Flight test of Quadcopter Guidance with Vision-Based Reinforcement Learning

Siddiquee, Manan; Junell, J.; van Kampen, Erik-jan

DOI
10.2514/6.2019-0142

Publication date
2019

Document Version
Final published version

Published in
AIAA Scitech 2019 Forum

AIAA SciTech Forum
7-11 January 2019, San Diego, California
AIAA Scitech 2019 Forum

10.2514/6.2019-0142

Flight test of Quadcopter Guidance with Vision-Based Reinforcement Learning

Manan Siddiquee*, Jaime Junell† and Erik-Jan van Kampen‡

Reinforcement Learning (RL) has been applied to teach quadcopters guidance tasks. Most applications rely on position information from an absolute reference system such as Global Positioning System (GPS). The dependence on "absolute position" information is a general limitation in the autonomous flight of Unmanned Aerial Vehicles (UAVs). Environments that have weak or no GPS signals are difficult to traverse for them. Instead of using absolute position, it is possible to sense the environment and the information contained within it in order to come up with a "relative" description of the UAV's position. This paper presents the design of an RL agent with relative vision-based states and rewards for the teaching of a guidance task to a quadcopter. The agent is taught the task of turning towards a red marker and approaching it in simulation and in flight tests. A more complex task of travelling between a blue and a red marker is trained in simulation. This work shows that relative vision-based states and rewards can be used with RL to teach quadcopters simple guidance tasks. The performance of the trained agent is inconsistent in simulation and flight test due to the inherent partial observability in the relative description of the state.

Nomenclature

IMU	Inertial Measurement Unit	UAV	Unmanned Aerial Vehicle
GPS	Global Positioning System	MAV	Micro Aerial Vehicle
UAS	Unmanned Aerial System	RL	Reinforcement Learning
UDP	User Datagram Protocol	MDP	Markov Decision Process
HTTP	Hypertext Transfer Protocol	INDI	Incremental Non-linear Dynamic Inversion
RGB	Red Green Blue	YUV	

I. Introduction

APPLICATIONS such as urban search and rescue, surveillance and infrastructure monitoring require Unmanned Aerial Systems (UAS) which can safely and autonomously fly in unknown environments without position information from the Global Positioning System (GPS). For these types of applications the Unmanned Aerial Vehicle (UAV) must perceive the environment to get an idea of where it is, identify where it needs to go and guide itself to its goal. Perception uses vision, distance sensors and other on-board sensors to come up with a relative description of the position and attitude of the UAV. These approaches are either computationally expensive or they require heavy hardware (for example laser distance sensors or stereo-vision setups) which can be challenging for Micro Aerial Vehicles (MAVs) with limited payloads. Furthermore, if the UAV is limited to using pre-programmed flight plans and routines for autonomous behavior, it cannot adapt to changing environments or mission requirements without human supervision. The limitations of current methods and lack of methods which enable higher levels of autonomy constrain their use in the aforementioned applications.

Presently, there are three main approaches to guidance in GPS denied environments. The earliest and simplest approach is dead reckoning. Estimates for the acceleration is obtained from the IMU, but it can also be obtained using novel methods such as visual odometry [1]. Using dead reckoning alone is problematic for UAVs due

*MSc. Student, Department of Control & Simulation, Faculty of Aerospace Engneering, TU Delft
†PhD. Student, Department of Control & Simulation, Faculty of Aerospace Engneering, TU Delft
‡Assistant Professor, Department of Control & Simulation, Faculty of Aerospace Engneering, TU Delft

to sensor noise and drift. Therefore, it is often fused with other localizing systems in order to improve their accuracy [2–5]. Another approach is replacing GPS with some other local integrated positioning system [5,6]. The problem with this method is the requirement of an external system. The final approach is using a laser rangefinder or a camera to create a map of the environment and localize the UAV within the generated map. This approach is called Simultaneous Localization and Mapping (SLAM) [7]. The drawback of this method is its computational cost which makes real-time applications challenging. Research into this method has yielded numerous guidance and navigation systems which use some form of speeded up SLAM, enabling GPS free navigation [8–10]. An example of a map generated using SLAM is shown in Fig. 1.

Figure 1: Maps of the first floor of MIT's Stata Center as presented in [8]. The left image shows the map created by the autonomously flying UAV developed in [8] using laser range finders. The right image shows the architectural floor plan.

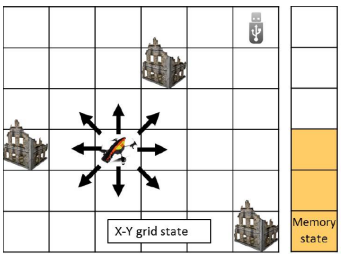
Vision is often a key element in many of these non-GPS localizing methods. The volume of information in vision and the lightweight nature of most cameras make it a desirable source of information required for guidance and navigation [5, 9, 11–13]. Bipin, Duggal and Krishna [14] used supervised learning to train depth estimates from visual data which is later used for trajectory planning. Purely vision-based SLAM is not very common. Weiss et al. [9] and Shen et al. [15] use purely vision-based SLAM for the autonomous navigation of MAVs. Research efforts to incorporate vision into the autonomy of UAVs has focused on specific tasks [9] such as landing [16, 17], hovering [18], corridor following and obstacle avoidance [19]. Some of these methods calculate optic flow [20] to estimate the attitude of the UAV and generate control actions. The accessibility of visual data and the relatively smaller amount of work carried out on the guidance of UAVs using vision-based reinforcement learning (RL) motivates its use in this study.

The core characteristic of autonomous systems is their ability to perform tasks with limited or no human interaction. Many of the work on the autonomous flight of UAVs in the previous decade used hard-coded software that has little or no learning capacity [13]; this limits their autonomy. In this regard, techniques from the field of Artificial Intelligence (AI) and robotics have been applied in UAS to incorporate learning and adaptation into its flight. Out of the machine learning methods available, RL promises the distinct advantage of being able to learn without supervision [21].

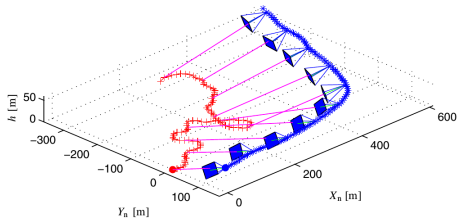
The increase in sensing and acting potential of UAVs brought on by improved electronics has encouraged the application of RL in its guidance and control [22–25]. Abbeel, et al. [26] demonstrated autonomous aerobatic maneuvers by a helicopter using inverse RL. It has been used to train optimal shapes for a morphing UAV at different flight conditions by Valasek et al. [27]. Valasek et al. [28] has also used RL to teach a simulated fixed wing UAV to track a target for surveillance using vision feedback; the path taken by one of their trained controller is presented in Fig. 2b. The performance of a non-linear autopilot is compared to a controller learned using RL in [29]. The study found that the designed non-linear controller slightly outperformed the RL taught controller. It has been used by Sharma [30] to train a UAV autopilot using a method called Fuzzy Q-learning and by Junell et al. [31] to tune the gains of a quadcopter using policy gradient RL. Further, it has been used for exploring an unknown environment and find paths to defined goals in [32] (see Fig. 2a). The ideas of RL have been used in [33] and [34] to come up with methods that enable UAV guidance and navigation in unknown environments with obstacles in the former study and cooperatively plan paths to avoid threats in the latter study.

A broader use of RL in UAVs is restricted due to several challenges with its implementation. Issues such as dealing with partial observability, the intractable state-action space for complex tasks and the trade-off between exploration and exploitation lead to problems in robotics applications [35]. Specifically for UAVs, RL implementations that aim to carry out on-line learning needs to ensure safe exploration [36] and fast convergence. This

factor discourages learning in flight tests with such systems.



(a) Representation of the guidance problem that was solved using RL in [32]. The agent is positioned in a 6x6 gridworld with the goal of finding an optimum route to photograph the ruins all the while accounting for available memory.



(b) Simulation results from [28]. Here an RL agent controlling a fixed wing UAV (blue) learns to give the right bank angle commands in order to keep a tracked target (red) in its view. This plot shows the trained agent tracking an erratically moving target.

Figure 2: Application of RL to UAV guidance

UAV guidance and vision-based RL have been researched as separate topic. However, there has been relatively little work on the application of vision-based RL for UAV guidance. This work aims to contribute to the filling of this "knowledge-gap" by implementing a vision-based RL guidance system in an AR Drone 2. First an RL agent learning a guidance task using vision-based states and rewards is simulated. Following this the agent is implemented in an AR Drone 2 and real life tests carried out. The differences between the simulation and practical tests (i.e. the "reality gap") is studied. The developed RL agent uses a vision-based state, thus there is no need for absolute position information. Furthermore, the UAV explores the environment by itself in order to find "good" paths to perform the navigation task. This makes the system more autonomous than one with a pre-programmed flight plan. The proposed method abstracts the mapping and localization by using relative vision-based states.

Simulation of the guidance tasks are carried out. The tasks consist of traveling to goal locations marked by colored rectangles. PaparazziUAV and FlightGear are used to simulate the RL agent. The learned policies are consequently uploaded to an AR Drone 2 and the performance of the real and simulated quadcopter are compared.

The next section (Section II) provides some background on RL and describes the algorithms used in this study. Section III discusses the navigation tasks, the environment of the guidance tasks and details about the agent. After this the simulations and the flight tests are expanded upon and their results explained in Sections IV and V. Finally, the report is concluded in Section VI with a statement of the findings of this paper and directions for further work.

II. Background on Reinforcement Learning

The following subsection provides an explanation of the elements that make up RL and its working principle. After that, an overview of two key concepts of RL, the ideas of value and policy, are explained. Lastly the learning method used in this work is described. The information provided in this section is sufficient to create the vision-based RL agent that will be taught a guidance task.

A. Overview of RL

RL is a machine learning method which can be used to teach a software agent sequential decision making tasks that have delayed rewards. In RL, the learning entity is called the agent. It acts in an environment, in order to accomplish a defined goal. While acting in the environment the agent senses its state and a scalar reward accompanied with every state transition. The reward is a scalar measure of how "good" it was to take the previous action from the previous state. The goal of an RL agent is to maximize the sum of future rewards that it can accumulate.

One of the most basic forms of the RL problem is characterized by a finite and discrete time Markov Decision Process (MDP). A MDP is a decision process whose state and state transitions satisfy the Markov Property. For a description of the state and the consequent state transition brought on by the environment to be Markov, future

states must only depend on the current state and the current action. This property is important for many RL methods as it allows learning based on the current state. Since all future states the agent can be in are fully determinable by its current state and action, the expected sum of rewards the agent can gather from any state is also function of its current state and action.

A finite MDP can be described using the five following elements: a finite **state space** (S), a finite **action space** (A), the **transition probability** ($P(s_{t-1}, s_t, a)$) between the states, a **reward model** ($R(s, a)$) and the **discount factor** (γ).

State Space (S) The state space consists of all the possible discrete states the agent can be in.

Action Space (A) The action space consists of all the actions the agent may take.

Transition Probability ($P(s_{t-1}, s_t, a)$) The transition probability is the system model. It describes the likelihood of ending up in state s_t given the agent takes action a from state s_{t-1}.

Reward Model ($R(s, a)$) The reward model describes the reward, r_t, that the agent obtains when it transitions to a new state.

Discount Factor (γ) RL tasks with a fixed ends are called episodic tasks. In general an RL task does not need to have a fixed ending, i.e. they may be non-episodic. The discount factor is used to make the sum of rewards to infinity bounded for non-episodic tasks.

B. Return, Value and Policy

The sum of rewards the agent obtains is called the *return*. RL agents aim to maximize the return. The *value* ($V(s)$) of a state is given by the expectation of the discounted sum of rewards from that state to the terminal state (Eq. (1)). An RL agent learns better ways of performing a task by acting in its environment and estimating the value of each of the states. The value is an estimate of the return that may be obtained from a state while following a certain policy. Being in states of higher value implies the agent will be able to accumulate more rewards.

$$V^{\pi}(s_t) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1}$$

The policy ($\pi(s, a)$) expresses the probability of taking an action when in a specific state. The value of a state depends on the way the agent acts, thus the estimated values in RL depend on a specific policy ($V^{\pi}(s)$). The optimal policy (π^*) maps the action which leads to the maximum return to every state (i.e. the best action is selected 100% of the time). A goal of RL is to find the optimal policy (π^*). Choosing an action which leads to the state of highest value is referred to as acting "*greedily*".

$$\pi^*(s_{t-1}) = \arg\max_a \sum_{s_t} P(s_{t-1}, s_t, a) V^*(s_t) \tag{2}$$

C. Q-Learning

Temporal difference (TD) learning [21] is a RL method which can be used to find optimal policies by estimating the value of each state through interactions with the environment. TD learning works by adjusting the estimate of the value of a state based on the reward and the value of the next state.

$$V(s_{t-1}) = V(s_{t-1}) + \alpha(r_t + \gamma V(s_t) - V(s_{t-1})) \tag{3}$$

On the one hand, the agent must explore the environment to estimate the value of the states. On the other hand, it must use the estimated values to act optimally. These two contradictory requirements lead to the dilemma of exploration versus exploitation in RL.

An approach to deal with the dilemma is using a policy that encourages random actions at the start of learning and optimal actions at the end of learning. This study uses the ϵ-greedy policy which picks a random action with probability $1 - \epsilon$ and a greedy (or optimal) action with probability ϵ[a]

Through the use of such a policy, a value approximation relation of Eq. (3) and an optimal action definition of Eq. (2), it is possible to come up with progressively better policies by interacting with the environment. With

[a]Many prefer to use another interpretation of ϵ, where a higher ϵ implies more random actions

sufficient exploration and improvement of the policy, the optimal policy will be found given the problem is a discrete time MDP [37].

However, without a model of the system one cannot find the optimal actions from the state values ($V(s)$) alone as the agent has no way of knowing how every action causes the state to transition. The model is needed to estimate the optimal action shown in Eq. (2). There are numerous solutions to this problem; the one used in this study is Q-Learning as proposed by Watkins [38]. In Q-learning state-action values (or Q-values) are used instead of state values to bypass the need for a model.

$$Q(s_{t-1}, a_{t-1}) = Q(s_{t-1}, a_{t-1}) + \alpha \left[r_t + \gamma \max_a Q^\pi(s_t, a) - Q(s_{t-1}, a_{t-1}) \right] \tag{4}$$

$$\pi^*(s_t) = \arg\max_a Q^*(s_t, a) \tag{5}$$

Estimating the Q-values for acting optimally allows the agent to select the best actions without a transition probability as it can pick the action with the maximum value from the state it is in (Eqs. (4), (5)). An added advantage of Q-learning is its off-policy learning capacity. This makes any policy that sufficiently explores the state-action space a viable option for approximation of the Q-values. This is the case as Q-learning always makes the Q-value updates based on the next optimal action (Eq. 4).

III. Tasks and Agent

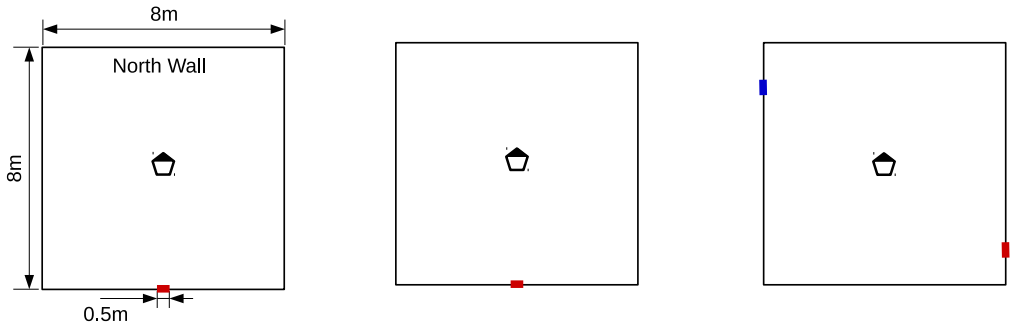
This section describes the guidance tasks, the designed RL agent and a rule based controller for performance comparison with the RL agent. The exact nature of the vision-based state and the reward scheme selected are described. Following this, the performed simulations and their results are discussed in Section IV.

A. Guidance Tasks

The guidance task consists of approaching fixed markers in an obstacle free 8m by 8m square room (Fig. 3a). The goals are physically represented by colored rectangles of dimensions approximately 42 cm by 60 cm (the dimension of an A2 papers). Two guidance tasks are defined:

One goal One red goal is placed at the middle of the South wall. The quadcopter must turn and approach the goal until a threshold amount of red pixel is seen. With fixed initializations, the quadcopter starts facing 180° away from the goal at a distance of 4m from the goal (Fig. 3b).

Two goal A blue goal is placed near the northern side of the East wall and a red goal is placed near the southern side of the West wall (Fig. 3c). The quadcopter must approach both goals until the threshold amount of the specific goal is seen. With fixed initializations, the quadcopter starts in the middle of the room facing the North wall; one goal is in its front left and another in its rear right .



(a) Dimensions **(b) One goal task** **(c) Two goals task**

Figure 3: The dimensions of the environment, position of the goals for the two tasks and their initialization state

The room for the flight test is a 10m by 10m enclosure called the Cyber Zoo. It is equipped with a motion sensing system[b] which accommodates experiments with robots. The quadcopters environment is constricted to an 8m by 8m region bounded by virtual walls inside the Cyber Zoo. For the simulation, an approximate 3D model of the 8m by 8m environment of the quadcopter in the Cyber Zoo is created. Snapshots from the real and the simulated environments are presented in Fig. 4.

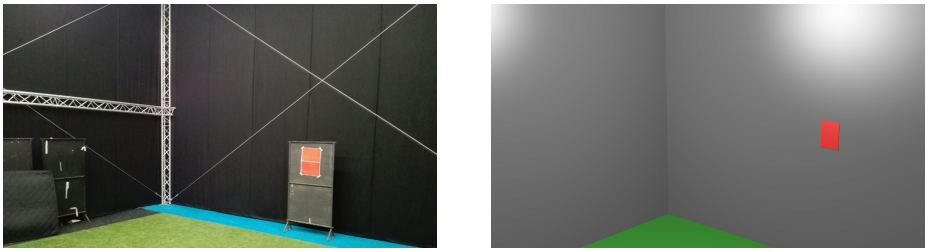


Figure 4: A picture of the real environment (left) and simulated environment (right).

The tasks are simulated with two sets of actions in order to look at the effect of incorporating expert knowledge into the RL agent through pre-programmed actions. Additionally, the effects of fixed and random initialization are studied.

B. Agent

This subsection discusses details about the agent: the vision-based state used by the agent is described, its learning and exploration scheme are discussed, the two action sets used in this study are presented and the reward scheme is characterized.

1. Vision-Based State

The separate components of the vision-based state is shown in Fig. 5. The first component of the state consists of three integers which represent seeing the goal at different parts of the quadcopter's field of view. The second component represents the amount of pixels being seen by the quadcopter and provides a sense of distance to the goal. The third and fourth components are memory states; the third component represents which goals have been visited and the fourth component represents a collision with the wall during the previous action.

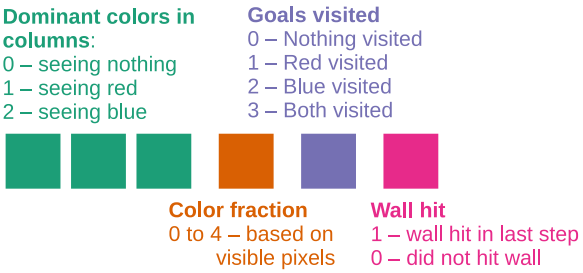


Figure 5: Representation of the vision-based state and its constituents

The first and second components of the state contain the information required to guide the agent towards the goal. The task requires the agent to approach colored markers (goals). Thus, the agent has to be able to distinguish the colored markers and get an idea of where it is relative to the markers from vision information. The first component of the state represents the information about the specific colored marker being seen and the relative lateral location of the marker with respect to the quadcopter.

The image frame is divided into three columns and the number of pixels above predefined thresholds are counted in each of columns. The three integers represent the detection (or no detection) of colored pixels above the threshold in the three columns. Some examples of what the quadcopter sees and the corresponding first

[b]Optitrack: Motive Tracker http://www.naturalpoint.com

component of the vision-based state are shown in Fig. 6. The integers take a value of "0" if there are no pixels above the threshold, a value of "1" if there are pixels above the red threshold and a value of "2" if there pixels above the blue threshold. Therefore the state "0,0,1" represents the presence of red pixels (above the threshold) in the right column.

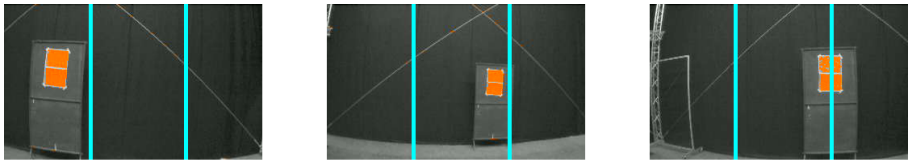


Figure 6: Three snapshots of the quadcopter's vision during simulation. The segmentation of the image into three columns and the consequent dominant columns part of the vision state have been depicted.

The detection of the red or the blue goal is based on the detection of pixels whose values are higher than a threshold. The *thresholding* for the detection is in RGB color space in the simulation and in YUV color space in the flight tests. In simulation, the RGB values of each pixel is summed and the percentage of red and blue calculated. A pixel is considered a red or a blue goal if more than 75% of its value is in the respective color. In flight tests, a pixel is considered red if its U components is less than 0.0 and its V component is greater than 0.13. The thresholds for detecting the blue are not defined for the flight tests as only the one goal task is performed in real life.

The second component (fourth integer) of the state represents how much of a goal is being seen by the quadcopter, and is a relative description of the distance to the goal. The sum of pixels above the threshold is divided by 5000 and the resulting number floored to obtain a discretized representation of the number of pixels above the threshold which are being seen. This integer is named the "Color Fraction" in this study. Its other use is deciding if a goal has been visited. In the simulations, the red goal is considered visited if the color fraction is greater than "3" and the blue goal is considered visited if the color fraction is greater than "2". The difference in the two thresholds is due to the lighting in the models of the environment which makes the blue goal less visible. Hence the range of values for this component of the state is between "0" and "3".

The third component (fifth integer) is a memory state which is relevant for the two goal task. This integer remembers which of the goals has been visited. It has a value of "0" when neither goals have been visited, a value of "1" if the red goal has been visited, a value of "2" if the blue goal has been visited and a value of "3" if both goals have been visited. For the one goal task this component can take one value; for the two goal task this state has three possible values.

The fourth component (sixth integer) of the state represents a collision with the wall caused by the previous action. This information is used to teach the agent to avoid walls . Its value is "1" if a forward movement in the previous action would have resulted in a collision with the wall. In such a case, the quadcopter does not make a forward movement[c]. For all other actions, its value is "0".

Such a state description leads to a total permutation of 64 states for the one goal task[d] and 748 states for the two goal task[e]. However, in reality the number of states are lower than this due to the definition of the environment. For example, it is not possible to see a disjoint goal [f] neither is it possible to see two goals at the same time[g].

2. *Learning scheme and parameters*

This study uses Q-learning with an ϵ-greedy policy. The values for the learning rate (α) and the discount factor (γ) are kept constant at 0.3 and 0.9 respectively. As the goal of this work is the real life application of a vision-based RL agent on a quadcopter, a sensitivity of the agent to the RL parameters is not performed, as long as there is learning and convergence with the used values.

[c] It is assumed that it has some kind of distance sensor to obtain this information
[d] $2^3 \times 4 \times 1 \times 2$
[e] $3^3 \times 4 \times 3 \times 2$
[f] the state "1,0,1" is not possible
[g] the state "1,2,0" is not possible

The exploration-exploitation problem is handled by increasing the greediness of the agent sequentially as the training progresses. A number of starting exploration rates (i.e. ϵ_0) were tried out in simulation; the trials led to a starting value of 0.60. A high early exploration rate was found to be wasteful in terms of learning rate as the number of possible states in either of the tasks are not significantly high. The schedule of increasing the ϵ for the two simulated tasks and the flight test is presented in Fig. 7.

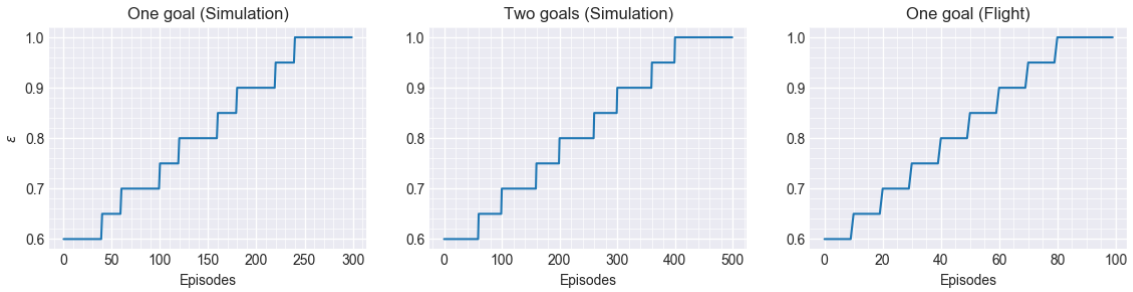


Figure 7: The ϵ schedule for training the RL agents in this study

3. *Actions*

Two sets of actions are defined for the agent. The *primitive action set* consists of going forward, turning left by 22.5° or turning right by 22.5°. The *extended action set* consists of the actions in the primitive set supplanted with an additional action. The extra action is a temporally extended action, called an *option* [39], which causes the agent to keep turning right until it sees a goal. The option is available to the agent only when it is seeing nothing (i.e. in the state 0,0,0;0;x;x). It is implemented with the intention of speeding up learning by incorporating external knowledge. The two sets of actions available to the agent are visualized in Fig. 8.

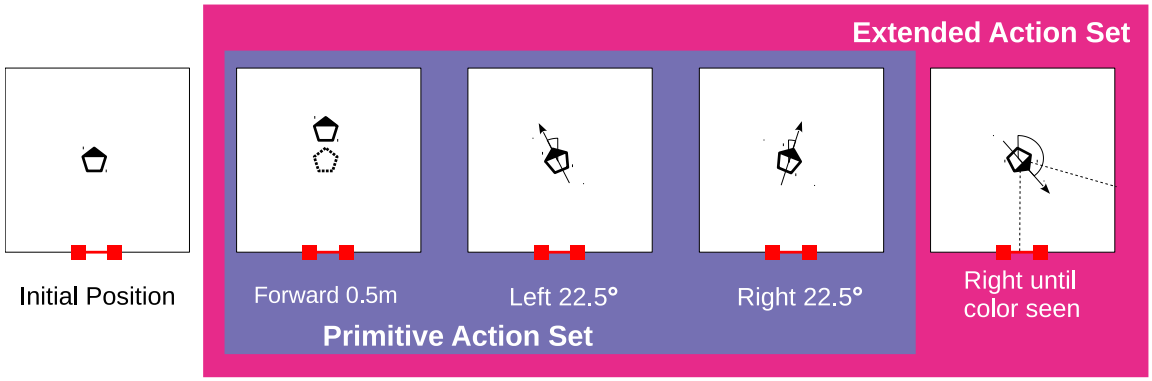


Figure 8: Depiction of the primitive actions and the extended action that are avalable to the agent.

4. *Reward scheme*

The reward scheme is defined with the objective of teaching the RL agent the task of guiding itself to (and between the) goals in the least number of steps. The designed reward accomplishes this objective by encouraging (or discouraging) three things: all movements are penalized to minimize the number of steps, seeing an unvisited goal is encouraged while seeing a visited goal is discouraged and finally visiting an unvisited goal is encouraged. A flow chart depicting the logic of the reward scheme is presented in Fig. 9.

The logic for these three encouragements to the agent through the reward scheme are colorcoded in Fig. 9. The green rhombuses are the initial penalty applied to every action as it is desired to minimize the number of steps required to finish the task. Simulations found that incorporating the knowledge of moving forward when seeing a goal and turning when not seeing anything in the reward scheme speeds up learning.

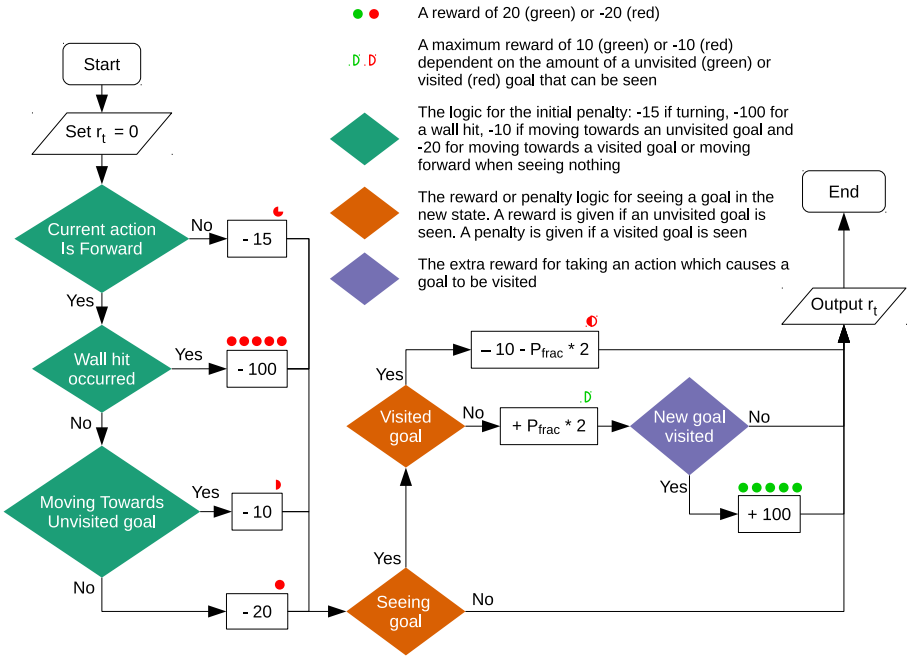


Figure 9: Flowchart depicting the selected reward scheme for the task. r_t represents the current reward and P_{frac} represents the count fraction. The reward for each step is initialized at 0 and modified according to the logic presented in this chart.

The orange rhombuses represent the reward for taking an action that ends up with the goal in sight. This reward is higher if more of the goal is seen and thus depends on the color fraction. For the two goal task, there is a penalty for having a visited goal in sight. This encourages the agent to visit the other goal. Finally, the agent is awarded a big positive reward for visiting an unvisited goal through the logic in the purple rhombus.

C. Rule based controller

The performance[h] of the trained RL agent is compared to a rule based controller using the same vision-based states as for the RL agent. The action selection logic of the rule based controller is presented below:

Action logic for approaching one goal

1. If not seeing goal → *Turn right*
2. Else if seeing goal:
   (a) If "Hitwall" = 0 (False):
      i. if "Color Fraction" <= 2
         A. If goal ONLY in middle column : → *Move forward*
         B. Else : → *Turn towards goal*
      ii. Else if "Color Fraction" > 2 : → *Move forward*
   (b) Else if "Hitwall"= 1 (True): → *Turn away from goal*

Action logic for approaching two goals

1. If seeing visited goal → *Turn right*
2. Else → Use logic for approaching one goal

[h] in terms of the steps required to reach the goal

IV. Simulation

The goal of this work is the implementation of a vision-based RL guidance agent in an AR Drone 2. A simulation is performed before trying out a real life implementation. This section describes the setup and results of the simulations.

A. Setup

The software used, the training scheme it's scheme are described in this section.

1. *Software*

The simulations are carried out using a combination of the PaparazziUAV's [40] simulator and FlightGear [41]. PaparazziUAV uses JSBSim [42] to simulate the autopilot software that it generates. FlightGear is an open source flight simulator which generates the vision information required for simulation.

The two features which make FlightGear appropriate for vision simulation with PaparazziUAV are its capacity to obtain state information from an external software and its ability to run a local HTTP server on a specified port. The server can be used to obtain screen shots of FlightGear's rendering of the environment. PaparazziUAV has built in infrastructure to interface with FlightGear and send it the aircraft state information. FlightGear can use this state information to visualize the environment of the aircraft.

The software for the RL agent has been implemented as a module inside PaparazziUAV. The state and rewards of the RL agent are defined based on visual data. A submodule for vision simulation is created which downloads screenshots from a pre-programmed HTTP address. The flight dynamics of the agent is simulated using PaparazziUAV's JSBSim. This data is fed to FlightGear through a prescribed UDP port[i]. FlightGear renders the environment and uploads its current view to the screenshot path of the HTTP server upon receiving a HTTP request on that path (e.g. http://localhost:9723/screenshot). This is the pre-programmed address in the vision simulation submodule. The screenshot is then unencoded as a RGB bitmap from JPEG and used to generate the vision based state and the reward.

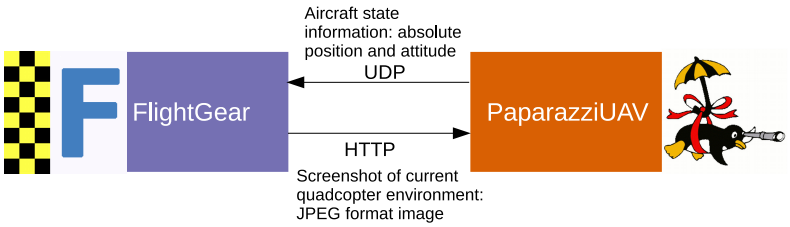


Figure 10: The interactions between the simulated autopilot software and FlightGear.

2. *Task Training*

The parameters and the training scheme are described in the following paragraphs. Both tasks are simulated with the fixed and random initializations; each initialization is simulated with the primitive and the extended action sets (Fig. 8). This leads to a total of four factor levels for each of the tasks. First the learning in both the tasks are presented. The four cases trained are : (a) Fixed initialization; Primitive action set (b) Fixed initialization; Extended action set (c) Random initialization; Primitive action set (d) Random initialization; Extended action set. Furthermore, the performance of the trained RL agents are compared to a rule based autopilot (see Section III-C).

The one goal task is trained for 300 episodes[j] using Q-learning. The training runs are repeated 50 times to obtain statistical estimates of the performance. The schedule for increasing the greediness (ϵ) is presented in Fig. 7. The learning rate (α) is kept at 0.30 and the discount factor (γ) is kept at 0.90. The two goal task is trained for

[i]The FlightGear and the simulation software need to be run with specific arguments for this to work

[j]An episode is the RL agent performing an episodic task from start to finish

500 episodes with 50 repetitions. The ϵ is increased according to Fig. 7 and like the one goal task, the α and γ are kept at 0.30 and 0.90 respectively.

B. Results

This section discusses the learning performance in terms of the number of steps required to reach the goal with an ϵ-greedy policy.

The learning of the agents for the one goal and the two goal task are summarized in Figs. 11 and 12. In both figures, the left plot shows the change in steps, the middle plot shows the sum of changes to the Q-values and the right plot shows the sum of reward over the 50 runs. The sum of changes to the Q-values are calculated by summing the changes made to the Q-values at each step, over the whole episode.

Two means are taken in order to obtain the statistics of the run as presented in Figs. 11 and 12. For both tasks the aforementioned quantities at each episode is averaged over the 50 repetitions of the training. Than the means over 50 repetitions are split in groups of 60 episodes for the one goal task and 100 episodes for the two goal task. The data points in Figs. 11 and 12 represent the grouped run means. The reason for grouping 60 episodes for the one goal task and 100 episodes for the two goal task is based on the number of episodes over which the greediness (ϵ) is increased by 10%.

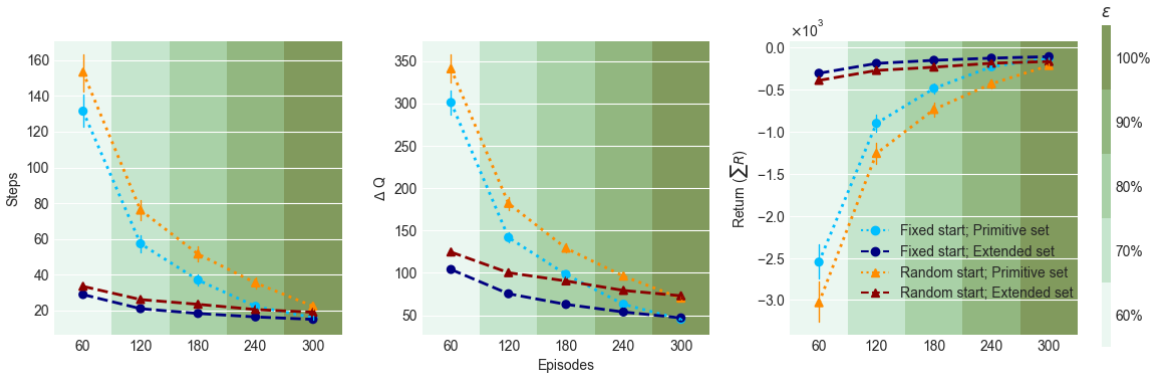


Figure 11: Learning of the one goal task for four conditions

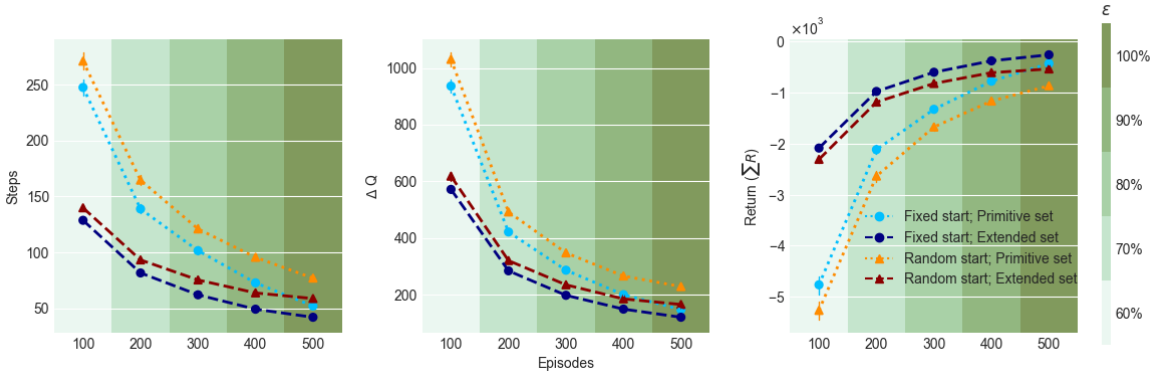


Figure 12: Learning of the two goal task for four conditions

The decreasing trend of the steps and the increasing trend of the returns in Figs. 11 and 12 show that, for both tasks and all simulated factors, the agent manages to learn to perform the task better as it accumulates more experience. However, there are differences in the learning and final performance of the trained agents.

ONE GOAL TASK: Table 1 compares the performance of the trained agents and the rule based autopilot. The table shows that the randomly initialized agents take more steps to learn and perform the task than the agents with

fixed initializations. It also shows that the agents perform worse than the rule based autopilot in all the simulated conditions. The agent performs nearly as good as the rule based autopilot only for the one goal task with fixed initializations.

The exclusion of the episodes, where the rule based autopilot gets stuck in infinite loops, is one of the reason for its lower mean steps to reach the goal with random initializations. The rule based autopilot gets into infinite loops when it is initialized near the wall, while also seeing a goal. The inability of the vision-based state to remember a collision with the wall for more than one step causes this infinite loop. When the rule based autopilot collides with a wall, while it is seeing a goal, it first tries to turn away from the goal. On the very next step it turns towards the goal to put it in the center of its field of view. After having the goal in the center, it tries to move forward again. The forward movement results in a collision, restarting the cycle.

However, the agent does not necessarily get stuck forever in these kinds of loops. The value of the learned actions keep changing while it goes goes through the loop. At some point, the values of the actions causing the loop may decrease low enough that another action become more valuable in comparison to it. This causes the agent to pick the other action, breaking the loop.

Table 1: Performance metrics of the agent and the rule based autopilot for the two tasks over 50 runs

Initialization	Fixed		Random	
Action set	Primitive	Extended	Primitive	Extended
	One goal task			
Sum of steps	15839.08	5960.18	20317.54	7336.20
Steps to goal (RL)	14.89	14.91	24.22	19.27
Steps to goal (Rule based)	14.56		15.26	
	Two goals task			
Sum of steps	61311.12	36419.14	72941.62	43149.54
Steps to goal (RL)	52.25	41.22	74.40	58.40
Steps to goal (Rule based)	35.04		33.57	

The extended action of turning until a goal is seen, decreases the steps required to learn the one goal task by a factor of approximately 2.5 (Table 1). This is on account of the way the tasks and the agents are formulated in this study. There is greatest ambiguity in the states when the agent does not see anything. Most of the real positions and headings the agent can be in, maps to seeing nothing. The positions for four different headings where the agent sees nothing is illustrated in Fig. 13. Having the option to turn until a goal is seen allows the agent to circumvent part of this ambiguity; the agent can use the option to transition from the ambiguous state of not seeing any goals to a less ambiguous one of seeing a goal.

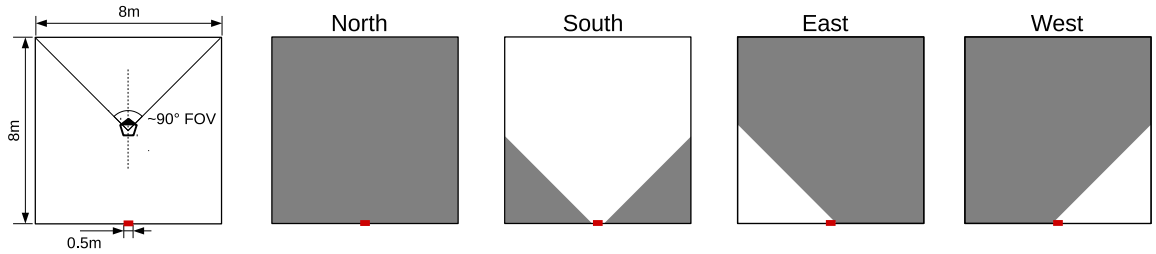


Figure 13: The regions in the environment where the quadcopter cannot see anything for four different headings. Grey represents seeing nothing and white represents seeing a goal. The leftmost figure shows the dimension of the environment, the goal and the field of view of the quadcopter. The leftmost figure shows the quadcopter facing the reference North.

The agents using the extended action set have a better final performance than the ones using the primitive action

set (see Table 1). Due to the ambiguity in the state description, the optimal policy is dependent on the initialization of the quadcopter. For example, of the environment and the task implies that for some initial headings the goal will be in the left region of the quadcopter, while for others it will be in it's right. The optimal actions for these different initial headings are left and right turns respectively. However, as both initializations of the heading have the same vision-based state (i.e. "0,0,0"), they can map to one action. The vision-based state does not encode any information about the relative heading of the quadcopter with respect to the goal. Thus the agent cannot learn optimal actions for this case, and performs sub-optimally.

TWO GOALS TASK: The use of the extended action decreases the steps required to learn the two goal task by a factor of approximately 1.5. The improvement in learning is less than in the one goal task because the option is less effective in reducing the ambiguity for the two goal task. Firstly, the option does not always transition the state of the agent from seeing nothing to seeing an unvisited goal. If, for example, the agent has already visited the blue goal and turns left a number of times resulting in a state where it sees nothing, choosing the option will transition the quadcopter back to seeing the blue goal again. Secondly, the agent has to spend more time in a region of the state-space where the option is not available. The agent cannot choose the option when it is seeing a goal. In the two goal task, the goal is in sight of the quadcopter for a bigger part of the environment and it has to spend more time learning the right values for each action while seeing either of the goal.

Both initializations with the two goal task perform worse than the rule based autopilot (see Table 1). The two goal task is harder to learn than the one goal task. The presence of a second goal and the need of a memory state for goals visited nearly doubles the state space. Further, the behavior required of the agent to finish the task is more complex, as it must now learn to visit two goals instead of one. Additionally, the presence of two goals lead to more points in the environment where the quadcopter is next to the wall with the goal in front of it. Thus, the quadcopter gets stuck in a loop more often in the two goal task. As the episodes where the agent gets into this loop are ignored for the rule based autopilot but not for the RL agent, the rule based autopilot gets a bigger advantage when estimating its final performance in the two goal task than in the one goal task.

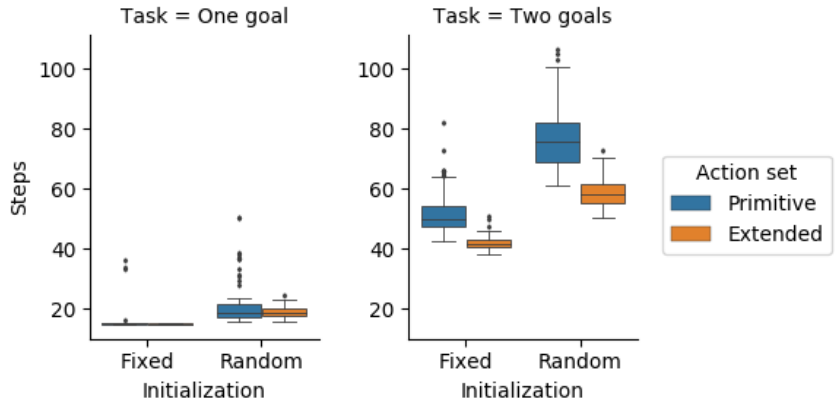


Figure 14: Box plots of the steps taken by the agent for the two tasks after training. Results from 50 runs are presented for each of the cases.

The means and standard deviations of the final performance for both tasks with both initializations is presented in Fig. 14. The box plots are created using the final fully greedy parts of the training for the two tasks[k]. Due to the small variance for the one goal task with fixed initialization, their boxes appear as lines in Fig. 14. The smallness of the variance in the final performance of the one goal agent with fixed initialization is evidence of the convergence of the learning.

A difference is observed in the number of outliers with the primitive and extended action sets. The number of outliers are higher for the primitive action set for both tasks, with both initializations. This indicates that with the primitive action set the agent may get lost, but with the extended action set the agent performs consistently.

There is a bigger gap in the mean performance of the trained agent between the two action sets for the two goal task than for the one goal task. This can be explained by the difference in the "usefulness" of the option in

[k] i.e. the last 60 episodes for the one goal task and last 100 episodes for the two goal task

performing the different tasks. As the one goal task comprises of turning to the red goal and approaching it, it is simpler and there is a smaller chance for the quadcopter to get lost. Hence near optimal performance is easier to obtain without the need for an action which helps bring the quadcopter back to the more relevant region of the state space (i.e. when it is seeing a goal). The two goal task on the other hand comprises of four steps: turning towards the closest goal, approaching it, turning towards the other goal and approaching it. There is a greater chance of getting lost and following a trajectory which adversely influences the Q-value estimates this case. With the extended actions, the quadcopter stays in trajectories which reduce the variance in the value estimate that result from the state ambiguity. Therefore, the final performance is better and the variance in the performance is lower with the extended action set for the two goal task.

V. Flight test

The RL agent is implemented in an AR Drone 2 and trained to perform the one goal task in real life flights. The flight tests aim to show that a vision-based RL agent can be trained to perform a guidance task in real life. Q-values learned in simultion are tested on the AR Drone 2 to study the reality gap in the simulations.

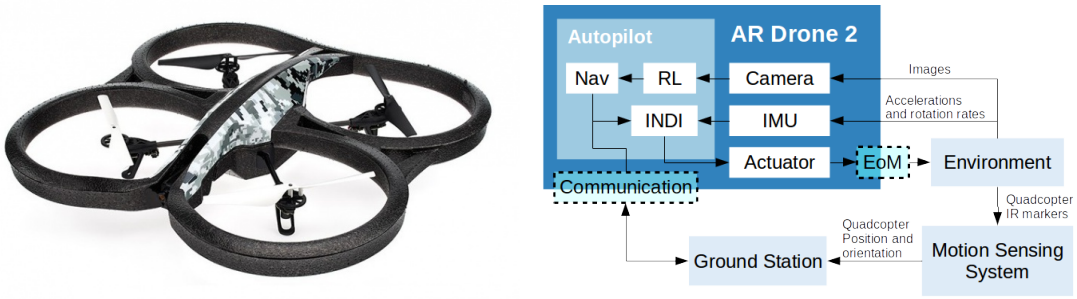
A. Setup

The setup of the one goal task has been described in Section III. This section discusses the other elements of the flight tests. First, the AR Drone 2 and its applicability to this study is outlined. After this, an overview of the software and hardware elements of the autopilot, and their interactions is provided. Lastly, differences between the simulated and the real life vision are discussed.

1. AR Drone 2

This study uses a Parrot AR Drone 2 Elite Edition (see Fig. 15a) to perform flight test on the developed RL agent. The key features of the quadcopter which make it suitable for these flight tests are its onboard camera and its use of an open source operating system.

The specifications of the AR Drone 2 meet the requirement of this study. The forward facing camera is capable of producing 1280 by 720 pixel video at 30 FPS. The simulations are carried out at 800 by 600 resolution and the designed agent takes at least 1 second for each step. As it takes 1 second for each step, it requires a new estimate of the vision information at approximately 1 Hertz (i.e. it requires images to be processed at 1 FPS). Therefore both the resolution and the video frame rate of the AR Drone 2 are sufficient. It uses BusyBox, a distribution of Linux, as its operating system. This enables the use of open source autopilots such as PaparazziUAV for its control.



(a) Parrot AR Drone 2 Elite edition **(b) Controller architecture**

Figure 15: The AR Drone 2 and the controller architecture

A key limitation of small quadcopter, such as the AR Drone 2, is their short flight times. On full battery, the AR Drone 2 is capable of approximately 12 minutes of flight. This creates a challenge for RL which requires

a thorough exploration of the state-space. The quadcopter is tethered to an external power supply to enable the prolonged flights required for RL.

2. *Autopilot Elements*

An overview of the interaction between the different elements of the system is presented in Fig. 15b. Three parts of the PaparazziUAV autopilot are visualized: the implemented "RL" module, the position controlling navigation module ("Nav") and the rate controlling Incremental Non-linear Dynamic Inversion ("INDI") [43] module. The hardwares these modules interact with are the actuators, the communication system, the IMU and the camera.

PaparazziUAV's autopilot controls the internal rate, position and attitude loops using feedback from the quadcopters IMU and the motion sensing system of the CyberZoo (see Section III). The RL agent is a higher level controller which decides between going forward or turning in either direction so that the quadcopter can reach the goal.

The movements of the quadcopter are also detected by the external motion sensing system of the Cyber Zoo. This information is passed to the autopilot through the ground station. Although the autopilot uses absolute position and attitude information for stabilization and position control, the RL agent makes higher level guidance choices based solely on the visual data. The inner loop control, although a challenging fields in its own regard, is out of the scope of this study.

3. *Differences with simulated vision*

The visual stream from the real runs have the following differences in comparison to the simulation runs:

- There is no noise in the vision data in the simulation but real life vision data from the quadcopter is noisy (see Fig. 16).

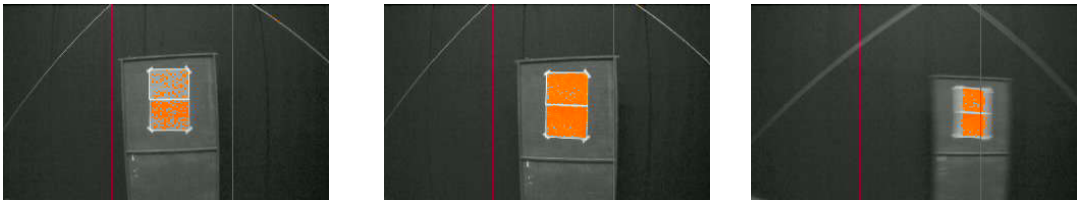


Figure 16: A noisy viewing (left), a less noisy viewing (middle) and a blurred viewing (right) of the red goal captured during flight tests. The red pixels are detections.

  The simulations are carried out with the time of day and the lighting conditions frozen at a specific point. Although the CyberZoo is enclosed in three directions, ambient light can still get in and change the nature of the visual data depending on the time of day. Further, blurring of the visual data (see right image in Fig. 16) caused by the motion of the quadcopter is not present in the simulated vision as FlightGear does not support it.
- The resolution of the image, the field of view of the camera and the colorspace are different between the simulation and the real-life system. The real system is tuned such that its outputs resemble that of the vision module in simulation.
- The computer vision module operates at a frequency of about 1 Hz in the real-life implementation. The total time to perform one action is between 1 to 3 seconds. There is a chance for the agent to use an image frame from before an action is over to estimate its vision state after the action is taken. This can result in the RL agent sensing the wrong vision-state. Fig. 17 illustrates the delay in perception. This does not happen in simulation as the RL agent grabs a frame from FlightGear only after performing an action. The problem of delayed perception can be mitigated by modifying the implementation; modifications to alleviate this problem are not attempted due to time constraints of the project.
- The camera distorts the visual data due to its
  - varying sensitivity to the different spectrums of visual light
  - fixed focal length which blurs objects out of its depth of field
  - imperfections (or flaws) in the lens

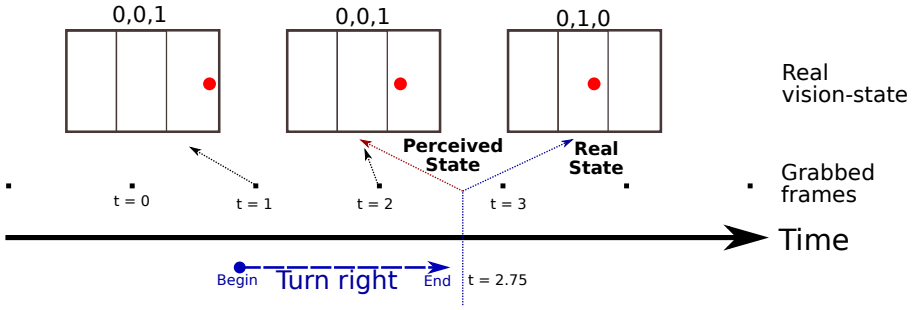


Figure 17: Illustration of the erroneous perception resulting from the fixed update rate of the real-life vision module.

B. Results

Flight tests are performed to demonstrate the learning in real life and study the reality gap between the learning in simulation and in real life.

FLIGHT TESTS: The one goal task is trained for the shorter duration of 100 episodes in the flight tests. Simulations of the one goal task with 100 episodes of training showed that the agent can learn the task with 100 episodes of training. The leftmost and rightmost plots in Fig. 19 visualizes the steps required by the simulated agent during and after training with 100 episodes.

The scheme for increasing the ϵ is justified in Section III and visualized in Fig. 7. Fig. 18 shows the mean trends in the learning over the five runs of the flight tests. In contrast to the datapoints in the simulation plots (i.e. Figs. 11 and 12), the datapoints in Fig 18 represent means over 10 episodes; this corresponds to a 5% increase of ϵ.

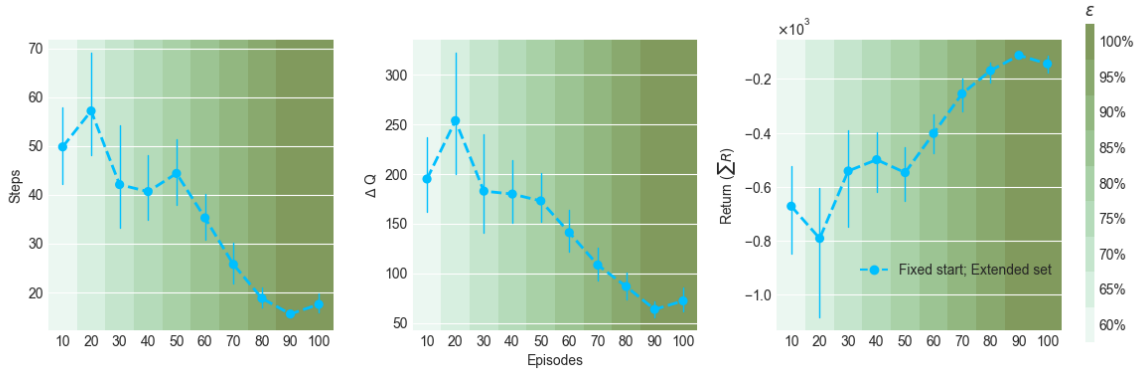


Figure 18: The learning metrics from flight tests of the one goal task averaged over five runs

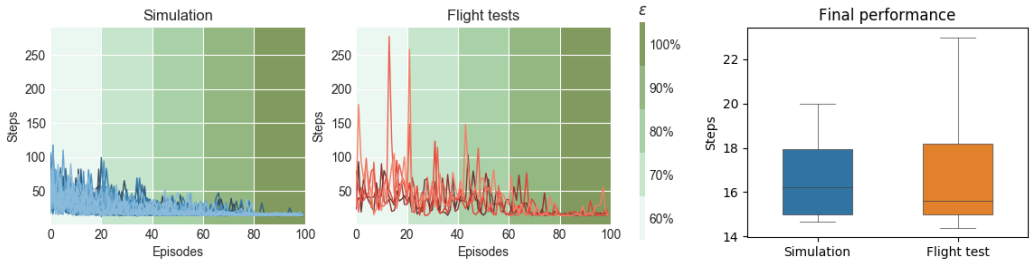


Figure 19: The two left figures show steps against episode for the 50 simulated runs and the five flight test runs. The rightmost figure shows the statistics of the fully trained performance.

Fig. 18 shows a decreasing trend in the steps and the sum of changes to the Q-values along with an increasing trend in the returns over the training. These indicate that the agent manages to learn the task better as it explores the environment. The performance of the agent gets worse when learning with full greed; this is seen in the rightmost parts of the plots in Fig. 18. The cause of this apparent degradation in the performance is the small number of runs used to estimate the mean performance. Due to the small number of runs, aberrations in the performance of the agent in one run has a significant effect on the mean. This can be seen in the rightmost part of the middle plot of Fig. 19. One of the lines in the middle plot of Fig. 19, representing one of the flight test runs, shows how the agent required more steps to reach the goal in the final phase of the training than the others. This can be caused by noise in the experiment or the ambiguity in the state description. The fluctuation in the performance on that run causes the mean over the five runs

Some of the differences between the real life learning and the simulated learning can be seen when comparing the left and right plots of Fig. 19. The figure shows the steps required to reach the goal for the five flight test runs, and the 50 runs of the simulation with 100 episodes of training.

The trajectories taken by the real and the simulated quadcopter, at different episodes of the training, from a specific training run are visualized in Fig. 20. The trajectories of the agent in the flight test are more erratic because of the greater amount of noise present in real flights. Besides the noisy motion, there are visible loops in the quadcopter's trajectory during the flight tests. These occur when the quadcopter chooses the option and keeps yawing until it sees a goal. The instability in the yawing motion of quadcopters and the introduced forces by the power tether contributes to the drift of quadcopter while it yaws, which results in the aforementioned loops. The simulated quadcopter also has these loops, but their diameter is smaller.

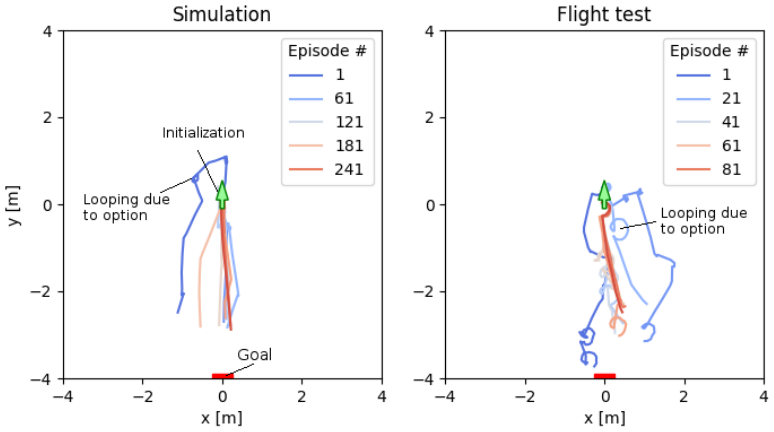


Figure 20: Trajectories taken by the simulated and the real quadcopter during one training

There is more variance in the steps required to reach the goal in flight tests in comparison to the simulations. The noise in the setup of real-life flights and in the flights themselves cause this variance. The noise in the flight introduces variance in the result of the state transitions, which gets reflected in the Q-values. Noise in the vision information makes the quadcopter perceive erroneous things. Erroneous state perception leads to erroneous Q-value updates, leading to a greater variance in its estimate. Further, there are inconsistencies in the initialization of the quadcopter and on the accuracy of the motion sensing system. Small variations in the initialization and calibration of the motion sensing system make the motion of the real quadcopter different from the simulated quadcopter.

Another observable feature is the maximum number of steps required to reach the goal in simulation and in flight. Fig. 19 shows two of the episodes in the flight test requiring more than 250 episodes to reach the goal. The maximum steps required to reach the goal in simulation out of all the episodes from all the runs is about 125 steps. In general the learning performance indicates that the performance in simulation is more consistent. This is expected as the simulation does not include many factors, such as the noise in the vision, which makes learning the task harder in reality. The fully trained quadcopter performs nearly as well as the quadcopter trained in simulation (rightmost plot in Fig. 19) but with higher variance in the final performance.

SIMULATION VALIDATION: The applicability of the policy learned in simulation to real life is studied. This information is desirable because online learning with quadcopters is often challenging, and learning in simulation is the preferable way to attempt RL based solutions.

The Q-values learned by an agent over 300 episodes of training in simulation are uploaded and tested in the AR Drone 2. The Q-values at specific points in the training is uploaded to the quadcopter and the task performed. The performance of the real life agent is compared to the simulated agent at the consequent point of training. The comparison is visualized in using a bar chart in Fig. 21.

Due to time constraints, no repetitions are carried out for the episodes performed with the real quadcopter. Secondly, the seed for the random number generator in the simulation and the flight test are not the same; the random actions picked by the policy are different between the simulated agent and the real life agent.

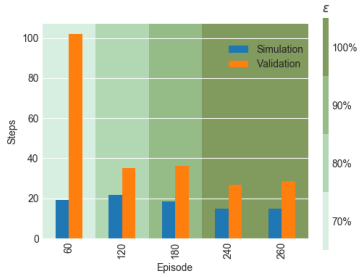


Figure 21: The steps taken in the simulation run and in the validation flight tests

The difference in the steps required to perform the task at the five validation points are visualized in Fig. 21. There is a difference of about 80 steps between the simulated and the real-life agent with the Q-values after 60 episodes of training. Q-values from consequent phases of the training have a difference of between 10 to 20 steps with the simulated agent.

Besides the noise, the delayed perception in the flight test (Fig. 17) causes the performance of the agent trained in simulation to degrade when transferred to a real-life quadcopter. When the agent is trained in flight, this delay is incorporated into the learning of the agent. However, in the validation flights, the Q-values on which the real life agent acts are trained without the delayed perception of the real world. The underlying transition model between the states in the simulation and the flight test are different.

This validation study led to three findings: First, the real life agent is bad at incorporating Q-values from early stages of the training. Second, there is a near constant offset between the performance of the simulated agent and the real life agent for Q-values which are relatively stable. Third, the delayed perception in the flight tests is one of the causes for the difference in behavior.

VI. Conclusions

This work explores the possibility of a learning vision-based guidance controller for a quadcopter. Such systems may improve the flight of Unmanned Aerial Vehicles (UAVs) in dynamic and Global Positioning Systems (GPS) denied environments. The possibility for such a guidance controller is investigated by developing a vision-based Reinforcement Learning (RL) controller for an AR Drone 2 and training it to perform simple guidance tasks first in simulation and later in real flights.

The descriptions of the state, reward and terminal condition for the tasks are derived from vision data. The training of one task, consisting of turning to a red marker, is performed in simulation and in flight test. A more complex task of approaching two markers (one red and one blue) is trained in simulation. The effect of incorporating knowledge into the agent is studied by expanding the actions it can take to include a multi-step action, called an *option*, designed to decrease the ambiguity in the state perception of the agent. The robustness of the learning is examined by simulating it with fixed and random initializations.

The simulation results show that the agent performs[1] the tasks better with increasing exploration of the environment. The option of turning until a goal is seen improves the learning rate. Further, if the task is relatively

[1]Note that "performance" is used to refer to the steps required to perform the task after full training and "learning performance" is used to refer to the steps required to learn the task.

simple and the *options* are made to consist of a sequence of primitive actions, both the set of primitive actions and the set of extended actions can converge to the optimal performance. As the complexity of the task is increased, for example by using a random starting position, the final performance of the agent with the extended set of action is better than the agent with the primitive set of actions. This is on account of the increased ambiguity in the state perception with random initializations which the *option* is better able to mitigate.

The agent in the flight test also manages to learn the task, albeit with worse learning and final performance than in simulation. The noise in the visual perception and motion of the quadcopter in flight tests lead to greater variance in the real-life learning in comparison to the simulated learning. Other factors that contribute to the reality gap are the delay in perception of the vision-based state in the real-life agent and the greater drift of the quadcopter position while yawing in real life.

The simulations and flight tests are carried out without any tuning or sensitivity analysis on the hyperparameters in order to meet time constraints. This does not have consequences for the final performance of the simulated agent performing the one goal task with fixed initialization. The observed final performance of that agent is close to the performance of the rule based autopilot. This is not true for any of the other simulations or the flight tests. Hence, the final performance for the other agents and the learning characteristics of all the trained agents in this study can potentially be improved by tuning the learning rate (α), the discount factor (γ) and trying out other policies. Namely a higher discount factor should speed up learning as more accurate information of the value of a state-action gets propagated backwards.

The development of a learning and vision-based guidance system for UAVs will broaden their domain of operation and thus increase their demand. This technology will enable the development of generic Unmanned Aerial Systems (UAS) that can be targeted towards specific markets. For example, a generic learning and vision-based UAS that is designed for checking the inventory will be usable in warehouses, supermarkets, workshops etc. Using learning and vision-based systems for guidance will remove the need to setup a local positioning system or programming the UAS autopilot for the environment on an ad-hoc basis. Furthermore, as one software architecture can be used to learn different tasks, there are potential savings in terms of software development for the UAV manufacturers. If vision-based learning can be made generic, it can be marketed to mass consumers who train their UAVs for personalized applications they desire. As the trained agent takes over the task of guiding the UAV, operating them will require less man power.

New regulatory bodies will need to be created to certify the systems and operators of learning based UAVs like the ones presented in this study. Social norms and outlooks will need to adapt to the emergence of mobile robots that are not defined by their programming, but have the capacity to improve on their designed roles over time. Before the mass marketing of learning based UAS, the manufacturers will have to ensure the safety and security of their owners and the societies where they are being marketed. Typical questions pertaining to the use of artificial intelligent systems in daily life will need to be answered. Questions such as who will be held accountable for damages caused by the autonomous operation of these systems and how to ensure the livelihood of people whose jobs are going to be replaced by such systems will need to be answered.

Perhaps the biggest factor which prevents the use of RL based controllers in current UAS is the time they take to learn. The agent designed in this study takes about two hours of flight to learn the relatively simple task of turning and approaching a goal. The learning time increases as the dimensionality of the state space is increased to enable more complex tasks. The results show that incorporating programmed behaviors can speed up learning. However, it also increases complexity of the design and implies the encoding of expert knowledge into the system. The time required for an agent to learn forces most applications to carry out the learning offline and then implement the learned policies in the UAS. Besides the slow learning in RL, vision-based applications need to deal with the problem of making sense of the large volume data that are contained in images. Generalized RL structures to map pixel based vision data to actions exist [44], although these methods are still expensive in terms of memory and computations.

References

[1] Nister, D., Naroditsky, O., and Bergen, J., "Visual odometry," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, Vol. 1, June 2004, pp. I–652–I–659 Vol.1.

[2] Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., Grixa, I. L., Ruess, F., Suppa, M., and Burschka, D., "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue," *IEEE Robotics Automation Magazine*, Vol. 19, No. 3, Sept. 2012, pp. 46–56.

[3] Kelly, J., Saripalli, S., and Sukhatme, G. S., "Combined visual and inertial navigation for an unmanned aerial vehicle," *Field and Service Robotics*, Springer, 2008, pp. 255–264.

[4] Krajnk, T., Nitsche, M., Pedre, S., Peuil, L., and Mejail, M. E., "A simple visual navigation system for an UAV," *International Multi-Conference on Systems, Sygnals Devices*, March 2012, pp. 1–6, 00034.

[5] Kendoul, F., "Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems," *Journal of Field Robotics*, Vol. 29, No. 2, March 2012, pp. 315–378.

[6] Fischer, C. and Gellersen, H., "Location and Navigation Support for Emergency Responders: A Survey," *IEEE Pervasive Computing*, Vol. 9, No. 1, Jan. 2010, pp. 38–47.

[7] Durrant-Whyte, H. and Bailey, T., "Simultaneous Localization and Mapping: Part I," *IEEE Robotics Automation Magazine*, Vol. 13, No. 2, June 2006, pp. 99–110.

[8] Bachrach, A., Prentice, S., He, R., and Roy, N., "RANGE–Robust Autonomous Navigation in GPS-Denied Environments," *Journal of Field Robotics*, Vol. 28, No. 5, Sept. 2011, pp. 644–666.

[9] Weiss, S., Scaramuzza, D., and Siegwart, R., "Monocular-SLAM–based Navigation for Autonomous Micro Helicopters in GPS-Denied Environments," *Journal of Field Robotics*, Vol. 28, No. 6, Nov. 2011, pp. 854–874.

[10] Grzonka, S., Grisetti, G., and Burgard, W., "A Fully Autonomous Indoor Quadrotor," *IEEE Transactions on Robotics*, Vol. 28, No. 1, Feb. 2012, pp. 90–100.

[11] Campoy, P., Correa, J. F., Mondragn, I., Martnez, C., Olivares, M., Mejas, L., and Artieda, J., "Computer Vision Onboard UAVs for Civilian Tasks," *Journal of Intelligent and Robotic Systems*, Vol. 54, No. 1-3, Aug. 2008, pp. 105.

[12] Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N., "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera," *Robotics Research*, edited by H. I. Christensen and O. Khatib, No. 100 in Springer Tracts in Advanced Robotics, Springer International Publishing, 2017, pp. 235–252, DOI: 10.1007/978-3-319-29363-9_14.

[13] Bonin-Font, F., Ortiz, A., and Oliver, G., "Visual Navigation for Mobile Robots: A Survey," *Journal of Intelligent and Robotic Systems*, Vol. 53, No. 3, May 2008, pp. 263.

[14] Bipin, K., Duggal, V., and Krishna, K. M., "Autonomous navigation of generic monocular quadcopter in natural environment," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 1063–1070.

[15] Shen, S., Michael, N., and Kumar, V., "Obtaining Liftoff Indoors: Autonomous Navigation in Confined Indoor Environments," Vol. 20, No. 4, 12 2013, pp. 40–48.

[16] Cesetti, A., Frontoni, E., Mancini, A., Zingaretti, P., and Longhi, S., "A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks," *Journal of Intelligent and Robotic Systems*, Vol. 57, No. 1-4, Oct. 2009, pp. 233.

[17] Izzo, D. and Croon, G. D., "Landing with Time-to-Contact and Ventral Optic Flow Estimates," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 4, 2012, pp. 1362–1367, 00022.

[18] Kendoul, F., Fantoni, I., and Nonami, K., "Optic Flow-Based Vision System for Autonomous 3D Localization and Control of Small Aerial Vehicles," *Robotics and Autonomous Systems*, Vol. 57, No. 6–7, June 2009, pp. 591–602.

[19] Hrabar, S., Sukhatme, G. S., Corke, P., Usher, K., and Roberts, J., "Combined optic-flow and stereo-based navigation of urban canyons for a UAV," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Aug. 2005, pp. 3309–3316.

[20] Horn, B. and Schunck, B., "Determining optical flow," *Artificial Intelligence*, Vol. 17, No. 1-3, 1981, pp. 185–203, 11476.

[21] Sutton, R. S. and Barto, A. G., *Reinforcement Learning: An Introduction*, Vol. 1, MIT press Cambridge, 1998.

[22] van Kampen, E.-J., Chu, Q., and Mulder, J., "Continuous adaptive critic flight control aided with approximated plant dynamics," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2006, p. 6429.

[23] Zhou, Y., Kampen, E.-J. v., and Chu, Q., "Nonlinear adaptive flight control using incremental approximate dynamic programming and output feedback," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2016, pp. 493–496.

[24] Zhou, Y., Van Kampen, E., and Chu, Q. P., "Incremental approximate dynamic programming for nonlinear flight control design," *Proceedings of the 3rd CEAS EuroGNC: Specialist Conference on Guidance, Navigation and Control, Toulouse, France, 13-15 April 2015*, 2015.

[25] Zhou, Y., van Kampen, E., and Chu, Q., "Incremental model based heuristic dynamic programming for nonlinear adaptive flight control," *Proceedings of the International Micro Air Vehicles Conference and Competition 2016, Beijing, China*, 2016.

[26] Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y., "An Application of Reinforcement Learning to Aerobatic Helicopter Flight," *Advances in neural information processing systems*, Vol. 19, 2007, pp. 1.

[27] Valasek, J., Doebbler, J., Tandale, M. D., and Meade, A. J., "Improved Adaptive-Reinforcement Learning Control for Morphing Unmanned Air Vehicles," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 38, No. 4, 8 2008, pp. 1014–1020.

[28] Valasek, J., Kirkpatrick, K., May, J., and Harris, J., "Intelligent Motion Video Guidance for Unmanned Air System Ground Target Surveillance," *Journal of Aerospace Information Systems*, Vol. 13, No. 1, 2016, pp. 10–26.

[29] Bou-Ammar, H., Voos, H., and Ertel, W., "Controller Design for Quadrotor UAVs Using Reinforcement Learning," *2010 IEEE International Conference on Control Applications*, IEEE, 2010, pp. 2130–2135.

[30] Sharma, R., "Fuzzy Q Learning Based UAV Autopilot," *2014 Innovative Applications of Computational Intelligence on Power, Energy and Controls with Their Impact on Humanity (Cipech)*, 2014, pp. 29–33.

[31] Junell, J., Mannucci, T., Zhou, Y., and van Kampen, E.-J., "Self-Tuning Gains of a Quadrotor Using a Simple Model for Policy Gradient Reinforcement Learning," *Policy*, Vol. 9, 2016, p. 10.

[32] Junell, J., Van Kampen, E.-J., de Visser, C., and Chu, Q., "Reinforcement Learning Applied to a Quadrotor Guidance Law in Autonomous Flight," *AIAA Guidance, Navigation, and Control Conference*, 2015, p. 1990.

[33] Yijing, Z., Zheng, Z., Xiaoyi, Z., and Yang, L., "Q learning algorithm based UAV path learning and obstacle avoidence approach," *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 3397–3402, 00000.

[34] Zhang, B., Mao, Z., Liu, W., and Liu, J., "Geometric Reinforcement Learning for Path Planning of UAVs," *Journal of Intelligent & Robotic Systems*, Vol. 77, No. 2, Feb. 2015, pp. 391–409.

[35] Kober, J., Bagnell, J., and Peters, J., "Reinforcement Learning in Robotics: A Survey," *International Journal of Robotics Research*, Vol. 32, No. 11, 2013, pp. 1238–1274.

[36] Mannucci, T., Kampen, E. J. v., Visser, C. d., and Chu, Q., "Safe Exploration Algorithms for Reinforcement Learning Controllers," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. PP, No. 99, 2018, pp. 1–13, 00001.

[37] Bertsekas, D. P. and Castanon, D. A., "Adaptive aggregation methods for infinite horizon dynamic programming," *IEEE Transactions on Automatic Control*, Vol. 34, No. 6, June 1989, pp. 589–598.

[38] Watkins, C. J. C. H., *Learning from delayed rewards*, Ph.D. thesis, University of Cambridge England, 1989, 05115.

[39] Sutton, R. S., Precup, D., and Singh, S., "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, Vol. 112, No. 1-2, Aug. 1999, pp. 181–211.

[40] Hattenberger, G., Bronz, M., and Gorraz, M., "Using the paparazzi uav system for scientific research," *IMAV 2014, International Micro Air Vehicle Conference and Competition 2014*, 2014, pp. pp–247, 00012.

[41] Perry, A. R., "The flightgear flight simulator," *Proceedings of the USENIX Annual Technical Conference*, 2004.

[42] Berndt, J., "JSBSim: An Open Source Flight Dynamics Model in C++," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Guidance, Navigation, and Control and Co-located Conferences, American Institute of Aeronautics and Astronautics, Aug. 2004, DOI: 10.2514/6.2004-4923.

[43] Sieberling, S., Chu, Q. P., and Mulder, J. A., "Robust Flight Control Using Incremental Nonlinear Dynamic Inversion and Angular Acceleration Prediction," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 6, 2010, pp. 1732–1742.

[44] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and others, "Human-Level Control through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, pp. 529–533.