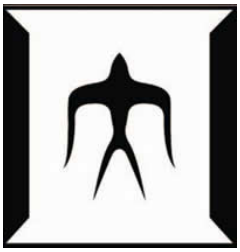
論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Associative-memory based learning and concurrent deep reinforcement learning with application to autonomous aerial maneuver
著者(和文)	HuangPei-Hua
Author(English)	Pei-Hua Huang
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第10706号, 授与年月日:2017年12月31日, 学位の種別:課程博士, 審査員:長谷川 修,山村 雅幸,寺野 隆雄,青西 亨,石井 秀明
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第10706号, Conferred date:2017/12/31, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Associative-memory based learning and concurrent deep reinforcement learning with application to autonomous aerial maneuver



Huang Pei-Hua

Supervisor: Dr. Hasegawa Osamu

The Interdisciplinary Graduate School of Science and Engineering
Tokyo Institute of Technology

This dissertation is submitted for the degree of
Doctor of Engineering

November 2017

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Huang Pei-Hua
November 2017

Acknowledgements

This dissertation is the results of a very close collaboration with my supervisor, Dr. Hasegawa Osamu. Hasegawa sensei has been an amazing mentor throughout my Ph.D. student career. I have learned a great many things from him. Just to name a few that stand out: the way he goes about choosing and solving research problems, how he teaches, and how he communicates research through papers and presentations. I have also over and over benefited from his genuine enthusiasm about our research, especially on self-organizing incremental neural network and his ever willingness to make time to meet with me.

Many thanks to my many friends from Taiwan and United Kingdom, who had nothing to do with any of the work in this thesis, but who always warmly welcomed me back whenever I was visiting.

To my younger sister, Cindy Huang, thanks for being such loving, caring and fun companions throughout my life, and looking after and support our parents throughout my Ph.D., so I could fully concentrate on my studies.

Last and foremost, I am thanking my parents, Ernest Huang and Yu-Shu Ling, who have given me unconditional love, support and encouragement throughout my life.

Abstract

In this thesis, we study the issues of policy learning efficiency and policy model optimization which arise when an unmanned aerial vehicle (UAV) learning to maneuver for achieving tasks in an initially unknown environment. While robust policy model allows stable and sophisticate control of the aerial vehicle. Due to the limited payload of UAV, especially, a micro aerial vehicle (MAV) of a quad-rotor helicopter (quadcopter), the availability of sufficient on-board computation is restricted. We put our focus on model-free methods by describing two settings of learning algorithms: (i) the associative memory based self-organizing incremental neural network learning (SOIAM) and (ii) the end-to-end model-free reinforcement learning combines with concurrent learning frameworks (CRL), they achieve the same goal of learning a good control policy with formal performance guarantees for autonomous navigation of quadcopter.

We show the SOIAM method leverage manual demonstrations to learn hovering task for a quadcopter, this setting is guaranteed to learn a control policy with performance comparable to the manual demonstrations and traditional control method. On the other hand, the CRL demonstrates that the state-of-the-art synchronous and asynchronous framework is particularly effective and efficient with larger scale learning in continuous control tasks. We evaluate performance in terms of the convergence speed and resulting maneuvers using simulated 3D environments of the Mujoco continuous control simulation and quadcopter simulation. The results show significant improvements in training time with higher reward. The trained quadcopter has performed a challenging maneuver of balancing an inverted pole, which even expert pilot cannot fly properly.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Imagining after catastrophic disturbance, instead of risking human life for the on-site investigation, single or multiple aerial robots automatically navigating through collapsed buildings to conduct search and rescue mission. They enter building, a few start mapping the internal structure of the building, and a few joint the fleet to search for survivors. They have not seen the environment before, nor have they seen the particular obstacles before, yet they are capable of self-localizing that they know where they are in the new environment, and manages to avoid obstacles efficiently along the way of searching for targets.

Aerial robotic operations can be seen from recent year in many industries across the world, i.e., agriculture, construction inspection, and land surveying and mapping, today they can also be seen around our household. They have the ability to lift payload, travel through space with incredible speed, and taking high resolution aerial photograph on the guided way-points using globally consistent signal, such as global positioning system (GPS). Yet, despite this, fully stand-alone autonomous aerial vehicles are still limited to laboratory environments.

Despite their impressive repertoire of capabilities, current industrial aerial robots will fail to adapt when presented in a GPS-denied environment, as well as an unconstructed and unfamiliar indoor environment that lack the support of map. This comes down not only to the dynamic and unpredictable nature of human environments which cannot be pre-programmed, or re-programmed during airborne, but also the lack of generalization about understanding the continuously changing environments which the robots must learn first hand. A successful navigation of robot requires two key factors: control autonomy and localization, where control autonomy is about how robot understanding its capabilities and constraints of controlling itself, localization is how robot perceive the world that it interacts within. In nature, human and animals use their eye as a primary sensor that provides the richest external information about the surrounding world, and through combining the

perceived visual information with other sources of sensory information consciously and subconsciously, such as gravity, hearing, taste, touch, sense fo space, and smell, there is also additional senses that only prosessed by specific animals, such as the inclination and rotation of magnetic field, the sense of position of their joints with respect to each other, these gained information can then be "computed" by the brain to make reasonable observations about our world, and develop an optimal strategy to react to specific observations. However vision does not have to be the primary sensor, some animals have different perception mechanisms, like bats, instead of vision information, sonar-like sensors with remarkable properties [7] help to serve the function of localization, also some birds have a compass-like sense, which is perceptive to the Earth's inclination and the directional information of celestial navigation.

State-of-art technology in aerial robotic has demonstrated impressive control of large scale art work with GPS or external beacons from the likes of Intel's drone light show [80] and Verity studio's drone show (Raffaello D'Andrea, 2016). In coming years, the number of aerial robots inhabiting our living spaces is expected to raise dramatically. However, before robots can realize their full potential in everyday life, they need the ability to manipulate certain degree of autonomy and intelligence to deal with the changing world around them.

There is no doubt, through understanding the mechanisms of nature, help us to develop powerful perceptual and control systems for robotic navigation. Best example is the sensing devices that found in today's applications of both industrial and mobile robots. Theses advanced artificial sensing devices have been implemented explicitly to emulate various perceptional mechanisms in humans and animals, which have evolved naturally to excellence. It is difficult to name all of them, but significant sensors, such as visual sensor; camera has been applied in image processing for robotics; inertial sensor and compass for state estimation; sonars, range finders and bulky laser devices for distant measurement and obstacle detection and identification. These sensors hold the key to successful navigation of robots, which give information about their surrounding 3-D spaces, by merging multiple sensor information, leading to the emergence of many mapping applications, such as technique called the Simultaneous Localization And Mapping (SLAM). With sufficient computation memory, precise self-localization assists robots to recall all the visited scenes, this enable robots to be superior in recognizing their environment. However when compare robots with our nature, the critical advantage of nature is not only capable of performing fast feature extraction and optimal fusion of information from multiple sources of sense, but also capable of subconsciously transferring the responded strategy into a sequential body motion. In the field of practical robotics, finding and manipulating efficient control policy from sensory information is still a remaining challenge.

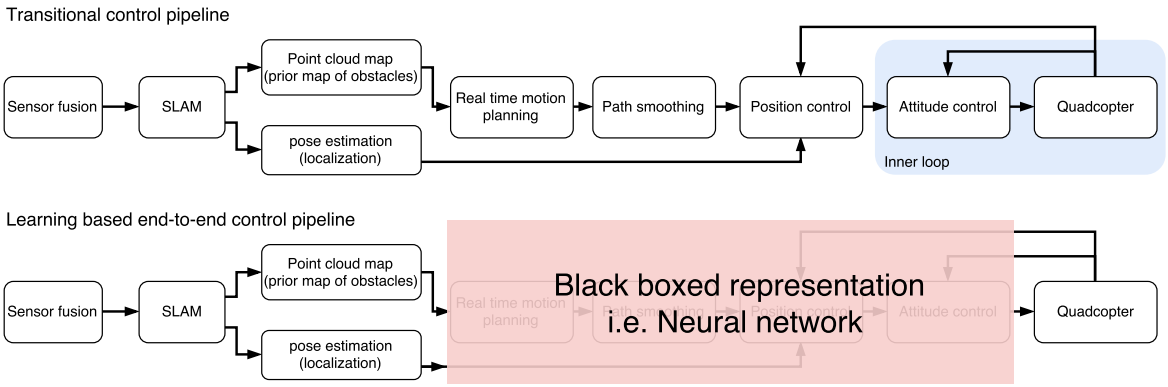


Fig. 1.1 A comparison of the architectures of a traditional control pipeline and a learning based model-free control pipeline.

Learning is a gift that given to human, however understanding the mechanisms of how human brain is involved in controlling our body for learning and executing new motor skills are still a remaining mystery, but from observations, the general outcome of leaning new motor skills for human is through brain simulation for strategies and continuous practice, the performance of specific motor skill can be refined through accumulating the practice and experience, eventually the skill becomes a permanent and autonomous action. In robotic applications, the model-free learning method studies the problem of how robots can emulate the human learning behavior to learn motion control strategies from sensory information for real-world tasks, robots can not only learn and perform generalized motor skills like human that are capable of coping new environment and situation, also conversely improve the understanding of the mechanisms of human brain during the learning procedure. In the nutshell, Figure 1.1 shows the key idea of benefit behind learning based model-free methods is to avoid the explicit construction of the complex configuration space (such as the complex planning problems, or even the computationally intense localization problems) and instead conduct a training of possibly a complex learning system, such as a parametric neural network, which the trained representation of the learning algorithm considers as a "black box". In this way, a series of complex navigation control problems can be simplified into a representation learning problem.

The ultimate goal in our research is applying model-free methods to achieve autonomous navigation from initially unknown environments for computationally constrained micro aerial vehicles (MAV), and discover efficient control policy representations with the focus on learning based methods, we seek to identify appropriate model-free approaches for both, the learning efficiency and (near-) optimal control policy. In this dissertation, we describe learning algorithms with formal performance guarantees which show that each of proposed methods can effectively solve control problems of maneuvering micro aerial vehicles —

mainly address the problems of model-free learning methods for (i) Provide algorithms for improving data collection and data fitting challenges, improve the exploration. (ii) learn to perform maneuver tasks without any prior knowledge.

1.1 Objectives

Our goal is to research the feasibility of using 3D simulations or real-world hardware to train a control policy for micro aerial vehicle (MAV) to perform various tasks of maneuver manipulation from scratch, the MAV will learn to navigate without any prior knowledge of the environment, using only sensory data as input. Achieving this would help us to discover a generalized model for navigation tasks, solve practical problems of model-free algorithms, and open up the possibility of transferring learned policy representation from a simulation setting to a physical hardware setting without any reinforced training. In order to accomplish this, we have to:

1. **Understand existing learning based control solutions.** The first aim is to get a good understanding of the underlying area of model free learning. This requires detailed knowledge of core concepts of learning algorithms in both supervised learning and deep reinforcement learning. Once confident with these core concepts, related state-of-the-art works need to be implemented and exam the improvements of our proposed works.

2. **Explore machine learning libraries and robot simulations.** A large proportion of time was spent developing and running experiments using different techniques. It was therefore necessary to explore a range of machine learning libraries (such as tensorfow and theano) and robot simulations (V-REP, gazebo, and unreal engine) that would be most beneficial to us.

3. **Implement a training framework.** A training framework that implements our supervised and deep reinforcement learning solutions needed to be created. This framework includes the simulation environment that suitable for acquiring MAV sensory data for testing the learning algorithms, especially for deep reinforcement learning, this needs to be developed such that swapping from simulation settings to real-world required as little effort as possible.

4. **Experiment iteration.** We look to evaluate how supervised learning and deep reinforcement learning techniques are suited to solving the problems of navigation. This requires us to iteratively collect training data and train learning algorithms that would

either converge or diverge, evaluate and optimize the result with experiments and parameter tuning to study the feasibility of applying learning based methods to the simulated and real robotics, and finally improve the system based on these results.

5. **Communicate with real-world robot.** Ultimately or possibly, we hoped that the trained agents in simulation could run in a real world robotic settings. This required us to develop an interface to the real-world settings of the MAV that could interpret commands from the simulation-trained network, such as robotic operating system (ROS) framework.

1.2 Challenges

The work of training robots with model-free based methods is very challenging, particularly in real world settings, it is due to the difficulty of data acquisition (collecting data is very expensive in real world) and efficiency of learning algorithms, it is hard, even with the state-of-the-art research and development (the deep reinforcement learning methods, i.e., Deep Q Network (DQN)). This work entailed gaining a full understanding and appreciation of the field with almost no prior experience. During the research of applying learning based approaches to robotics, the biggest challenges encountered were:

1. **Inaccurate state estimation.** For ground operating robots, the measurement of so-called "zero position" exists, it is when all actuators stop and the resulting velocity is equal to zero. However, this does not exist in the nature that has demonstrated to have excellent 3-D navigation and motor skills. Alike the nature, the "zero position" is not applicable to a micro aerial vehicle (MAV), because it is impossible to stop all actuators for an airborne vehicle. Hence it is particularly challenging to apply existed control systems of ground robots directly to MAV that is naturally unstable, such as motion planning. A precise state estimation plays a critical role in the accuracy of controlling MAV, and state must be made available at all times when in the air. Although research in the perception for state estimate of MAV is not our main focus, the main state estimation entity of our indoor experiment for MAV is fully depended on the fusion of the vision and inertial measurement. Therefore, we would encounter the challenges of seeking to identify well established and well developed sensory fusion approaches for estimating the state to make sure the noise of state estimate is minimized.

2. **Long training times.** A single experiment would usually run for many days before producing any conclusive results. The progress can be slow down in many ways,

despite the fact that when experiments crash due to bugs or our machine being powered down unexpectedly, there are also problems of hyper-parameter tuning, requirement for large amount of real world data for training, and moreover the slow convergence property of learning based methods or the probability of divergence during training.

3. **Experimental hardware setup.** Due to the long data collecting and training times, it was often challenging to decide what experiments would be most beneficial to run. Making large changes from one experiment to another would make it difficult to know why an experiment failed, yet on the other hand too little of a change would result in slow iteration progress. Moreover, the hardware limitation is also an challenge when carry out the experiment, such as the battery limitation makes quadcopter can only operate for 10 minutes, therefore can only produce limited amount of data for single charge of battery.

4. **Dynamic models of quadcopter simulation.** With large amount of possible robot configuration settings and body dynamic models, only a general underactuated rigid body model is commonly available to the micro aerial vehicle (such as the micro quadcoper), because the ground robots, such as rover, robotic arm, and full dynamics are well-studied for these robot configurations rather than the newly evolved quadcopter. Hence it is challenging for simulated quadcopter to tackle all the dynamics of the real world settings.

5. **Machine learning libraries.** As today's open sourced machine learning libraries for deep learning are too new, such as the tensorflow: Google's new machine learning library, was used to build and train our neural networks. Although the community for deep learning is still growing fast, the tutorials on specific features are still lacking or missing. This made development more difficult than choosing an appropriate and well-established library that has been in production for years. Having said that, we are glad to have had the chance to help testing and developing the library in an advanced setting.

1.3 Contributions

We have demonstrated in the chapter 3 of a supervised method with manual demonstration of maneuvers in real world settings and chapter 4 of policy gradient method with deep model in simulation settings, both have the ability for learning control policies for aerial robot maneuvers using sensory data alone as input. Along the way we have tried many variations that we hope will be insightful to future work. Our key contributions are:

1. **Learning aerial vehicle control policies.** We present two different perspectives of learning control policies: an associative memory based self-organized incremental neural network learning system (SOIAM) and concurrent deep reinforcement learning system (CRL) for aerial robotic, we trained and evaluated the system in real world robotic setting and 3D simulation settings respectively. We train and evaluate our systems that learning policies from both real world and 3D simulation is possible.

2. **Generalization of agents.** Experiments are trained with variations in aerial robot configuration and target positions. The agents are able not only to generalize to unseen states, but also to slight variations in the target dimensions and ignore the introduction of small amounts of clutter into the scene.

3. **3D micro aerial vehicle simulator.** We developed a 3D aerial robotic (specifically a quadcopter) simulation that replicates the retailed quadcopter, the AR.Drone. The simulation is fully customizable, and we implement based on the Openai reinforcement learning platform that enable the chance for faster training on other algorithms with our simulation.

4. **Concurrent approaches to learning algorithms.** We proposed a concurrent learning algorithms (ATRPO) to combine concurrency with the state-of-the-art techniques of trust region policy and generalize advantage estimation, this ensures multiple agents explores interesting areas in multiple environment instance. We show that this method reduces training time and in some tasks reaching higher episodic reward than other methods.

5. **Training SOIAM directly to real-world setting.** We developed the SOIAM based training framework. As well as training the SOIAM for control policy in real world settings with real hardware of quadcopter, we also evaluate the SOIAM in the real-world without any additional training. Ours approach is the initial and first work that attempts to apply SOIAM algorithm for both training and evaluating the control policy in the real-world and demonstrate that this is possible.

6. **Learning from sensors alone.** In recent research, related work in learning discrete control policies with image resources directly, such as playing Atari games. Our work uses sensors alone as input, and perform action in continuous domain.

1.4 Accompanying publications

1. **Huang Pei-Hua** and H. Osamu, "Associative-memory-recall-based control system for learning hovering manoeuvres," 2015 International Joint Conference on Neural Networks (IJCNN), Killarney, 2015, pp. 1-8.

2. **Huang Pei-Hua**, and Hasegawa Osamu: "Learning Quadcopter Maneuvers with Concurrent Methods of Policy Optimization." Journal of Advanced Computational Intelligence and Intelligent Informatics 21.4 (2017): 639-49. Web.

1.5 Structure of this dissertation

We summarize the following structural outline in the respective chapters of this thesis:

- **Chapter 2.** Following with Chapter 1 of an comprehensive introduction of the thesis, it introduces the background of core material covering the fields of hardware development of micro aerial vehicle (MAV), and its general dynamic models that most simulator used. We then cover the perception part of the MAVs, especially the self-localization concept for MAVs state estimation. Finally, provide relevant state-of-the-art works in these respective fields.

- **Chapter 3.** We provide, a presentation of our first achievement on real world MAVs - with the associative memory based method to self-organizing incremental neural network (SOIAM). In this first method – a model free supervised learning method – we highlight our modifications to adapt the associative memory concept to the self-organizing neural network (SOINN), in this chapter, we introduce the training and evaluating procedure of the quadcopter manuevers in real world settings for hovering task. The results show that it is sufficient to learn a dynamics model from manual demonstrations.

- **Chapter 4.** After we analyzed previous algorithms that described in Chapter 3, in order to resolve the problems raised in Chapter 3, we concentrate on our second method in concurrent deep reinforcement learning – the asynchronous trust region policy optimization (ATRPO) – In Chapter 4, we present both asynchronous and synchronous frameworks combined with reinforcement learning algorithm that resolve the exploration problem in both the SOIAM method and general model-free learning methods, we also provided empirical studies of continuous Mujoco simulation, eventually evaluated the results of empirical studies of the proposed method, thtat

demonstrated improvement on the learning efficiency of tasks with different forms of parallel learning paradigm. Additionally, described how the proposed system can improve the issues of data acquisition and exploration in our first method that described in Chapter 3. Furthermore, the resulting design of the system will then enable us later to manipulate velocity control of the the simulated quadcopter for specific tasks.

- **Chapter 5.** In this chapter, we test our previously proposed systems that are theoretically analyzed in a simulated quadcopter setting in real-world scenario, with tasks: hovering and inverted pole balancing. These experiments examined and evaluated different sensitivities of the proposed system according to various quadcopter task complexities and demonstrated a functioning, model-free concurrently learning based system for quadcopter application. Also the experimental results show the proposed frameworks outperforming the conventional methods by gaining faster convergence of learning a good control policy.

- **Chapter 6.** This chapter comprehensively concluded this thesis with additional literature reviews that related to the achievements of this thesis, and discussed the feasibility of future works of combining SOIAM method and ATRPO method to improve the stability of learning, as well as some high level applications such as transferring from simulation setting into real world setting.

Chapter 2

Background

In this chapter, we provide contents to fully explain some core concepts that the rest of this dissertation is based on. The sections introduce concepts in both aerial vehicle hardware and SLAM — both of which are vast fields in themselves, even though they are not the main contribution in our research, they serve a crucial role to successfully complete the experiments. Then section followed by a detailed discussion of relevant work in the field.

2.1 Micro Unmanned Aerial Vehicle (MAV)

The official name for this technology is the Unmanned Aerial Vehicles (UAVs). the public and mainstream media generally referred them as 'drones'. Since the term was actually taken from the long-standing use of military drones but this is not necessarily the best description for today's consumer and commercial aerial vehicle. Although the legal requirements are still quite restrictive, UAV applications are becoming widespread, from military usage to civil applications, such as aerial imaging, agriculture and even delivery services.

The MAVs refer to the size of airframe, micro size has smaller frame when in operation, and they offer new perspectives for civilian transportation and services with their agility, speed, and occupied space for storage. However the MAVs have three major constraints:

1. **Payload constraint.** For a commercial MAV to hover, every 10 grams in increasing payload weight will require roughly 1 Watt of additional lifting power. Hence it is a trade off between the limited payload budget and lightweight sensors to provide information that is rich enough for the maneuver tasks.

2. **Computational constraint.** Generally, more powerful and robust onboard sensors and computational units consume more power when operating a MAV, as the result that less power is available for the MAV's propulsion system. Both the autonomy capability

Table 2.1 UAV airframe comparison

Airframe type	Hover and vertical takeoff	Mechanical complexity	Electronics complexity	Cost and difficulty of repairing after crash
Airplane	No	Medium	Low	Low
Multicopter	Yes	Low	High	Low
Helicopter	Yes	High	Medium	High

and operating time of MAV are indirectly or directly affected by first two constraints that aforementioned.

3. **Remote sensing constrain.** The strength and availability of the sensor signal. By definition, most ground operating robots have limited range of operation, which it will limit the field of operation to be as closely to the scene as possible, but aerial vehicles can travel some distance away in a short period of time from the ground control, such that remote sensing has higher chance of failure (e.g. distance measurement sensor). In addition, some limited conditions, such as indoor that global position system (GPS) cannot reach or high magnetic field interference area where may only receive a extremely noisy signal (e.g. magnetometer sensor or GPS compass).

2.1.1 Hardware architecture of MAV

The most popular UAV airframe can be categorized into three main types: airplanes, multicopters and helicopters. Table 2.1 summarized the advantages and disadvantages that must be considered when operating a UAV.

We can see from table 2.1, airplane of fix-wing aerial vehicle is not a good fit for our research objectives, despite the fact that this airframe type is the most aerodynamically efficient, which provides the longest flight time and range, it is not capable of hovering and require longer distance to take off, operating this type airframe indoor with obstacles is considering difficult. The multicopter type has advantage on the cost and simplicity of repairmen after crash, and it is one of the most popular types of UAV airframe configurations in recent years. The advances in electronics have made components smaller and cheaper for the multicopter to be built and repaired, hence it becomes a viable alternative for general UAV applications. The multicopter uses multiple propellers to lift vehicle by providing downward thrust force, instead of using a single lifting rotor like a helicopter. The multicopter is capable of controlling its orientation by adjusting the speed of each of the propellers independently,

causing differential thrusts and torques. Multiple types of multicopter exist, and are identified according to the number of propellers they mount and how they are arranged.

We found that for most quadcopter control literature uses a "+" configuration instead of "x" configuration, however through out this dissertation we use "x" configuration for all the experiments and simulations. As the reason for choosing an "x" configure been:

1. **Camera clearance.** An "x" can have larger clearance than a "+" configure to mount a camera pointed forward, the view is not obstructed by the frame arm more easily.

2. **Visibility of Yaw.** A "+" will have one marked front arm. An "x" will have two. As an operator of quadcopter, we consider it is easier to identify the orientation of the quadcopter when flying in third-person-view (TPS), because the front arc with two front lights/marks is better to recognize than one. This is a very safety critical consideration during the situation of recovering.

3. **More torque for rotation.** The moment of inertia is the same for both type of frame, therefore the difference is really that you can torque with all four motors, and have 2 to the square more available torque to rotate. You can get about 41 percent more rotational acceleration from an "x" than a "+".

A quadcopter has four rotors in cross multicopter configuration. As we can see from Figure 2.1, the two diagonal pairs of motors (w1, w2) and (w3, w4) always turn in same directions. By changing the motor speed, we can move the vehicles in different directions in 3D space. Initially, suppose all the motors have the same speed as shown in Figure 2.1(a); rising the speeds of all four motors altogether generate upward movement. Changing the speeds of motor pair of 2 and 3 (w2, w3) or 1 and 4 (w1, w4) change the rotation of roll angle as well as a lateral motion. Changing the speed of motor pairs of 1 and 3 (w1, w3) or 2 and 4 (w2, w4) result in the rotation of pitch angle as well as the longitudinal movements (see Figure 2.1(c),(d)). Finally, the counter-torque resulting from motor pair 1 and 2 (w1, w3) or 3 and 4 (w3, w4), generate the yaw rotations as shown in Figures 2.1(b).

2.1.2 Dynamics model

In practice, to apply learning based methods for optimal control in real world settings, the dynamics of the system are generally not known in advance, and often need to be learned from iterative observations of the system. However in the case of simulation, the dynamics are initially required to be defined. This section we introduce a general model using Newton's law for simulating the quadcopter. To note that, this model does not reveal an identical settings as the real world quadcopter, since it is simply a simulation.

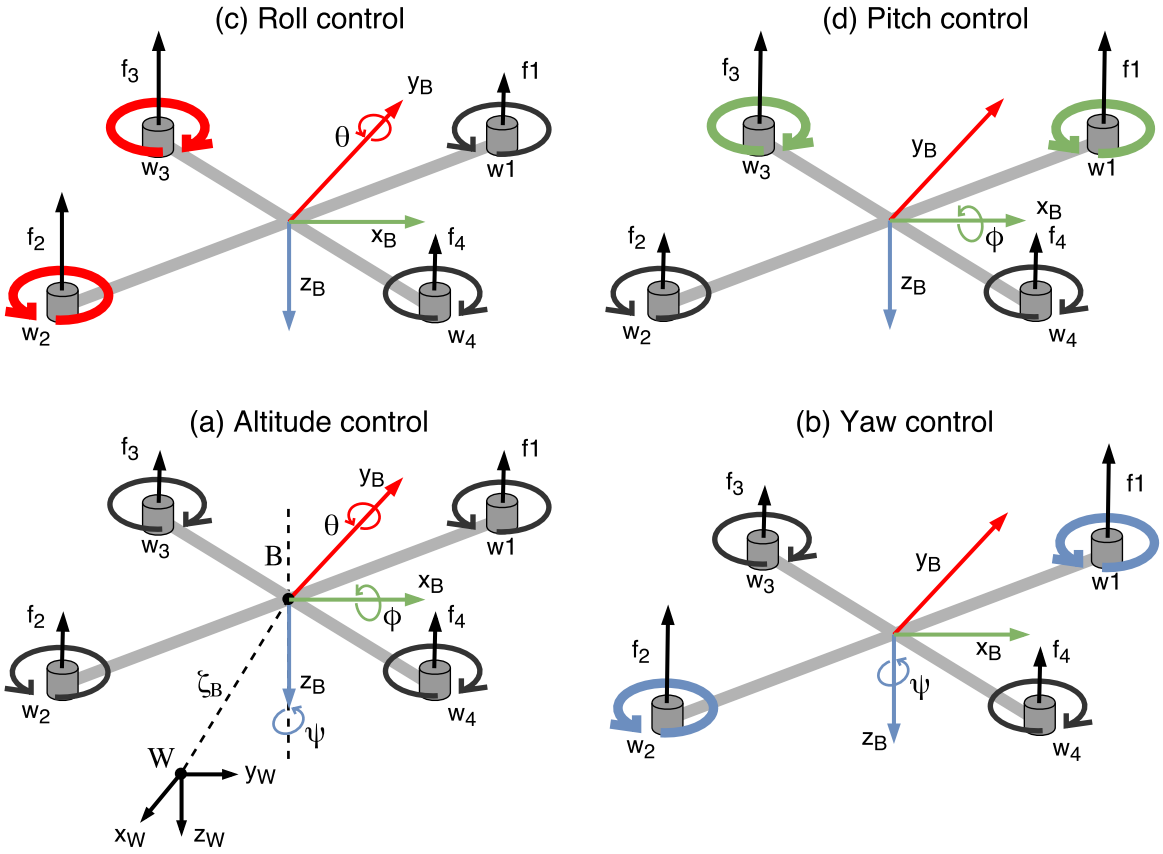


Fig. 2.1 The quadcopter schematic of "x" configuration with world and body frame. The arrow width is proportional to the rotation speed of corresponding motors, and the color represent resulting torque that applied to the quadcopter frame.

Figure 2.1(a) shows the general scheme of a quadcopter with "x" configure and the relevant coordinate frames and variables for the quadcopter. The quadcopter is modeled as an underactuated rigid body where net thrust is constrained along the $-z_B$ axis, where B denotes the body-fix frame of the quadcopter, which attached to the center of mass of the quadcopter, it translate and rotate with the quadcopter: x_B, y_B and z_B co-respond to the coordinates of the body-fix frame. The world frame is denoted as W, it is also called the inertial frame, in our case it is implemented in with a North-East-Down (NED) orientation; the ϕ, θ and ψ correspond to the conventional roll, pitch and yaw angles; and to note that the w_i marks the angular velocity of each motor respectively. The dynamics of quadcopter is defined as [51]:

$$\begin{aligned} \dot{\zeta} &= \frac{d^W \zeta_B}{dt}, & m\ddot{\zeta}_B &= mgz_W - u_1 z_B \\ \dot{R}_{BW} &= R_{BW}\hat{\Omega}_{BW}, \quad J_B\dot{\Omega}_{BW} = \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} - \omega_{BW} \times J_B\omega_{BW} \end{aligned} \tag{2.1}$$

where the ζ_B is the body frame position, $\dot{\zeta}_B$ is the body frame velocity. The R_{BW} is the rotation matrix from the body frame B to the world frame W, the Ω_{BW} is the angular velocity of the B frame with respect to the W frame. The g is the gravity acceleration and m is the mass of the quadcopter. The quadcopter will then have a state of $[\zeta_B, \dot{\zeta}_B, R_{BW}, \Omega_{BW}]$ and control set of $[F_{zB}, M_{xB}, M_{yB}, M_{zB}]$, where F_{zB} is the force applied along the z-axis of B due to the net thrust; and M_{xB}, M_{yB}, and M_{zB} are the moments about x, y, and z axes of B, respectively, due to individual motor thrusts or torque. Through adjusting the parameters of Eq 2.1, the control principle can be transferred from one quadcopter to another.

2.2 Current research state of MAV navigation

Autonomous MAV research is relatively not as mature as the other research of ground-operating robotics, such as rover and robotic arms for picking operation, but advancing very fast in recent years due the operational needs for various applications. Although a lot of effort has been put into this research topic during the past few years, we still endeavor to find micro autonomous aerial system with learning based method in unknown, mapless, and GPS-denied environments. For learning based methods to tackle higher level tasks of MAV navigation, i.e., the autonomous and rapid indoor exploration, swarm formation fly, and long term global trajectory planning and optimization, it is necessary to solve the elemental issues of navigation.

Robust autonomous MAV navigation system requires the combination of the precise perception and the generalized control policy, navigation system will not work well without one of them. In order for MAV to perceive the world of known the positional information, the most common way is to use GPS or DGPS (an approach by fusing IMU and GPS data together) on the MAV. The MAV can be fully stabilized and controlled. The major issue of adopting GPS-based solutions, is limited location and precision for receiving adequate GPS satellite signals. even though industrial graded GPS can have accuracy of position estimation up to few centimeters depends on the location on earth, the consumer GPS is degraded to meters, and it is completely unavailable in indoor environments. Alternative solutions of adopting laser-based range finders have recently succeed in replacing the GPS for indoor navigation, such as LIDAR (the Light Detection and Ranging). Bachrach et al. performed autonomous maze navigation of ground robot with the use of a Hokuyo single plane laser scanner to construct map with a 2D SLAM. Although very convincing results for indoor navigation tasks, the range finders are still expensive solution for robotic navigation, due to not only its cost, also the limitation of their perceptive distance and field of view (FOV, typically in single plane) and are considerably heavy for MAVs, hence they are not optimal for MAVs that have restricted payload weight.

Vision is a viable solution for GPS denied environments. The most common way in a laboratory environment is to install external tracking cameras or motion capture cameras in known and limited locations and to have them track the MAVs within a flying arena[55]). However this motion capture approaches are not feasible and practical enough for large environments and for missions, but for testing model development and deployment purposes, they are very efficient and robust, also provide reliable ground truth reference to the evaluation of other new control algorithms and SLAM approaches. Impressive works, such as cooperative ball throwing and catching of MAVs (Robin Ritz et al. [67]), constructions built by flying machines(Augugliaro et al.[31]), or quadcopters ball juggling (Muller et al. [59]) have been achieved recently using the external tracking system.

In other hand, even with perfect precision of state estimation, without a continuously stable and accurate performance of motion control, the MAVs will not be able to complete any task or maneuver, or even to stabilize itself, this is due to their instability and non-linearity in nature, the solutions normally yield to a complex combination of sensory feedback and model controller. The simplest approach of modeling the control system of MAVs is to construct a "double-feedback" loop control structure, each feedback loop is considered as single subsystem: One control loop (as the inner-loop) for the attitude of the MAVs (responsible for stabilizing the 3-D angular orientation), it can be handled by simple model-based controller, where the other one (as the outer loop) is covered for the 3-D position control. In practice,

to tackle the MAVs' high dynamics and agility, stabilization of MAVs needs to take place all the time, therefore the attitude control loop has to run faster than the position control loop. Many different and robust control theories and methods are employed to design the attitude stabilizer or motion controller for MAVs[53]. Bouabdallah et al.[12] in 2007, first apply two different control techniques, PID and linear quadratic (LQ) techniques to the MAVs. Subsequently, other research works found out a PID- or a simple PD-controller is suitable enough for serving general purpose[30][69] of motion control. In early stage of MAV development for model control, Bouabdallah et al.[12] proposed Backstepping and sliding-mode control method to handle the nonlinear models and dynamic uncertainty to cope with external disturbances for MAV (quadcopter), and H infinite control[29] algorithms are used to improve the robustness of the system.

There are two well-known issues with the aforementioned model based controllers: (i) The performance degradation in case of model uncertainties and unknown disturbances. (ii) The requirement for bridging the gap between state estimator and control algorithms with path or trajectory planning for position control, thus, increase the complexity of the system. The potential ways to solve these problems is intelligent control methods such as fuzzy logic control[18], neural networks control[82], and learning based control[13]. The challenge of utilizing these techniques is the slow convergence or divergence may cause failure in real time control system when training the controllers, and the efficiency of data requirement, since learning based method require large amount of real data to search for a good policy. Throughout this dissertation we address the issues by proposing two approaches: supervised learning based approach and deep reinforcement learning approach, which is a combination of deep neural network and reinforcement learning.

2.3 Simultaneous localization and mapping (SLAM)

SLAM systems assist robot to estimate both its relative state of motion and the structure of surroundings for re-localization, when the 3-D position is not known in prior to current environment, this is achieved through the extraction of distinctive features from the surrounding environment. There are two popular detection methods in SLAM system that are commonly deployed in practical robotics: LIDAR (the Light Detection and Ranging or the Laser Imaging Detection and Ranging) and vision (i.e., rolling/global shutter camera, RGB-D camera). LIDAR is a surveying method that illuminate the target with a pulsed laser light, and detect the differences in the laser returned times and wavelengths, the information can then be used to generate digital representations of the 3D space. The commonly known LIDAR mapping algorithms for robotics are the GMapping in openSLAM (Giorgio Grisetti,

et al. [31][32]) and the hector slam proposed by S. Kohlbrecher et al.[49]. The vision method is considered of using on-board camera as a real-time motion sensor, it utilizes the image frames from camera to calculate the relative 3D position and orientation between the scene and camera.

We generally consider using vision based SLAM (or visual SLAM) with on-board cameras to conduct long-term task of navigation in an initially unknown environment, because camera is rather light weight and cheap when compare to LIDAR solution, which camera is beneficial for the MAVs that have limited computational power and payload allowance. In other words, we do not need any prior information about the environment, nor the need of external supports, i.e., a Motion capture camera system (The Kinect or Vicon) and heavier LIDAR equipment, in order to obtain a reliable state information for MAV control. In this thesis, we applied a feature based visual SLAM approach, called visual parallel tracking and mapping (PTAM) that proposed by Klein and Murray[46][47], using a monocular camera. PTAM generates a point cloud map that is considered as an entity of represented information about the environment, the map is automatically built by captured frames of the MAV's on-board camera, and at the same time the map is used for localization as a real-time pose estimator that provides information of MAV's 6 Degree-of-Freedom (DoF) poses (position and attitude).

However, there are two issues when applying PTAM in practice. Firstly, we need to figure out a robust way to process the vast amount of imagery information, which frequently captured by the on-board camera of the MAV that has restricted on-board computational power. The best solution is to bypass this issue by performing our PTAM offline, the MAV serves as a streaming server for transferring camera stream wireless to a remote PC that is capable of running PTAM. Secondly, the drift of visual SLAM; the problem is reported by Ahrens et al., 2009[4] which the accuracy can decay when perform navigation in large environments where requires longer period of operation and evaluation. This is a well known issue in the field of SLAM research, and a well studied research direction to improve the SLAM. The drift is commonly reduced by fusing the camera information with an inertial measurement unit (IMU) sensor in a extended Kalman Filter (EKF) SLAM framework.

2.3.1 A monocular SLAM: parallel tracking and mapping (PTAM)

The visual SLAM algorithm PTAM is used in order for the MAV to localize itself with a single camera in 6 DoF from an initially unknown environment. In computer vision, PTAM is a map based approach that is similar to structure-from-motion methods, it is using key-frame-based algorithm to extract the features from multiple successive frames, Strasdat et al.[75] demonstrated to be more computationally efficient and robust among other visual SLAM

algorithms. To summarize, PTAM splits the SLAM framework into two separated processes: the mapping and the tracking. The mapping is straight forward for storing the salient features that extracted from the camera frames as a search-able point cloud map, where the tracking is functioned as pose estimator to determine the 6 DoF poses of the camera, by comparing the extracted salient point features with the stored one from the mapping process. The processes of the tracking and mapping function is described in detail as:

1. **Data association.** In order to determine the new poses for the camera. The tracking process will employ a simple motion model to search for the features that are matching to the stored feature map from the mapping process in the newly incoming camera frame.

2. **Key-frame selection and addition.** When explore new portion of the unknown environment, to improve the corresponding estimation error between the current frame that projected with mapped features and the observed ones, the tracking process continues to adjust and re-estimate the camera's pose (orientation and position), by adding new key-frame to construct and improve a map of existed 3-D point cloud of the environments. This process of adding new key-frames to the map is considered as computationally critical. A key-frame is selected mainly based on the measured distance between feature points, and proportion of overlap between the previously added key-frame. Detailed selections regarding to criteria are described in depth by Strasdat et al.[75].

3. **Bundle adjustment.** To ensure the consistency of the feature map after new key-frame is added, bundle adjustment is applied to diminish the comparable error between the key-frame poses and feature map positions. Among other visual SLAM algorithms, the bundle adjustment of PTAM is considered as light weight for computation, because it only process the tracking if the newly appeared area is already stored in the points map.

In code level implementation, the mapping and the tracking processes are executed in different computer threads, hence they can obtain benefit from running at different update frequencies. The estimation of the camera poses can run at higher frequency with the tracking thread. However the mapping is the most time consuming process in PTAM, which includes construction of the point cloud map and addition of new key-frames, the performance and robustness can be largely improved through the utilization of the entire computer thread for the heavily loaded tasks. A improved algorithm proposed by Klein and Murray[47] through skipping imagery frames for the mapping process, this considerably reduces computational

complexity. Particularly when using a wide FOV camera, a lot of unnecessary information can be ignored or even removed from a full sequence of successful frames. It is also worth to mention, for slowly moving camera or the one that is stationary in fixed position, the new frame is barely evaluated by the mapping process, thus saving the computational power for processing.

2.3.2 State estimation: Extended Kalman Filter (EKF)

Drifts of the estimated state in the visual SLAM of MAV navigation will cause negative impact to the control accuracy of the vehicle, especially for the evaluation and operation of a long term navigation. The IMU or visual SLAM alone are well known to suffer from constant drifts in their state estimation of velocity and position, hence the EKF is kicked in to minimize the drifts. EKF is a non-linear version of Kalman filter(KF), which is a well-known method to filter and fuse noisy measurements of a dynamic system to get a good estimate of the current state, and EKF is making it applicable to a much wider range of real-world problems. In practice, it is common to mitigate this drift by fusing extra sensor information altogether such as cameras, IMU, or even GPS.

2.3.3 Visual Inertial state estimator

Referring to the remote sensor constraint of MAVs mentioned in Section 2.1, single sensor will not in general adequately provide consecutive information to estimate the MAV state, the EKF approach mentioned in Section 2.3.2 enables the collaboration of multiple types of sensors to improve stability and redundancy of state estimation through fusing and/or inter-swapping their information while airborne. It is popularly studied topic of fusing the IMUs information with the estimated state of a visual SLAM framework. Re-initialization is required if failure occurred in monocular vision SLAM, since the metric scale is not recoverable due lost tracking of the key-frame pose information. However an IMU with known extrinsic parameters can be incorporated to solve the problem by bridging the short failure periods of the unrecoverable metric scale of the vision SLAM framework. As suggested by Corke et al.[17] in practice aspects, the key to successfully combine IMUs and vision SLAM is on the calibration process of the IMU and camera, precise timing matching and known calibration parameters are critically important to enhance the accuracy of motion estimation and consistency of the absolute scale of observed map. The timing incorporation of multiple sensors can be classified by two concepts: the loosely-coupled and the tightly-coupled: The loosely-coupled method approaches the fusion process by matching the closest time stamps, the IMUs and vision units are considered as separated modules that

individual sensor processed at different sampling frequency, such as the works proposed by Kneip et al.[48], Armesto et al.[5], and Weiss and Siegwart[81], on the other hand, the tightly-coupled approach normally performed with hardware matching with a precise clocked frequency, i.e., FPGA, signal processing ICs, the IMUs and vision information are combined together into a single, optimal entity as those described by Lupton and Sukkarieh[42] and Jones and Soatto[40].

In practice, we employ the loosely-coupled EKF method similar to Jakob Engel et al.[24], in order to keep the scale and orientation of the generated map correctly aligning with the PTAM algorithm by Klein and Murray[47]. The EKF is also used to compensate for time delays in the SLAM system.

Chapter 3

Learning maneuvers via manual demonstrations

This study presents our initial experiment on aerial robotic application using neural associative memory-based control system, that imparts online learning and predictive control strategies to a cost-effective quadcopter helicopter, the Parrot AR.Drone 2.0. The control system is extended to tackle a fundamental and challenging problem for the quadcoptor: hovering and trajectory control. We proposed system based on self-organizing incremental neural network with inclusion of an associative memory algorithm. The algorithm can cope with a hierarchical data space and complex time-transition dynamics; it enables online incremental learning from manual control, thereby gradually improve the stability against interference such as drift caused by either mechanical impairment or external excitation. In particular, after continuously learning the associative state-command pair of hovering maneuver, the system can execute the command associated with current state. The proposed system is evaluated on a realistic AR.Drone quadcoptor to test its capacity to tackle the hovering control problem. The results demonstrate that for the first time, the proposed system effectively offers a novel approach to quadcoptor application of an associative memory-based neural network by successfully tackling a hover task through iterative on-line learning.

3.1 Introduction

Recent progress in autonomous control systems has led to a growing interest in quadcoptor applications and research. This is because of their ability to explore real world space through rapid maneuvers and mobility. Consequently, companies recently intensified their interest in achieving greater maneuverability and control stability by offering small scale, commonly

available, commercialized, and low-cost developmental platform for such quadcopter system, which have provided an ideal testing bench for not only the research purposes on sophisticated control technologies but also for entertainment applications. However, despite the fact that a light-weight sensor solution for self-localization is still a critical and popular problems in robotic system, in terms of systematic, the main challenge of the platform that we are focusing on is the absence of an autonomous control solution. Historically, the control of robots is pervasively pre-programmed to safely meet the expectations, such as navigating through space and interact with people in everyday life. We assume that the program is based on code and models that make interaction with them challenging.

In this chapter, we consider the concept of an associative memory (AM)-based approach, which decreases the complexity of interaction between robots and human by enabling the program to learn from environments, and recalling what has been learned in the past to impart a generalized estimation about the current environments. The AM is inspired by the brain mechanism using which it acquires new knowledge, with memorization being a critical mechanism of human intelligence. We assume that the AM is also crucial for applying human intelligence to permit mechanical agents and robots to acquire new motion or knowledge, which allows a robot to complete complex tasks with sufficient accuracy and experience through practice that previously only humans could perform. Few studies have adopted the AM approach for robotic applications in intelligent robots [9] [19]; the AM is expected to efficiently process with large data storage capacity through compression, and it can perform well on online incremental robot learning. This study provides an AM-based neural network to synthesize autonomous control of a quadcopter; the AM-based neural network demonstrated guaranteed performance with generalized estimation of control, when dealing with a noisy environment in real practice.

We chose quadcopter for our study because it allows more freedom for analysing our proposed system. A quadcopter is a helicopter that is mounted with four independent, fixed propellers and motors; varying the speeds of the four corresponding motors interactively provides it with six degrees of freedom to navigate around space. Generally, a commercial quadcopter is also equipped with adequate inexpensive on-board features, such as monocular camera, inertial navigation system (INS), ultrasonic sensor (used for altitude less than around 1 m), and controller processor. Siegwart et al.[73] has analysed and described the abilities of quadcopters in detail. In this study, we address the problem of hovering control by reducing the quadcopter drift error in GPS-denied unknown environments using a learning based approach. The preliminarily described aspects of the proposed algorithm are based on a neural AM network that adapts the drift reduction to autonomous hovering control in disruptive conditions; the overall goal of the system is to offer an alternative method to achieve

stable and precise hovering maneuvers by learning control commands supervised by a human operator. In the present demonstration of our system, it is using only an on-board camera, essentially with a monocular Simultaneous localization and mapping system (MonoSLAM) [46], which has been demonstrated to work well enough with a single inexpensive camera[3]; while the quadrocopter is flying blindly in space, MonoSLAM provides posture estimation to the control system. A survey revealed that the fully functional MonoSLAM system working in real-time visual tracking is the Parallel Tracking and Mapping (PTAM). PTAM has been used to conduct similar researches using an on-board camera [25] to stabilize a quadcoptor with modeling control. The results showed that PTAM provides a robust framework for visual tracking that is adaptable to the testing platform of our proposed algorithm.

The motivation behind our work is to showcase a scale-aware and simple topological AM-based control synthesis that is feasible for the hovering of a low-cost quadcoptor. As the hardware platform is limited, we use the Parrot AR.Drone that is available commonly in toy store, with a light weight of only 420 g and a protective hull, safe to use in public. As the on-board computational resources are very limited, all computations including the proposed algorithm are performed externally on a modest notebook processor through Wi-Fi port, the AR.Drone transmits visual information to PTAM for posture estimation and receives estimated maneuvering commands with the given posture from the system. During the experimental procedure, the AR.Drone is initially controlled by human operator, who "instructs" the AR.Drone to perform hovering maneuvers through a game-pad controller. Sets of associative state-command pair are created by collecting data from PTAM and controller. The system learns the maneuver from these associative pairs in real-time, once sufficient association are learned, the neural AM-based control system can recall the associative command response by providing the current posture to perform hovering control. The system also allows real-time updating to improve the setting of tasks without an off-line retrain. The contribution of this study is two-fold: first, our approach leverage the use of model-free algorithm for robotic exploration, and second, it demonstrates a neural AM-based control algorithm is feasible for enabling a highly dynamic robot to learn from the unknown environment in real-time, without any advance knowledge from static or dynamic modeling.

3.2 The self-organized incremental neural network (SOINN)

The concept of SOINN[28], which was proposed for clustering data and learning topology, it comprises two layers. The first layer is used to generate a topological structure of presented patterns, and the second layer outputs the number of clusters and distribution of the patterns. The learning algorithms of both layers are almost identical; however, they differ in their

inputs. The inputs of the first layer are the data presented to SOINN. After the first layer finishes learning them, the second layer replicates the vectors in accordance with the weights of the nodes that were generated in the first layer. The distances are derived for both the nearest node and the second nearest node from the given input (they are known as the 1st and 2nd winner). A new node with weight equal to that of the input is generated if the distances are sufficiently great. Otherwise, a new edge is formed between the nearest and the second nearest node; the weights are updated for the nearest node and its neighbors. The growth of the network is important for adapting to non-stationary environments. Many conventional clustering algorithms such as k-means require that a user predefined how many clusters are to be generated, also in topology learning problems, the number of nodes is needed to be predetermined in advance to learn. However, the quantities of both clusters and nodes are chosen adaptively by SOINN during learning.

3.3 SOINN with Associated Memory (AM): SOIAM

The Self-Organizing and Incremental neural Associative Memory (SOIAM) is an extension of SOINN that incorporates the AM, which is an incremental, on-line algorithm that is developed to deal efficiently with associative pair data, Figure 3.1 shows the training framework. In previous works, its performance has been demonstrated to be better than comparable methods[76]. SOIAM generally begins training from an empty set that considers the first two input associative data as the starting nodes, and then, when SOIAM holds more than two nodes, it finds the first- and second-winner nodes (I) by calculating the minimum distance between the given associative a_t and the dimensional weight vector W_i of the ith node in all SOIAM nodes (S), i.e.

$$I_{1st} = \text{argmin}_{i \in S} \|a_t - W_i\| \tag{3.1}$$

$$I_{2nd} = \text{argmin}_{i \in S - W_{1st}} \|a_t - W_i\| \tag{3.2}$$

If the distances between new associative input and the first- or second-winner are less than the similarity threshold that defined in [28], the input pattern becomes the first-winner node. Otherwise, SOIAM creates a new node for the new associative input in current network. In the case in which a new input associative pattern is assigned to the nearest node in the SOIAM, the weight vector is updated by the value of new input pattern and an edge between the first and the second winners is created (if edge does not exist). The SOIAM network clustering behaves similarly to SOINN[28], the cluster is formed by connecting existing nodes in SOIAM, this is largely different from other clustering techniques, such as k-means, where a

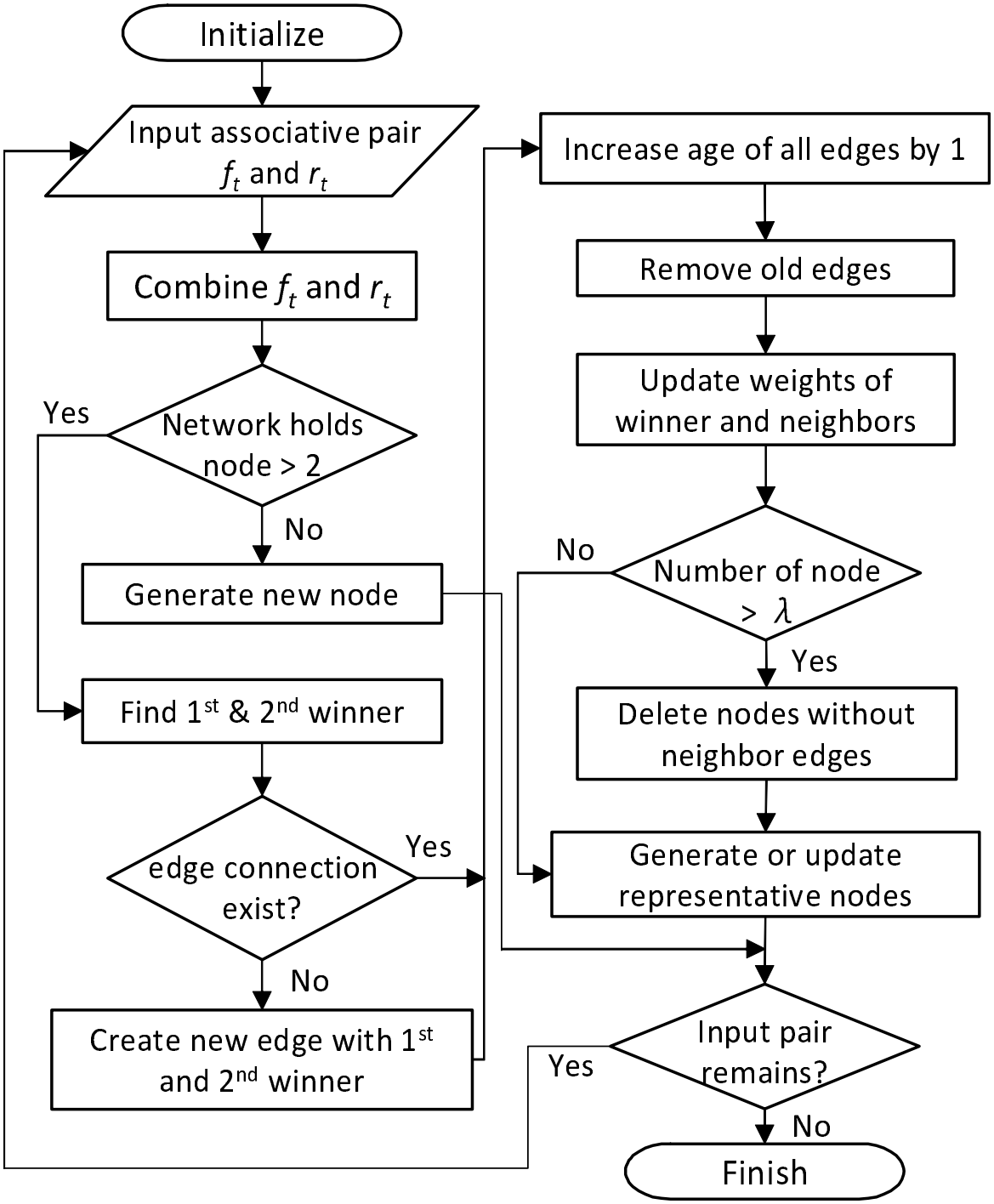


Fig. 3.1 The flowchart for SOIAM algorithm training an associative pair of input data ft and rt.

new input is directly added to form the cluster. Therefore significantly saves computational memory when running for an extended time. Within the SOIAM network, each edge is assigned an age value, therefore every time a node is selected as the first winner, the age increase by 1, the entity permits each node to be removable by setting an age threshold Λ_{edge} and removal threshold λ (the Λ_{edge}=750 and λ=4000 are examined to be practical enough to carry out the experiment), two steps of activities can kill the node: (1) When node exists for a long time without winning any new input pattern, once Λ_{edge} is reached, all connected edges die and are eliminated. (2) As the accumulated nodes increase to reach threshold λ, the node without bonded edge is eliminated from the network.

3.3.1 Associated memory (AM): Value-key pair

Given that substantial navigation information and manual control commands are collectible from the AR.Drone platform, we assume that there are unexplored associations among these data, which largely related to the dynamic control model of the AR.Drone. The proposed system uses the Self-Organized Incremental Neural Network, SOINN, combed with AM-based predictive control method, which is capable of capturing the associations among the data. This turns AR.Drone behavior into a regular pattern, and uses the pattern to estimate control command from the current state of drift error.

The concept of associative pair represents as a fuzzy rule, which in a form of "If *input* THEN *output*"; this study defines the input as an associative key (the posture state) and the output is a corresponding value(responded command). For every given input associative pair a with respect to time t, such that $a_t \in \mathbb{R}^{\omega+\omega}$ for a given length of interval of time series input (ω), which can be derived from

$$a_t = \begin{bmatrix} f_t \\ r_t \end{bmatrix} \tag{3.3}$$

where the $f \in \mathbb{R}^{\omega}$ is the feature vector of that associative key that comprises $(f_t^1, f_t^2, ..., f_t^{\omega})$, and $r \in \mathbb{R}^{\omega}$ is the corresponding value, which comes with $(f_t^1, f_t^2, ..., f_t^{\omega})$. The associative pair f and r rise the concern regard to different measurement unit (i.e. posture state(m) $\leftrightarrow$ control command) during clustering, which may cause error when the euclidean distance is calculated due difference in scale size. Therefore, the associative pair with same unit is grouped when performing training and testing, i.e., posture state with posture state and command with command.

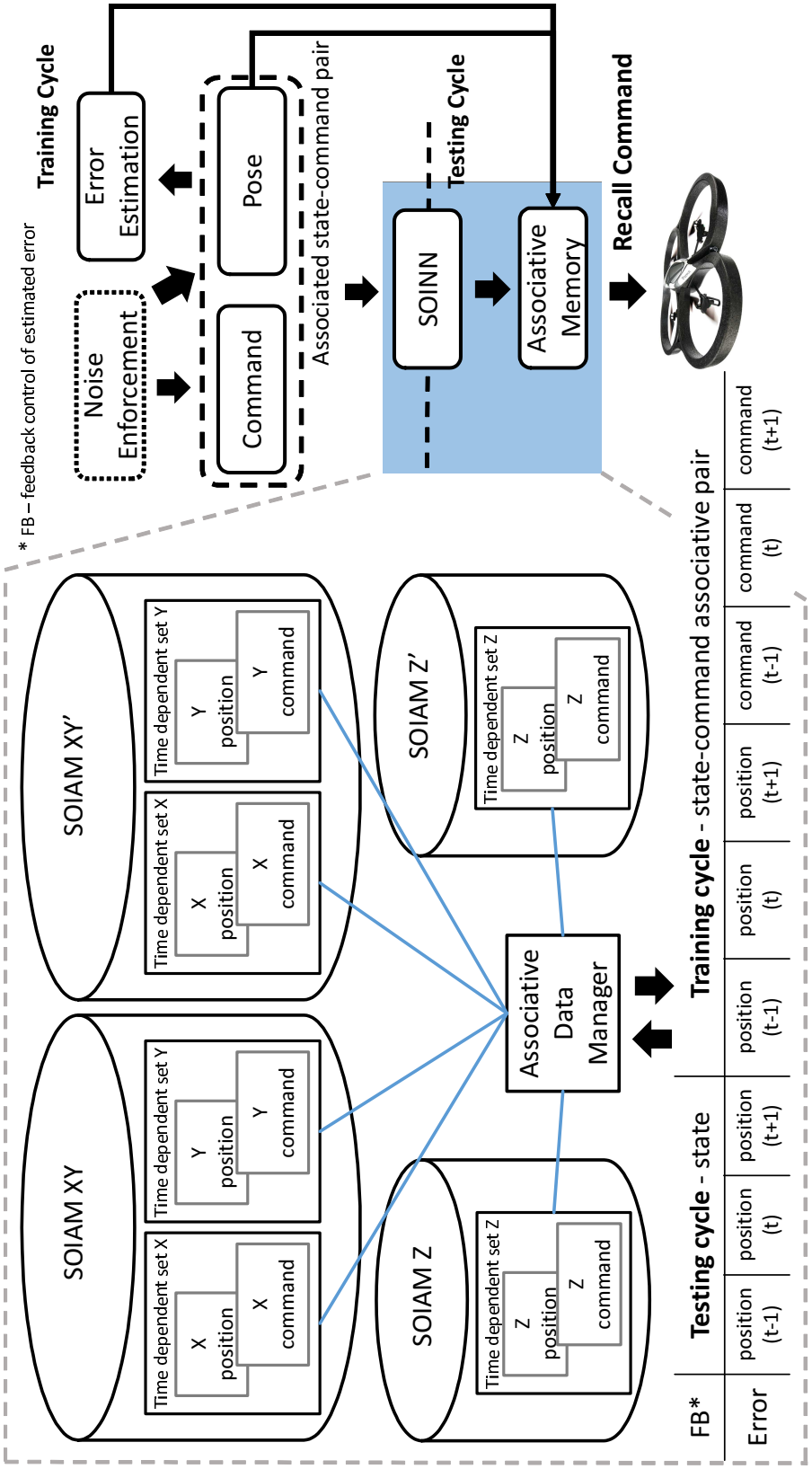


Fig. 3.2 Designed database structure for learning and recalling of the system organized with an associative data manager that deals with incoming and outgoing associative paired data.

3.3.2 Data-Pool Structure

The system contains a database with prior knowledge, which can be incrementally trained in real-time while the AR.Drone is in the air. A six-dimensional vector of training data is received at each time-step as a state-command paired set,$\{p_x, cmd_x\}$, $\{p_y, cmd_y\}$ and $\{p_z, cmd_z\}$, obtained from the estimated position of PTAM, with an associated AR.Drone control command that executed from a USB game-pad controller manually operated by an operator. The training and testing phase of the proposed SOIAM-based system is detailed in Figure 3.2.

The trained SOIAM network maintains a data pool, that stores the associative pairs of the state-command sets in a time series format. Left-hand side in Figure 3.2 shows the designed structure of the data pool, where SOIAM XY and SOIAM Z are the coarse learning sets, which are trained SOIAM network with untuned control command, that gives a full range of command values (ranging from 0 to 1, higher value provides more aggressive fly experience to the AR.Drone), this provides generalized and dynamical manoeuvre of piloted control, whereas SOIAM XY' and SOIAM Z' hold an identical data structure; however these are fine learning set only trained with finely tuned command values(from 0 to 0.34). They are not involved in the cycle of incremental training. The advance of having this fine learning set is to improve the accuracy of control, while maintain the generalization of task performance. In particular, an associative data manager (ADM) is embedded to select which SOIAM network to be incrementally trained or recalled, based on the feedback error estimation given by system, as the error is less than 0.3 m, fine learning set is selected, otherwise coarse learning set is selected.

3.3.3 Time Dependent Learning and Recall of SOIAM

The proposed AM-based neural network control algorithm has two phases: learning (training) and predicted control (testing). The proposed system can switch back and forth between these two phases at any time, allowing it to achieve consistent learning from the complex environments in real time.

This section, we defines the time series data as successive postures measured over time, and assumes that each measured posture links with a corresponding control command. Fig. 3.3 details the composition of time series training and test data, where the time series is represented by a sequence of $t-1, t, t+1$ and $t+2...etc$, and so on. To efficiently predict from the previously learned time-series data, SOIAM is trained through employing a sliding window technique with shifting steps. This process is performed by the ADM, shown in Figure 3.2. It helps to collect and link the time series training and testing data. The number

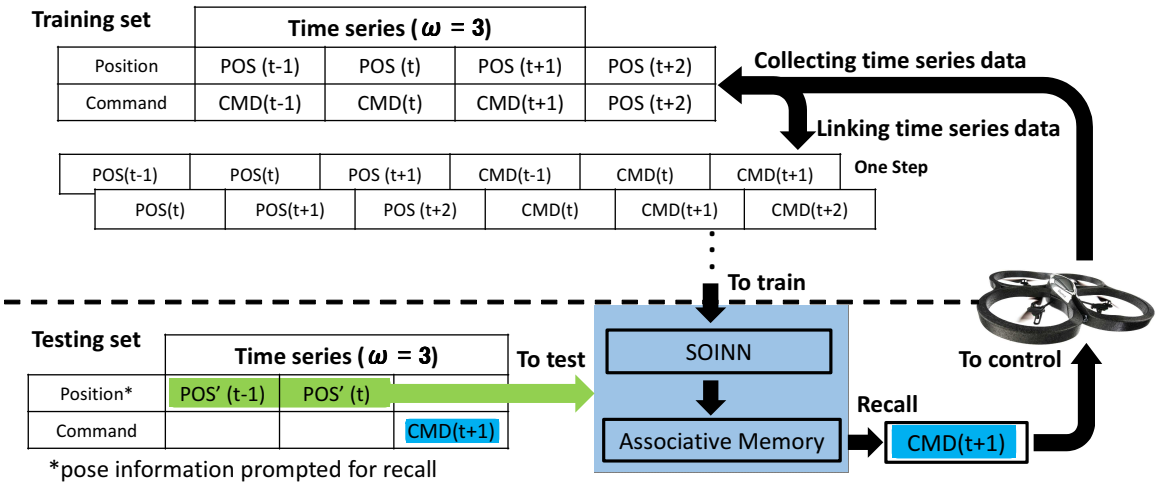


Fig. 3.3 Process of associative state-command pair dataset from the proposed self-organized incremental neural network associative memory, in both training and testing phase.

of training steps at time t is given by $\frac{(s \times t)}{\omega} + 1$, where s is the data sampling rate and ω indicates the time series length of associative pair. This study uses ω=3, a value fast enough to allow the system to respond to the motion of AR.Drone. To perform recall after sufficient steps of the learning cycle, during which the estimated position of AR.Drone continuously provided, the trained system is prompted to estimate the associated commands from the data pool. The performance of the system can be evaluated by logging the series of position changes.

3.4 Implementation of the system

Figure 3.4 summarized the whole implementation, our system consists of the following two major components:

1. Visual SLAM - A monocular SLAM algorithm as described in Chapter 2 (PTAM) is applied to captured video frames and computes an estimate of the AR.Drone's pose with its on-board camera, then using the EKF to fuse the pose estimate from PTAM with available sensor information to predict a more precise pose information.

2. SOIAM based learner and controller - based on an estimate of the drone's position and speed provided by the EKF, the SOIAM based method introduced in Section 3.3 is used to learn and predict appropriate control commands to fly and hold to a given target position.

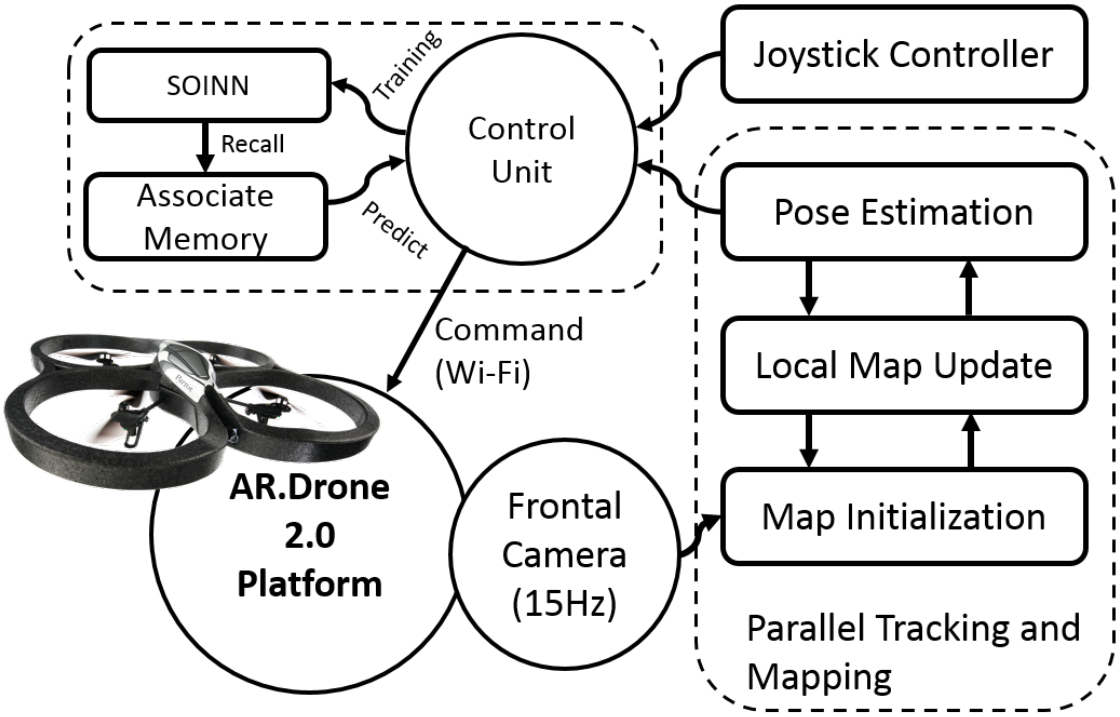


Fig. 3.4 Topological implementation of proposed AR.Drone system.

SOIAM, the AM-based SOINN, is embedded in the proposed system. The entire system including SOIAM, was developed in C++ on a GNU linux platform, Figure 3.4 generalize the system assembly workflow, in which each block is developed separately and then integrated using the open source Robot Operating System (ROS) [65], which provides well-structured interfaces for all blocks. The open sourced AR.Drone Application Programming Interfaces (API) of CVDrone (OpenCV plus AR.Drone, https://github.com/puku0x/cvdrone) are used as the primary control unit, in which OpenCV library is utilized for image handling and processing. The API provides basic control functionality, enabling communication between AR.Drone and notebook for transferring sensor data and sending commands.

3.4.1 The training procedure of system

Figure 3.5 illustrates the experiment setup of hardware for training a SOIAM based controller. To collect training data, we used an XBOX controller to generate the AR.Drone commands (rate of pitch, roll, yaw, and thrust), and auto-associated the commands with the corresponding state estimation of the AR.Drone in a closely matched time frame. Following describes the procedure of how the training data is collected:

Fig. 3.5 Hardware equipment setup for training the SOIAM based controller, including a ground control laptop that exchanges sensing data and control command with AR.Drone through Wi-Fi ad-hoc network. An XBOX gaming controller generates commands from human demonstration for training.

1. Arm the AR.Drone, randomly place the AR.Drone in an initial position within the map that generated by the PTAM.

2. From the Initial position, perform maneuver commands with the XBOX controller, for hovering example, we manually demonstrated the commands of moving toward the target position from the initial position.

3. Continue to collect the training data through repeating Step 1 with different initial positions. We collect X-Y plan and Z plan of training data individually to simplify the procedure, so we can re-use the X-Y plan control with different Z plan control, and they are trained with separated SOIAMs.

4. The use of the XBOX controller leads to dead-zone value in the recorded commands (especially when accurate control for small movements of vehicle), which may not be too much problems for SOIAM during training. However, we provide a controller calibration application that allows to smooth out the recorded dead-zone values through linear mapping.

Large number of repetitions of this training procedure is critical to ensure that SOIAM has enough examples to predict the control commands according to a sequence of states (details are discussed in Section 3.2 and 3.4). The same procedure applies to other tasks, i.e., flying in an interrupted environment with wind.

3.4.2 The AR.Drone 2.0 Hardware platform

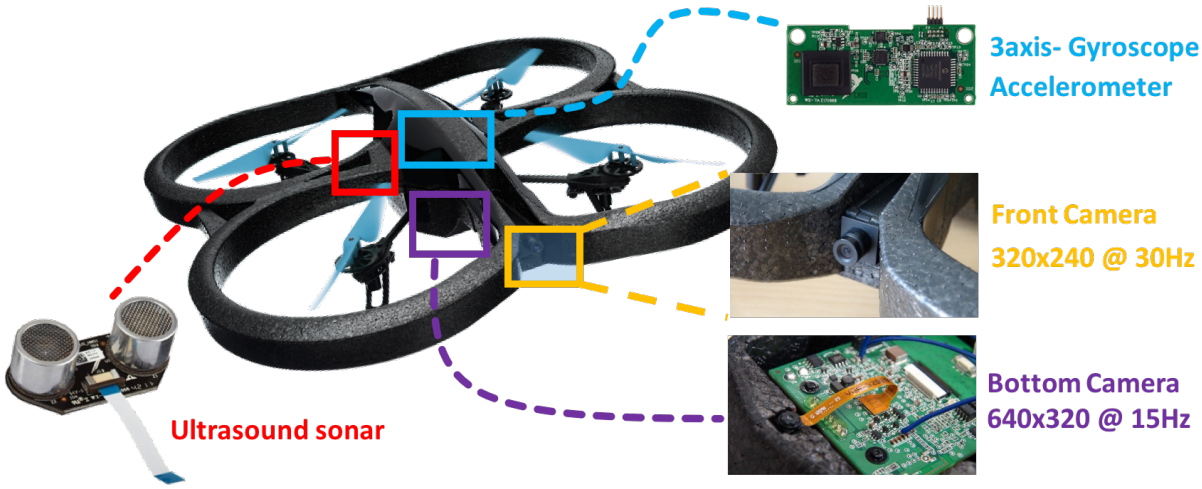


Fig. 3.6 The hardware identification of the Parrot AR.Drone 2.0 platform

The main MAV hardware that used in the experiment is a quadcopter. During last few couple of years, the quadcoptor platform has become widely available to the public, regardless of whether it is commercialized or open-sourced. This study uses the quadcoptor that is available from Parrot Corporation, named AR.Drone 2.0, detailed shown in Figure 3.6, a low-cost, light-weight quadcopter, compared with other research purposed quadcopters, such as Ascending Technology's Pelican or Hummingbird quadcopters. AR.Drone was released to the market in 2010 as a high-technology toy, this type of quadcopter has been used in several researches. Its robustness against accidental crashes is particularly suitable for studying learning based algorithms because it requires practice during training, and it can safely be used in public for demonstration. However a trade-off exists between its price and flexibility of customization, the on-board hardware or firmware cannot easily be modified, and that communication with the quadrocopter is only through 802.11g Wi-Fi connection, this limits the range of flying. The AR.Drone weights roughly 420g, when fully geared with the 1500 mAh battery and protective hull, measures $53cm \times 52cm$, it has an endurance of operation with fully charge batter of 15 minutes when carrying no payload.

The AR.Drone is mounted with four brushless inrunner motor, each motor is capable of 28,500 revolution per minutes, which generate thrust to have an all-up-weight of 500 gram, it is equipped internally with inertial measurement units (IMU), a 3-axis gyroscope that measures angular velocity, and accelerometer sensors for measuring linear acceleration. The measurement of roll and pitch angles are accurate and not subject to drift over time, with

a deviation of only up to 0.5°. The yaw measurements however drift significantly over time (with up to 60° per minute). Furthermore a downward facing ultrasonic altimeter provides relative altitude information to the ground, can measure up to a maximal range of 6 m.

On-board low level computation

The on-board control firmware busy-box version 1.8.1 is running on 1 GHz ARM cortex A8 processor, it is used as low level processing unit (LLP) for performs sensors data fusion for attitude control. Also with a separated video Digital Signal Processing (DSP) unit running at 800 MHz, used to process the video image data.

On-board cameras

The AR.Drone has two on-board cameras, one looking forward and one downward. The forward pointing camera runs at 18 frame-per-second (fps) with a resolution of 640×480 pixels, covering a field of view of $73.5° \times 58.5°$. The used of fish eye lens caused the image is subject to significant radial distortion. Furthermore rapid drone movements produce strong motion blur, as well as linear distortion due to the camera's rolling shutter. (A delay of 40 ms between the first and the last line captured).

The downward pointing camera capturing at faster speed of 60 fps with a lower resolution of 176×144 pixels, covering a field of view of only $47.5° \times 36.5°$, but is afflicted only by negligible radial distortion, motion blur or rolling shutter effects. Both cameras are using lossy compression, and subject to an automatic brightness and contrast adjustment.

3.4.3 The state estimator of AR.Drone

Due to the limitation of our hardware platform of AR.Drone described in Section 3.4.1, the state estimation of AR.Drone is limited to the use of a monocular SLAM, as no plan of modification to mount another camera on-board, hence our solution is based on the PTAM, the well-known key-frame-based monocular SLAM system as described in Section 2.3. It is augmented to make use of the additional sensor information available, in particular to resolve the monocular SLAM problem of scale recovery from the scene.

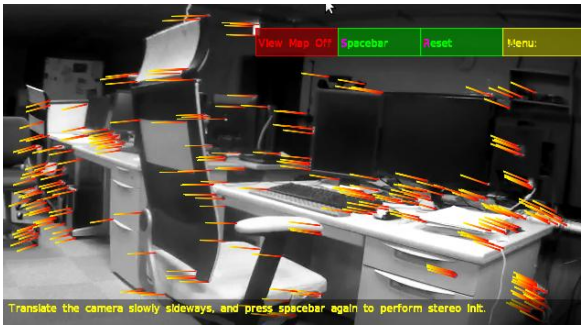
Estimating the state is essential for autonomous navigation of AR.Drone. The estimate is required to not only be accurate, but also the latency as small as possible: the quicker the update, the attitude and position controller can respond more accurately to stabilize the quadcopter. We use an EKF to estimate the state of the drone by fusing the pose estimates of PTAM with sensor measurements provided by the internal IMU. Figure 3.7 shows the

procedures of the implemented PTAM for state estimation by using AR.Drone's onboard forward facing camera. Enabling EKF fusion has following benefit to state estimation:

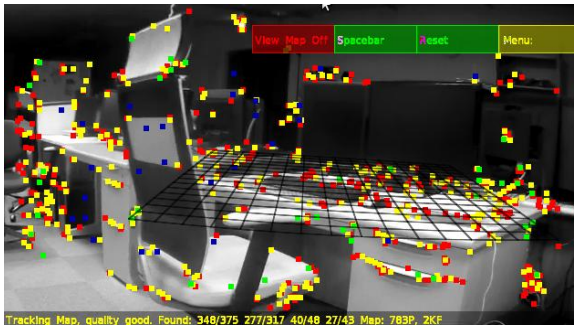
1. It can be used as initialization for tracking process, replacing PTAM's originally built-in decaying velocity model.

2. It can be used as an alternative initialization or speed up recovery for the tracking process when tracking is lost.

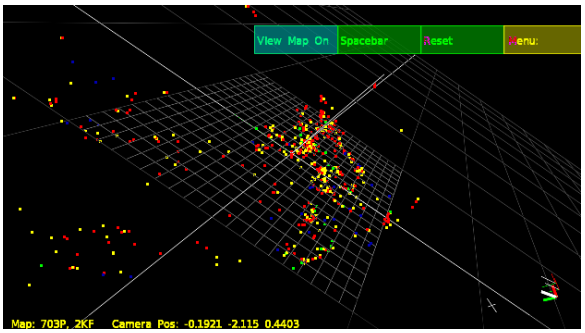
3. It helps preventing inclusion of false key-frames by performing constant checking, as the measurement of roll or pitch angle of the IMU deviates severely from PTAM's pose estimation, the tracking simply discard the respective result, new key-frame is not added to the map.



(a) PTAM key-frame Initialization



(b) Key point location used for tracking



(c) 3D view of built dense map

Fig. 3.7 Screenshots of the procedures of running the implemented PTAM algorithm: (a) Initializing the map with two consecutive frames that generate the initial map from the extracted keypoints. (b) Identifying new key-frame by the tracking, and generate new landmarks and added to the map to perform pose estimation. (c)The generated 3D dense map that maintained by PTAM, it shows the location of all the added keypoints for tracking the currently fused pose of the camera, which on board the AR.Drone.

State space for estimation

The estimated state of EKF is defined to be

$$\{x, y, z, \dot{x}, \dot{y}, \dot{z}, \Phi, \Theta, \Psi, \dot{\Psi}^T\} \in \mathbb{R} \tag{3.4}$$

where

- x, y, and z correspond to the world-coordinates of the AR.Drone's center of gravity in meters.
- $\dot{x}$, $\dot{y}$, and $\dot{z}$ correspond to the velocity of the AR.Drone in meters per second (m/s), expressed in the world coordinate system.
- Φ, Θ, and Ψ representing the AR.Drone's orientation, which correspond to the angle of roll, pitch and yaw respectively in degree.
- $\dot{\Psi}$ corresponds to the velocity of yaw rotational in degree per second.

As the state changes over time and measurements are integrated in irregular intervals, the respective values will be treated as continuous functions of time when appropriate.

3.4.4 Communication functions of implemented system

As soon as the battery is connected on AR.Drone, it broadcasts an wireless LAN network in ad-hoc mode to let any device connect to the network. Once connection is established, the drone immediately starts to communicate by initially sending drone's status to the end-user who connected to it, after AR.Drone receive control command, it will acknowledge to the end-user's signal by continuously sending the navigation data to the end-user every 10 ms through connection. The communication of sending data and receiving navigational commands is handled mainly on four separate channels: navigation channel (UDP port 5554), video channel (UDP port 5555), command channel (UDP port 5556), control port (TCP port 5559, optional). Note that the three major communication channels are UDP channels, hence there may have chance of package lost or receiving in the wrong order. Also note, since this is for experimental use, there is no security measures implement for the data flow control, so someone may connect to and control a running drone at any time.

To integrate and manage the data flow more easily, so the AR.Drone can efficiently and safely communicate through the Wi-Fi network with the high level processing unit (HLP), which is an off-board computer with Intel core-i7 processor, running customer position control algorithms including SOIAM algorithm, the robotic operating system (ROS) framework

is deployed as a communication protocol for sensor messages exchange, parameters tuning for SOIAM and data flow monitoring for interactive command. The AR.Drone sends all the navigation measurements including IMU data at 200 Hz, the ultrasound measurements at 25 Hz to the remote HLP via wireless LAN.

Command channel

The AR.Drone is commanded by broadcasting a stream of command packages, each defining the following parameters:

1. desired roll and pitch angle, yaw rotational speed as well as vertical speed, each as fraction of the allowed maximum, i.e. as value between -1 and 1,
2. one bit switching between hover-mode (the drone tries to keep its position, ignoring any other control commands) and manual control mode,
3. one bit indicating whether the drone is supposed to enter or exit an error-state, immediately switching off all engines,
4. one bit indicating whether the drone is supposed to take off or land.

Being sent over an UDP channel, reception of any one command package cannot be guaranteed. In the ROS implementation the command is therefore re-sent approximately every 10 ms, allowing for smoothly controlling the drone.

3.5 Experiment results

To evaluate the performance of the proposed system, fundamental studies in the flight mechanism of hovering are featured to train the system. In general a hovering task, the controller seeks dynamics on the basis of the position data collected from flight and adjust the dynamics to minimize the errors from a fixed position. The proposed SOIAM-based system is expected to estimate appropriate command outputs to control the AR.Drone. All experiments are conducted in an indoor of laboratory space: autonomous hovering control in stationary and interruptive environments as shown in Figure 3.8, and the hovering results are evaluated in terms of precision and stability, where the precision shows how accurate the system can hover at same place over a period of time, and the stability accesses the average error of hovering over time for the system.

(a) Stationary hovering (b) Interruptive hovering

Fig. 3.8 Experimental environments for testing fly of AR.Drone. A yellow 5cm-by-5cm square marker that placed below the drone indicates the attempted hovering point.

3.5.1 Hovering

In general hovering task, the controller seeks dynamics that based on position data collected from real flight, and adjust the dynamics to minimize the errors from a fixed position. Hovering is a fundamental maneuver of flight that other complex maneuver based on, so it can be challenging for several reasons, the changing dynamics from time to time of a hovering task are complex, particularly when there are sources of interference presented, the state of dynamics and trajectory spaces are continuously correlated to each other since there are a series of motion with respect to time, incorrect dynamics lead to an escalation of errors, which generate bad control trajectories that direct the consequence to a crash of the quadcopter.

In addition, throughout the experimental observations, we found that the proposed AM-based controller generates similar control outputs to the modern optimal control strategy, e.g., Linear Quadratic Regulator(LQR)[56], hence the learned dynamics of the proposed system can be evaluated by comparing with tuned PID control (assumed a simplified dynamics) in terms of precision and stabilization.

Stationary hovering

We first consider autonomous hovering in a stationary 3-D space, where stationary space is defined here as stable indoor environment without wind, Fig. 3.8 (a) shows the AR.Drone perform the test hovering fly in stationary environment, the error is primarily driven by the drift of AR.Drone, owing to its imprecisely tuned mechanics, such as propellers. Fig. 3.12

shows a 180 s testing results for different steps of training cycle over the x- and y- axis. This experiment provides results with 3000, 9800, and 11000 training cycle steps. The total training time is 796 s, resulting from the battery power constraints, which limited the flying time. For each stage of training cycle, the performance was studied for only 184s, with each data logging rate being 0.046 s; hence a total of 4000 data points per testing cycle are processed for evaluation. The results demonstrate the qualitative improvements in behaviour in the hovering precision (reduction of drifting error) over iteratively training of hovering manoeuvres. At the beginning of few training trials, the AR.Drone sometimes performs aggressive manoeuvres owing to the limited ability of the human pilot, leading the AR.Drone to pass the target position and cast away; however, during later iterative learning trials, the AR.Drone gradually learns how to retrieve from casting away from target point. After 11000 steps of training, the standard deviations of the measured x and y position are $\{\sigma_x, \sigma_y\} = \{0.107, 0.06\}$m.

Along z-axis, The AR.Drone is controlled to hover at approximately 0.64 m above the ground. Similarly, Fig. 6 shows the testing results last 184 s after 11000 steps of training, with a standard deviation of $\sigma_z = 0.026$ m. Additionally, it shows the given state estimation comparison between measured height from the on-board Altitude Sensor and PTAM estimation of z position.

Fig. 3.9 The positional transition of commanded AR.Drone for autonomous hovering that measured along x-, y- and z-axis with different training iterations in stationary condition. The x and y plot on a scale of [1, -1] *m* are on left-hand side, where the sign represents direction of displacement from the hovering target, z plot shows hovering 0.64 m above ground.

Table 3.1 Experimental Estimations on Precision and Stability of hovering

Experimental Condition		On-board Sensors		Proposed SOIAM		Classical PID	
		Precision	Stability	Precision	Stability	Precision	Stability
Stationary hovering	along x-axis	259.3	0.647	62.9	0.138	63.4	0.206
	along y-axis	249.8	0.541	50.7	0.114	54.5	0.196
Interruptive hovering	along x-axis	319.4	0.785	76.7	0.176	84.1	0.258
	along y-axis	298.4	0.622	54.2	0.115	72.4	0.266

Interruptive hovering

In this experiment, an interruptive environment is provided with artificial wind driven by an electrical fan, Fig. 3.8 (b) shows the AR.Drone performs hovering, flying in the interruptive environment. In this case, the error becomes more complex than for stationary hovering not only due to the mechanical problem but also the external wind excitation. For the experimental set-up, the fan is placed roughly 10 cm away from the right-hand-side of the AR.Drone, a single direction of continuous wind (approximated wind speed of 170 m/min) along the x-axis is assumed to perform the test and evaluation. On the basis of our observation, the pattern of the associative position input and its corresponding command output is stable across the stationary environment indicating that the previously trained system in stationary condition alone is unlikely to have good performance due to the absence of dynamics. Hence the sampling is not enough for performance robustness; the solution is to adapt the system to this new environment through on-line incremental learning, and the system is additionally trained to perform hovering in this interruptive environment. TABLE 3.1 concludes the quantitative results of the estimated precision and stability, on both stationary and interruptive environments. The evaluation is focused on the trajectories along the x- and y-axis respectively. Given a single direction wind along the x-axis of the AR.Drone, a continuous changing position causes the wind to attack at different angles, therefore mainly affecting the control on x- and y-axis. The test flying data are analyzed along x- and y-axis separately, recorded at 180-s intervals and logged with a sampling rate of 21.7 Hz.

Precision

Hovering precision is estimated with the integration value for the trajectory from the hovering point(zero point). The integration value statistically has a unit, Absement, which is mathematically measured in *ms* (metre second), as the smaller value estimates more precise trajectory control of hovering. In general, a similar pattern can be observed across the x-

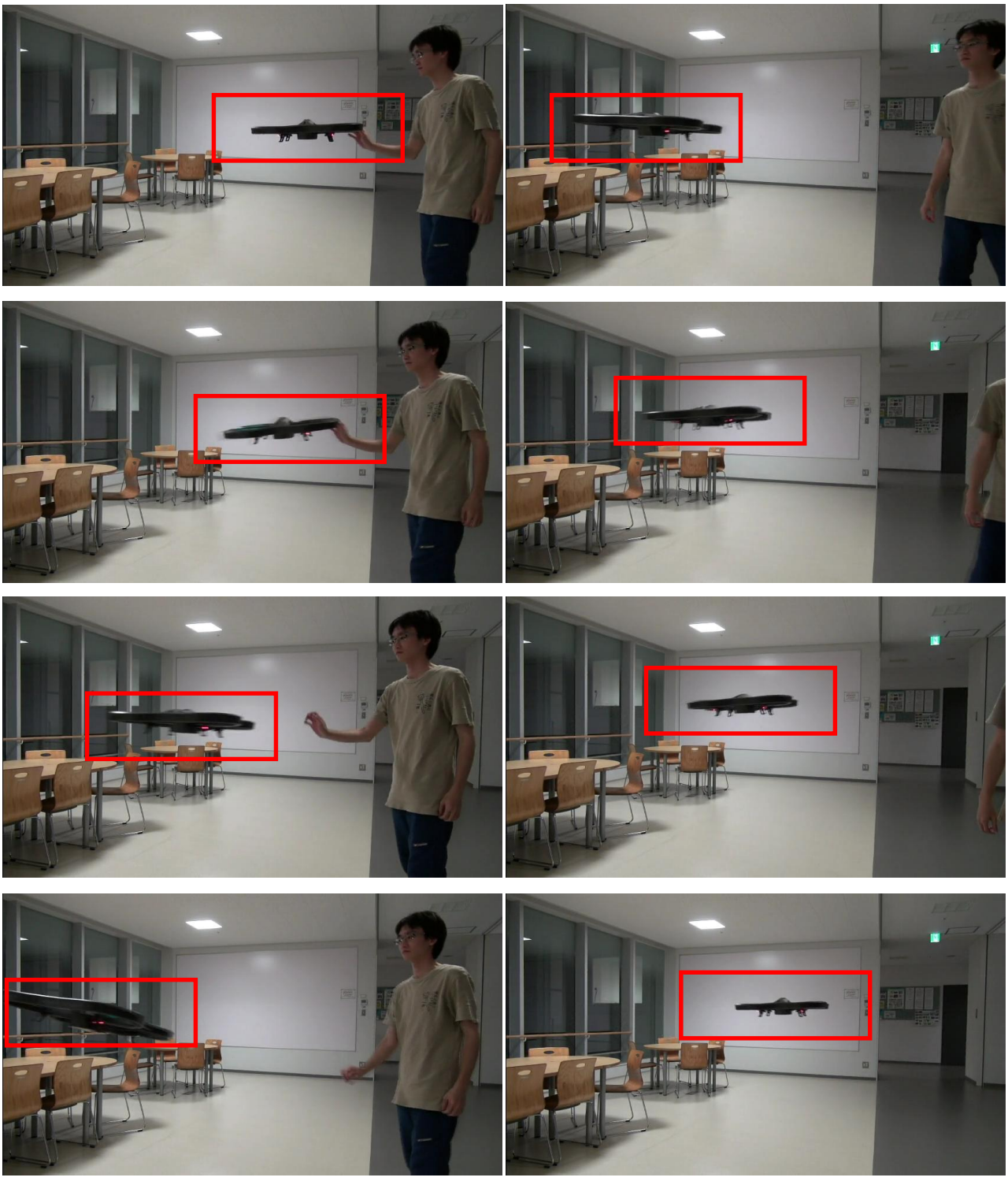


Fig. 3.10 Frame sequence for AR.Drone stationary hovering. (a) Left column shows the frame sequence of interrupting the AR.Drone by pushing it (b) Right column frames demonstrate the recovery from the interruption in order to hover at target position.

and y-axis, and as expected, the hovering trajectories for control methods in TABLE **??** are less precise when in an interruptive environment, because the integration values ascend; however, for both environments, the integration value of hovering control, which relies on the on-board sensors of AR.drone (without learning control), is greater than the integration value, which is controlled by the proposed SOIAM-based system (with learning control). Therefore, SOIAM-based system is shown to improve and advance the control precision of the system. Additionally, referencing the SOIAM-based system to a classical approach of PID control (Proportional-Integral-Derivative Control), which assumed the PID parameters are finely tuned for the hovering test during stationary hover, the precision of SOIAM-based system stays closely to the PID control, whereas in interruptive hover, the SOIAM-based system shows more precise performance by providing a smaller integration value than that provided by PID control. This indicates adaptation is required to cope with dynamical change in the hovering environment.

Stability

Hovering control maintains the AR.Drone near a targeted position; hence, the system must not only to flexibly adapt the error but also to keep the error as small as possible. The square-rooted variation of the averaged integration value over periods of time gives the stability estimation, which is attempting to identify the volatility of the hovering trajectories. The averaged integration value is obtained by dividing the entire trajectory into small sections of sub-trajectory. During this evaluation, the trajectory data is divided into 3-s sub-trajectory, with a total of 60 sub-trajectories. By working out the integration value of each sub-trajectory, the square-rooted variation can be estimated by calculating the standard deviation from the averaged integration value; hence, a stable flying trajectory features a small value of estimated result. Considering the results shown in TABLE I, SOIAM-based control provides comparatively lower stability value than the on-board sensor and PID control across both experimental environments. This validates a reasonable stability across the 180s flying test in both environments.

3.5.2 Trajectory following

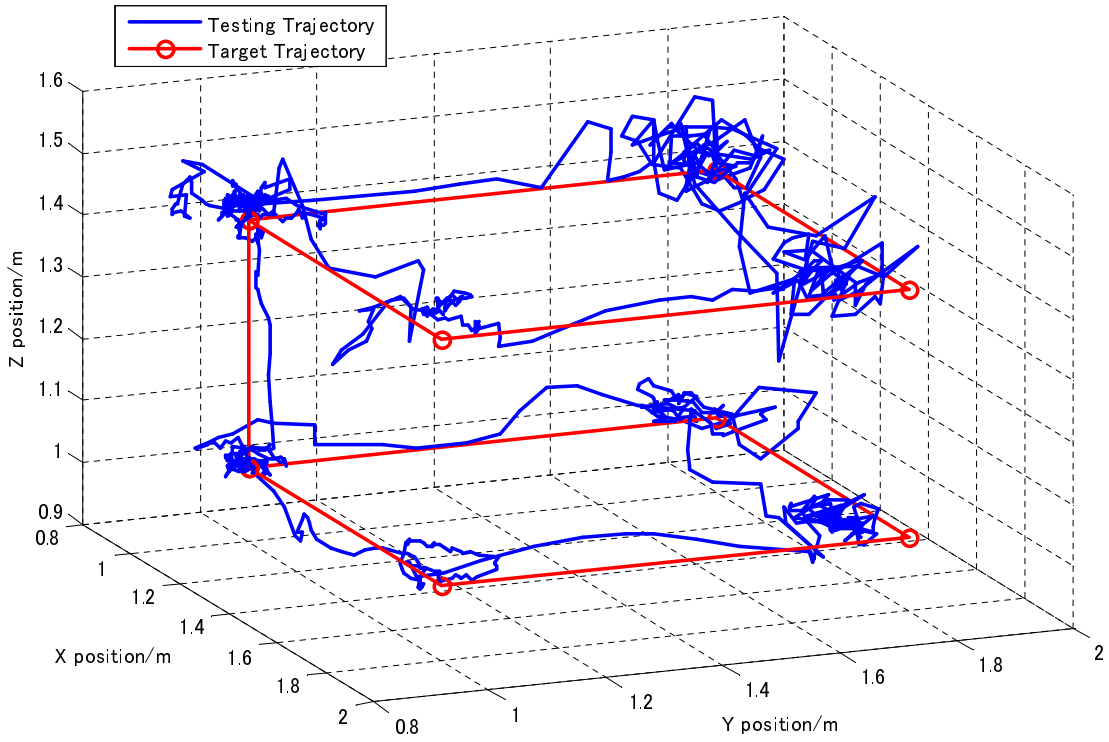


Fig. 3.11 Studied trajectories that consists of two 0.8 $\times$ 0.8-m squares in altitudes of 1.02 m and 1.42 m respectively: the trained SOIAM system is tested by giving four hovering way-points for each square.

We now consider a study of autonomous navigation, simplifying the problem by generating a rectangular trajectory in a 3-D space. This is performed using the previously trained autonomous hovering system to command the AR.Drone, and by giving a sequence of way-points representing the different hovering targets for AR.Drone to approach. Fig. 3.11 presents the qualitatively resulting trajectories of a double tier rectangular trajectory (in blue) using 8 way-points to fly the planned trajectory, the trajectory is reconstructed from the estimated position given by PTAM.

3.6 Related works

In the context of developing a flight control system, examples include the dynamic control approach on large-scale quadcoptor that is designed by Pounds et al.[64], and the model-based algorithm presented by Hoffmann et al.[37] for autonomous fly. However, the conventional model-based approaches can not satisfy our requirements for autonomous navigation throughout space, because most of these control system are task-specified, which commonly rely on discovered or pre-defined dynamic flying models of the robots and their interactive environments, these control approaches are impractical for autonomous navigation in complex indoor or outdoor environments, which are subject to changes. Additionally, The requirements for prior knowledge of interpretable and meaningful dynamic models cause the target robots to perform aggressive movements with no generalization, and significant issues of maneuvers replication are persistent in other robots.

In contrast to learning based methods, which often proceed in reverse, by relying on model identification or system synthesis, constructing system model from data-driven observations. This process requires data collection to learn effective control strategies. The learning based approach benefits from having no need for pre-defined complex models to develop a system. Research has shown good performance using learning based control approach in largely two categories, open- and close-looped control, for close-loop, e.g., Calise[15] and Chowdhary et al. [16] outline the framework for concurrent learning adaptive control, which guarantee improved learning and stability with selected and recorded data, David Nodland et al.[61] have developed neural network-based optimal control with output feedback, Efe[23] have introduced a neural network-based controller that learned Proportional, Integral and Derivative(PID) control. These previously proposed active control algorithms are mostly based on the strategy of closed-loop controller, which by optimizing the feedback of control output from robot, this results in the risk of oscillatory output response, also makes the system relatively more complex and costlier to construct than open-loop controller; Open-looped control system is not engaged in learning based method, however it does not encounter

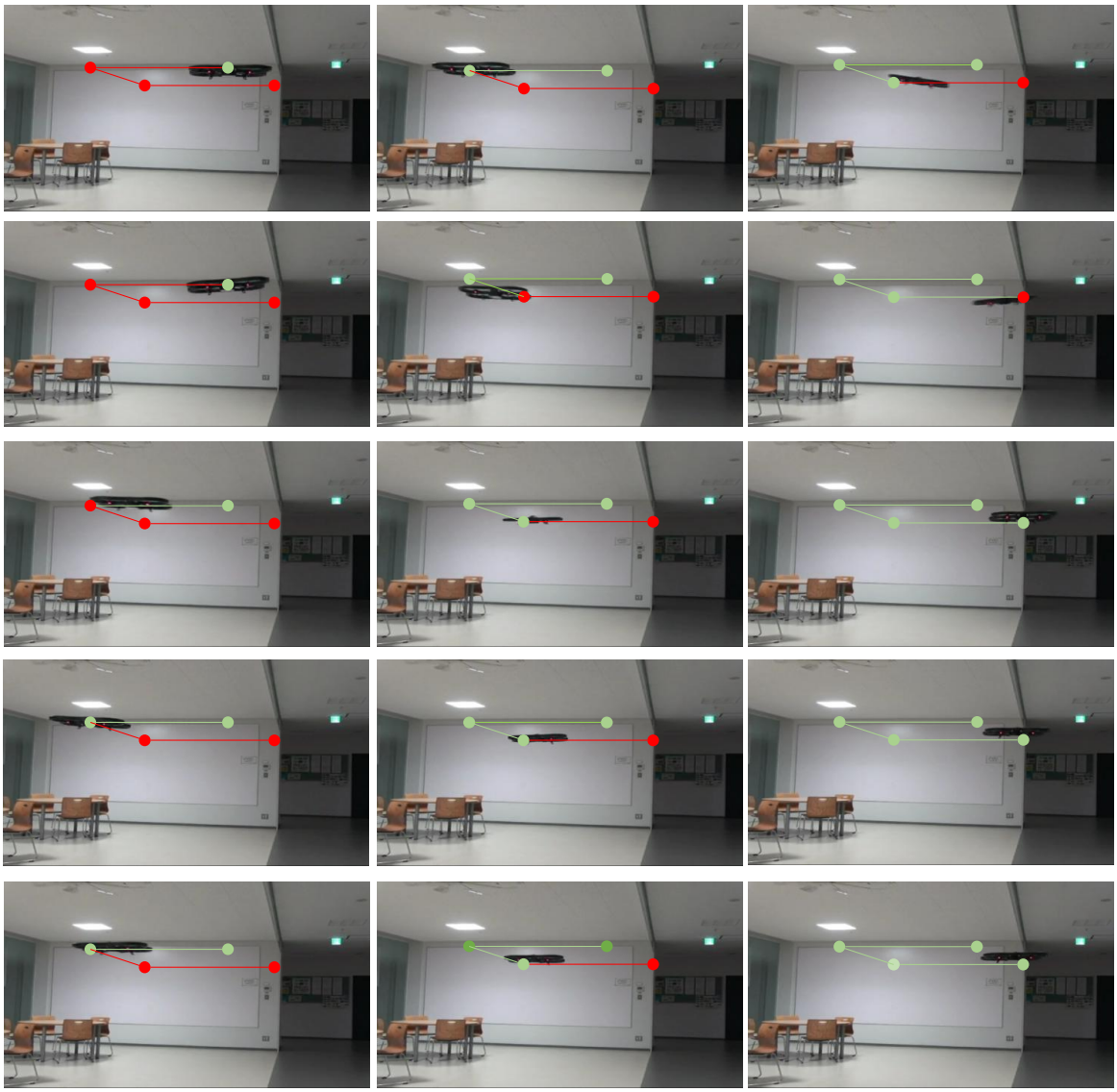


Fig. 3.12 Partial frame sequence demonstration for AR.Drone trajectory following using learned SOIAM. Trying complete 4 way-points on same altitude. Imposed lines represent the trajectory; green indicates the planned trajectory, and red is the traveled trajectory. (a) Left column shows AR.Drone follow trajectory in a transition along y-axis, (b) transition along x-axis, (c)and finally transition along y-axis in reverse direction to (a) to complete the trajectory.

the problem of oscillation, hence can be embedded to improve the performance of control system, e.g., Abbeel[1] have shown an example of applying open-looped control through mimicking an expert performance in the learning process for ensuring effective performance learning. The proposed AM-based system can be seen as dual controller architecture through the combination of both open- and close-loop approaches, as the system self-organizes and incrementally learns from the closed-loop control fashion for a period of time. Then, it controls the robot in an open-loop fashion. This maintains the stability and simplicity in the open-loop part, as well as the accuracy and adaptation to dynamic changes in close-loop part.

3.7 Discussion

In this study, we introduce an SOIAM-base control system to allow generalized and precise autonomous control of low-cost quadcoptor in unconstructed environments. Validating the system through real world experiments, we benchmark the robustness of the learning-based control system that minimized the drift error of AR.Drone, and detailed the performance of the system with respect to its incremental manner from noisy environments. Our model-free approach was demonstrated, in particular, for quadcoptor control estimation with given estimated posture states while attempting hovering tasks with minimal prerequisites, i.e., additional sensors, advanced knowledge of flight models. Although a visual system for self-localization was not fully considered in this dissertation, we must address this problem in order to have more precise and globally consistent control freedom of aerial vehicle. We are also interested in enabling the quadcoptor to learn more complex maneuvers.

However, we would like to point out few problems after conducting experiments with the SOIAM based method, these problems are prior to be addressed, in order the improve the algorithm:

1. **Intelligent exploration for data acquisition.** We demonstrated the SOIAM based control system is feasible for learning manual demonstration, and the performance is fully data dependent. The process of getting manual data is very time consuming and difficult, especially in a real world setting, also to achieve good performance, getting expert demonstrations are required. We consider to solve this by reinforcement learning. With recent great achievements of learning algorithm, agent can randomly explore the unconstructed environments by itself and collect data for improving it own policy.

2. **A more generalized representation.** This is a very common problem to learning based methods, because the collected data only reveals partial information about the

world, for example, data for specific task, i.e., hovering, therefore the learned system can only perform hovering out of box, the AR.Drone cannot flip or avoid obstacle. In following chapter of this dissertation, we consider to deal with these two problems, and develop learning methods to tackle more aerial robotic problems.

Chapter 4

Concurrent deep reinforcement learning (CRL) for continuous control

In the previous chapter we described how we can leverage manual demonstrations to learn a control policy through learning a neural network that performed as a black box, which alleviates the need for designing complex model-based system with unknown dynamics. The challenges when training a neural network often require large amount of data to be collected from expert demonstrations (a very time-consuming process) , and the process of hand-engineering an appropriate training data set (in general for supervised learning). In this chapter we describe completely different approaches to eliminate the need of expert demonstrations and handcrafting data, and to show how we can obtain a good parameterized policy model more efficiently, by leveraging the concurrent training methods with the reinforcement learning methods and policy optimization.

4.1 Introduction

Recent progress in autonomous control system has led to a growing interest in deep learning applications. This is because of its ability to learn useful features directly from supervised, unsupervised learning, and reinforcement learning, avoiding the need for cumbersome and time-consuming hand-engineered feature selection. The challenges that we are interested to address of applying deep learning and reinforcement learning to autonomous control directly is the efficiency of convergence and generalized policy models for continuous control domain, which interact with changes in environments. As a typical example, it is always valuable for a robot to adapt its speed parametrically depending on the wind conditions or location in the environment to improve performance, energy efficiency, and safety.

Reinforcement learning (RL) is considered an effective solution for autonomous control applications in recent years; it learns iteratively to perform, by trial and error, and search for the most appropriate actions for a given situation in an environment rather than requiring a fully developed environmental model to perform accordingly. However, training real world robots with RL remains an open problem, which can be decomposed into three broad categories: (1) Failsafe training, (2) convergence guarantee, and (3) faster learning for a generalized policy. This study addresses (3), in particular, impressive research has been conducted that combines RL algorithms with deep neural networks to successfully solve the control problems associated with playing Atari games[58][33]. However, most of the approaches are designed for the discrete action domain. Most real-world robotic control problems have continuous action spaces for accurate movement; therefore, the aforementioned approaches cannot be directly applied to the continuous action domain. There are few solutions to address this problem, but the most obvious one is to discretize the continuous action space. However the action dimensions will increase rapidly if precise controls are needed. This is known as the curse of dimensionality[54], which makes policy exploration inefficient. Another common approach is to use actor-critic (AC) algorithm. Significant approaches such as those of Kimura et al.[44] and Degris et al.[21] combined an AC algorithm with a policy gradient method to learn a robotic platform. Both studies provide an important footprint toward the continuous control of real robots.

In this article, we bring together a novel approach of the policy gradient optimization method proposed by John Schulman et al.[71], known as trust region policy optimization (TRPO), with an asynchronous RL-AC framework[57] to form a practical algorithm for our continuous control tasks. The critical benefit is the speeding-up of learning without the use of dedicate GPUs. Instead, a multi-core CPU on single machine will handle the algorithm and experiments. In this article, we first introduce the algorithm and frameworks that we worked on. Then, we examine the serial, synchronous, and asynchronous AC frameworks of TRPO by summarizing the performance of empirical studies in a MuJoCo physical simulator.

4.2 Problem description

The problems we are concerned with in this chapter are continuous control tasks that can be solved with the model-free deep reinforcement learning methods. We state the problems in the perspective of the practical implementation for a robotic control system rather than a general formulation of control framework. We consider two specific classes of state-action spaces and data sample efficiency, which we will refer to as continuous state-action spaces and convergence efficiency respectively.

1. Continuous state-action spaces. With recent great achievement of Deep Q Network (DQN) algorithm by Mnih et al.,[58], it is capable of human level performance on controlling game agents using unprocessed pixels for input. However DQN can only handle discrete and low-dimensional action spaces. Many tasks of our interests in robotic control tasks, have continuous (real valued) and high dimensional action spaces. There is needs for adapting the ideas underlying the success of Deep RL to the continuous action domain.

2. Convergence efficiency. Often the reason of Deep RL has being impractical to real physical systems, is due to its high sample complexity. Recent techniques have only provided partial solution to the issue of instability and sample complexity challenges, such by Hasselt et al.[34]. In order to make deep reinforcement learning practical enough to tackle real-world robotic problems, we must develop learning methods that are both data efficient and stable.

4.2.1 Extensions to our previous work

Previous work described in chapter 3 has shown the effectiveness of using manual or expert demonstrations (called associative memory based learning of SOINN, or SOIAM) in various ways for direct robotic control, without taking care of the continuous action domain of task, which described in problem 1. In fact, a large fraction of the literature considers the problem of modeling control system from data typically referred to as system identification. Interestingly, SOIAM algorithm can be seen as choosing a very specific set of paired memory, namely expert demonstrations of the task at hand. Our results show that, if our autonomous control policy acts on the same system and in the same conditions as the expert.

In this study, we not only tend to find solutions to the described problems, but also by means of finding the feasibility to improve the previous work, that is, the SOIAM algorithm, which has the same issues as other model-free learning methods, as the high sample complexity, and requires large amount of sample data for a good control policy, or leverage our previous work for the Deep RL, or both.

4.3 Preliminaries

In an ideal system, or any given state s, and action a, we consider a state transition probability from time t to $t+1$ as:

$$P(s_{t+1}, r_{t+1} \mid s_t, a_t) = P(s_{t+1}, r_{t+1} \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_0, a_0, r_0) \tag{4.1}$$

where the state s_t is considered as a Markov property, when the state preserves all the relevant information in the past. That is, the given reward from the environment at $t+1$ depends only on the state s and action a at time t. Also, given the present state s_t, such that s_t captures all information of past events, the future is independent of the past.

A reinforcement learning (RL) setting that satisfies the Markov property is recognized to be a Markov Decision Process (MDP), and for a finite set of state and action space, it is considered to be a finite Markov Decision Process (finite MDP). The following sections summarize the necessary background of RL to solve high-dimensional continuous control problem for the given finite MDPs.

4.3.1 Reinforcement learning (RL) paradigm

A RL scenario considers a discrete-time-step behaved model of a stochastic control system that interacts with an environment E, which has the dynamics of $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. Furthermore, we assume that the model incorporates a parametrized policy π with a probability distribution $a_t \sim \pi(a_t|s_t;\theta)$, which can be optimized by policy gradient methods with respect to the expected reward of the envirоonment$E$. An initial state s_t is observable at time step t, hence an action a_t can be executed by sampling the policy π with respect to s_t. In return, a new state s_{t+1} according to the dynamics and a scalar reward $r_t = r(s_t, a_t)$ are received. As the process continues to roll out towards a terminal state, it is considered a serial rollout (refer to Algorithm 1), and the results form a trajectory that consists of a historical sequence of states and actions. Also, the sum of discounted expected rewards $R_t = \mathbb{E}[\sum_{t=0}^{H} \gamma^t r_t)]$ for the trajectory, where H is the horizon, which is assumed to be finite for all policies in practice and $\gamma \in (0,1]$ denotes a discount factor. Hence, the general goal of the policy gradient is to maximize this discounted total expected reward by iteratively approximating $(\nabla_\theta \mathbb{E}[\sum_{t=0}^{H} \gamma^t r_t])$ and updating the gradient for the parametrized policy π. There are several expressions for the policy gradient estimate. In the literature, Schulman et al.[72] summarized the general form for a policy gradient as

$$\mathbb{E}\left[\sum_{t=0}^{H} \Psi_t \nabla_\theta log \pi_\theta(a_t|s_t)\right] \tag{4.2}$$

where Ψ_t denotes a variety of different approaches that are applicable to the policy gradient estimate, and $\nabla_\theta log \pi_\theta(a_t|s_t)$ denotes the policy derivatives.

Algorithm 1 Serial roll-out in environment E

Initialize iteration counter $i = 0$, step counter $t = 0$.
repeat
  Reset E and step counter $t \leftarrow 0$
  repeat
    Observe s_t from E
    Act with a_t according to policy $\sim \pi(a_t|s_t)$
    Return new state s_{t+1} and expected reward r_t
    Update step counter $t \leftarrow t+1$
    Store transition $(s_t; a_t; r_t; s_{t+1})$ as batch
  until state s_{t+1} is terminal
  $i \leftarrow i+t$
until $i == N$

4.3.2 Trust region policy optimization (TRPO)

TRPO is an optimization technique based on the policy gradient, and it shares some similarities with a natural policy gradient[41] and natural actor-critic[63], in which they all converge toward the same step direction when updating the parametrized policy. However, TRPO achieves the same goal with a different approach by approximately solving a Kullback-Leibler(KL) divergence constraint for each iteration of the policy update. Hence the TRPO is selected to solve our quadcopter tasks, it is not only able to solve high-dimensional continuous control problems, but also monotonically improve the optimized control policy with minimal amount of hyperparameter tuning. If we let the parameters of a policy that we are trying to update be θ_{old}, the TRPO update (Schulman et al.[71]) attempts to solve the KL divergence constraint as:

$$\begin{aligned} &minimize \left[Loss_{\theta_{old}}(\theta) = \tfrac{1}{H} \sum_{t=1}^{H} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \\ &\text{subject to } \overline{D}_{KL}^{\theta_{old}} = \frac{1}{H} \sum_{t=1}^{H} D_{KL}(\pi_{\theta_{old}}(\cdot|s_t) \parallel \pi_\theta(\cdot|s_t)) \end{aligned} \tag{4.3}$$

where $\hat{A}_t$ is the advantage estimator function that scales the policy gradient. It is defined as $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, in which the state-action value function $Q^\pi(s_t, a_t)$ estimates the expected return R_t for selecting an action a according to policy π in state s, and the value function $V^\pi(s_t)$ is a baseline that is the expected return R_t for following policy π from state s. Hence the advantage provides a measurement of how well the selected action performs compare to the default policy. Furthermore, the advantage estimate can be generalized as $\hat{A}_t := \sum_{n=0}^{H} (\gamma\lambda)^n \delta_{t+n}$, where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, and λ is used to adjust the amount

of bias and variance. Substituting $\Psi_t = \hat{A}_t$ into Eq.(4.2), the policy gradient is now expressed as:

$$\mathbb{E}\left[\sum_{t=0}^{H} \hat{A}_t \nabla_\theta log \pi_\theta(a_t|s_t)\right] \tag{4.4}$$

Because the value function leads Eq.(4.4) to a lowest variance estimate of the policy gradient, we use this expression throughout this study. Notably, Eq.(4.3) approximates the policy gradient in a similar way to Eq.(4.4), but introduces a trust region constrain to bind the KL divergence between the new policy with parameters(θ) and the old policy(θ_{old}).

4.3.3 Deep neural network (DNN)

Deep neural network by an abstract definition has more hidden layers than "normal" neural network, this difference provides more robust and sophisticated models can be achieved. the DNN parametrization minimizes the need for manual feature and policy engineering, and it also serves as a "black box" that allows mapping an end-to-end policy from high-dimensional inputs, such as images and fused sensory data, directly to discrete/continuous output, such as control actions. However, training a DNN with such expressive parametrization introduces a number of practical problem, which it tend to be sensitive to hyper-parameter settings, often requiring extensive hyper-parameter tuning to find good convergence property. Poor hyper-parameter settings tend to produce unstable or divergent learning.

4.3.4 Actor-critic (AC) method

The AC algorithm and many other iterative policy architectures such as policy gradient consist of two processes, policy evaluation and policy improvement. While the policy improvement is required to improve the policy on every step of update until convergence, and the advantage of introducing the policy evaluation is that it makes efficient usage of experienced data, hence more intelligently than traditional policy gradient methods. In Figure 4.1, the policy π is known as the actor for policy improvement, and the value function V is referred to as the critic for policy evaluation. This AC architecture is represented by Sutton et al.[77] that shows the policy is independent to the value function. The actor generates an action for a given state of the environment, and the critic produces a TD (Temporal-Difference) error signal for the given state and total expected reward, this signal from the critic drives updating in both the actor and the critic. If the critic is estimating the action-value function, such as Q-function $Q^\pi(s_t, a_t)$, it will also need the output of the actor. In Deep RL, AC combines the benefits of both approaches of value-iteration methods such as value function update

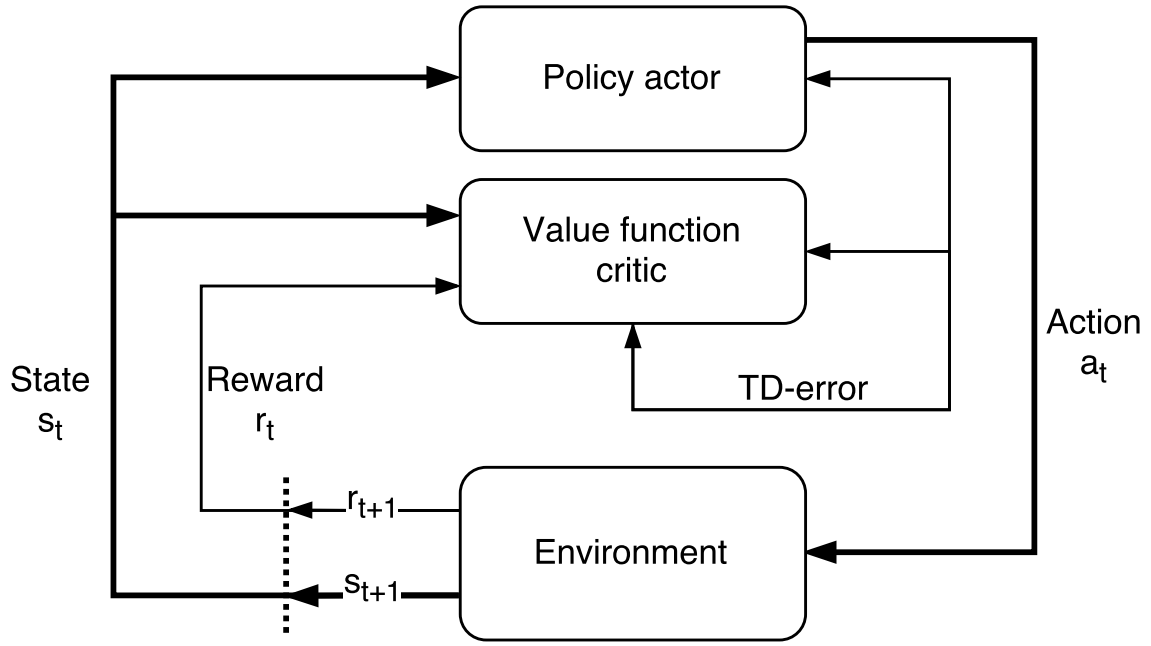


Fig. 4.1 The actor-critic architecture that proposed by Sutton et al. in 1998 [77].

by linear regression and policy actor update using stochastic policy gradients, and we can represent the actor and critic structures by DNN features independently.

Value function

Value function is used to estimate how valuable the state is when an agent enters into that state. In our study, we measure the value of the state with state-value functions. The policy is evaluated with the expected return R when starting in a state s, therefore the expected value of the value function is defined as:

$$V_\pi(s) = E_\pi(R_t|_{s_t=s}) = E_\pi(\sum_{k=0}^{H} \gamma^k r_{t+k+1}|_{s_t=s}) \tag{4.5}$$

where γ is the discount for the immediate reward.

In other AC literature the value is also measured with a action-value function for the policy, it is the expected return R when starting in state s, by taking action a, and following thereafter, the value function becomes:

$$Q_\pi(s) = E_\pi(R_t|_{s_t=s,a_t=a}) = E_\pi(\sum_{k=0}^{H} \gamma^k r_{t+k+1}|_{s_t=s,a_t=a}) \tag{4.6}$$

A fundamental property of value functions is the ability to satisfy a set of recursive consistency equations called the Bellman's equation:

$$\begin{aligned} V_\pi(s) &= E_\pi(R_t|_{s_t=s}) \\ &= E_\pi(r_{t+k+1} + \sum_{k=0}^{H} \gamma^k r_{t+k+2}|_{s_t=s}) \\ &= \sum_a \pi(s,a) \sum_{s'} P(s'|_{s,a})[R(s,a,s') + \gamma \sum_{k=0}^{H} \gamma^k r_{t+k+2}|_{s_{t+1}=s'}] \\ &= \sum_a \pi(s,a) \sum_{s'} P(s'|_{s,a})[R(s,a,s') + \gamma V_\pi(s')] \end{aligned} \tag{4.7}$$

The equation (4.7) is a simply dynamic programming equation that expresses a relationship between the value of a state s and the value of the valid state, which tells us that the current value is equal to the immediately reward plus the valid value after that. It allows us to sum over all the possibilities, weighting each by its probability of occurring. Moreover, we can see that the value of the first state must equal the reward for making a transition to together with the discounted value of s'. By cycling through the equation, we can fit a representation model of our value function for the states.

4.4 Concurrent training frameworks

We present the methodology in the perspective of the practical implementation of a robotic control system rather than a theoretical framework, because the theoretical foundation of existing techniques is well established and has been proven in numerous studies[66],[83],[60], [36],[39].

The asynchronous framework considers the parametric update scheme of HOGWILD![66], which allows a shared memory to be accessed by concurrent processes without any locking mechanism. With standard RL settings, multiple RL agents are run in parallel, and each agent contributes different parts of the observation within the environment through its own exploratory policy, thereby providing a maximal overall diversity to the exploration phase. In practice, a shared memory holds the policy parameter vectors across all concurrent agents that are multi-processed on a single machine. Algorithm 2 summarizes the asynchronous update.

Although the Algorithm 2 scheme can also be run in a distributed format across multiple machines, we would proceed to a multi-processed approach on a single machine for compu-

tational efficiency, which would eliminate the problem of communication among machines. Mnih et al.[57] demonstrated that this asynchronous approach helps to train neural networks with gradient decent methods more reliably and yields a speed improvement of the training time in discrete domain.

Algorithm 2 Generalized asynchronous update framework for single RL agent in environment E

// Initialize shared parameters θ and counter $T = 0$.
Initialize threaded parameters θ'.
Initialize threaded counter $t \leftarrow 1$.
repeat
  Reset gradient $d\theta' \leftarrow 0$
  Synchronize threaded parameter $\theta' = \theta$
  $t_{start} = t$
  Get state x_t
  repeat
    Roll-out according to policy $\pi(a_t|x_t;\theta')$
    $t \leftarrow t+1$
    $T \leftarrow T+1$(asynchronously)
    if terminal s_t **then**
      Reset E
      Get state x_t
    end if
  until $t - t_{start} == t_max$
  for batch $\in$ trajectory batches from roll-out **do**
    Accumulate gradients $d\theta$ w.r.t $d\theta'$
  end for
  Asynchronously update θ using $d\theta$
until $T > T_{max}$

4.4.1 Serial update framework

The serial method is the simplest form of training a controller: it is considered as a serial rollout of trajectories (Algorithm 1) when a single system interacts with a single environment. Referring to Eq.(4.3) and Eq.(4.4), we consider training the controller with an AC architecture in which the policy π is the actor and the baseline estimate of the value function $V^{\pi}(s_t)$ is the critic. Practically, we define two separate sets of parameters for the policy and value functions. Algorithm 3 provides details of training the system with the serial method.

Algorithm 3 Training system model with AC architecture in environment E

Initialize policy parameters θ_π.
Initialize value function parameters θ_v.
Initialize step counter $t \leftarrow 1$.
loop
  Serial roll-out with current policy, Algorithm 1
  Compute generalized advantage $\hat{A}_t$
  Update policy $\theta_{\pi_{new}}$ with TRPO, Eq.(4.3)
  Update value function $\theta_{v_{new}}$, Eq.(4.7)
end loop

4.4.2 Synchronous update framework

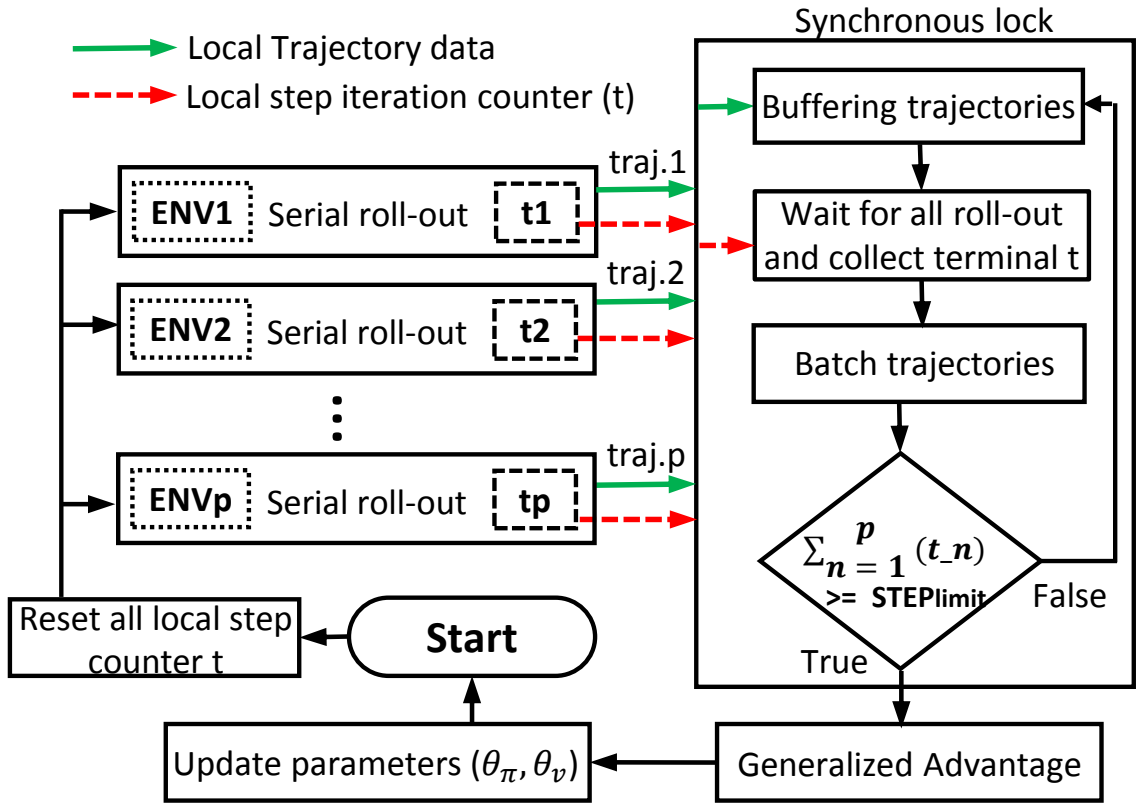


Fig. 4.2 System flowchart of the synchronous method, each serial roll-out includes an environment(ENV) and a local step counter(t). $STEP_{limit}$ is a constant that decides the amount of steps in the batch. Local step counter (t) is reset after each parameter update.

The synchronous method runs multiple serial rollouts in parallel. The implementation of this method involves a locking mechanism for all the concurrent rollouts to be synchronized, which ensures the batched trajectories follow the same policy rule, thereby safely updating the

system in a synchronous manner. Fig.4.2 presents the system flowchart of the synchronous method.

With this method, the procedure for gathering data and updating parameters is identical to the serial method. If we reduce the number of concurrencies of the rollout to one, they will have same performance on evaluation. It should be noted that in practice, the parallel rollout is carried out using a multi-process approach; therefore, each rollout individually holds the same copy of the policy rule to generate actions. In order to provide diversity to the trajectories, processes are initialized individually with different random seed numbers.

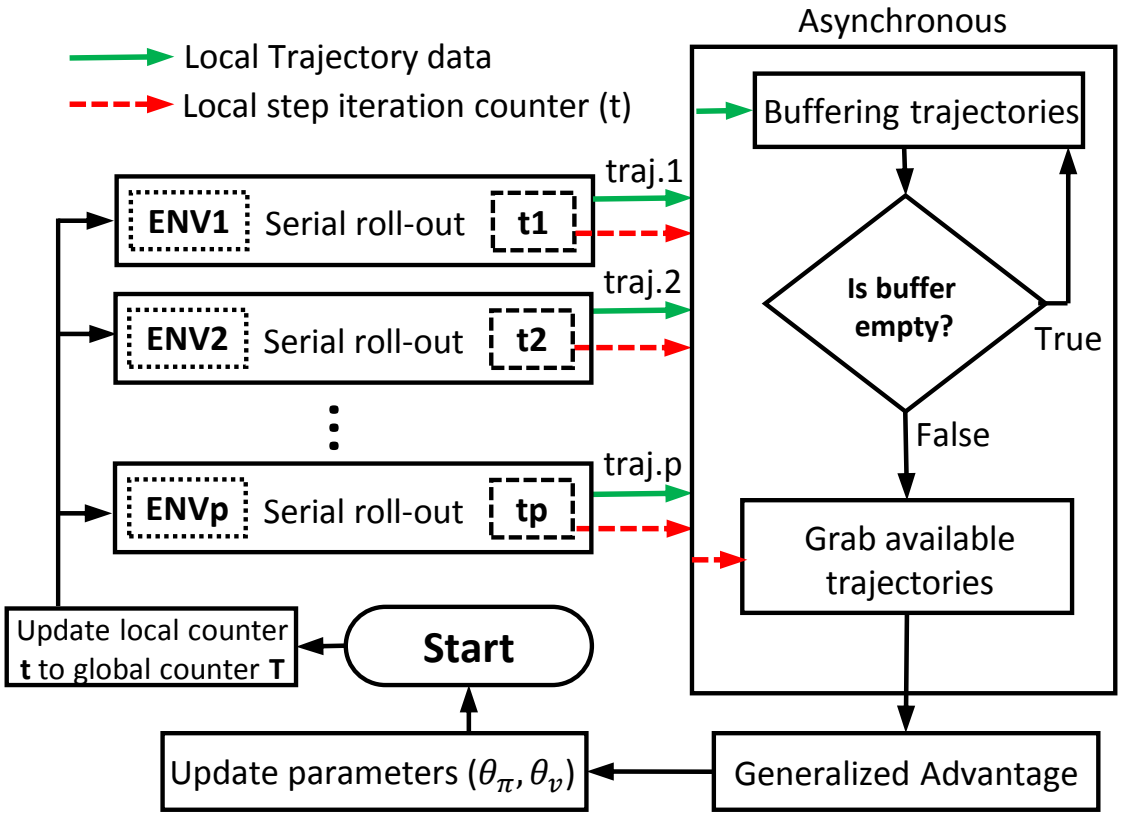


Fig. 4.3 System flowchart of the asynchronous method, each serial roll-out includes an environment(ENV) and a local step counter(t).

4.4.3 Asynchronous update framework

Here, we present the practical asynchronous TRPO with generalized advantage estimation (ATRPO) by following the protocol defined in Algorithm 2 and incorporate it with the iterative optimization in the serial method (Algorithm 3), Fig.4.3 presents the data flow of the ATRPO. Compared to the synchronous method, the asynchronous method is more advanced in near-

constant usage of CPUs without waiting for other processes to complete, which prevents wasting computing cycles. It also leads to more frequent parameter updates, and therefore speeds up the convergence of learning. We present the asynchronous method in Algorithm 4. Unlike the Gorila framework proposed by Nair et al.[60], we run Algorithm 4 concurrently in a single machine with only one CPU by creating multi-processes to achieve parallelism. Moreover, instead of accumulating the gradients iteratively as in the asynchronous advantage actor-critic (A3C) method proposed by Mnih et al.[57], we compute the gradients using a batch method for all time steps of the trajectory, which we found to be more faster and stable.

Algorithm 4 Asynchronous TRPO framework for single RL agent in environment E

// Initialize shared policy parameters θ
// Initialize shared value function parameters ϕ
// Global counter $T = 0$.
Initialize threaded policy parameters θ'.
Initialize threaded value function parameters ϕ'.
repeat
  Reset gradient $d\theta' \leftarrow 0$ and $d\phi' \leftarrow 0$
  Synchronize threaded parameter $\theta' = \theta$ and $\phi' = \phi$
  Serial roll-out with policy θ', Algorithm 3
  $T \leftarrow T + 1$
  Compute generalized advantage $\hat{A}_t$
  Compute gradients $d\theta$ w.r.t $d\theta'$, Eq.(4.3)
  Compute gradients $d\phi$ w.r.t $d\phi'$, Eq.(4.7)
  Asynchronous update θ using $d\theta$, ϕ using $d\phi$
until $T > T_{max}$

4.5 Implementation

In code based implementation, we develop all the deep learning algorithm in Python with Theano library, the proposed DNN models is constructed with Keras, which is a high-level neural networks API that built upon Theano and Tensorflow.

4.5.1 The DNN models

Both the policy and value functions were parametrized with a neural network that used one fully-connected (dense) hidden layer with 128 tanh units to map the input state vector to a hidden state, followed by a single recursive layer of 128 LSTM cells. The output layer of the policy network consisted of two outputs: one was the mean vector modeled by a

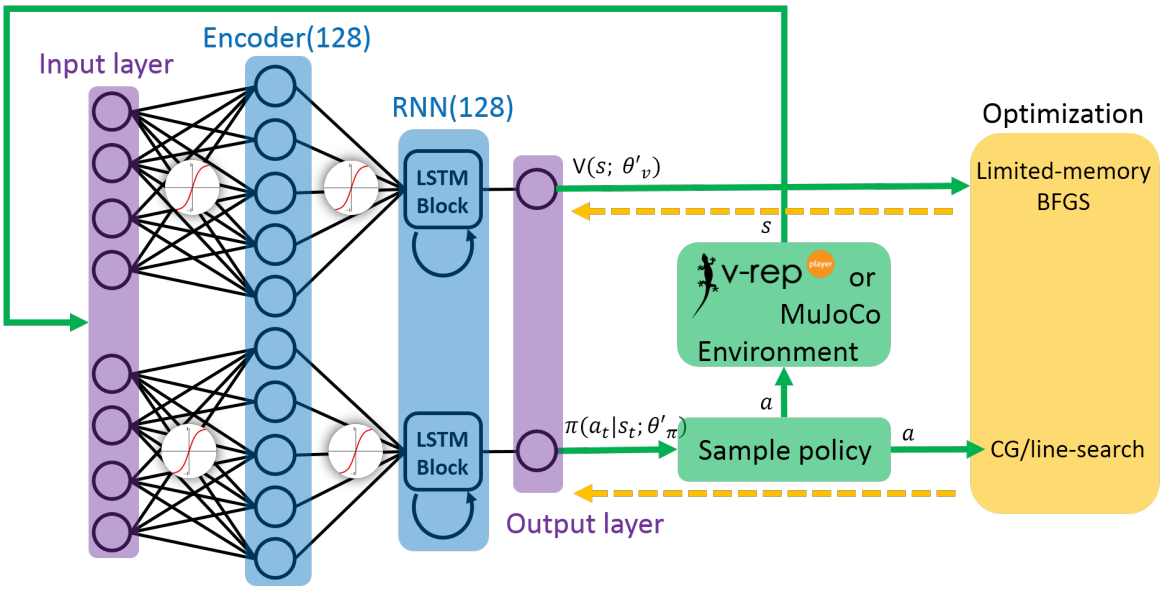


Fig. 4.4 A tuned deep neural network model that detail is described in section 4.5 for training our Mujoco simulation experiment.

linear layer; the other was the log variance modeled by a separate linear layer using SoftPlus ($\log(1+\exp(x))$) activation. Furthermore, in feedforward mode, the policy network defined by the normal distribution took the input state s toward the output layer, where it sampled a pair of mean μ and log variance $log(\sigma)$ of the Gaussian distribution with a spherical covariance, and each output had the same dimension as the action space of the task. The value function followed the same structure as the policy network, but with a single linear output layer. In this study, the policy and value network did not share any parameters. We used the identical neural network architecture for all of the tasks that performed in this study.

4.5.2 Update of value function

Following the expression of value function that derived in Eq.(4.7) of section 4.3.4. We simply consider the estimation of the value function as solving a nonlinear regression problem by minimizing the squared sum:

$$minimize \sum_{t=1}^{H} \|V_{\theta_v}(s_t) - V_t\|^2 \tag{4.8}$$

where V_t is the target value of the sum of discounted rewards: $\sum_{l=0}^{H} \gamma^l r_{t+l}$, and $V_{\theta_v}(s_t)$ is the predicted value of the state s_t. We batch each episode of trajectories for a single update; thus, t indexes all of the time steps of a batch. This is known as the Monte Carlo approach

for estimating the value function. To update the value function, we use the method of limited-memory Broyden Fletcher Goldfarb Shanno algorithm (L-BFGS), which has been demonstrated as much more stable to train[50] and requires less tuning for parameters such as the learning rate and convergence properties.

4.6 Experiment results of empirical studies

We conducted the experiments using the MuJoCo[79] simulator, for different tasks and purposes, where the MuJoCo is more empirical and precise. However, not only does it work as a benchmarking tools for the proposed approaches[26], but the chosen tasks also shared a similar design basis and characteristics with the quadcopter tasks, such as the design of the cost function. The empirical tasks were simulated by the MuJoCo physical engine, and were ported to interface with the OpenAI gym open-source library[14].

4.6.1 Mujoco simulation for continuous control tasks

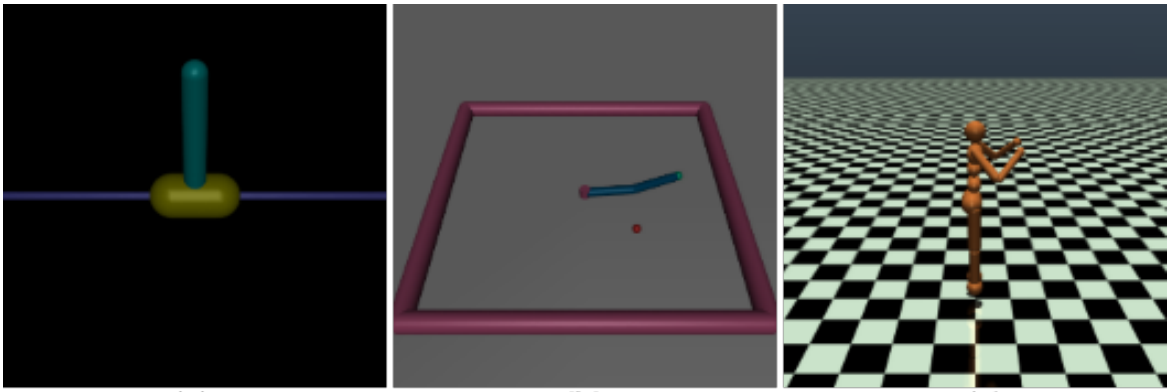


Fig. 4.5 The simulated MuJoCo environments and models for the presented continuous control tasks in this chapter:(a) 1-D inverted pendulum, (b) 2-D reacher, and (c) humanoid, respectively

Inverted Pendulum

The 1-D inverted-pendulum balancing task in Fig.4.5(a) followed the physical settings that are described by Barto et al.[8]. It has one continuous action space and four continuous state dimensions. As for the single parametric update, we batch at least five trajectories, which in

Table 4.1 The Mujoco simulation of continuous control tasks

Environments	Inverted Pendulum	Reacher	Humanoid
State dimensions	4	11	375
Action dimensions	1	2	17
Max step limit	3000	3000	5000
Simulation engine	MuJoCo Euler	MuJoCo Euler	MuJoCo Euler
Purpose	Balance an inverted pole on a cart	Move 3-DOF arm from random start positions to target positions	3-D bipedal robot walk forward as fast as possible.

total contained a minimum length of 5000 time steps per batch. We used a binary reward function:

$$r(t) = \begin{cases} 1 & \text{if not terminal state} \\ 0 & \text{otherwise} \end{cases}$$

Reacher

The robotic arm reaching task in Fig.4.5(b) was trained to reach random targets within a 2-D space; it had two action spaces and eight state dimensions. This task has no specific condition for termination; The robotic arm continued to act, and learned to obtain largest reward by approaching the target as closely as possible. We used a minimum length of 15,000 time steps per batch. We used a reward function: $(D_{body-target}) + \sum \|a\|^2$, where $D_{body-target}$ is the distance to a reachable target, and a is the continuous actions taken.

Humanoid

The humanoid walking task in Fig.4.5(c) has 10 action spaces and 33 state dimensions. Each episode is initialized with a state that consists of a uniform distribution centered on a reference model configuration. We used 50,000 time steps per batch. There were two conditions for an episode to be considered as terminal: when (1) The central mass of the model was outside the range of a predefined height of 1.0 and 2.0m; (2) The terminal state was not reached in 2000 time steps.

4.6.2 Experiment setup

The experimental results are compared in terms of averaged episodic rewards against the wall-clock time, which it is a more appropriate comparison across the different optimization

algorithms that considered in this study. It is also a crucial measure of the computational cost for a real-world application. We used the same machine to conduct all of the tasks. It ran on a 64-bit Linux platform with a single 3.2 GHz quad-core CPU (i.e., four physical cores) and a total of 16GB of RAM. This machine was dedicated to this study, to ensured that only a single task was running during each experiment session. Also full CPU usage was observed during all sessions (all eight CPU processes were utilized at 100 % capacity).

4.6.3 Simulation results

Figure 4.6 shows a comparison of the best testing results for the methods we considered: The *synchronous* and *asynchronous* frameworks described in Sections 3.2 and 3.3, with the serial method being the baseline of comparison, and all methods found a solution for solving problems in the MuJoCo domains. For the serial and synchronous methods, we saved and tested the trained model in each task for every five updates and we selected and averaged the rewarded scores of five best runs during the testing phase for each saved model. With the increased amount of updates in the asynchronous method, we evaluated the model for every 50 updates.

Overall, of the results shown in Fig. 4.6, the asynchronous method with two processes yields the best convergence over the other two methods among all tasks in terms of computational time; it also has the potential of achieving higher rewards. The synchronous method of using two and four processes learn slightly slower than the asynchronous one. Following the increased scales of complexity and dimensionality of the tasks, the asynchronous and synchronous methods performed significantly well, e.g., in the humanoid of Fig. 4.6(c), the speed is improved by a factor of five for reaching an average score of 5000. We also noticed that, although leaning is largely sped up by moving from a single process to multiple processes, we could not find much performance improvement when moving the processes from two to four. This was due to a bottleneck in the inter-process communication overhead when sharing memory across different processes as well as the requirement for optimal load balancing over the four physical cores of the experimental machine. Although this is also an interesting topic for further improvement of the results, it is far beyond the scope of this study.

It should be noted from Figure 4.6 that the best results use a smaller batch size in the asynchronous method. We swept through varieties of batch size in steps of 1000 for each task and summarized the more significant ones in Figure 4.7. The results suggested that the asynchronous method performs better with a smaller batch size, which is a crucial advantage in real robot learning, as the acquisition of training data is more expensive. In contrast with the synchronous method, a decrease in the batch size of the asynchronous method

demonstrated an efficient usage of the computational cycle. This fulfilled the idle cycle with an increased model update frequency. However, when there are no changes in update frequency for the synchronous method, idle cycles are wasted during the synchronization of other processes. In any case, batch size must be properly selected according to the tasks: Too small a batch size leads to unstable and noisy learning, and causes poor evaluation performance.

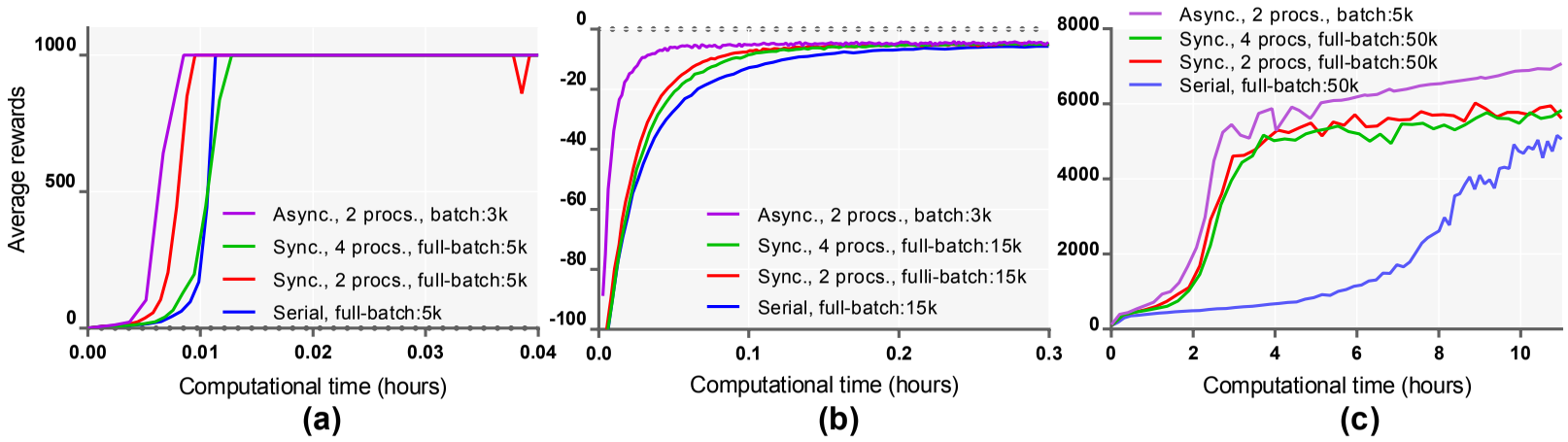


Fig. 4.6 The best testing results of trained models for the simulated MuJoCo environments: (a) Inverted pendulum balancing; (b) Reacher; (c) Humanoid, using serial, synchronous(labeled as Sync.) and asynchronous (labeled as Async.) approaches. All rewards were averaged over the five best runs for each experiment.

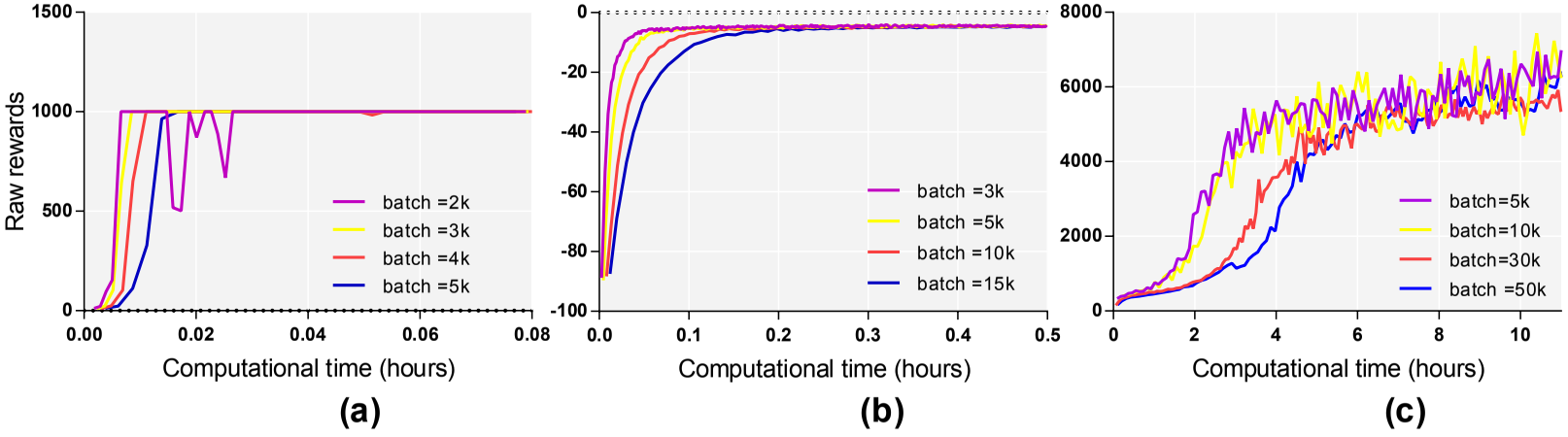


Fig. 4.7 Learning curves for the asynchronous method with different sizes of training batches in simulated MuJoCo models: (a) Inverted pendulum balancing, (b) Reacher, (c) Humanoid. All rewards were presented in raw scores of every 5 episodes.

4.7 Discussion

The work presented in this chapter draws large influence from Google DeepMind's work [58] on training deep neural network to play classic Atari-2600 games through the use of Q reinforcement learning, and works of John Schulman et al.[71] on TRPO and GAE for improving learning properties of reinforcement learning. We use deep neural network like DeepMind, but the underlying task we are attempting to achieve is vastly different. We proposed methods for learning a continuous control policy model that captures the system's dynamics on larger scales of instance than a single instance. In our empirical studies, this method outperformed the conventional single instance of the TRPO + GAE, the performance is measured as a function of the training time. As the policy for batching new data can affect the learned control model, it could be interesting to characterize this dependency, and explore the possible benefits of an diversely explored strategy, during which the current policy is in charged for batching new data.

Chapter 5

Learning maneuvers for MAV using the CRL framework

This chapter presents an aerial robotic application of deep reinforcement learning that imparts an concurrent learning framework and trust region policy optimization (details described in chapter 4) to a simulated quad-rotor helicopter (quadcopter) environment. In particular, we optimized a control policy asynchronously through interaction with concurrent instances of the environment. The control system was benchmarked in previous chapter and here we extended with examples to tackle continuous state-action tasks for the quadcoptor maneuvers: hovering control and balancing an inverted pole. Performing these maneuvers required continuous actions for sensitive control of small acceleration changes of the quadcoptor, thereby maximizing the scalar reward of the defined tasks. The simulation results demonstrated an enhancement of the learning speed and feasibility for learning tasks.

5.1 Introduction

Recent progress in autonomous control system has led to a growing interest in quad-rotor helicopter (quadcopter) applications. This is because of their ability to explore real-world space through rapid maneuvers. Companies have recently intensified their interest in offering a small-scale and low-cost development platform for such a quadcopter system, which has provided an ideal test bench not only for research purposes on sophisticated control technologies but also for performance art applications[70]. However, despite the fact that the model of a quadcopter itself has been well studied in the past[11][38], the main challenge of autonomous control that we are focusing on is the policy models, which interact with changes in environment. As a typical example, it is always valuable for a quadcopter to adapt

its speed parametrically depending on the wind conditions or location in the environment to improve performance, energy efficiency, and safety.

Following up on our work described in Chapter 3 one can learn an neural network so as to optimize manual human demonstrated commands that directly associated from a given state — i.e., predictive control output of an unknown dynamic model over long time scales. However, the AM based SOINN algorithm described in Chapter 3 can be heavily data dependent and expensive in data acquisition when applied in a real-world continuous state-space setting. In this chapter, we apply our proposed learning methods from chapter 4 to a simulated quadcopter system for autonomous maneuvers: hovering and balancing an inverted pole. The system uses a deep actor-critic neural network (DACNN) that play the role of determining the quadcopter's control signal for moving for-/back-ward, and rotation. The computed control signal is then sample with Gaussian policy for continuous control to allow the quadcopter to complete tasks. In addition, vision based self-localization for environmental awareness enable the quadcopter to generate local map for surrounding obstacles, as well as to localize itself for the purpose of navigation. All subsystems described in this chapter run simultaneously in real time simulation using the Virtual Experimentation Platform (V-REP) with ODE simulation engine to rendering the quadcopter and its dynamic models. The simulation results demonstrate that our concurrent trust region policy optimization (TRPO) architecture (described in chapter 4) is practical for learning efficiency and effectively finding (near-) optimal control policy of the quadcopter for specified tasks.

5.2 Preliminaries

The simulated quadcopter state Γ comprises its position (x, y, z), orientation angle (roll ϕ, pitch θ, yaw ψ), velocity ($\dot{x}$, $\dot{y}$, $\dot{z}$) and angular velocity ($\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$). The quadcopter is controlled via a 4-dimensional action space of acceleration $[u_1, u_2, u_3, u_4] = [\ddot{x_\theta}, \ddot{y_\phi}, \ddot{z_{alt}}, R_\psi]$:

1. u_1 and u_2 : The longitudinal (front-back) and latitudinal (left-right) cyclic pitch controls cause the quadcopter to pitch forward/backward or roll sideways, and can thereby also affect acceleration in the longitudinal and latitudinal directions.

2. u_3. The four main rotors collective control affects the altitude of the main quadcopter, by rotating all the propellers at the same speed can theoretically hover the quadcopter at certain altitude. As all the propellers sweep through the air with higher rotation, that is increasing value of u_3, the resulting amount of upward thrust (generally) increases with this command; thus this control affects the main quadcopter's thrust.

3. u_4. The collective pitch control u_4 affects diagnostic propeller thrust, and can be used to yaw (turn) the helicopter.

We used above described notations to define our problem statements (section 5.2.2) of controlling the simulated quadcopter.

5.2.1 Reward function

In this section, we identify the importance of the reward function regards to the reinforcement learning (RL). In previous chapter of 3, 4, we did not specifically study and design reward function for the individual task, since the tasks in empirical study of chapter 4 are well-studied tasks, the reward functions are predefined respectively for each of task. In Chapter 3, there was no requirement of defining reward function for the AM-based SOINN method, SOIAM, the reward function is provided by the expert demonstration that extracts a definition of the task in the form of a reward, which is guaranteed to result in an (near-)optimally good policy that have the same skill as the expert.

In the perspectives of the RL and optimal control, the ultimate aim is to find a good control policy, i.e., a prescription that tells us a satisfied action to take for every possible state of the system, as the method with expert demonstration in Chapter 3. To generate this kind of look-up policy is rather difficult and cumbersome for RL, hence for most control problems, It is more useful to specify a reward function than to directly specify the optimal policy. However, practically defining reward function is not simple either, even for experienced control engineers and field experts, because it is hard to formulated the task in a form of reward function, consider this, rather than ask expert to describe exactly what is a good aerobatic quadcopter flip entails, it is much easier for expert quadcopter pilot to just fly the quadcopter and demonstrate a few aerobatic flips. There are many practical control tasks, for instance, what would be the correct objective functions for having a quadcopter fly an acrobatic maneuver or navigate in the obstacle-rich environments; or for having a ground vehicles navigate in highway traffic? It is very challenging to formally specify the reward function individually, this means that there are no general rules of articulating the reward functions for many control problems, it is often manually tweaked until the desired behavior is obtained. Therefore the difficulty of manually specifying a reward function represents a significant barrier to the broader applicability of RL and optimal control algorithms.

The key to find the reward function is making assumption that the set of features is sufficiently rich, and it linearly trades off between a known set of features, that features are the state-space that corresponds to the task to be executed. This assumption is fairly unrestrictive. Features can be discrete valued, for example, whether the quadcopter is

airborne, whether the quadcopter is in a collision, etc. Features can also be continuous valued, for example, we could have features in quadratic forms that corresponding to the squared distance to the target position or orientation. In the implementation section 5.3 of this dissertation, we described more details on the reward functions for hovering and pole balancing tasks.

So far we have mentioned that upon advancing to find good reward function, as it can largely affect the RL learning results, as the RL agent acts toward the next time step, it receives a reward from the reward function. Informally, the RL's goal is to maximise the cumulative these rewards in the long run of tasks. In general, the RL agent attempts to maximise its expected return, which in the simplest case is the sum of rewards:

$$R_t = r_{t+1} + r_{t+2} + ... + r_H \tag{5.1}$$

where H is the final time-step through a finite horizon, R is the respected reward, and r is the immediate reward that obtained from reward function. Here the final-time step is a natural break point at the end of a sequence which puts the RL agent in a terminal state (the end of single episode). Following this, the agent is forced to re-initialize to standard starting state or a state sampled from a standard distribution of starting states. Hence as a ripple effect, the reward function will have large influence on the convergence properties and learning efficiency of RL algorithms.

5.2.2 Problem statement

This article is concerned with learning efficiency of continuous control problems of quadcopter: Hovering and inverted pole balancing (IPB), using gradient-based policy optimization with RL techniques. We initially define the problematic quadcopter tasks that are considered in this study. Assuming the quadcopter's center of mass has a position and orientation of $\Gamma = [x_\theta, y_\phi, z_{alt}, R_\phi, R_\theta, R_\psi]$, with respect to the world-frame (W), this scheme controlled the translational acceleration: $[\ddot{x}_\theta, \ddot{y}_\phi, \ddot{z}_{alt}]$ and rotational acceleration: $\ddot{R}_\psi$ of the quadcopter (see Fig.5.1). To simplify the tasks, we kept the initial altitude z_{alt} and heading R_ψ constant at all times, and the controls satisfied $\ddot{z}_{alt} = \ddot{R}_\psi = 0$. Thus, the valid control commands for quadcopter to balance a pole and hover is $[\ddot{x}_\theta, \ddot{y}_\phi]$.

Problem 5.2.1 (Hovering task) *Given positional initial and target states of a quadcotper, Γ_{init}, Γ_{targ}, find an optimal control command $[\ddot{x}_\theta, \ddot{y}_\phi]$ for a holonomic quadcopter to stay as close to the target state as possible. The task is completed when the quadcopter comes to*

rest at the target state. This is when $\Gamma = \Gamma_{targ}$ *and* $\dot{\Gamma} = 0$*. In practice, we consider the task is done when* $|\Gamma - \Gamma_{targ}|$ *and* $|\dot{\Gamma}|$ *is sufficiently small.*

Problem 5.2.2 (Inverted pole balancing task) *Considering the pole as a point mass that was rigidly attached to the center of mass of the quadcopter. Given an initial position and speed of the pole center of mass,* $P_0 = [p_x, p_y]$*, and pole displacement-velocity constraint* $[\varepsilon_P, \varepsilon_{\dot{p}}]$*, find optimal control commands* $[\ddot{x}_\theta, \ddot{y}_\phi]$ *for a holonomic quadcopter to keep the pole on top of the quadcopter. The task is completed when the pole comes to rest, and the quadcopter reaches a hovering state, as* $P = \dot{P} = \dot{\Gamma} = 0$*. In practical aspect, the task is satisfied when* $|P| < \varepsilon_P$*,* $|\dot{P}| < \varepsilon_{\dot{P}}$*, and* $|\dot{\Gamma}|$ *is sufficiently small.*

5.3 Implementation

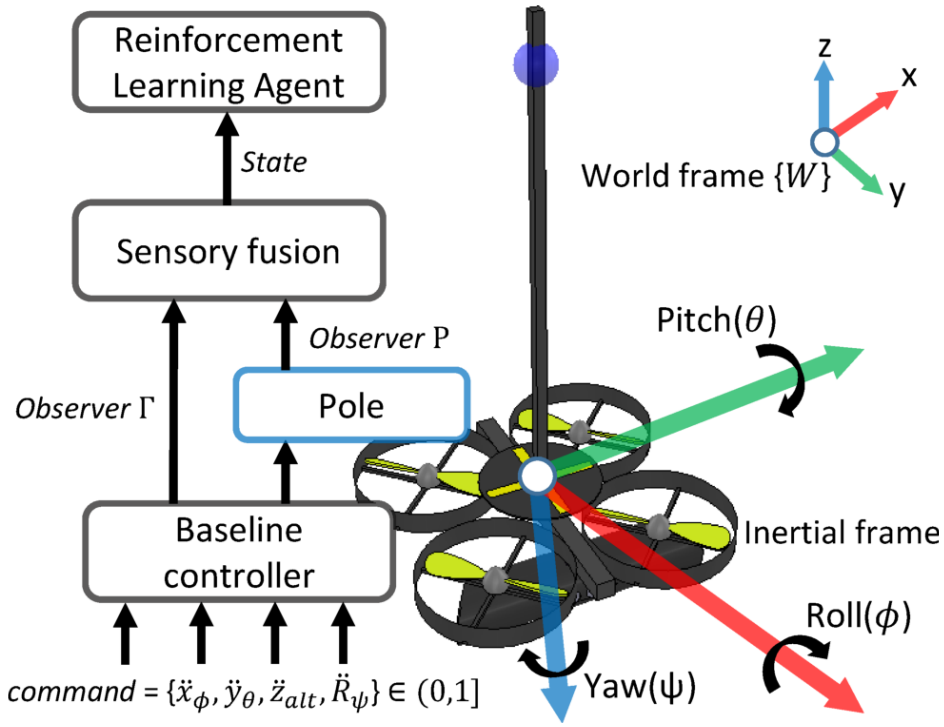
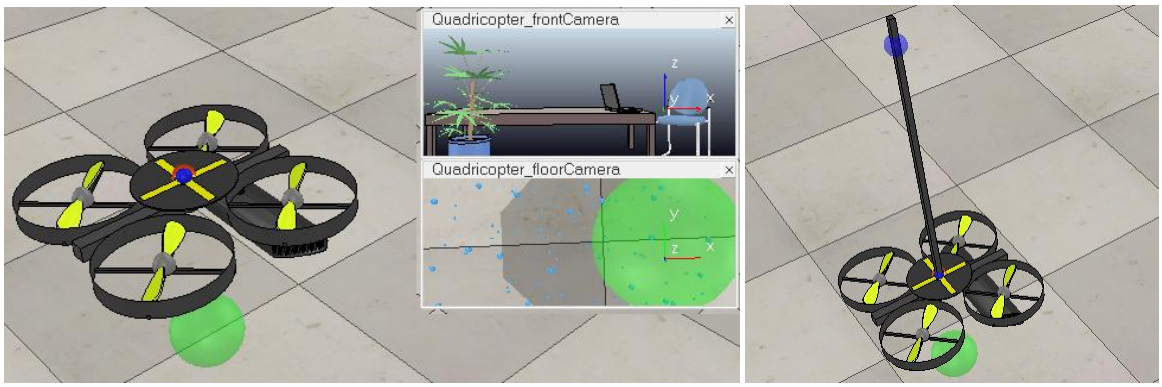


Fig. 5.1 Simulated quadcopter model and its control system block diagram. The system learns a specific task of balancing an inverted pole; The blue sphere on the pole represents the trackable landmark of the pole for external vision sensors. It queries the simulator, fuse sensory data and receives a reward for a state. Settings are practically achievable in a real-world situation. The hovering task follows the same control protocol but excludes the observable pole.



(a) (b)

Fig. 5.2 Simulated environments and models for the presented tasks in this study. The V-REP quadcopter simulator: (a) Quadcopter hovering and (b) inverted pole balancing on the quadcopter. In quadcopter tasks, the green sphere is the referenced area for quadcopter hovering.

We trained the hovering and inverted pole balancing (IPB) tasks using a quadcopter simulation, that is, the Virtual Experimentation Platform (V-REP) with real-time simulation[68]. V-REP is more of an application-based simulation. It was easier to interface for robotic tasks, which were simulated in the Open Dynamics Engine (ODE) physics engine with asynchronous real-time operation.

5.3.1 V-REP simulator

We simulated the quadcopter model (Fig. 5.1) in a real-world paradigm, as described in our prior experiment[62], by including a simulated Kinect vision sensor to generate a world map (W) using the parallel tracking and mapping (PTAM)[47] algorithm; In an IPB task, we obtained the pole states by using external vision sensors to track the pole, which was similar to the settings described by Bethke et al.[10]. This merged the acquired information for the localized states of the quadcopter and pole, instead of using the absolute world coordinates provided by the simulator. We also considered a disruptive environment that can cause the quadcopter to deviate away from a course or hovering point. The error is primarily driven by the drift of the quadcopter, which can easily be picked up during hovering. In real-world situation, the error is caused by imprecisely tuned internal mechanics, such as propeller vibration or external sources, e.g., wind. In this study, we simulated the disruptive space by continuously exerting a force on the quadcopter.

We implemented a baseline controller for the simulated quadcopter, based on linear proportional-integral-differential (PID) model, to translate the desired input acceleration into

motor rotation. The implemented controller was independent from the performed tasks. We designed the simulated quadcopter tasks following the problem statements: The hovering (*Problem 2.1*) and the IPB (*Problem 2.2*) task. All tasks had two action spaces for controlling the for/backward (induced by a velocity change of pitch θ) and left/right (by roll ϕ velocity change) acceleration $[\ddot{x}_\theta, \ddot{y}_\phi]$. The altitude of the quadcopter stayed fixed at 5.1 m above the ground and its heading aligned with the x-axis. The hovering task had eight state dimensions representing the linear and angular states of the quadcopter Γ and $\dot{\Gamma}$, each episode was initialized with a state of $\Gamma = \dot{\Gamma} = 0$, whereas the IPB task had 12 state dimensions that extensionally included the pole's relative position to the quadcopter and its joint angular velocity, P and $\dot{P}$, initialized with $\Gamma = \dot{\Gamma} = P = \dot{P} = 0$. The pole had a length of 1.15 m and its center of mass was 0.62 m away from its base, which was attached to the quadcopter.

Moreover, in the hovering task, an episode was terminated if the quadcopter failed to hover around the origin of the world map W by crashing or flying outside a square area of 5.0 by 5.0 m. In the IPB task, terminal conditions also included when the pole fell below a predefined height of 0.8 m with respect to the height of the quadcopter. In all of the attempted tasks, we introduced random noise to the quadcopter model to encourage learning. Otherwise, it might have ended up learning a policy that stopped the quadcopter from making any movement from the origin.

The following step reward function is used for hovering task:

$$r(t) = \begin{cases} -1000, & \text{if terminal} \\ -(\Gamma_{err}^2 + 0.01\dot{\Gamma}^2), & \text{otherwise} \end{cases}$$

where the Γ_{err} is the distance to the hovering target from current position $\Gamma(x_\phi, y_\theta)$ of quadcopter. The inclusion of velocity term $\dot{\Gamma}(\dot{x}, \dot{y})$ helps to learn less aggressive maneuvers. Moreover, for the task of balancing the inverted pole, the step reward is given as

$$r(t) = \begin{cases} -2000, & \text{if terminal} \\ 20 - (0.01P^2 + 0.005\Gamma^2), & \text{otherwise} \end{cases}$$

Here, the design of the reward function is key to agents learning a good control policy and encourage faster convergence. We used a quadratic form for the reward function in both tasks, which are described by Kim et al.[43] and Bagnell et al.[6]. The reward became more negative as the distance from the target increased. We also found that a large negative penalty reward for termination is required in order to train our models properly, because the experiments were designed for the quadcopter to operate as long as possible in the simulated

environment. For the hovering task, the total episodic reward could become very negative for a long episode. The trained model did not yield good performance in the testing phase, so the large negative penalty prevented this from occurring, and for the inverted pole balancing task, we additionally introduced a value of 20 as a survival bonus in the step reward.

5.3.2 Integrating concurrent framework to flight simulation

We develop a Python script using the asynchronous mode V-REP API framework and robotic operating system (ROS) framework to interact with V-REP environment. The remote API functions of V-REP allow third party application or hardware to control a simulation, it is achieved through socket communication in a way that reduces lag and network load to a great extent, this is all happened in a hidden fashion to the user. During learning, we run V-REP in headless mode to relief the computation resources of the V-REP interface.

5.4 Autonomous flight simulation results

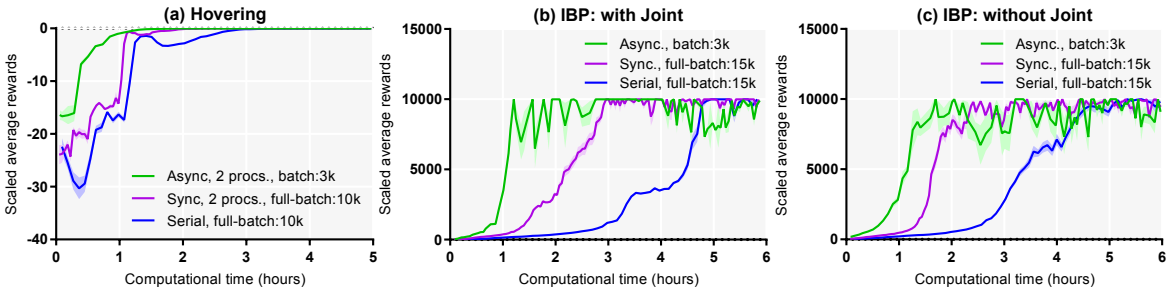


Fig. 5.3 Learning results for the hovering and inverted pole balancing (IPB) tasks of the simulated quadcopter model in the V-REP environment. Rewards are averaged across the 10 best testing runs: (a) Hovering is plotted with a scaled average reward to facilitate easier presentation and comparison.

To test out the trained policy model, we followed the same evaluation protocol as is described in the empirical studies of the MuJoCo simulator. The convergent speeds of all methods for the hovering and IPB tasks of the V-REP-simulated quadcopter are summarized in Fig. 5.3. The results show that all methods find solutions to both tasks and have a similar pattern to the empirical studies, which is that the asynchronous method has the fastest convergence speed. It took about 1 hour for the hovering task to converge to reasonable reward scores and 2 hours to learn the balancing task. Moreover, we used the asynchronously trained model to investigate the quality of performed maneuvers of the quadcopter and plotted the resulting trajectories in Figs. 5.4 and 5.5.

5.4.1 Hovering task: comparing with our previous study

Fig. 5.4(a) shows the trajectory of the quadcopter flying in a disruptive environment without any control model applied. The noise was randomly simulated in the task and caused the quadcopter to drift away from the hovering target. During the asynchronous training of the quadcopter, it generally showed qualitative improvements in the behavior of the hovering precision (reduction of drifting error). However, during iteratively training of randomly explored hovering maneuvers, it sometimes performed aggressive maneuvers because of the stochastic policy properties, which lead the quadcopter to pass the target position and cast away. However, no critical crash was recorded during the training, such as the quadcopter flip-over. Fig. 5.4(b) is the trajectory of the quadcopter performing the test hover flying around the origin (0,0) in the disruptive space for 40 s. We logged the x and y positions of the quadcopter every 50 ms, and the standard deviations of the logged x and y position are $\{\sigma_x, \sigma_y\} = \{0.0029, 0.0025\}$m. Eventually the quadcopter learned how to recover from casting away. Furthermore, we examined the generalization of the trained model with different initial conditions during testing. The trajectories are shown in Fig. 5.4(c) and (d) with different starting coordinates of (0.3 m, 0.2 m) and (-0.5 m, -0.5 m) respectively, all trajectories last 40 s, and the x- and y- positions were logged at 20 Hz. The results demonstrated that, within the policy exploratory region of a 5 x 5-m square area, the trained model was generalized enough to be able to reach the hovering target from different points and hover at the target.

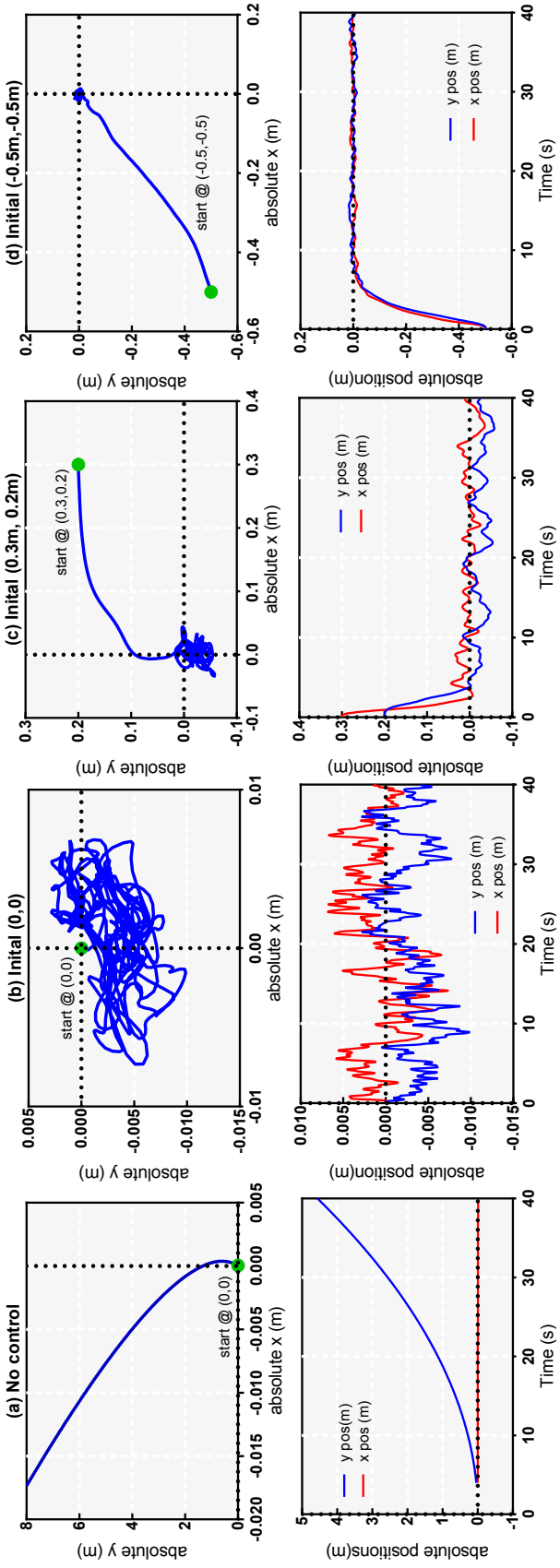


Fig. 5.4 Simulated quadcopter trajectories for control model of hovering task; The top row shows the x-y trajectory plots of the quadcopter, where the green dots represent the starting position of quadcopter in the world map. The bottom row plots the corresponding x and y trajectories against the wall-clock time, which lasted for 40 s.

We defined the **precision** of a trajectory in our prior study[62] as the total area covered by the trajectory from the hovering point(i.e.,the origin). The total area is estimated by integrating the trajectory, and the obtained value statistically has a unit, absement, which is mathematically measured in $m \cdot s$ (meter second); as the smaller the value, the more precise the trajectory control of hovering is. To make the previous results comparable, we tested a self-organized incremental neural network with associative memory (SOIAM)-based method in the hovering simulation task. The obtained total values (sum of the values of the x and y trajectories) were 60.766 for no control, 0.743 for the SOIAM-based control system and 0.231 for the asynchronous method from this study. Therefore, the learned policy model of the asynchronous method in this study was demonstrated to improve and advance the control precision.

5.4.2 Inverted pole balancing task

An inverted pole on the quadcopter made the whole model dynamically unstable. In this study, the pole was attached to the quadcopter through a revolutionary joint, which caused interaction between the quadcopter and pole. It was guaranteed to fall when no control was applied to the quadcopter, which was considered an extension of the 1-D cart-pole balancing task shown in Fig.4.5(a) from chapter 4. A few prior works have studied the dynamic equilibrium of similar models, such as Hehn et al.[35] and Figueroa et al.[27]. They conducted the experiment with the pole placed directly on the quadcopter without attaching it, and the effect of the pole becomes illegible during balancing. Figueroa et al.[27] also studied a control system for this type of model using a value fitting method with multiple policy models to complete various subtasks of balancing, whereas in this work, we used a single policy model with a minimally informative reward function and a policy gradient search method to accomplish a similar task. We summarized the testing results of the trained control policy for both models in Fig.5.3(b) and (c). For both models, we used the same reward function and settings. The learning speeds were very close to each other. We noticed a fluctuation of the averaged rewards in both test results. This is because the trained policies did not have a specific trajectory to complete an episode, and they exploited various paths to survive the terminal conditions.

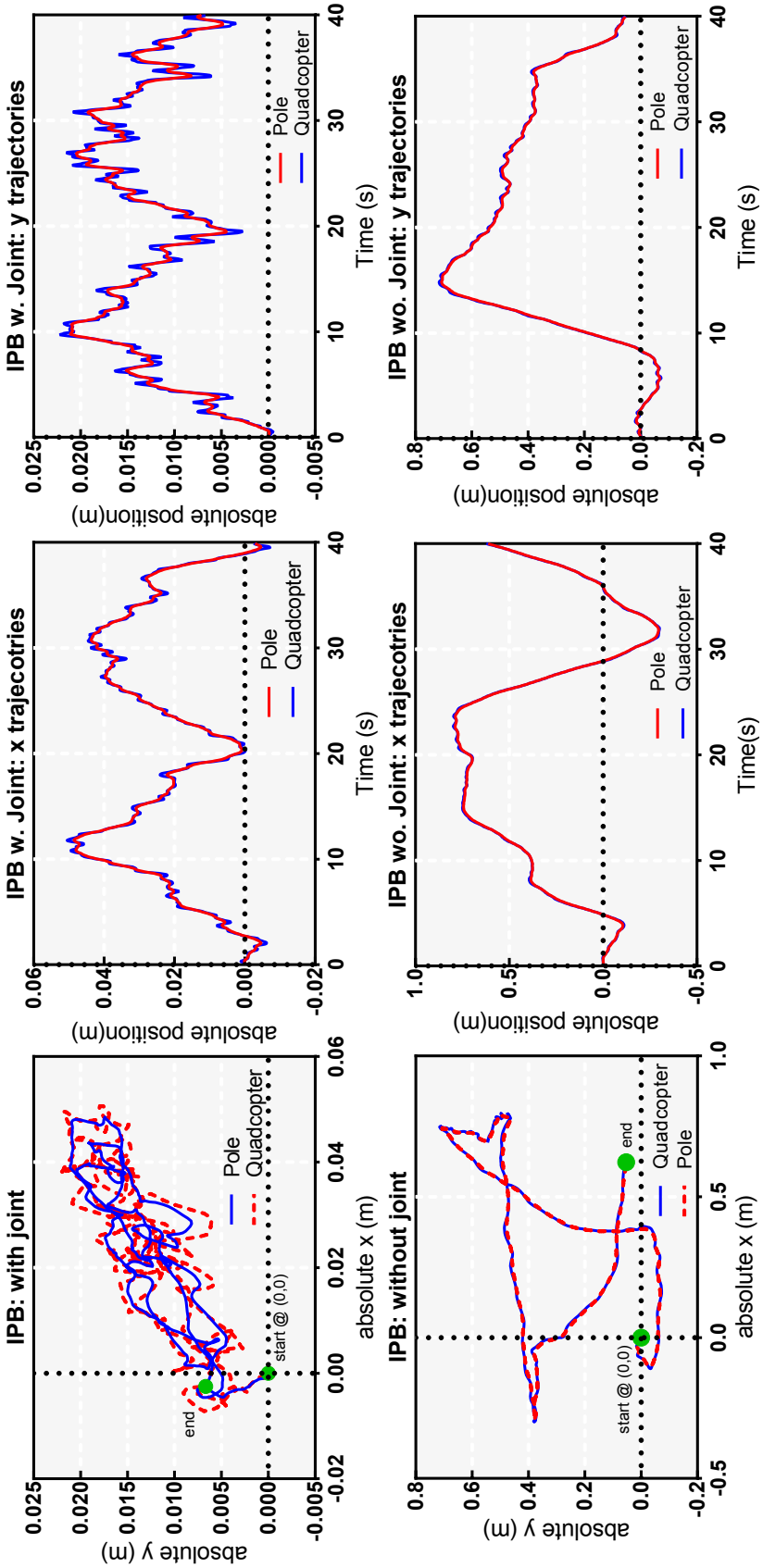


Fig. 5.5 Trajectory plots of a simulated quadcopter for IPB tasks. Top row: the pole is attached to quadcopter through a joint; Bottom row: without the joint; Left column: The x-y plots of quadcopter and pole positions, where the green dots denote the start and end of a 40 s trajectory; Middle and right columns: The corresponding x and y positions against wall-clock time for 40 s.

We plotted the trajectories of the quadcopter balancing the pole by means of attached joint in Fig. 5.5, where the model without a joint behaves similarly. The key to balancing the pole successfully is for the learned controller to command the quadcopter to closely follow the center of mass of the pole. However from our observations, if we only try to balance the pole, the quadcopter will have a tendency to fly away from the origin (hovering target) until it terminates at the 5 x 5-m boundary in order to catch up to the continuously falling pole and not turning back. Therefore, the results shown in Fig.5.5 are not only for balancing the pole, but also attempting to hover at the origin by manipulating the pole in a reverse falling direction. Furthermore, the maneuver speed of quadcopter in the task without a joint is much slower and hovering is less precise than the task with a joint, it is because when the pole is not firmly attached to the quadcopter body, the vibration of the quadcopter causes the pole to drift on the platform. Therefore, resulting a trained policy with slower maneuverability of the quadcopter, hence it is more difficult for the quadcopter to turn back to the hovering position when attempting to balance the pole. We considered these tasks to be solved as the inverted pole was balanced and survived from reaching the terminal conditions for more than 40 s.

5.5 Discussion

We combined parallelism methods, in particular of the synchronous and asynchronous actor-critic frameworks, with insights from trust region optimization methods. This resulted in algorithms that successfully sped up the learning of stochastic control policies with stabilized iterative updates by asynchronously training with smaller step-size batches, and improved the efficiency of computational costs. We used a recurrent neural network to learn control policy from scratch and examined tasks with different continuous action dimensions running in two different types of physical simulation engines. Moreover, this study showcased the learning of generalized quadcopter controllers for performing various maneuvers. Our quadcopter simulation design also provided real-world perspectives by utilizing a simultaneous localization and mapping (SLAM) system for the quadcopter to perform self-localization in the simulated environments, as close as possible to a real-world situation.

5.5.1 Stability

The stability of training can also be an issue in real-world settings for a model-free scenario. In this case, during the exploration phases, robots rely on the stochastic policy to interact with the environment and update the model accordingly; therefore, unstable training can

possibly lead to the catastrophe of losing the quadcopter. Stochastic gradient decent (SGD) is conventionally employed in machine learning as an optimization algorithm for solving non-convex problems, and its behavior in an asynchronous update manner within distributed systems has also been investigated in many prior works (Dean et al.[20], Heigold et al.[36]). This also includes the study of other popular SGD extensions such as ADAGRAD (An adaptive gradient algorithm proposed by Duch et al.[22]), RMSProp (Introduced by Tieleman and Hinton[78]) and Adam (Adaptive moment estimation[45]), but the difficulty of training with SDG is in optimizing the parameters, as it is especially sensitive to learning rate and convergence criteria; the worst case can cause network to diverge during training. In this study we used an optimization algorithm that combines L-BFGS and conjugates the gradient (CG) with a line search to optimize our AC network architecture. We experienced that these methods are much more stable during synchronous and asynchronous training frameworks, and particularly in the asynchronous method with a smaller batch size, significant speed and rewarded score improvements were observed. Although comparing the speed of L-BFGS/CG methods with SGD for an asynchronous update perspective is beyond the scope of this study, some studies have found competitive improvement to SGD on speed and accuracy by using L-BFGS/CG methods[50].

5.5.2 Scalability

This work focused on the experimental tasks of quadcopter. We hope that the proposed methods can serve as a bridge for connecting our prior work with future works, in which there is a possibility of scaling up the training with multiple real-world aerial vehicles in parallel, as the robotic arm farm described by Levine et al.[52]. Our experimental results demonstrated that less data batches are required to achieve the desired reward scores. Therefore, incorporation of this work not only speeds up the training but also reduces the complexity of the data acquisition. To transfer the quadcopter problems from training in simulation to real-world settings, there are few methods that can be applied to train the quadcopter: Tasks are trained directly using real-world quadcopter from scratch, as our prior method[62] of employing an additional supervision of professional human operator to maneuver the AR drone quadcopter, and then learn the maneuvers using SOIAM to predict the control commands of a quadcopter from real-world data. To remove this dependency on directly training in real-world settings and instead train in simulation. That is taking the deep neural network trained in simulation and applying it directly on the real-world quadcopters, however the difficulties are that, the simulated environment and quadcopter models should resemble the real-world settings as much as possible, and the noises introduced by the dynamics of a real quadcopter are difficult to accurately simulate, such as the vibration of

motors and the precision of the inertial and visual sensors. There are solutions of using hybrid techniques that utilize the supervised training data for transferring the network to real-robot settings. These include the recent achievements of AlphaGo[74] by initially manipulating the human play of Go games, and later refine its policy through a gradient method to learn how to win the game in real world. There is also the apprenticeship learning proposed by Pieter Abbeel et al.[2], to train a helicopter to performing aerobatics with data obtained from expert demonstrations.

Chapter 6

Conclusion and outlooks

The major challenges we encounter when applying model-free learning based control to aerial robotic control are: (i) We need to explore the state and action space of the control task for valuable training data. (ii) We need to identify an optimal set of hyper-parameter for learning the system. (iii) Even when these two issues are resolved, it can be computationally expensive and time consuming to find good control policy.

In this dissertation, we described efficient algorithms that address each of these three issues in the associative memory based self-organized incremental neural network learning (SOIAM) setting and concurrent deep reinforcement learning (CRL) setting. In SOIAM setting, we initially developed a system that enables a low-cost quadcopter - such as the Parrot AR.Drone - to localize itself in initially unknown, no trackable artificial markers, no external sensors and GPS-denied environments using on-board monocular camera of AR.Drone, this is achieved by employing a keyframe-based simultaneous localization and mapping (SLAM) algorithm - parallel tracking and mapping (PTAM). After we have collected manual demonstrations of the task with the associative pair - visual pose estimate and control command. Our SOIAM based learning algorithm leverage manual demonstrations to (i) Learn a description of the hovering control task in the form of intended trajectories. (ii) Learn a continuous control policy without the trial-and-error paradigm of requiring explicit and diverse exploration. Our SOIAM based learning algorithms are guaranteed to return a control policy that performs comparably to the manual control and the traditional proportional-integral-differential (PID) control in terms of precision.

In the CRL setting, we have also presented asynchronous and synchronous frameworks combined with RL algorithm that resolve the exploration problem in both the SOIAM setting and general model-free learning settings. We have not only provided empirical studies of continuous Mujoco simulation, but have also shown how these frameworks have enabled us to solve some very challenging tasks of simulated quadcopter, they have demonstrated

significant improvements on the training time for the highest reward. They have enabled a multi-instance training of robots to provide large diversity of exploration. They have significantly extended the state-of-the-art of learning based control in autonomous quadcopter flight. Our simulated quadcopter has performed by far the most challenging aerobatic maneuvers that we ever encountered, including maneuver such as inverted pole balancing, which even expert human pilots cannot fly properly.

6.1 Future work

There are still fundamental problems waiting to be solved, which concerning both the simulation and real world setting.

- Transferring from simulation setting to physical world setting. Our work has shown that simulations can indeed be used to learn policies for quadcopter control. Moreover, we are looking into methods of how to transfer learned policies from simulation to the real world quadcopter, and proving their practicality. Since these methods can definitely improve the learning capability of quadcopter, as simulation is more tolerant of trial-and-error processes.

- Large scale training. In the chapter 3 and 4, we showed that the concurrent learning instance with maximum of 2 instance, due to the computational bottleneck, we would like increase the number of instance to a larger scales of learning. Possibly with distributed computing frameworks.

- Combining the CRL with SOIAM. We would like to incorporate both settings together to form a consistent learning algorithm. We considered the CRL as an on-policy method, we hope to combine it with SOIAM that as an off-policy method, which can maintain a associative network of experience. Through iterative procedure, we can prioritized the experience learned by SOIAM, and selectively train the CRL with prioritized experience. This can help improve the learning property and sample efficiency of the CRL.

- Investigating more complicated tasks and environments. The ultimate goal for machine learning or artificial intelligence is to create generalized agent, which is capable of understanding the world in a general way like human. In robotics, the agents must also be able to deal with new objects without obstructing its ability to perform a task. The future work could explore the effectiveness of training multiple agents with CRL

frameworks in a number of different environments in which adding variations to the policy.

References

[1] Abbeel, P., Coates, A., and Ng, A. Y. (2010a). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639.

[2] Abbeel, P., Coates, A., and Ng, A. Y. (2010b). Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29:1608–1639.

[3] Achtelik, M., Weiss, S., and Siegwart, R. (2011). Onboard imu and monocular vision based control for mavs in unknown in-and outdoor environments. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 3056–3063. IEEE.

[4] Ahrens, S., Levine, D., Andrews, G., and How, J. P. (2009). Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments. In *2009 IEEE International Conference on Robotics and Automation*, pages 2643–2648.

[5] Armesto, L., Tornero, J., and Vincze, M. (2007). Fast ego-motion estimation with multi-rate fusion of inertial and vision. *The International Journal of Robotics Research*, 26(6):577–589.

[6] Bagnell, J. A. and Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1615–1620. IEEE.

[7] Barshan, B. and Kuc, R. (1992). A bat-like sonar system for obstacle localization. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(4):636–646.

[8] Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846.

[9] Benavidez, P. and Jamshidi, M. (2011). Mobile robot navigation and target tracking system. In *System of Systems Engineering (SoSE), 2011 6th International Conference on*, pages 299–304. IEEE.

[10] Bethke, B., Valenti, M., and How, J. (2007). Cooperative vision based estimation and tracking using multiple uavs. In *Advances in Cooperative Control and Optimization*, pages 179–189. Springer.

[11] Bouabdallah, S. (2007). *Design and control of quadrotors with application to autonomous flying*. PhD thesis, Ecole Polytechnique Federale de Lausanne.

[12] Bouabdallah, S., Murrieri, P., and Siegwart, R. (2004). Design and control of an indoor micro quadrotor. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4393–4398 Vol.5.

[13] Bouffard, P., Aswani, A., and Tomlin, C. (2012). Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. In *2012 IEEE International Conference on Robotics and Automation*, pages 279–284.

[14] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[15] Calise, A. J., Hovakimyan, N., and Idan, M. (2001). Adaptive output feedback control of nonlinear systems using neural networks. *Automatica*, 37(8):1201–1211.

[16] Chowdhary, G. V. and Johnson, E. N. (2011). Theory and flight-test validation of a concurrent-learning adaptive controller. *Journal of Guidance, Control, and Dynamics*, 34(2):592–607.

[17] Corke, P., Lobo, J., and Dias, J. (2007). An introduction to inertial and visual sensing. *The International Journal of Robotics Research*, 26(6):519–535.

[18] Coza, C., Nicol, C., Macnab, C. J. B., and Ramirez-Serrano, A. (2011). Adaptive fuzzy control for a quadrotor helicopter robust to wind buffeting. *J. Intell. Fuzzy Syst.*, 22(5,6):267–283.

[19] de Rengervé, A., Boucenna, S., Andry, P., and Gaussier, P. (2010). Emergent imitative behavior on a robotic arm based on visuo-motor associative memories. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1754–1759. IEEE.

[20] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. (2012). Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, pages 1223–1231.

[21] Degris, T., Pilarski, P. M., and Sutton, R. S. (2012). Model-free reinforcement learning with continuous action in practice. In *Proceedings of the 2012 American Control Conference (ACC)*, pages 2177–2182. IEEE.

[22] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[23] Efe, M. Ö. (2011). Neural network assisted computationally simple pi d control of a quadrotor uav. *Industrial Informatics, IEEE Transactions on*, 7(2):354–361.

[24] Engel, J., Sturm, J., and Cremers, D. (2012). Camera-based navigation of a low-cost quadrocopter. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2815–2821.

[25] Engel, J., Sturm, J., and Cremers, D. (2014). Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems*.

[26] Erez, T., Tassa, Y., and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4397–4404. IEEE.

[27] Figueroa, R., Faust, A., Cruz, P., Tapia, L., and Fierro, R. (2014). Reinforcement learning for balancing a flying inverted pendulum. In *Proceedings of the 11th World Congress on Intelligent Control and Automation*, pages 1787–1793. IEEE.

[28] Furao, S. and Hasegawa, O. (2006). An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, 19(1):90–106.

[29] Gadewadikar, J., Lewis, F. L., Subbarao, K., Peng, K., and Chen, B. M. (2009). H-infinity static output-feedback control for rotorcraft. *Journal of Intelligent and Robotic Systems*, 54(4):629–646.

[30] Gonzalez-Vazquez, S. and Moreno-Valenzuela, J. (2010). A new nonlinear pi/pid controller for quadrotor posture regulation. In *2010 IEEE Electronics, Robotics and Automotive Mechanics Conference*, pages 642–647.

[31] Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46.

[32] Grisettiyz, G., Stachniss, C., and Burgard, W. (2005). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2432–2437.

[33] Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems 27*, pages 3338–3346.

[34] Hasselt, H. V. (2010). Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621.

[35] Hehn, M. and D'Andrea, R. (2011). A flying inverted pendulum. In *Proceedings of 2011 IEEE International Conference on Robotics and Automation*, pages 763–770.

[36] Heigold, G., McDermott, E., Vanhoucke, V., Senior, A., and Bacchiani, M. (2014). Asynchronous stochastic optimization for sequence training of deep neural networks. In *Proceedings of the 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5587–5591.

[37] Hoffmann, G. M., Huang, H., Waslander, S. L., and Tomlin, C. J. (2011a). Precision flight control for a multi-vehicle quadrotor helicopter testbed. *Control engineering practice*, 19(9):1023–1036.

[38] Hoffmann, G. M., Huang, H., Waslander, S. L., and Tomlin, C. J. (2011b). Precision flight control for a multi-vehicle quadrotor helicopter testbed. *Control engineering practice*, 19(9):1023–1036.

[39] J. Reddi, S., Hefny, A., Sra, S., Poczos, B., and Smola, A. J. (2015). On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems 28*, pages 2647–2655.

[40] Jones, E. S. and Soatto, S. (2011). Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research*, 30(4):407–430.

[41] Kakade, S. (2002). A natural policy gradient. *Advances in neural information processing systems*, 2:1531–1538.

[42] Kelly, J. and Sukhatme, G. S. (2009). Visual-inertial simultaneous localization, mapping and sensor-to-sensor self-calibration. In *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*, pages 360–368.

[43] Kim, H. J., Jordan, M. I., Sastry, S., and Ng, A. Y. (2004). Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*, pages 799–806. MIT Press.

[44] Kimura, H., Yamashita, T., and Kobayashi, S. (2001). Reinforcement learning of walking behavior for a four-legged robot. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, volume 1, pages 411–416. IEEE.

[45] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[46] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan.

[47] Klein, G. and Murray, D. (2009). Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE.

[48] Kneip, L., Weiss, S., and Siegwart, R. (2011). Deterministic initialization of metric state estimation filters for loosely-coupled monocular vision-inertial systems. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2235–2241.

[49] Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U. (2011). A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160.

[50] Le, Q., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., and Ng, A. (2011). On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 265–272.

[51] Lee, T., Leok, M., and McClamroch, N. H. (2013). Nonlinear robust tracking control of a quadrotor uav on se (3). *Asian Journal of Control*, 15(2):391–408.

[52] Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *arXiv preprint arXiv:1603.02199*.

[53] Li, L., Sun, L., and Jin, J. (2015). Survey of advances in control algorithms of quadrotor unmanned aerial vehicle. In *2015 IEEE 16th International Conference on Communication Technology (ICCT)*, pages 107–111.

[54] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[55] Lupashin, S., Schollig, A., Hehn, M., and D'Andrea, R. (2011). The flying machine arena as of 2010. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2970–2971.

[56] Minh, L. D. and Ha, C. (2010). Modeling and control of quadrotor mav using vision-based measurement. In *Strategic Technology (IFOST), 2010 International Forum on*, pages 70–75. IEEE.

[57] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*.

[58] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[59] Muller, M., Lupashin, S., and D'Andrea, R. (2011). Quadrocopter ball juggling. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 5113–5120. IEEE.

[60] Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., et al. (2015). Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*.

[61] Nodland, D., Zargarzadeh, H., and Jagannathan, S. (2013). Neural network-based optimal adaptive output feedback control of a helicopter uav. *Neural Networks and Learning Systems, IEEE Transactions on*, 24(7):1061–1073.

[62] Pei-Hua, H. and Osamu, H. (2015). Associative-memory-recall-based control system for learning hovering manoeuvres. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[63] Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

[64] Pounds, P., Mahony, R., Gresham, J., Corke, P., and Roberts, J. M. (2004). Towards dynamically-favourable quad-rotor aerial robots. In *Proceedings of the 2004 Australasian Conference on Robotics & Automation*. Australian Robotics & Automation Association.

[65] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5.

[66] Recht, B., Re, C., Wright, S., and Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, pages 693–701.

[67] Ritz, R., Mueller, M., and D'Andrea, R. (2012). Cooperative quadrocopter ball throwing and catching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4972–4978. IEEE.

[68] Rohmer, E., Singh, S. P., and Freese, M. (2013). V-rep: A versatile and scalable robot simulation framework. In *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE.

[69] Salih, A. L., Moghavvemi, M., Mohamed, H. A. F., and Gaeid, K. S. (2010). Modelling and pid controller design for a quadrotor unmanned air vehicle. In *2010 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, volume 1, pages 1–5.

[70] Salmon, P. C. and Meissner, P. L. (2015). Mobile bot swarms: They're closer than you might think! *IEEE Consumer Electronics Magazine*, 4(1):58–65.

[71] Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015a). Trust region policy optimization. In *Proceedings of the the 32nd International Conference on Machine Learning*, pages 1889–1897.

[72] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[73] Siegwart, R., Bouabdallah, S., et al. (2007). Design and control of quadrotors with application to autonomous flying.

[74] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[75] Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-time monocular slam: Why filter? In *2010 IEEE International Conference on Robotics and Automation*, pages 2657–2664.

[76] Sudo, A., Sato, A., and Hasegawa, O. (2009). Associative memory for online learning in noisy environments using self-organizing incremental neural network. *Neural Networks, IEEE Transactions on*, 20(6):964–972.

[77] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

[78] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2).

[79] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

[80] Vásárhelyi, G., Virágh, C., Somorjai, G., Tarcai, N., Szörényi, T., Nepusz, T., and Vicsek, T. (2014). Outdoor flocking and formation flight with autonomous aerial robots. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3866–3873.

[81] Weiss, S. and Siegwart, R. (2011). Real-time metric state estimation for modular vision-inertial systems. In *2011 IEEE International Conference on Robotics and Automation*, pages 4531–4537.

[82] Xu, B., Shi, Z., Yang, C., and Wang, S. (2013). Neural control of hypersonic flight vehicle model via time-scale decomposition with throttle setting constraint. *Nonlinear Dynamics*, 73(3):1849–1861.

[83] Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. (2010). Parallelized stochastic gradient descent. In *Advances in neural information processing systems 23*, pages 2595–2603.