstrates a reinforcement learning approach to the optimization of power consumption in a UAV system in a simplified data collection task. Here, the architecture consists of two common reinforcement learning algorithms, Q-learning and Sarsa, which are implemented through a combination of robot operating system (ROS) and Gazebo. The effect of wind as an influential factor was simulated. The implemented algorithm resulted in reasonable adjustment of UAV actions to the wind field in order to minimize its power consumption during task completion over the domain.

1. Introduction
The growth of global population by the yearly rate of more than 1% have raised the expectancy for food demand to be doubled by 2050.[1, 2] Recent statistics show that by applying digital agriculture, the yield of current worth 1.2 Trillion USD in world will increase by 70% and digital agriculture will touch market of 9.7 Billion by 2050. Particularly, the productivity of the farms can increase by 67% by that time, via the assistance of data-driven techniques.[3] Collecting this data and incorporating it into practical input for robots is a tedious portion of this required technological revolution in agriculture, and thusly a topic of extensive research.

One of the most common ways to collect data from the farms is using agriculture sensors, e.g., soil moisture, water sensor, and weather sensors. However, data collection through sensors suffer from some major limitations such as high installation cost and limited spatial coverage, and therefore is inflexible with on-demand collection needs. An alternative way for data collection and farm monitoring is to use unmanned aerial vehicles (UAV).[4] The attraction of UAV in practical data collection tasks can be attributed to the drop in cost of onboard sensors, flight control units and most significantly, small-scale embedded computing platforms. The development of UAV systems is however difficult, especially during test trails, due to the problems with site availability, weather conditions, potentially dangerous operations, and considerable resource requirements.[5]

Today, although quadrotors are popularly in use for farm data collection, they suffer from limited battery life. Sufficient aerial imagery is only achieved by multiple drone flights, which beside long wait time for recharging are challenging the applicability of these UAVs. Generally, wind plays the most significant role in power consumption of drones. It is shown that only change of yaw with respect to wind speed can improve the area covered by a single drone flight by 30%.[6] Therefore, power consumption optimization and path planning control of the UAVs with respect to wind is an attractive research topic during any intelligent task completion mission.

There are two folds to UAV control while addressing an autonomous task completion problem. Firstly, flight control inherently implies stabilization and control of aircraft typically done through an onboard
Flight Control Unit (FCU) in an “inner loop” level. Secondly, a control unit in “outer loop” level, is typically responsible for mission level objectives such as path planning, collision avoidance and navigation. Model-based UAV controllers are offer a path for accomplishing substantial autonomy in this secondary level. However, modeling the dynamics of UAVs in practice is extremely difficult and sometimes impossible, due to abundant uncertainties in novel conditions such as unforeseen environments. Another approach is to implement a learning algorithm that gives the agent an ability to adapt its behavior in case of changes in conditions. Classic reinforcement Learning (RL) is one of the most common learning frameworks in robotic applications that allows the agent to utilize direct interaction with its environment for training, with little to no prior knowledge. In this work, we aim to simulate an implementation of two common RL algorithms, Q-learning and SARSA for coverage path planning of a single quadrotor under various linear wind fields. The framework we present here employs conventional tools in robotics field, such as robot operating system (ROS) and Gazebo simulation environment.

The rest of this article is organized as follows: In Section 2, we provide a baseline for RL and how it is derived, alongside a brief description of Q-learning and SARSA, both well-known RL algorithms. In Section 3, we demonstrate the flight dynamic of a quadrotor and extract the parameters that are further used to define the problem statement in the simulation presented in Section 4. Section 5 gives the simulation results and adaptability of the proposed solution as well as its validation in a tabular case. Finally, Section 6 provides the closing remarks.

2. Reinforcement Learning Algorithms: Q-Learning and SARSA

Robot reinforcement learning is an increasingly popular method that can offer the ability to learn previously missing abilities. These can include behaviors that are priorly unknown are not facile to code or optimizing problems without an accepted closed solution. The behavior optimization occurs through repetitive trial and error interaction with its environment. This machine learning method can be defined as a Markov decision process (MDP) through which the agent is trained by an action-sense-learn cycle. In a standard model-based RL algorithm (Figure 1), the agent observes the state $s_t \in S$ from its environment, and takes action $a_t \in A$ based on the prior knowledge resulting in its current policy $\pi_t$. The taken action results in a new state $s_{t+1}$, which here can be determined from the state transition distribution $P(s_{t+1}|s, a)$ and leads to the reward $R(s, a)$.

![Figure 1. Standard network structure for reinforcement learning algorithm.](image-url)
The expected return (sum of discounted rewards) can consequently be used to give the optimal state-action value function for a given state-action pair \((s, a)\):

\[
Q^*(s_t, a_t) = R(s_t, a_t) + \gamma \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) \max_{a_{t+1} \in A} Q^*(s_{t+1}, a_{t+1})
\]

(1)

Where \(t\) can be an iteration numerator (or time-step), \(\gamma \in (0,1)\) is a pre-defined discount factor. Therefore, the agent learns to modify his action policy based on the cumulative rewards over iteration. Using this state-action value function, we can calculate the optimal policy, \(\pi^*\) by:

\[
\pi^* (s, a) = \arg\max_{a \in A} Q^*(s, a)
\]

(2)

Various RL algorithms mostly vary in terms of trade-off between exploration and exploitation in creating and updating the value function.[10] Here we will describe two of them and further implement them in the experimental scenarios.

2.2 Q-learning

Q-learning is an Off-Policy algorithm for temporal difference (TD) learning. While not requiring a model of the environment, based on an exploratory or random policy, Q-learning learns to optimize the policy when the actions are selected. In Q-learning, the learned action value function, \(Q\), directly approximates \(Q^*\) independent of the followed policy.[10]

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [R_{t+1} + \gamma \max_{a} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]
\]

(3)

Where \(\alpha \in (0,1)\) is the learning rate which is a hyperparameter like \(\gamma\).

2.3 Sarsa

Sarsa is an On-Policy temporal difference (TD) learning which uses the following algorithm to update its action value function, \(Q\):

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]
\]

(4)

The major difference between Sarsa and Q-learning is that Q-values are not necessarily being updated based on the maximum rewards. Instead, Sarsa uses every element of these five events: \((s_t, a_t, R_{t+1}, s_{t+1}, a_{t+1})\) that creates the transition from one state-action pair to the next.[10]

3. Quadrotor Flight Dynamics

In this section, we define a planar problem with 3 DoF which includes position in \(x\) and \(y\) axis and the heading angle \(\psi\) while the altitude of the quadrotor (position in \(z\) axis) remains constant. It is reasonable to consider the quadrotor as a rigid body, which accelerates by the torques and forces applied form its four rotors.
The velocity and applied wind is simplified in order to reduce the complexities derived by the physics of the UAV, and shown schematically in Figure 1. For the given velocities, we can have:

\[ X_G = V_w \sin \sin (\psi) + W_x \]  \hspace{1cm} (5)

\[ Y_G = V_w \cos \cos (\psi) + W_y \]  \hspace{1cm} (6)

\[ \dot{\psi} = \frac{V_w}{R_{\text{min}}} U (-1 < U < 1) \]  \hspace{1cm} (7)

Equation 4 is integrated with respect to time to give:

\[ \psi = \psi_0 + \frac{V_w}{R_{\text{min}}} Ut \]  \hspace{1cm} (8)

\[ \dot{X}_G = V_w \sin \sin \left( \psi_0 + \frac{V_w}{R_{\text{min}}} Ut \right) + W_x \]  \hspace{1cm} (9)

\[ \dot{X}_G = V_w \left[ \left( \cos \cos \left( \frac{V_w}{R_{\text{min}}} Ut \right) \sin \sin (\psi_0) \right) + \left( \cos \cos \left( \frac{V_w}{R_{\text{min}}} Yt \right) \cos \cos (\psi_0) \right) Ut \right] + W_x \]  \hspace{1cm} (10)

Equation 6 is integrated giving:

\[ X_G = -\frac{R_{\text{min}}}{U} \cos \cos \left( \psi_0 + \frac{V_w}{R_{\text{min}}} Ut \right) + W_x t + X_{G_0} \]  \hspace{1cm} (11)

Equation 2 is substituted into Equation 5 and then integrated to give:

\[ Y_G = \frac{R_{\text{min}}}{U} \sin \sin \left( \psi_0 + \frac{V_w}{R_{\text{min}}} Ut \right) + W_y t + Y_{G_0} \]  \hspace{1cm} (12)
The variables $\dot{X}_G$ and $\dot{Y}_G$, represent the UAV’s total velocity in the $x$ and $y$ direction respectively, relative to the ground. $W_x$ and $W_y$ are the wind speeds in the $x$ and $y$ directions respectively. $\psi$ gives the angular velocity, $R_{min}$ and $V_w$ represents the UAV’s minimum turning radius and speed. Finally, $X_G, Y_G$ gives $x$ and $y$ coordinate of the UAV while $\psi$ describes its heading angle. [12]

4. Problem Definition

The most prominent limiting factor for the autonomous flight of a quadrotor is its battery level, which should be constantly monitored by the RL agent and makes rerouting decisions such as returning to ground control station or (maybe more efficiently) going for less demanding objectives. Let us assume a very general task of gathering aerial images in a pre-defined domain while the validity of each image at each location deteriorate with a constant defined rate. The agent is supposed to keep his image map as updated as possible but the visits should not be ‘too frequent’. An actual application of this scenario can be labeling each image with the amount of crop, or being able to estimate the moisture of soil by certain confidence over time and different weather conditions.

Suppose the task is successfully completed at each episode if $\Phi(x, y, t) > \Phi_c$, while $\Phi$ is the objective function and $\Phi_c$ is a critical constant threshold value ($0 < \Phi < 1$). Meanwhile, the value of objective function can degrade with an arbitrary function as a characteristic of environment, $\frac{\partial \Phi(x, y, t)}{\partial t} = \lambda(x, y, t)$. The value of degradation rate can be constant in a very basic case and can vary to a fully stochastic model, and in real-world problem will be influence by several factors such as type of crop, temperature, wind direction, location of each point in domain. Figure 3 demonstrates the $20 \times 20, x - y$ domain. The ground control station is located in the first cell where the UAV can fully recharge its battery level in $T_B$ time.

![Figure 3. Problem domain with ground control station in very first cell and wind field.](image)

The effect of wind on the power consumption of any UAV, is dependent on various parameters, notably the geometry, weight and material and the components of the wind field. In a 2D case, wind field can have two components in $x$ and $y$ direction which can vary during the time: $W(t) = (W_x(t), W_y(t))$. We defined a simplified power consumption model, $P_t$, in the simulation environment for the basic scenario:
\[ P_t = c_1 \sqrt{((x_t - x_{t-1})^2 + (y_t - y_{t-1})^2 + c_2 (x_t - x_{t-1}) - w_{x_t}) + c_3 ((y_t - y_{t-1}) - w_{y_t}) + T(x, y, t)} \]

Through which \(c_1, c_2, c_3\) are custom set coefficients which respectively control the influence of change in location, and effect of wind field in x and y direction, and passage of time (which works in the agents disadvantage if it wants to stick to a certain location). The last term \(T(x, y, t)\) is a turbulence factor that can be any function, and typically exists in a farm due to agricultural machinery, and presence of other drones for multi-agent tasks. Having the power consumption, we can calculate the battery level \(s_b \in [0, 100]\). Similar to [8] the state of the agent at each time-step is \(S = \{s_x, s_y, s_b\}\), where \(s_x \in [1, 20]\) and \(s_y \in [1, 20]\), are integers representing the location of the agent based on the cells in the domain. The agent can chose to move to any adjacent cell, providing 8 possible actions at each cell, including the cells on the edge of the domain which will result in the reward of -100. An additional negative reward for charging was set to -30 for each time that the agent visits the ground control station. In the real-world problem, the whole scenario is nothing but providing an accurate map of \(\Phi(x, y)\) with the least consumed power. Therefore, the reward function is calculated based on a combination of battery level and accuracy of objective function given in equation 13 (\(x, y, t\) for all parameters is removed for simplicity).

\[ R(s) = c_{\Phi} (\Phi / \Phi_c)^u + c_p s_b ; \quad s_b(t) = s_b(t-1) - P_t \]  

Here, \(c_{\Phi}\) and \(c_p\) are the coefficients responsible for tuning the significance of the objective function in comparison with battery level and \(u\) is the parameter that represents a urgency of the maintenance for objective function. The algorithm to address this problem is given in Figure 4. For all the cases the discount factor, learning rate, were set to \(\gamma = 0.9\) and \(\alpha = 0.2\). For the cases with Q-learning algorithm, we have used an epsilon value that decays over time, starting greedily from \(\epsilon = 0.9\) and decaying by a factor of \(\epsilon \times 0.998\) until it reaches to the minimum value of \(\epsilon = 0.01\). This means that the agent starts operating by making stochastic decisions 90% of the time and eventually relies on the converge policy.

**Figure 4. Network structure for reinforcement learning algorithm**
5. Results

The experiments were implemented through a combination of ROS and Gazebo. Gazebo is equipped with a robust physics engine, which paves the way for taking in actual wind history and resulting in realistic power responses in various conditions. The \((x, y)\) location of UAV is extracted from the global position of the quadrotor in Gazebo. For the cells as big as \(1 \times 1 \, m\), the height of the quadrotor was assumed to be at \(z = 3 \, m\) to ensure that it will match the reality of the case as much as possible (Figure 5).

\[\begin{align*}
\text{Figure 5. Quadrotor in simulation domain.}
\end{align*}\]

In order to validate the implementation of the RL algorithms, the Q-learning scenario was compared to a MDP direct path planning problem in a \(7 \times 7\) domain, described here.\cite{12} Figure 6 shows the MDP simulation results for the given wind field in an agreement with RL simulation results which is achieved in 57 iterations. While a simple Q-learning algorithm converges to the diagonal path for a constant wind condition, due to non-linearity of the wind field in this validation case, the number of iterations for reaching a perfect agreement between the methods was extremely higher.

\[\begin{align*}
\text{Figure 6. A comparison of MDP (given in black) and RL (given in red) simulation results. The objective is to depart from the cell pointed by red dot reach to the black one.}
\end{align*}\]
After validating the implementation of the algorithm, the define problem in section 4 was initialized with drone starting at \((x, y) = (1,1)\). The threshold of objective function was set to a constant of \(\Phi_c = 0.7\), and wind field was remained constant. The resulting state value contour is given in Figure 7.

In order to compare Q-learning and Sarsa the average reward for the first 200 episodes is given numerically in Table 1. It was resulted that Q-learning algorithm learns faster due to the greedy search with high epsilon. In addition Q-learning can learn a policy even if taken actions are chosen randomly and evidently it showed a slightly larger standard deviation especially in early episodes. It seems that Sarsa demands the information to be stored for more iterations before the action values can be updated which on the other hand can be interpreted that the agent would make risky moves more frequently. [11] Due to the linearities in this study (constant wind field and constant fading time for the objective function) and limited size of the domain the heat map and value contour barely changed after 200 episodes.

| Episode Interval | Q-learning | Sarsa |
|------------------|------------|-------|
| 0-20             | -139       | -153  |
| 20-80            | -84        | -122  |
| 80-120           | -66        | -107  |
| 120-160          | -59        | -86   |
| 160-200          | -51        | -63   |

6. Conclusion and Future works

The future of this project involves replacement of the simulation environment with the in-field trials. Therefore, the process of policy optimization is happening in a learning training step, which in robot reinforcement learning is known as “mental rehearsal”. The agents have shown promising behavior in new wind conditions and our group is improving the simulation environment and tuning the developed model to be able to react efficiently to rapid wind changes or power consuming tasks.

Battery cost of an intelligent UAV is not only cause by the power consumption of the rotor, but also by the possible on-board processors which are much desired to eliminate the need to constant communication to ground control station. The present framework can be further expanded to include the power consumption of processors to make this simulation environment closer to the real-world application.
References

1. Cohen JE. Human population: the next half century. *Science* 2003; **302**: 1172–5.

2. Food Production Must Double by 2050 to Meet Demand from World’s Growing Population, Innovative Strategies Needed to Combat Hunger, Experts Tell Second Committee | Meetings Coverage and Press Releases. https://www.un.org/press/en/2009/gaef3242.doc.htm. Accessed 25 Sep 2019.

3. Godfray HCJ, Beddington JR, Crute IR, Haddad L, Lawrence D, Muir JF, et al. Food security: the challenge of feeding 9 billion people. *Science* 2010; **327**: 812–8.

4. Katsigiannis P, Misopolinos L, Liakopoulos V, Alexandridis TK, Zalidis G. An autonomous multi-sensor UAV system for reduced-input precision agriculture applications. *2016 24th Mediterr. Conf. Control Autom.* 2016. IEEE, pp 60–64.

5. Chen IY-H, MacDonald B, Wünsche B. Evaluating the Effectiveness of Mixed Reality Simulations for Developing UAV Systems. 2012. Springer, Berlin, Heidelberg, pp 388–399.

6. Kapetanovic Z, Diego S, Chandra R, Kapoor A, Sinha SN, Sudarshan M, et al. Farmbeats. 2017.

7. Koch W, Mancuso R, West R, Bestavros A. Reinforcement Learning for UAV Attitude Control. 2018; 1–13.

8. Imanberdiyev N, Fu C, Kayacan E, Chen IM. Autonomous navigation of UAV by using real-time model-based reinforcement learning. *2016 14th Int Conf Control Autom Robot Vision, ICARCV 2016 2017*; **2016**: 1–6.

9. Kober J, Bagnell JA, Peters J. Reinforcement learning in robotics: A survey. *Int J Rob Res* 2013; **32**: 1238–1274.

10. Sutton, R. S. and Barto AG. Reinforcement Learning: An Introduction (2nd Edition, in preparation). *MIT Press*. 2018.

11. Zamora I, Lopez NG, Vilches VM, Cordero AH. Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo. 2016.

12. Al-Sabban W, Gonzalez L. Wind-energy based path planning for electric unmanned aerial vehicles using Markov Decision Processes. *Proc ...* 2012.