Policy Hyperparameter Exploration for Behavioral Learning of Smartphone Robots

Jiexin Wang

A dissertation submitted to the Graduate School of Informatics in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Informatics

at Kyoto University

Acknowledgement

This work is accomplished by many people's help and support.

Firstly, I would like to thank Professor Kenji Doya, who introduced me into the smartphone robot project, guided me from the beginning of learning the Java programming language to the completeness of the robot platform. I feel deeply thankful and touched by his endless patience, inspiring ideas, consistent guidance and many nights reviewing and modifying our work dedicatedly. I am greatly indebted to him for supporting me attending Machine Learning summer school, many international and domestic conferences so that I gained the whole picture of Robotics and AI research and connections with other peers. I learned not only the techniques and the way of thinking, but also the research style and spirit from him. Joining his research group is a big fortune to my career and life.

I would like to thank Dr Eiji Uchibe, who is always there for me for any trivial questions. His endless patience and kindness is invaluble to this work. I learned enormously from so many enlightening discussions with him. I was amazed by his theoretical knowledge and technical skills.

I also feel very grateful to have Professor Shin Ishii as my supervisor in Kyoto University. Thanks to his encouragement for me to start the PhD study and setting up the collaboration with OIST so that I can work on the robot project for my main interest. I deeply appreciate his support for me moving from Okinawa to Kyoto so that I can continue my PhD research.

My great appreciation to Professor Sugie, and Professor Otsuka, who are the reviewers of my thesis. I'm thankful for their comments and suggestions to

improve the thesis.

Thanks to Dr Oba, who connected me with Ishii lab even I spent most of my PhD years in Okinawa. Every time I came back to Kyoto, he arranged seminar for me to obtain valuable feedback from Ishii lab students and staff. We had very deep discussions about my research status and how to be a good researcher.

I have to thank Dr Maeda, Dr Nakae, Dr Urakubo, Dr Meshgi in Ishii lab for their help of research advice and setup of research environment when I spent my days in Ishii lab.

Also my peers in OIST, they gave me lots of suggestions and encouragements during the time when I was in Okinawa. Thanks to Naoto Yoshida, who was the antecessor of the smartphone robot project, my work is based on his work. Thanks to Chris Reinke, who always made the theoretical issues easy for me to understand, and gave me professional advice of testing robots and building a good software system. Thanks to Peter Mekhail who designed the first chassis of the robot, Tosif Ahamed and Paavo Parmas for their technical supports when we worked on the project together.

The mental support of my family and sincere friends is also a credit to this work. I cannot finish this work without their understanding and being there for me.

My profound gratitude to everyone who was involved in my PhD study.

Funding

This study was supported by research funding of Okinawa Institute of Science and Technology Graduate University to the Neural Computation Unit.

List of publications

Journal

J. Wang, E. Uchibe, and K. Doya. Adaptive Baseline Enhances EM-based Policy Search: Validation in a View-based Positioning Task of a Smartphone Balancer. *Frontiers in Neurorobotics*. 11:1. DOI 10.3389/fnbot.2017.00001. 2017

J. Wang, E. Uchibe, and K. Doya. EM-based Policy Hyper Parameter Exploration: Application to Standing and Balancing of a Two-wheeled Smartphone Robot. *Artificial Life and Robotics*. 21:125. DOI 10.1007/s10015-015-0260-7. 2016

Conference

J. Wang, E. Uchibe, and K. Doya. Two-wheeled Smartphone Robot Learns to Stand Up and Balance By EM-based Policy Hyper Parameter Exploration. *In proceedings of the 20th International Symposium on Artificial Life and Robotics*, Beppu, Japan. (AROB 2015)

J. Wang, E. Uchibe, and K. Doya. Control of Two-Wheel Balancing and Standing-up Behaviors by an Android Phone Robot. *In Proceedings of the 32nd Annual Conference of the Robotics Society of Japan*, Kyushu Sangyo University, Fukuoka, Japan. (RSJ 2014)

J. Wang, E. Uchibe, and K. Doya. Standing-up And Balancing Behaviors of Android Phone Robot: Control of Spring-attached Wheeled Inverted Pendulum. *IEICE technical report*. Nonlinear problems 113(341), 49-54, Hongkong, China. (NLP 2013)

Abstract

Robots are the counterparts of human. Scientists in the field of Robotics and Artificial Intelligence spent decades on inventing human-like intelligence and generating biological agent-like behaviors on robots in a wide spectrum of applications from assisting human to understanding evolution. Our main research goal is to understand emergence of behaviors and minds of biological creatures by achieving various behaviors of robots. Such issues include investigating how an individual learns, how a group evolve, and how the reward systems are developed. To do so, we developed a smartphone based robotic platform and a novel Reinforcement Learning algorithm for the smartphone robots to achieve various behaviors.

Current robotic research platforms including humanoid robots, iCub and NAO, and small desktop miniatures, Khepera and e-puck are either too expensive, hard to maintain or with limited computational power and sensors. To overcome these limitations and enable variety of behaviors with more degree of freedom, we developed an affordable high-performance robotic platform, namely a smartphone balancer as an individual. A balancer's behavior contains basic mobilities of standing-up, balancing and stable running, and integrated complex behaviors like foraging, collision avoidance and mating etc. This platform is a practical testbed for designing optimal control and reinforcement learning algorithms, has high potential in the fields of behavior-based robotics and multiagent research etc, and also benefits education and hobby use.

In this thesis, we firstly propose the design and control architecture of the smartphone balancer and test the feasibility of the balancer to achieve standing-up and balancing behavior under the hardware constrains. Generally, it is

difficult to mount high torque motor on small-sized robot. We propose a spring-attached wheeled inverted pendulum model of the smartphone balancer to overcome the difficulty of motor torque limitation. We scan different combination of motor torque, spring coefficient and frequency of different controllers in both simulator and actual hardware system with hand-tuned policy parameters. Results show that the attachment of the elastic bumper simplified the control and assisted the robot to achieve the required behaviors successfully.

For a single agent to acquire basic behaviors more efficient and safe in practice, we developed a deterministic policy search method, so called EM-based Policy Hyperparameter Exploration (EPHE). EPHE integrates policy gradient method PGPE and EM-based policy search framework to avoid gradient calculation and learning rate tuning. It assumes a prior distribution over the policy parameters and updates the hyperparameters in a closed form solution of maximizing a lower bound of expected return. This results in a return weighted updating rule, when the prior distribution is from exponential family. However, updating with the fully observed dataset decelerates the learning performance due to the worse-behaving samples.

There are various discarding rules and weighting schemes based on the return history to preserve informative samples. Heuristically, we adopt a K-elite selection mechanism with EPHE (EPHE-K) to discard worse sample under a fixed baseline determined by the experimenter. Similar approaches were adopted in previous policy search methods like PoWER , FEM and CEM. Methods rescaled the returns in a convex transformation can also be equipped to our learning framework, including CMAES and REPS weighting schemes. CMAES reweighted the samples similarly by truncating the sample size and a logarithm transformation. REPS reweighted the samples in an exponential transformation of the corresponding return. However, all these methods have to determine a fixed baseline prerequisitely. To further avoid tuning the K parameter of the fixed baseline, we propose an adaptive baseline equipped to EPHE (EPHE-AB) to discard inferior samples below the average of the return history and further examine the effectiveness of different baselines including the mean and one and two standard deviations from the mean.

We implement EPHE-K and EPHE-AB with baselines of the mean, one and two standard deviation from the mean in three simulation tasks, including

pendulum swing-up with limited torque, cart pole balancing and standing up and balancing behavior in our smartphone balancer simulator. Results show that EPHE-K outperforms the previous policy gradient methods like PGPE and Finite Difference. And EPHE-AB-mean outperforms EPHE-K, EPHE with CMAES weighting scheme, EPHE with REPS weighting scheme, PGPE, NES, and FEM in the early stage of learning. It is shown that the choice of mean of the adaptive baseline is more effective in focusing on positive outliers than others in the beginning of learning, and leads to a steady decrease of discarding number of the samples during learning. We further implement EPHE-AB-mean in the real smartphone balancer system to achieve standing up and balancing behavior, and visual target approaching behavior. Results show that EPHE-AB-mean outperforms PGPE, and the smartphone balancer successfully achieved the behaviors.

Finally, to enable high level integrated behaviors, we implement EPHE-AB-mean for the balancer to learn foraging and mating behavior. Results show the successful achievement of the desired behaviors under low-cost hardware environment settings.

Contents

Figures, Tables and Algorithms

Chapter 1

Introduction

1.1 Why Smartphone Balancer?

Common robotic platforms which can facilitate researches by quick introduction at low cost and by sharing and comparison of programs are highly in demand for robotics and artificial intelligence society. Humanoid robots like NAO and iCub, small-sized robot like Khepera, e-puck and AIBO etc are the best examples (See Figure 1). However, they are either too expensive, hard to maintain or equipped with limited sensor and low computational power. Recent smart phone carries various features for constructing a robot, including a high-performance energy-efficient CPU, versatile sensors such as cameras, gyroscope, accelerometers, GPS, audio, wireless communications and software developing environment. By connecting actuators along with micro controller chips such as Arduino, Rasberry PI and IOIO board to the smartphone, we can construct a smartphone based robotic platform. Compared to traditional robots which require sophisticated structure design and time-consuming manufacture, smartphone based robots are more compact and handy for not only research teams but novice developers.

For a single agent, we consider a two-wheel balancer based on the proposed smartphone robotic platform. Balancing robot provides lively impression than quasi-static movements of tricycle or four-wheel robots. It has one more degree of freedom enabling various basic behaviors such as standing-up, balancing, and moving, and facilitating high-level integrated behaviors such as foraging, collision avoidance, and mating, etc. This bottom-up behavioral structure also corresponds to the emergence of natural behaviors from biological creatures. Under the framework of basic dynamics, it is fertile to test and develop control and learning algorithms. Under the higher level behavioral domain, it is primed for revealing the insights of learning and evolution.

In Chapter 2 we address the hardware construction and achievement of standing-up and balancing behavior of the individual smartphone robot. We propose a model of wheeled inverted pendulum with elastic bumper as the dynamics of the robot, and construct a simulator based on the model. We further utilize a switching control architecture incorporating a Linear Quadratic Regulator to stabilize the pendulum around the upright equilibrium and energy-charging controllers to destabilize it around the resting po-

sitions supported by the elastic bumpers to achieve the target behaviors. We investigate the relation among the coefficient of elastic bumper, the maximum motor torque, and the control parameters using the simulator. The simulation results show the attachment of the elastic bumper is able to overcome the difficulty of mounting high torque motor on small-sized robot under appropriate combinations. With the parameters, we reveal the hardware construction and implement the same control structure to the real robot system. The results show that with certain combinations of elastic bumper parameters and control parameters, the smartphone balancer succeeds to achieve standing-up and balancing behavior.

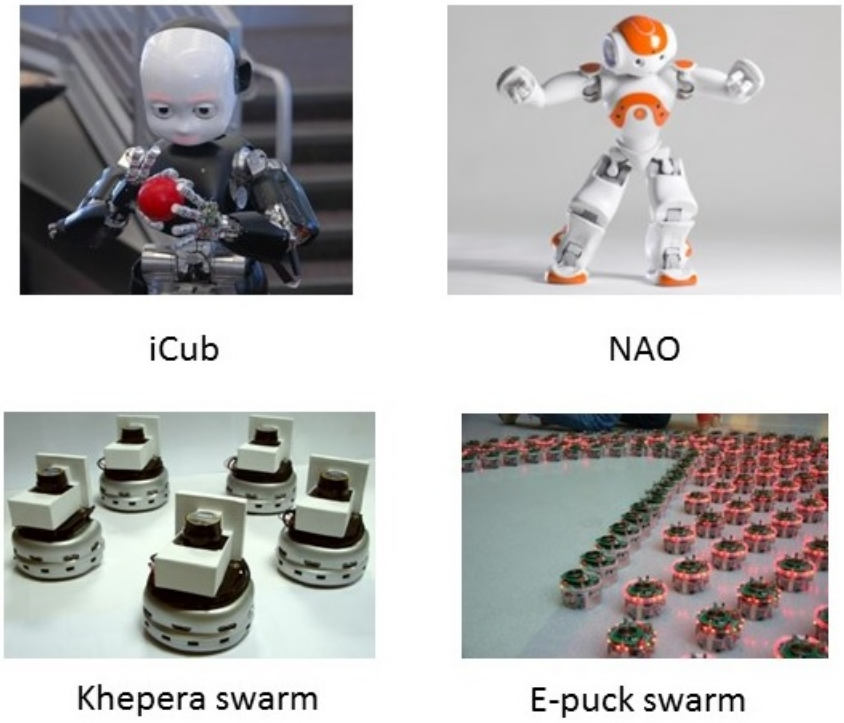


Figure 1. Current popular robotic platform for research

1.2 Policy-based Reinforcement Learning

Learning is an essential skill for animal and human to survive and nowadays machine learning technique is for the robot to equip intelligence and gain the autonomy. Reinforcement learning is one of the fields in machine learning inspired from behavioral psychology, concerned with how agents should take actions in an environment to maximize its expected return. It has plenty of stunning real world implementations for robots to acquire various capabilities, including Stanford's inverted autonomous helicopter [1], robot athletes playing sports like soccer in RoboCup, and table tennis [2], quadruped robot learning negotiating obstacles [3], deep reinforcement learning in autonomous car, Asimo doing housework, and iCub in embodied cognition [4], etc. Developing efficient and practical learning methods is in high demand.

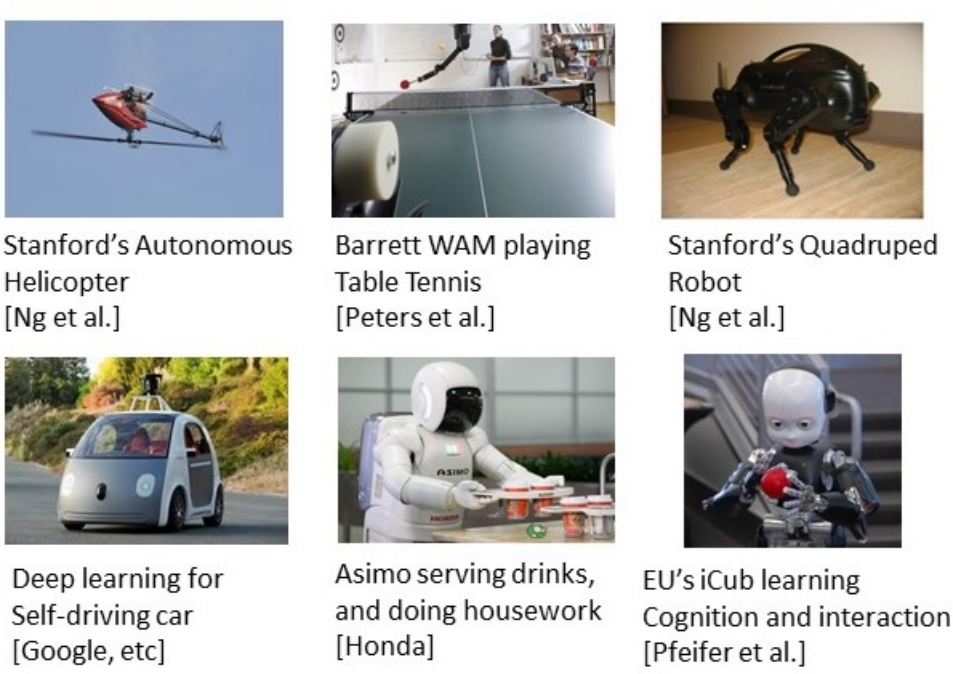


Figure 2. Robots' learning various behaviors

With the target for our smartphone robot to acquire behaviors more efficient and safe in practice, we consider a deterministic policy search method which is also applicable to other robotic platforms. Policy search methods are widely adopted in high-dimensional robotic problems with continuous states and actions among numerous reinforcement learning methods. They learn policies directly from reward functions as an alternative to value function-based reinforcement learning which requires a complete data filling of state-action space to assess the behavioral qualities. Classic policy gradient algorithms such as REINFORCE [5] and GPOMDP [6] suffer from high variance in the gradient estimates because of the noise injected at every time step for a stochastic policy. Policy Gradients with Parameter-based Exploration (PGPE) [7] addressed this problem by evaluating deterministic policies with the parameters sampled from a prior distribution. Natural Evolution Strategy (NES) [8] uses the natural gradient to update a parameterized search distribution in the direction of higher expected fitness, which can be applied under the PGPE framework. However, these methods require learning rate tuning when utilizing optimization method like gradient ascent.

Another trend is EM-based poilicy search, such methods as EM-based Reward-Weighted Regression (RWR) [9], EM Policy learning by Weighting Exploration with the Returns (PoWER) [10] and Fitness Expectation Maximization (FEM) [11], etc. These methods maximize the lower bound of the objective function with respect to the parameters of exponential family distributions, resulting in a reward-weighted averaging of the sampled parameters to avoid gradient calculation and learning rate tuning.

By combining PGPE and EM-based policy search, we propose a novel policy search algorithm so called EM-based Policy Hyperparameter Exploration (EPHE). EPHE has merits inherited from PGPE that 1) the use of deterministic policy reduces the variance of the gradient with respect to the hyperparameters, 2) the policy does not need to be differentiable so that a wide range of controllers can be chosen, and from RWR, 3) the learning rate tuning is avoided which is practical in real robot system.

However, updating with the fully observed dataset decelerates learning due to the worse-behaving samples. Heuristically, we adopt a K-elite selection mechanism with EPHE (EPHE-K) in sampling process of each hyperparameter updating to discard worse samples below a fixed baseline. We compare

EPHE-K with PGPE and Finite Difference method in benchmarks of pendulum swing-up with limited torque, cart-pole balancing, and standing-up and balancing behavior in the smartphone balancer simulator. The results show that EPHE-K outperforms previous methods.

Among the reward weighting algorithms, PoWER implicitly illustrated the discarding mechanism, and EPHE-K, CEM [12], and FEM are similar in preserving informative samples by considering a fixed baseline with the parameter determined by the experimenter. To further avoid the parameter tuning of K, we improve EPHE with an adaptive baseline (EPHE-AB) to discard worse samples below the average of the reward history in each batch updating. The simulation results of two benchmarks and the smartphone balancer show the improvements of performances compared to EPHE-K, EPHE with CMAES weighting scheme, EPHE with REPS weighting scheme, PGPE, NES and FEM. By investigating different adaptive baselines with the threshold of the mean, one and two deviations from the mean, we found that baseline of the mean discards most of the samples in the early stage of learning and has a steady decrease of number of discards samples during learning. We further implement EPHE-AB with the baseline of mean in the real smartphone balancer system to test its performance in the standing-up and balancing task and a view-based approaching task. The results show that EPHE-AB-mean outperforms the previous algorithms and realizes the behaviors more efficiently and reliably. In Chapter 3, we review policy based reinforcement learning algorithms and illustrate the results of our proposed methods.

EPHE-AB-mean is also supportive for the balancer to achieve high-level integrated behaviors including foraging and mating etc. We review the history of such survival behaviors and achieve the foraging and mating behaviors of the balancer in Chapter 4.

1.3 Towards High-level Behaviors

Behavior is a sequence of actions made by biological individuals in conjunction with the environment (including other systems or individuals) where the actions affect its own perceptions and consequently its future actions and perceptions. Current artificial intelligence and robotic technology allows

autonomous robots to become behavioral agents capable of habituating in partially unknown and changing environments. Such robotic platforms are powerful tools for not only understanding individual biological system and gaining insights in the emergence of intelligence, but also for revealing social psychological phenomenons and the nature of evolution by resembling biological colonies.

Behavior-based Robotics (BBR) [13] [14] is a field in robotics inspired from neuroscience, ethology and psychology, under which the robots are able to exhibit complex-appearing behaviors mostly by gradually correcting its actions via sensory-motor links, despite internal state to model its immediate environment. It assumes a subsumption architecture of the robot behaviors, which can be visualized as a layered stack of parallel behaviors where the behaviors at the bottom deal with the survival, and the behaviors at the top achieve desirable goals if opportunity exists. Higher layers encompass lower layers, and high-level intelligent competences such as communication, and reasoning are believed emerged from the basic survival abilities such as foraging, mating and escaping. Figure 3 shows the architecture.

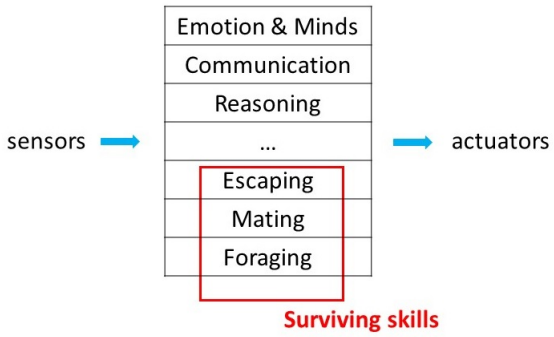


Figure 3. Subsumption architecture of robot behaviors

Reinforcement learning and genetic algorithms (implemented as evolutionary robotics [15]) are popular approaches for the robot to acquire basic and complex behaviors. Basic behaviors like standing-up and balancing, vision-based approaching are estabilished in Chapter 2 and Chapter 3. Here we focus on realizing survival capabilities regarded as prerequisite behaviors under the domain of BBRs. We define foraging behavior as approaching to a food source

and realizing battery charging, mating behavior as one agent approaching to another and exchanging genetic information through communication.

The first robot equipped with the foraging ability is the "Tortoise" developed by Walter in 1953 [16]. The behavior was replicated using a simple light following approach that allowed it to guide itself towards the recharge station in the arena. McFarland and Spier [17] defined a "basic cycle" consisting work - find station - recharge to investigate energy and utility in a self-sufficient autonomous mobile system [18]. Mataric proposed a reinforcement learning framework to achieve multi-robot foraging [19]. The behavior defines avoiding, dispersing, searching, homing and resting as behavior primitives. Floreano and Keller successfully achieved cooperative foraging using Khephera robots and further revealed the emergence of altruism [20]. Doya and Uchibe launched the "Cyber Rodent Project" [21], using groups of rodent like robots under the biological constraints of self-preservation (foraging) and self-reproduction (mating) to investigate the reward mechanism of robots and biological creatures. A Cyber Rodent can search for and recharge from battery packs and copy its program to another agent through infrared communication. The Cyber Rodent platform investigated different learning and evolution architectures for various behaviors [22] [23] [24], finding intrinsic and extrinsic rewards [25], examined Darwinian embodied evolution [26], and revealed the insights in the emergence of polymorphic mating strategies [27].

In Chapter 4, we introduce a real-world environment for the smartphone balancer to habitat. We focus on the surviving skills of the agent, namely foraging and mating. The hardware construction is modified based on the version in Chapter 2, and the learning method is consistent with the policy based reinforcement learning we propose in Chapter 3. The results show that the robots succeed in the surviving tasks and indicating a potential to understand more about emergence of behaviors and a practical framework to investigate reward systems.

Chapter 2

Design and Control of a Smartphone Balancer

2.1 Introduction

Small robots that can work on desktop spaces are in high demand for research education and hobby use. A common robot platform can facilitate research by quick introduction at low cost and by sharing and comparison of programs. Khepera, developed at EPFL in the mid 90s [28] is the best example of a common desk-top robot platform. Recent smartphones are loaded with almost everything we need for a robot, including a high-performance CPU, cameras, gyros, accelerometers, GPS, wireless communication and software developing environment, except for actuators. We have proposed smartphone-based robots as a low-cost high-performance desktop robot platform [29][30]. In [29] [30] Yoshida et al. developed a two-wheeled Android phone robot platform utilizing Nesus S and Arduino board as the motor control interface. They successfully realized two-wheel balancing bahavior using sensor fusion of gyro and accelerometers built in the smartphone as robot's state variables.

Two-wheel robots give lively impressions than quasi-static movements of tricycle or four-wheel robots. It has one more degree of freedom enabling various basic behaviors such as standing-up, balancing, and stable running etcand it is fertile to test and develop control and learning algorithms. Control of wheeled inverted pendulum has been studied throroughly in previous researches [31] [32] and real-world applications like Segway [33] are in practice. The most typical approach is to linearize the system around the unstable equilibrium and then apply linear controller for stabilization.

Here we consider a dynamic standing-up and balancing behavior of a smartphone balancer. In order to realize standing up far away from the upright equilibrium, however, a nonlinear control strategy is required. Here we propose elastic bumpers attached in both sides of the robot to support its body at resting positions, and assist it naturally recover to the upright region. It also allows pumping up of energy, which enables standing up behavior using toy motors with small output torques. We adopt a switching control architecture for achieving the entire behavior, namely a stabilizing controller near the upright equilibrium and a distabilizing controller near the resting position.

In this chapter, we first build a realistic numerical model of the elastic bumper

attached wheeled inverted pendulum in simulator and investigate the combinations of the spring constant, maximum motor torque, and non-linear control strategy allow successful standing up and balancing behavior. The switch control architecture incorporates a Linear Quadratic Regulator to stabilize the pendulum around the upright equilibrium and destabilizing controllers, namely, an Energy Pumping controller, a square wave controller, and a sine wave controller around the resting position supported by elastic bumpers. The simulation results show this control architecture can successfully achieve standing-up and balancing behavior with small motor torque. The results also illustrate the relation between the spring coefficient, maximum motor torque, and control parameters, which is useful for real hardware construction.

We then construct the smartphone balancer using the suggested hardware from the simulation such as the spring coefficients of elastic bumpers etc, and implement the same control architecture to realize the behavior. The hardware components include an Android phone Nexus 4, an Android compatible interface board, IOIO-OTG, the HUB-EE wheels with built-in motor, motor controller and rotary encoder, and a wireless chargeable battery. The robot shell is designed in AUTOCAD and 3D printed out. For balancing, we apply the descrete-time Linear Quadratic Regulator (LQR) as the stabilizer. The system dynamics parameters are identified by the least squares method using the data collected from the real robot experiments. For standing-up from the resting positions, we adopt a sine wave controller as a destablizer. We hand-tune the stiffness of the springs, as well as the parameters of the switching threshold, the amplitude of the motor output, and the frequency of the sine wave controller. The results show that the proposed control architecture can achieve the desired behavior although the success rate and the efficiency remain to be improved.

The hardware construction of the smartphone balancer is illustrated in Section 2.2. The simulation experiment including the physical dynamics of the wheeled inverted pendulum model extracted from the smartphone balancer, control architecture and experiment results are discribed in Section 2.3. The hardware experiment is discussed in Section 2.4. The summary of the chapter is in Section 2.5.

2.2 Hardware Construction

2.2.1 Overview

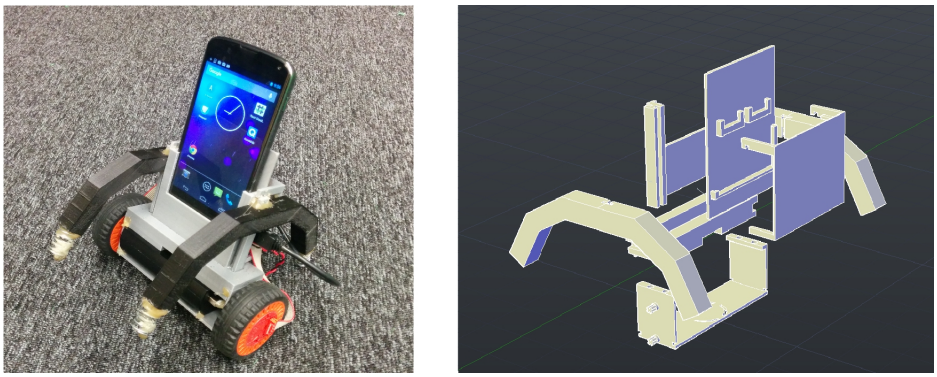


Figure 4. Current version of the smartphone balancer

Figure 4 shows the current version of our smartphone balancer. Its chassis is designed as an assembly kit for convenient composition and modification. The smartphone rests in the middle slot, a battery is clipped to the bottom container, two wheels are inserted into Lego cross sticks extruding from the body side, and the IOIO OTG board (the microcontroller chip) and break-out board slide into slots on the back. By sliding in two spring arms on both sides of the robot body, it becomes a spring-armed balancer that can achieve standing-up, balancing and approaching behaviors.

2.2.2 Hardware Connections

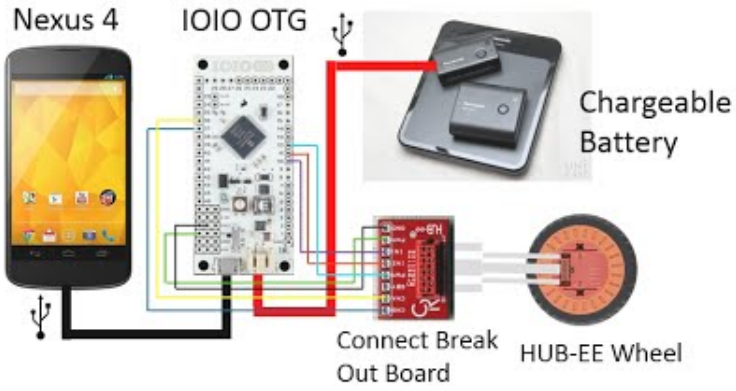


Figure 5. Hardware connection

As the balancers brain, we use Nexus 4, which was co-developed by Google and LG in November 2012. It has a 1.512 GHz quad-core Krait CPU, 2GB RAM and is operated on an Android Jelly Bean 4.2 system, which is compatible with IOIO OTG USB in the debugging mode. IOIO OTG is a board that provides a host machine that can interface with external hardware over a variety of commonly used protocols. Since it was specifically designed to work with Android devices, programming is on the Android side with IOIO libraries. We use the HUB-EE wheel as a robot servo. It integrates a DC motor, a Toshiba TB6593FNG motor driver, and a quadrature encoder, for real-time data recovery. The chassis is designed in AutoCAD and printed out by Ultimaker 2.

Figure 5 shows the hardwares connection. Nexus 4 is connected to IOIO through a USB cable. The battery outputs 5V into the IOIO board to supply power to both the IOIO board and the smartphone. The IOIO board also provides 5V to trigger the HUB-EE wheel. The connect break-out board clarifies communication channel and provides a neat socket connection between the HUB-EE wheel and the IOIO board.

Table 1 shows the details of pin connection. In1 and In2 transmit the digital signals from board to wheel for rotating direction. PWM transmits Pulse-Width Modulation signal from board to wheel to set the speed of the wheel.

chA and chB transmit the digital rotary encoder signals from wheel to board.

Left wheel		Right wheel	
Wheel channel	I/O pin	Wheel channel	I/O pin
In1	10	In1	5
In2	11	In2	6
Pwm	12	Pwm	7
chA	36	chA	34
chB	37	chB	35

Table 1. Connection between IOIO and HUB-EE wheel

2.2.3 Software Setup

The program is written in Java by developing an Android app in Android Studio. The IOIO developer provides high level Java APIs including IOIOLib Core, and IOIOLib Application to read in and out the digital input/output and the PWM input/output to transfer the sensory information and control signals. We also use the Open CV library to capture the target's visual information. In the standing-up and balancing task, three threads work simultaneously: the IOIO thread that sends control signals and retrieves encoder data every 1 ms (control cycle), a sensor thread that retrieves a fused gyro and an accelerometer every 6 ms, and a UI thread that visualizes the robots state and its learning process. In the approaching task, the camera sensor is activated and updated every 30 ms.

2.3 Simulation Experiment

2.3.1 Physical Dynamics

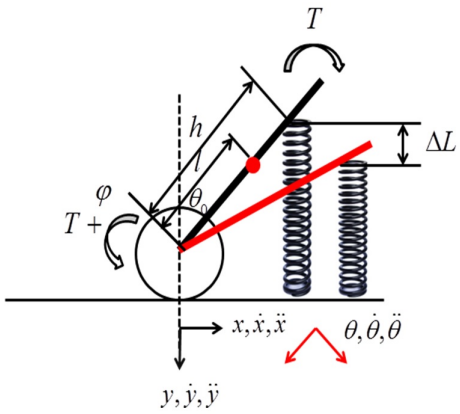


Figure 6. Modeling of spring-attached inverted pendulum

Figure 6 demonstrates the dynamical model of spring-attached inverted pendulum. We first define the coordinate of wheel position x and tilting angle θ. Let the vertical dash line be $\theta = 0^o$, clockwise be the positive direction of θ, counterclockwise be the positive direction of the motor torque T and wheel rotating angle φ. By using Newtons Approach to derive the equations of motion of the model with reference of Section 3.4 [32], we have the second derivative equations of system dynamics:

$$\begin{aligned}\ddot{\varphi} = [&(\alpha + m_p lr\cos\theta)(T_R + T_L) + m_p^2 l^2 rg\sin\theta\cos\theta - m_p lr\alpha\dot{\theta}^2\sin\theta \\ &-m_p lr\sin\theta\cos\theta F_s h - 2b\dot{\varphi}(m_p lr\cos\theta + \alpha)]/[\alpha\beta + m_p^2 l^2 r^2\cos^2\theta]\end{aligned} \tag{2.1}$$

$$\begin{aligned}\ddot{\theta} = [&(\beta - m_p lr\cos\theta)(T_R + T_L) + m_p^2 l^2 r^2\dot{\theta}^2\sin\theta\cos\theta + m_p l\beta g\sin\theta \\ &-\beta\sin\theta F_s h + 2b\dot{\varphi}(m_p lr\cos\theta - \beta)]/[\alpha\beta + m_p^2 l^2 r^2\cos^2\theta]\end{aligned} \tag{2.2}$$

where

$$\alpha = I_p + m_p l^2, \qquad \beta = 2I_w - m_p r^2 - 2m_c r^2$$

Specifically, the spring force is derived by Hooke's law as

$$F_s = k\Delta L + c\dot{\theta} = k(L_0 - L) + c\dot{\theta} = kh(\cos\theta_0 - \cos\theta) + c\dot{\theta}.$$

The notation is given by Table 2.

θ	tilting angle of the pole
φ	rotating angle of the wheel
x	position of the wheel
m_w	mass of the wheel
m_p	mass of the pole
l	length from the center of the pole
r	radius of the wheel
I_w	moment of inertia of the wheel
I_p	moment of inertia of the pole
g	gravity acceleration
F_s	spring force
c	damper coefficient of the spring
k	spring coefficient
h	spring location along the pole
θ_0	threshold angle for activating spring
F_f	friction force with the floor
T_R, T_L	applied torque of right and left wheel
u	control input

Table 2. Notation of the dynamics of wheeled inverted pendulum

The state variables of the system are $[\varphi, \dot{\varphi}, \theta, \dot{\theta}]^T$, and the control input is $u = T_L + T_R$. Table 3 shows the system specification measured from the lastest version of the smartphone robot which is implemented in the MATLAB simulator.

We set a threshold $\theta_0 = \pm 60^o$ of the tilting angle to activate the front and the rear springs. Normally, the spring remains its natural length before the

pole reaches $\pm 60^o$. After the pendulum enters the region where θ is larger than 60^o or smaller than -60^o, the spring begins to compress and generate force.

m_p	0.354 kg
m_w	0.024 kg $\times 2$
l	0.06 m
r	0.03 m
h	0.096 m
I_w	$\frac{1}{2} m_w r^2$ kgm^2
I_p	$\frac{4}{3} m_p l^2$ kgm^2
g	9.8 m/s^2
θ_0	$\pm 60^o$
b	0.0001 kgm^2/s
c	0.05 N^2/s

Table 3. Simulator specification

2.3.2 Control Architecture

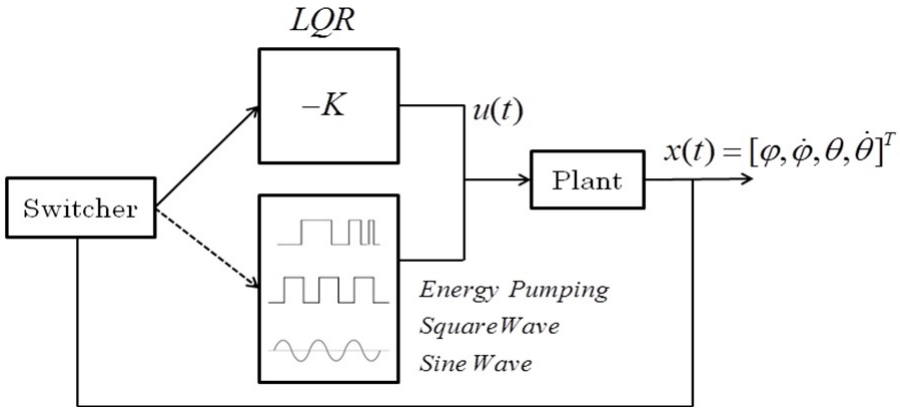


Figure 7. Control Architecture of achieving standing-up and balancing behavior in simulation

Figure 7 illustrates the control framework, where the control input $u(t)$ is the torque applied to the left and right wheel, and the control output $x(t)$ is the state variables based on the system dynamic equations (2.1) and (2.2). The goal of the control system is to stabilize the pole in the upright regions from any initial angle and destabilize it with the spring force when the robot body is in the lower side. The switcher is activated based on the tilting angle of the pendulum thresholded by the experimenter.

2.3.2.1 Stabilizer

To stabilize the pendulum, it is necessary to linearize the system dynamics around the equilibrium point, where θ should be close to zero, suggesting $\sin\theta \approx 0$ and $\cos\theta \approx 1$. If we assume angular velocity $\dot{\theta}$ is small enough, $\dot{\theta}^2 = 0$. The linearization of equation (2.1) and (2.2) are as follows:

$$\ddot{\varphi} = \frac{(\alpha + m_p lr)u + m_p^2 l^2 r g\theta - 2b\dot{\varphi}(m_p lr + \alpha)}{\alpha\beta + m_p^2 l^2 r^2}$$

$$\ddot{\theta} = \frac{(\beta - m_p lr)u + m_p l\beta g\theta + 2b\dot{\varphi}(m_p lr - \beta)}{\alpha\beta + m_p^2 l^2 r^2}$$

The resulting linearized dynamics can be further written into the state space representation and linear feedback control law

$$\dot{x} = Ax + Bu$$

$$u_S = -Kx$$

based on the optimal control theory, where $\dot{x} = \frac{d}{dt}[\varphi, \dot{\varphi}, \theta, \dot{\theta}]$,

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{2b(m_p lr + \alpha)}{\alpha\beta + m_p^2 l^2 r^2} & \frac{m_p^2 l^2 r g}{\alpha\beta + m_p^2 l^2 r^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{2b(m_p lr - \beta)}{\alpha\beta + m_p^2 l^2 r^2} & \frac{m_p g l \beta}{\alpha\beta + m_p^2 l^2 r^2} & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{\alpha + m_p lr}{\alpha\beta + m_p^2 l^2 r^2} \\ 0 \\ \frac{\beta - m_p lr}{\alpha\beta + m_p^2 l^2 r^2} \end{bmatrix}.$$

Note that u_S represents u of stabilizer.

In a Linear Quadratic Regulator (LQR) design of control system, the gain matrix K is found by minimizing a quadratic cost function:

$$J = \int_0^\infty x(t)^T Q x(t) + u(t)^T R u(t) dt$$

where Q and R are symmetric positive matrices to penalize certain states. The choice of Q and R depends on the amount of cost the designer attributes to each element of the state.

To obtain K, a matrix P should be solved according to Riccati Equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

where P is positive definite matrix that always exists when the closed loop system is stable. With the solution of P, K is obtained by

$$K = R^{-1}B^T P.$$

In our case, we set $Q = 0.01 \times I_4$, $R = 500$. Note that I_4 represents a 4 dimensinal identity matrix.

2.3.2.2 Destabilizers

To destabilize the pendulum in the spring-affected region, we adopt several simple controllers, including a positive feedback controller, namely, Energy Pumping controller:

$$u_D = T_{max}\text{sgn}(\dot{\theta})$$

and two time dependent controllers, which are Square wave and Sine wave controllers:

$$u_D = \pm T_{max}\text{sgn}(\sin 2\pi f t)$$

$$u_D = \pm T_{max} \sin 2\pi f t$$

where T_{max} is the maximum torque applied to the wheel and f is the frequency of the control input wave. Sine Wave controller provides smoother control input than Square Wave controller. Note that u_D represents u of destabilizer.

2.3.3 Experiment Results

By setting the boundary of maximum torque, the control input should follow:

$$u_{output} = \begin{cases} T_{max} & (\text{if } u > T_{max}) \\ -T_{max} & (\text{if } u < -T_{max}) \\ u & (\text{otherwise}) \end{cases}$$

The control cycle is 10 sec, with sampling rate 0.01 sec. We calculate settling time for each trial to evaluate the stabilizing behavior. Settling time is the time elasped from the starting time and remained within a specified error band (2% of LQR control domain) with duration for 3 sec. Since the spring is activated when $\theta_0 = \pm 60^o$, the initial state is fixed at $[0, 0, 60^o, 0]^T$.

2.3.3.1 Investigating Maximum Torque

We first show the result using only LQR controller within the entire control range $\theta \in [-60^o, 60^o]$ to reveal the effects of applying different maximum torques, where $u = u_S$ of u_{output}. The spring coefficient is set to 0 N/m.

The control is regarded as "failure" when θ no longer remains in $[-90^o, 90^o]$, in which case the trial ends before the control period terminates, or it fails to stabilize in the upright position within the control cycle.

We assign T_{max} from $0 \sim 0.65$ Nm to the motor and calcuate settling time for each T_{max}. As Figure 6 (up) shows, settling time of small T_{max} is set to 10 sec because trial ends earlier when it is unable to swing up the pendulum. Sometime, failure is caused by large $\dot{\theta}$, where the torque is not large enough to stabilize the factor. This situation happens when $T_{max} = 0.17$ Nm. Failure also causes large deviation of the wheel position. Figure 6 (down) shows the maximum distance of the wheel. The control period is able to be finished until $T_{max} = 0.18$ Nm, where the controller succeeded staying inside the boundary and stabilize the pendulum successfully. With the increase of T_{max}, settling time is reduced. Figure 8 (up) also suggested that 0.6 Nm is the maximum torque for the pendulum to swing up without torque limitation. Figure 8 (down) shows that even it is possible to swing up the pendulum, large torques lead to large displacements of the wheel, which might not be practical in real hardware experiments.

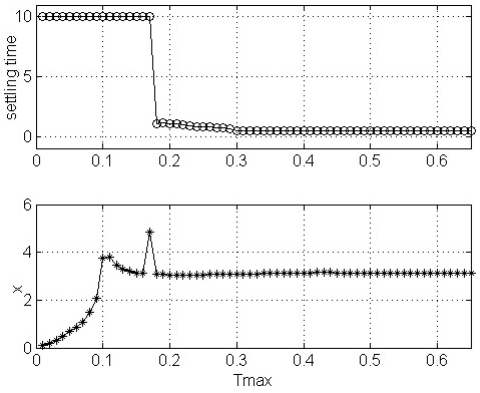


Figure 8. Settling time (up) and maximum horizontal position x(m) (down) for stabilizing with different maximum torque

2.3.3.2 Scanning the Coefficients

Here, we combine the stabilizer and the destabilizer to realize the entire control of standing-up and balancing of the wheeled inverted pendulum model. The threshold of the switcher is intuitively determined and u of u_{output} are defined as:

$$u = \begin{cases} u_S & (\text{if } -12^o < \theta < 12^o) \\ u_D & (\text{if } \theta < -40^o \text{ or } \theta > 40^o) \\ 0 & (\text{otherwise}) \end{cases}$$

We redefine the "failure" based on the spring limitation of its minimum length, where trials end when θ no longer stays in $[-80^o, 80^o]$.

Figure 9 shows the heatmap of settling time based on T_{max} from 0.01 $\sim$ 0.18 Nm and spring coefficient k from 500 $\sim$ 4000 N/m in the case of LQR combined with Energy Pumping control. We choose the upper bound of $T_{max} = 0.18$ Nm based on the result of effective T_{max} in the previous section.

Settling time equals 10 sec (blace) suggests the failures of the system. The whiter the patch suggests shorter settling time and better performances. Failures can be 1) the tiling angle is out of boundary, 2) the controller fails to stabilize in fixed control cycle, or 3) the performance dissatisfies the duration requirement, etc. Figure 9 reveals that small torques can cooperate with most of the springs, meanwhile, larger torques have a spread effect with respect to coefficients of springs. That is, the larger the torque, the wider the choice of effective springs. Failure occurs in the middle of the figure is mostly due to the incapability of LQR to stabilize high angular velocity. In the case of large torque with large spring coefficient, the bouncing time is less, usually once at most.

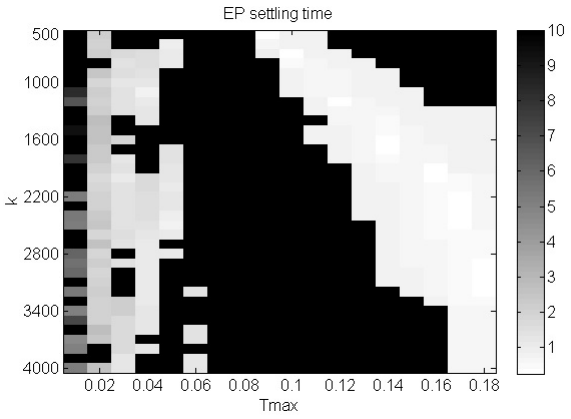


Figure 9. Settling time of LQR + Energy Pumping controller

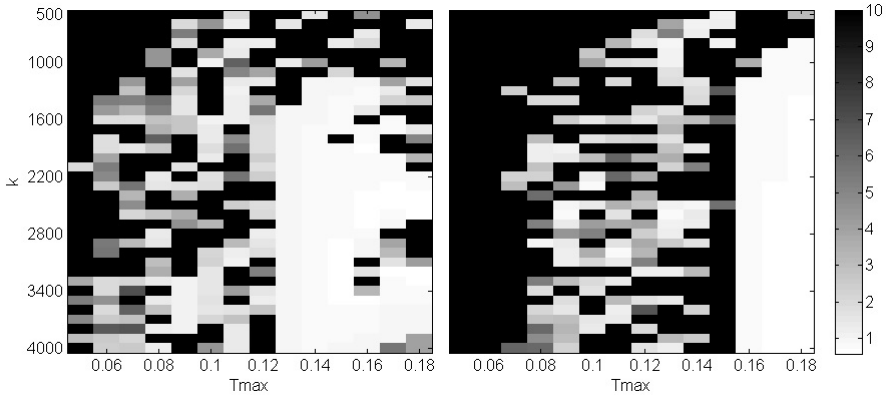


Figure 10. Settling time of LQR + Square Wave controller (left) and LQR + Sine Wave controller (right) when $f = 7$ Hz

In Square and Sine Wave controller cases, we compare two control inputs with the same shape but the opposite sign. We scan several reasonable control frequencies, and found that when $f = 7$ Hz, and input u_D is negative proportional to T_{max}, the successful rate of all the torques is the highest. Figure 10 illustrates both of the cases. The applicable torque starts from 0.05 Nm.

In most of the cases, spring can cooperate with small-torque control input to stabilize the pendulum successfully within duration of 3 sec. Especially, higher torques with higher spring coefficients reach the target in a short time.

Most importantly, the range of torque can be reduced into 0.05 Nm with the reducing rate 72%. Here we examine the relationship between control frequency f and spring coefficient k with fixed $T_{max} = 0.1$ Nm, and u_D is negatively proportional to T_{max}. Spring coefficient k is from $500 \sim 4000$ N/m, while f is from $1 \sim 12$ Hz.

An interesting behavior has been found when $f = 2$ Hz in Figure 11 (left). It succeeded because the control signal remains generating constant torque in a relatively long time, which is large enough to swing up then pendulum. In Sine Wave case Figure 11 (right), this would not happen due to the rapid change of the control input. The main difference between Square Wave and Sine Wave controllers is the changing rate, which highly depends on the type and performance of the motor. Figure 9 shows that both lower and higher control frequencies are not applicable to activate the spring.

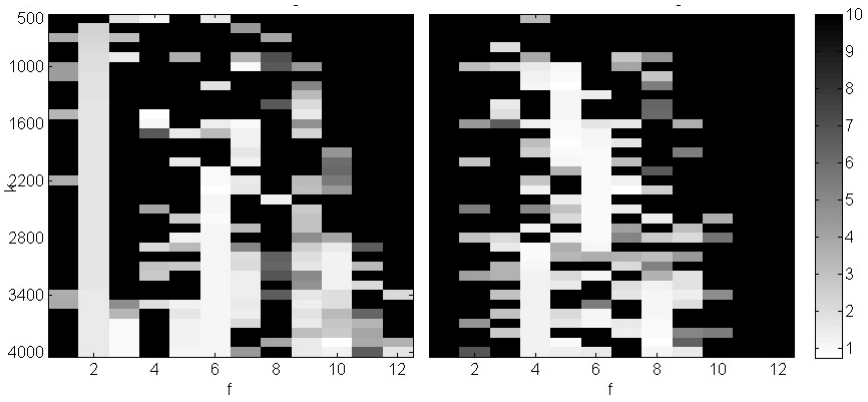


Figure 11. Settling time of LQR + Square Wave controller (left) and LQR + Sine Wave controller (right) when $T_{max} = 0.1$ Nm

Finally we show one specific control result in Figure 12 with $f = 9$ Hz, $k = 1000$ N/m, and $T_{max} = 0.1$ Nm. The settling time is 4.46 sec. The unit of theta and phi are degree. When we convert phi to x, we found the wheel position deviation is reduced.

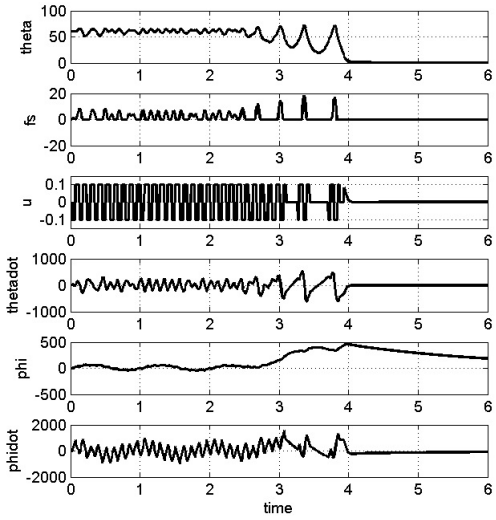


Figure 12. Specific control result

2.3.4 Section Summary

This section presented a new hardware model a wheeled inverted pendulum with elastic bumpers and a switching control system to solve the limited motor torque problem in real construction of the smartphone balancer. Dynamic equations have been derived and realistic robot specification is implemented in a MATLAB simulator. We adopt three basic destabilizing controllers to activate the spring. All of them succeeded achieving the required behaviors with spring coefficients from 500 $\sim$ 4000 N/m. We investigated the relation between appropriate maximum torque and spring coefficients with fixed control frequency, and relation between control frequency and spring coefficients with fixed maximum torque. The results showed that appropriate frequency setting can benefit the performance of Square Wave and Sine Wave controllers. In the fixed T_{max} case, we found most springs work along with the controller when f is from 4 $\sim$ 10 Hz, with the best performance of $f = 2$ Hz in Square Wave controller. By traversing frequencies from 4 $\sim$ 10 Hz, we also found $f = 7$ Hz has the highest successful rate for most maximum torques from 0.05 $\sim$ 0.18 Nm. By comparison of the results of single LQR stabilizer and our proposed control architecture, it is revealed that standing-up can be achieved by smaller maximum motor torque. Hence, useful hardware

information is provided for the construction of the actual robot.

2.4 Hardware Experiment

2.4.1 Control Architecture

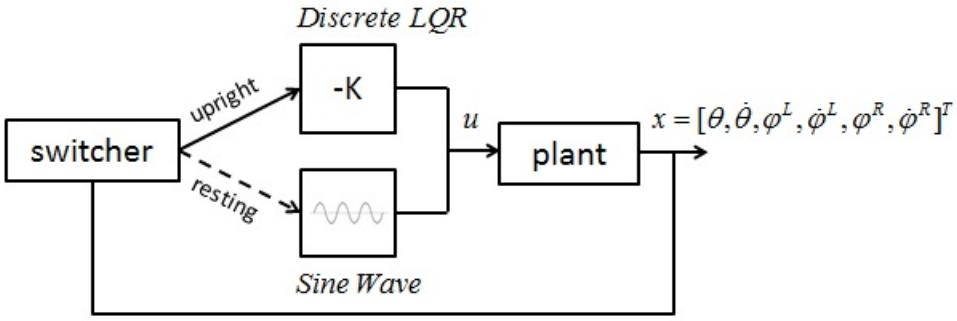


Figure 13. Control Architecture of achieving standing-up and balancing behavior in hardware

Figure 13 shows the control architecture used for real hardware implementation of the smartphone balancer. In the real smartphone robot system, the derivatives of state variables become noisier, especially the second derivatives of angular velocities. To avoid this, we use a discrete LQR as the stabilizer, which regards the system dynamics as

$$x_{t+1} = Ax_t + Bu_t.$$

where t is the time step in the time sequence with the length of T, namely $t \in \{1, ...t, t+1, ..., T\}$. We adopt only the Sine Wave controller among the destabilizers in the previous section. Here, we consider the wheel rotating information seperately in the state variables, therefore the state variables are defined as $x = [\theta, \dot{\theta}, \varphi_L, \varphi_R, \dot{\varphi}_L, \dot{\varphi}_R]^T$, and the control input which is the motor torque applied to the wheels is defined as:

$$u = \begin{cases} -Kx & (\text{if } -\theta_{thre} < \theta < \theta_{thre}) \\ p \sin 2\pi f t & (\text{otherwise}) \end{cases}$$

where p is the amplitude of the command to the wheel motor, and f is the frequency of the Sine Wave.

2.4.1.1 Obtain System Specification

In the real robot system, we also face the problems of unknown physical specifications such as the position of the center gravity, moment of inertia of the wheel and pendulum, friction coefficient of the floor, firction of the wheel axle, and torque constant of the motor, resulting in the difficulties to obtain precise system dynamic matrices.

Here we use one of the maching leanring technique, Least Square method to estimate the system matrices through the data collected from the real robotic experiments. Based on the system dynamics under the framework of discrete-time LQR, we rewrite the matrix into the form

$$x_{t+1} = CZ_t = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix}.$$

Here we define

$$X_T = [x_2, x_3, ..., x_T]$$

$$Z_{T-1} = \begin{bmatrix} x_1, & x_2, & ... & , x_{T-1} \\ u_1, & u_2, & ... & , u_{T-1} \end{bmatrix}$$

as the data can be obtained from the real hardware experiment, and the estimated system matrices are

$$\hat{C} = X_T Z_{T-1}^T (Z_{T-1} Z_{T-1}^T)^{-1}. \tag{2.3}$$

Note that the accuracy of the estimation is based on the size of the dataset and the spectrum of the visited states.

2.4.2 Experiment Results

2.4.2.1 Balancing

We developed a hand-tuning friendly interface in the Android system of the smartphone. As shown in Figure 14, the first seven rows are the state variables, control signal and sampling rate. The next four rows show the control gain parameters for each state, which can be roughly tuned by the yellow seekbar right below. The button "Reset" is used to reset all the state variables to 0, the button "Stop/Start" terminates connection between the phone and the motor, and the button "Default" is to set the control parameter back to the initial values. The rest of the buttons are used to mirco tuning of the control parameter in 100 or 10 magnitudes.

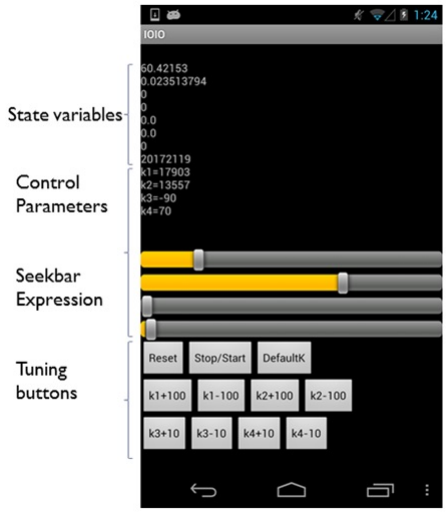


Figure 14. Control gain tuning UI

We randomly choose four sets of hand-tuned control parameters based on the observation to generate behaviors and collect the data. For each set we collect

a 5×1000 dataset for four times, the five features are $x = [\theta, \dot{\theta}, \dot{\varphi}_L, \dot{\varphi}_R]^T$ and u. We eliminate two variables φ_L and φ_R from the original state variables because of the drifting effect of the rotary encoder. Based on (2.3), we obtain $\hat{C}$ containing the information of the system identification matrices as follows:

$$A = \begin{bmatrix} 0.7278 & 0.6710 & -0.0013 & -4.0002e-4 \\ -0.0637 & 0.9080 & 0.0013 & 9.4295e-5 \\ -1.1297 & 5.5689 & 0.8299 & -0.0015 \\ -0.2241 & 0.3682 & 0.0018 & 0.9998 \end{bmatrix},$$

$$B = \begin{bmatrix} -2.7714e-5 \\ -5.8137e-7 \\ -5.1194e-6 \\ -8.0359e-7 \end{bmatrix}.$$

We use dlqr command provided in MATLAB to calculate the control gain vector based on the system identification matrices where

$$K = [22609, 17288, -99, -397].$$

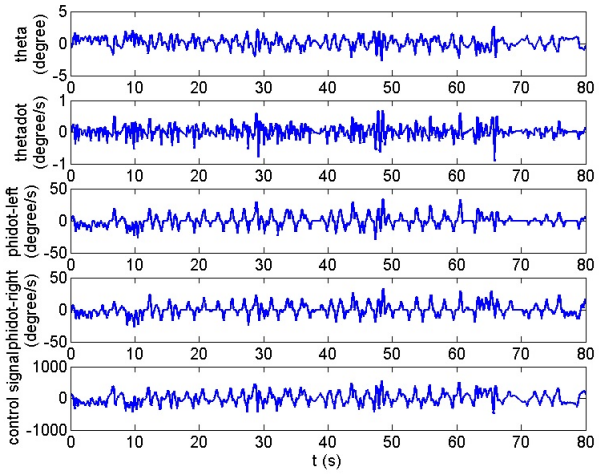


Figure 15. Dynamics of stabilizing behavior in hardware

Note that the angular information is translated in radius, and the amplitude of the command to the wheel motor is from $0 \sim 1000$. Figure 15 shows the results of the balancing behavior. We intercept 4000 time step of the data with the sampling rate of 20 ms, suggesting 80 sec sustainable balance. Note that all the variables are not strictly stabilized to 0^o equilibrium due to the sensitivity of the sensor, however, the robot is able to keep its tilting angle inside $[-2.6301^o, 2.6942^o]$.

2.4.2.2 Standing-up and Balancing

Based on the results in the previous section, we know that appropriate combination of spring, maximum torque, control wave frequency and threshold of switcher lead to the success of standing-up and balancing behavior. Here, we use two sets of springs specifying 0.3548 N/mm and 0.8869 N/mm with initial angles $\theta_0 = 20^o$ and $\theta_0 = 30^o$. We hand-tune the parameters of the switcher threshold θ_{thre} and the frequency of control signal f with fixed amplitude of the command $p = 1000$ applied to the wheel motor. We run 10 trials for each set of parameters with each trial lasting 1 min, and record the success rate of achieving the behavior with duration at least 5 sec. Table 4 reveals the results.

For spring coefficient $k = 0.3548$ N/mm

θ_0	20^o		30^o		
θ_{thre}	15^o	20^o	20^o	25^o	30^o
$f = 3$ Hz	4^*	2	0	1	2
$f = 4$ Hz	1	1	0	2	1
$f = 5$ Hz	0	3	0	0	0

For spring coefficient $k = 0.8869$ N/mm

θ_0	20^o		30^o		
θ_{thre}	15^o	20^o	20^o	25^o	30^o
$f = 3$ Hz	1	0	3	4^*	1
$f = 4$ Hz	2	1	2	1	1
$f = 5$ Hz	0	0	1	1	1

Table 4. Success rates of hardware parameter combinations

The results suggest that the spring with smaller coefficient performs better with the smaller initial resting angle. $f = 3$ Hz is the best control signal frequency among the other two. The threshold of switcher should be set lower than the initial resting angle, indicating that the destabilizer should start working before the spring arms touch the ground. $\theta_0 = 20^o$, $\theta_{thre} = 15^o$, and $f = 3$ Hz is the best combination of parameters using the spring with $k = 0.3548$ N/mm, and $\theta_0 = 30^o$, $\theta_{thre} = 25^o$, and $f = 3$ Hz is the best with $k = 0.8869$ N/mm. The overall success rate is relatively low and sloppy, due to the various failures in practice. For example, 1) the spring is too weak to bounce up the robot body, 2) LQR fails to stabilize in the upright position because the angular velocity is too high, even the spring bounced well, or 3) time goes out when it tried hard to bounce, etc. Figure 16 shows the angular and control information of a successful behavior with the parameter $\theta_0 = 20^o$, $\theta_{thre} = 15^o$, $f = 3$ Hz, and $k = 0.3548$ N/mm. Yellow bar areas illustrate the periods that the controller is switched to destabilizing mode.

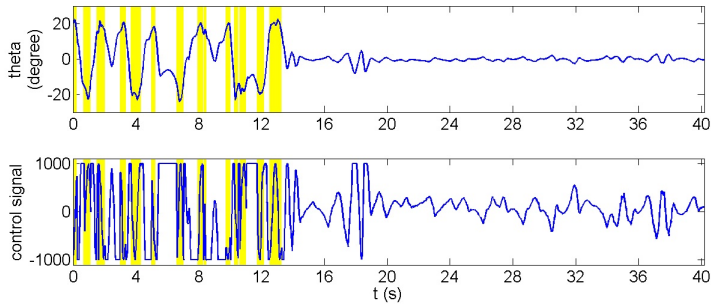


Figure 16. Dynamics of a successful standing-up and balancing behavior in hardware

2.4.3 Section Summary

This section tested the proposed switching control scheme in the real smartphone balancer system, and successfully achieved the standing-up and balancing behavior based on hand-tuned hardware parameter combinations obtained from the simulation results of previous section. The hardware parameters include initial resting angle, spring coefficient, maximum torque,

control signal frequency and the switching threshold. To reduce the influence of system noise and simplify the control, we adopted a discrete-time LQR framework as the stabilizer, and chose only the Sine Wave controller as the destabilizer. We created a control gain tuning Android application to collect random data of the dummy balancing behavior, and used least square method to obtain the system identification matrices. We tested several sets of hardware parameters, and it suggested successful achievement of the standing-up and balancing behavior although with relatively low success rate. The results showed that the spring with smaller coefficient performs well with smaller initial resting angle, and the switching threshold lower than the initial angle is necessary.

2.5 Chapter Summary

In this chapter, we focus on designing the prototype of a smartphone robot, namely a balancer based on the Android platform. We first illustrated the manual to build a real smartphone balancer, containing the hardware components, chassis design, connections, and software developing environment. We then proposed an elastic bumper attached wheeled inverted pendulum model to represent the dynamics of the smartphone balancer which enables various lively behaviors, including standing-up and balancing. We derived the equation of motion of the model and constructed a numerical simulator to investigate combinations of different hardware settings to accomplish the behavior. The hardware concern covers the spring coefficient, maximum motor torque, control signal frequency, and initial resting angle, etc. We also proposed a switching control scheme to realize the behavior, that is to stabilize the balancer in the upright equilibrium and destabilize it in the resting regions. The simulation results showed that with the elastic bumper, it is possible to achieve the behavior with smaller motor torque, and suggested wide choices of springs, and destabilizing controllers. We implemented the control architecture in the real robot system with several combinations of the hardware setting tuned by hand, and verified the feasibility of achieving the target behaviors. Hand-tuned parameters evinced successful behavior, however, were unable to guarantee the optimum. To further encourage the balancer to become an environment adaptive agent, we would like to in-

troduce intelligence of learning for the robot to search for optimal policy parameters.

Chapter 3

Policy-based Reinforcement Learning

3.1 Introduction

Reinforcement learning (RL) is likely a prototype of living organisms adapting to the unknown circumstances with trial-and-error strategies. The RL agent optimizes its policy in real time by experiencing different states and actions. Rather than traditional robotic research which assumes well-known environment, and requires sophisticated behavior design and tedious controller programming, robotic research under reinforcement learning framework has its flexibility and significance of intelligence. It provides a principled way to build agents whose actions are guided by a set of internal drives, and a sound theoretical basis that allows for handling stochastic environments and rewards that take multiple steps to obtain.

Classic RL considers learning with a value function, which is associated with the current state or the state-action pair and the expected sum of future rewards with respect to particular policies. The value function is used to evaluate the quality of an executing action in a certain state among the total state-action spaces. This quality evaluation is later on utilized to compute the policies by action selection or to update the policy. However, this approach requires a complete state-action space filling in with the data, causing a huge demand of storage and computational resources in the high-dimensional continuous states and actions robotic problems. Additionally, the value function is computed iteratively by bootstrapping, which often leads to a bias in the quality assessment of the state-action pairs when applied function approximation techniques in continuous state spaces.

As an alternative, policy search methods, which directly search for optima in the policy parameter space by examining each parameter's performances from objective functions usually an expectation of sumation of future reward, are more adoptable. In this chapter, we review two trends of policy search methods, namely policy gradient estimation and EM-based policy search, and propose a novel model-free policy search method called EM-based Policy Hyperparameter Exploration (EPHE) inspired from the merits of both trends.

Classic policy gradient methods such as REINFORCE [5] and GPOMDP [6] suffer from high variance in the gradient estimates due to the noise injected

at every time step for a stochastic policy even if the optimal baseline is subtracted [34]. Policy Gradients with Parameter-based Exploration (PGPE) [7] method addressed this problem by evaluating deterministic policies with the parameters sampled from a prior distribution defined by policy hyperparameters. The use of deterministic policy reduces the variance of the performance gradient with respect to the hyper parameters and the deterministic policy does not need to be differentiable. Natural Evolution Strategy (NES) [8] method considers the natural gradient over the plain gradient to search in a higher expected evaluations. However, these approaches update the hyperparameters with gradient ascent of the estimated gradient requiring learning rate tuning.

Another trend is EM-based policy search methods based on closed-form maximization of the lower bound of the objective function with respect to the parameters of exponential family distributions. Those methods are realized as simple reward-weighted updating rule and do not require gradient computation and learning rate parameter tuning. Such methods include EM-inspired Reward-Weighted Regression (RWR) [9], EM Policy learning by Weighting Exploration with the Returns (PoWER) [10] and Fitness Expectation Maximization (FEM) [11]. RWR first introduced the framework of EM-based reinforcement learning and reduced the problem of learning with immediate rewards to a reward-weighted regression problem with an adaptive reward transformation for faster convergence [9]. However RWR considers a stochastic policy using additive Gaussian noise to the action, which could have problems of the perturbations being averaged out and the risk of damage to a hardware system with high-frequency noise. PoWER considers state-dependent exploration to address these problems and adjusts the exploration to stepwise, episode-wise and even through a slowly varying form [10]. FEM inherited the same EM-based policy search framework, and considers a rank-based return transformation function. It also uses an online mechanism to update policy sample by sample than batch by batch. A forgetting factor was introduced to modulate the speed at which the search policy adapts to the current sample [11]. REPS [35] has the similar idea of EM-based methods for bounding two distributions. The difference is, it bounds the information loss measured by relative entropy between the observed data distribution and the data distribution generated by the new policy to update the policy parameters.

Other state-of-the-art policy search methods include Cross Entropy Methods (CEM) [12], Covariance Matrix Adaptation Evolutionary Strategy (CMAES) [36], Policy Improvement with Path Integral (PI^2) [37] and PI^2-CMAES [38]. CEM and CMAES explore in the policy parameter space directly, and they both update mean and covariance matrix of a multivariate Gaussian search distribution in a weighting scheme. CEM updates the diagonal covariance while CMAES updates the full covariance matrix through incremental adaption along evolutionary paths. PI^2 is a probability weighting method derived from the first principles of optimal control, it perturbs the parameter and collects rewards at every time step during exploratory policy execution, and updates the policy parameter weighted by the probability of the rewards. PI^2-CMAES improved PI^2 by updating the full covariance matrix.

Our proposed method EPHE combines the ideas of PGPE to evaluate a deterministic policy in individual episodes with policy parameters sampled from a prior distrubition, and of EM-based framework to update the hyperparameters in a reward-weighting way. Heuristically, we propose an elite selection mechanism (EPHE-K) to preserve informative samples for hyperparameter updating, which is selecting the K best parameters based on the return history and discarding the rest of the sampled parameters. EPHE-K is tested in the benchmarks of pendulum swing-up task, cart-pole balancing task and simulator of our smartphone balancer task with a non-linear, non-differentiable controller. The experimental results show that EPHE-K outperformed other policy gradient methods such as Finite Difference and PGPE with the best learning rate. There is no significant difference between different setting of K, however, agents without the selection mechanism indicate much worse performances.

EPHE-K is similar to CEM and FEM, and is equivalent to episodic PoWER and PI^2 [38] [39].

Concerning the weighting scheme for each rollout, RWR, PoWER, FEM, and EPHE-K require the returns to be nonnegative and summed up to constants to implicitly resemble a proper distribution. PoWER implicitly realized the discarding rule, and FEM and EPHE-K discard samples below a fixed baseline determined by users, which is similar to CEM. CMAES reweighted the samples similarly by truncating the sample size and a transformation to a convex shape. REPS reweighted the samples in an exponential transforma-

tion of the corresponding return.

Hereafter, we propose an adaptive baseine based on EPHE (EPHE-AB) to discard worse samples below the average of the reward history in every hyperparameter updating, and further examine the learning performances of different baselines containing the mean, and one and two standard deviations from the mean. We implement EPHE-AB and evaluate several baseline functions in three simulation benchmarks. The results show that the adaptive baseline methods improved the performance over previous policy search methods including PGPE, NES and FEM, and outperform the CMAES weighting scheme and REPS weighting scheme. We further implement EPHE-AB-mean in the real smartphone balancer system in a standing-up and balancing task, and a view-based approaching task. Previous method like PoWER was implemented in a real robot arm for playing a ball-in-a-cup task using imitation learning for parameter initialization [10]. In our smartphone robot experiment of learning to stand-up and balance and learning to approach a visual target, the hyperparameters are learned from scratch, and the robustness against the variability of initial states is required. The results show that the adaptive baseline method with the mean successfully achieved the two behaviors in hardware experiments. The analyses of the number of discarded samples and the distribution of returns show that the choice of the baseline by the mean causes a steady decrease of the number of discarded samples with learning and a capability to select positive outliers in the early learning stage.

The related methods are illustrated in Section 3.1. EPHE-K and its implementation are discussed in Section 3.2. EPHE-AB and its implementation are discussed in Section 3.3. The summary of the chapter is in Section 3.4.

3.2 Related Methods

3.2.1 Problem Setting

Markov Decision Process is a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker widely adopted in reinforcement learning. Here we consider a discrete-time Markov Decision Process, defined as a 4-tuple (X, U, P, R), where

X: a finite set of states
U: a finite set of actions
$P_{xu}(\cdot)$: the state transition probability
$R_{xu}(\cdot)$: the reward function based on the states and actions.

At each time step t, the agent takes action u_t based on current state x_t according to policy $\pi(u_t|x_t, \theta)$ parameterized by vector θ. The dynamic environment makes a transition to next state x_{t+1} according to $p(x_{t+1}|x_t, u_t)$ and gives a scalar reward r_t. The history trajectory is denoted by a state-action-reward sequence as $h = [x_1, u_1, r_1, ..., x_t, u_t, r_t, x_{t+1}, ..., x_T, u_T, r_T]$ where T is the time length.

The goal of policy-based reinforcement learning is to find parameter θ that maximize an objective function defined as the agent's expected sum of reward:

$$J(\theta) = \int_H p(h|\theta)R(h)dh \tag{3.1}$$

where $R(h)$ is the return defined as the cumulative reward of sequence h and $p(h|\theta)$ is the probability of observing h. Under the Markovian environmental assumption, the probability of sequence $p(h|\theta)$ for a stochastic policy is given by:

$$p(h|\theta) = p(x_1)\prod_{t=1}^{T} p(x_{t+1}|x_t, u_t)\pi(u_t|x_t, \theta) \tag{3.2}$$

where $p(x_1)$ is the initial state distribution.

If the policy is deterministic, denoted by $u_t = \pi(x_t, \theta)$,

$$p(h|\theta) = p(x_1)\prod_{t=1}^{T} p(x_{t+1}|x_t, u_t)\pi(x_t, \theta).$$

We consider episode-wise parameter updating over step-wise in this chapter.

3.2.2 Policy Gradient Methods

Policy gradient methods use gradient ascent to update policy parameters based on the maximization of the objective function. The gradient is defined as $\nabla_\theta J(\theta)$ pointing the direction of the steepest ascent of the expected return. The policy gradient is given by

$$\nabla_\theta J(\theta) = \int_H \nabla_\theta p(h|\theta) R(h) dh. \tag{3.3}$$

The update is given by

$$\theta_{new} = \theta + \alpha \nabla_\theta J(\theta)$$

where α is the learning rate determined by the experimenter.

3.2.2.1 Finite Difference

The Finite Difference method (FD) [40] estimating the gradient by applying small perturbations $\delta\theta$ to the parameter vector θ is the simplest policy gradient method. The perturbation performs as the exploration strategy in parameter space, and can be defined as different distributions applied to each parameter separately or together. The gradient can generally be obtained in least-square regression sense

$$\nabla_\theta^{FD} J(\theta) = (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta^T \Delta R$$

where $\Delta\Theta = [\delta\theta_1, ..., \delta\theta_N]^T$, $\Delta R = [\delta R_1, ..., \delta R_N]$, for each $\delta R = R(\theta + \delta\theta) - R(\theta)$. Note that N is the number of collected trajectories.

Algorithm 1: Finite Difference

Input : Initialize policy parameter θ

repeat
- Perform N episodes and sample N trajectories:
  - for each episode n draw $\theta^n = \theta + \delta\theta$
  - collect $H = \{h^n\}_{n=1...N}$
  - evaluate $R(h^n)$
- Calculate $\nabla_\theta^{FD} J(\theta)$
- Update $\theta_{new} = \theta + \alpha \nabla_\theta^{FD} J(\theta)$

until *Convergence*;

3.2.2.2 REINFORCE

REINFORCE [5] is the most representative one among the likelihood-ratio methods, which utilizes "likelihood-ratio" trick given by the identity

$$\nabla_\theta p(h|\theta) = p(h|\theta) \nabla_\theta \log p(h|\theta).$$

By inserting the trick into (3.3), we have

$$\nabla_\theta J(\theta) = \int_H p(h|\theta)\nabla_\theta \log p(h|\theta)R(h)dh \quad (3.4)$$

$$= E_{p(h|\theta)}\left[\nabla_\theta \log p(h|\theta)R(h)\right]. \quad (3.5)$$

Substituting (3.4) with (3.2),we have

$$\nabla_\theta J(\theta) = \int_H p(h|\theta) \sum_{t=1}^{T} \nabla_\theta \log \pi(u_t|x_t,\theta)R(h)dh.$$

Although it is not practical to integrate over the entire space of histories, we can use sampling to obtain the estimate of the gradient

$$\nabla_\theta^{RF} J(\theta) \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T} \nabla_\theta \log \pi(u_t^n|x_t^n,\theta)(R(h^n) - b).$$

b is a reward baseline to reduce the variance of the gradient. The optimal baseline is given by

$$b^{RF} = \frac{\sum_{n=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi(u_t^n|x_t^n,\theta)\right)^2 R(h^n)}{\sum_{n=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi(u_t^n|x_t^n,\theta)\right)^2}.$$

Note that the policy $\pi(u_t|x_t,\theta)$ in REINFORCE should be differentiable.

3.2.2.3 PGPE

The problems of REINFORCE are that the policy has to be differentiable with respect to the policy parameters and that evaluation of a stochastic

Algorithm 2: REINFORCE

Input : Initialize policy parameter θ

repeat
  Perform N episodes and sample N trajectories:
    for each episode n draw $\theta^n \sim p(\cdot)$
    collect $H = \{h^n\}_{n=1...N}$
    evaluate $R(h^n)$
  Calculate baseline b^{RF}
  Calculate $\nabla_\theta^{RF} J(\theta)$
  Update $\theta_{new} = \theta + \alpha \nabla_\theta^{RF} J(\theta)$
until *Convergence*;

policy can lead to a high variance in the entire histories. Policy Gradients with Parameter-based Exploration (PGPE) [7] addressed these problems by considering a distribution of deterministic policies with the policy parameter θ sampled from a prior distributuion defined by the hyperparameters ρ, which are typically the mean and the variance of θ. The objective function is given by

$$J(\theta) = \int_\Theta \int_H p(h|\theta)p(\theta|\rho)R(h)dhd\theta \tag{3.6}$$

By updating the hyperparameter vector ρ, we can obtain the deterministic policy $\pi(u_t|x_t, \theta)$ where θ is eventually computed by the expectation of the prior distribution. Note that the variance of $p(h|\theta)$ of PGPE can be kept small because a deterministic policy is adopted.

Differentiating (3.6) with respect to ρ by using "likelihood-ratio" trick gives us

$$\nabla_\rho J(\rho) = \int_\Theta \int_H p(h|\theta)p(\theta|\rho)\nabla_\rho \log p(\theta|\rho)R(h)dhd\theta.$$

Again by sampling, we have the gradient estimate

$$\nabla_{\rho}^{PGPE} J(\rho) \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_{\rho} \log p(\theta|\rho)(R(h^n) - b).$$

The optimal baseline is given by

$$b^{PGPE} = \frac{\sum_{n=1}^{N} \left(\nabla_{\rho} \log p(\theta|\rho)\right)^2 R(h^n)}{\sum_{n=1}^{N} \left(\nabla_{\rho} \log p(\theta|\rho)\right)^2}.$$

Algorithm 3: Policy Gradients with Parameter-based Exploration

Input : Initialize policy hyperparameter ρ

repeat
- Perform N episodes and sample N trajectories:
  - for each episode n draw $\theta^n \sim p(\theta|\rho)$
  - collect $H = \{h^n\}_{n=1...N}$
  - evaluate $R(h^n)$
- Calculate baseline b^{PGPE}
- Calculate $\nabla_{\rho}^{PGPE} J(\rho)$
- Update $\rho_{new} = \rho + \alpha \nabla_{\rho}^{PGPE} J(\rho)$

until *Convergence*;

3.2.2.4 NES

Natural Evolution Strategies (NES) [8] is policy gradient method using the natural gradient to update the parameterized search distribution in the direction of higher expected fitness. The intuition of the natural gradients is to limit the distance between two subsequent distributions. Other than the Euclidian metric of $\Delta\theta^T \Delta\theta$, we can use Kullback-Leibler (KL) divergence to measure the closeness, which can be approximated by the second-order Taylor expansion as follows

$$\mathbf{KL}(p(h|\theta)||p(h|\theta + \Delta\theta)) \approx \Delta\theta^T \boldsymbol{F}_{\theta} \Delta\theta \leq \epsilon.$$

where the Fisher information matrix can be obtained by

$$\begin{aligned} \boldsymbol{F}_\theta &= E_{p(h|\theta)}\left[\nabla_\theta \log p(h|\theta) \nabla_\theta \log p(h|\theta)^T\right] \\ &= \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \log p(h|\theta) \nabla_\theta \log p(h|\theta)^T \end{aligned}$$

The natural policy gradient is therefore given by

$$\nabla_\theta^{NES} J(\theta) = \boldsymbol{F}_\theta^{-1} \nabla_\theta J(\theta).$$

Algorithm 4: Natural Evolution Strategies

Input : Initialize policy parameter θ

repeat
- Perform N episodes and sample N trajectories:
  - for each episode n draw $\theta^n \sim p(\cdot)$
  - collect $H = \{h^n\}_{n=1\ldots N}$
  - evaluate $R(h^n)$
  - calculate $\nabla_\theta \log p(h|\theta)$
- Calculate $\nabla_\theta J = \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta \log p(h|\theta) R(h^n)$
- Calculate $\boldsymbol{F}_\theta$
- Update $\theta_{new} = \theta + \alpha \nabla_\theta^{NES} J(\theta)$

until *Convergence*;

3.2.3 EM-based Policy Search

EM-based Policy Search [9] estimates a lower bound of the expected return from histories and iteratively updates the policy parameter using an analytic solution for the maximum of the lower bound. This way, there is no learning rate parameter tuning compared with policy gradient methods. The original

idea of expectation-maximization (EM) is developed by Dempster et al [41]. We match the current policy parameters θ with a new policy parameterized by θ'. By assuming the return $R(h)$ is nonnegative, Jensen's inequality [42] yields the following bound of the log-expected return

$$\log J(\theta') \geq \int_H \frac{R(h)p(h|\theta)}{J(\theta)} \log \frac{p(h|\theta')}{p(h|\theta)} dh + \log J(\theta) \equiv \log J_L(\theta').$$

We defined $J_L(\theta')$ as the lower bound and its log-derivative is

$$\nabla_{\theta'} \log J_L(\theta') = \int_H p(h|\theta) R(h) \nabla_{\theta'} \log p(h|\theta') dh \tag{3.7}$$

$$\approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \nabla_{\theta'} \log \pi(u_t^n | x_t^n, \theta') R(h^n) \tag{3.8}$$

since $J(\theta)$ is constant. By solving the log-derivative of the lower bound equals 0, we can obtain the new policy parameter θ'.

3.2.3.1 RWR

Reward-Weighted Regression [9] considered a stochastic policy using additive exploration ϵ as Gaussian noise. The policy action is defined as

$$u = \theta^T \Phi(x) + \epsilon,\ \epsilon \sim \mathcal{N}(0, \Sigma)$$

where $\Phi(x)$ is the basis functions representing the states. By solving equation (3.7), it naturally yields a weighted regression method with the return as weights.

For each policy parameter θ_i,

$$\theta_{inew} = (\boldsymbol{\Phi}_i^T \boldsymbol{R} \boldsymbol{\Phi}_i)^{-1} \boldsymbol{\Phi}_i^T \boldsymbol{R} \boldsymbol{U}_i,$$

where

$$\boldsymbol{\Phi}_i = \left[\Phi_1^1, ..., \Phi_T^1, \Phi_1^2, ..., \Phi_T^2, ..., \Phi_1^N, ..., \Phi_T^N\right]^T,$$

$$\boldsymbol{U}_i = \left[u_1^1, ..., u_T^1, u_1^2, ..., u_T^2, ..., u_1^N, ..., u_T^N\right]^T,$$

$$\boldsymbol{R}_i = \text{diag}\left(R_1^1, ..., R_T^1, R_1^2, ..., R_T^2, ..., R_1^N, ..., R_T^N\right)^T.$$

Note that N is the number of sampled trajectories, and T is the time length for one episode.

Algorithm 5: episodic Reward Weighted Regression

Input : Initialize policy parameter θ

repeat
- Perform N episodes and sample N trajectories:
  - for each time step t, draw $u_t = \theta^T \Phi(x_t) + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \Sigma)$
  - collect $H = \{h^n\}_{n=1...N}$
  - evaluate $R(h^n)$
- Update $\theta_{new} = (\boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{R} \boldsymbol{U}$

until *Convergence*;

3.2.3.2 PoWER

The unstructured exploration of the policy at each step of eRWR has several disadvantages 1) it causes a large variance in parameter updates that grows with the number of time-steps, 2) it perturbs actions too frequently as the system acts as a low pass filter, and the perturbations average out, thus their effect is washed out, and 3) it can damage the system executing the trajectory. To address these problems, Policy learning by Weighting Exploration with the Returns (PoWER) [10] considers a structured state-dependent exploration to improve eRWR.

The policy is defined as

$$u = (\theta + \epsilon)^T \Phi(x),$$
$$\epsilon^T \Phi(x) \sim \mathcal{N}(0, \Phi(x)^T \Sigma \Phi(x)).$$

Inserting the policy into (3.7), the updating rule is derived as

$$\theta_{new} = \theta + \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} W(x) R(h) \right)^{-1} \left(\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} W(x) \epsilon R(h) \right)$$
$$\Sigma_{new} = \frac{\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} \epsilon_t \epsilon_t^T R(h)}{\frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} R(h)},$$

where $W(x) = \Phi(x)\Phi(x)^T (\Phi(x)^T \Sigma \Phi(x))^{-1}$.

Algorithm 6: Policy learning by Weighting Exploration with the Returns

Input : Initialize policy parameter θ

repeat
- Perform N episodes and sample N trajectories:
  - for each time step t, draw $u_t = (\theta + \epsilon_t)^T \Phi(x_t)$ with
  - $\epsilon_t^T \Phi(x_t) \sim \mathcal{N}(0, \Phi(x_t)^T \Sigma \Phi(x_t))$
  - collect $H = \{h^n\}_{n=1...N}$
  - evaluate $R(h^n)$
- Compute importance weights and reweight rollouts, discard low-importance rollouts
- Update $\theta_{new}, \Sigma_{new}$

until *Convergence*;

3.2.3.3 FEM

Fitness Expectation Maximization (FEM) [11] inherited the same EM-base policy search framework, and considers a simple rank-based utility transformation function (regarded as the return transformation function), which first ranks all samples based on fitness value, then assigns 0 to the $N - m$ worst ones and assigns values linearly from 0 to 1 to the m best samples. It also uses an online mechanism to update policy sample by sample than batch by batch. A forget factor β is introduced to modulate the speed at which the search policy adapts to the current sample. Batch size N is only used for utility ranking function, same as $R(t)$ which ranks the current sample among the N last seen samples.

Algorithm 7: Fitness Expectation Maximization

Input : Initialize policy parameter θ

repeat

  For each time step t, draw sample $u_t = (\theta + \epsilon_t)^T \Phi(x_t)$ with

    $\epsilon_t^T \Phi(x_t) \sim \mathcal{N}(0, \Phi(x_t)^T \Sigma \Phi(x_t))$

  evaluate fitness $R(x_t, u_t)$

  compute rank-based fitness shaping

    $F_t = F(R(x_t, u_t) | R(x_{t-1}, u_{t-1}), ..., R(x_{t-N}, u_{t-N}))$

  update

    $\theta_{new} = (1 - \beta F_k)\theta + \beta F_k(\theta + \epsilon_t)$

    $\Sigma_{new} = (1 - \beta F_k)\Sigma + \beta F_k \epsilon_t \epsilon_t^T$

until *Convergence*;

3.3 EPHE-K

3.3.1 Method

Here, we describe our proposed method, EM-based Policy Hyperparameter Exploration (EPHE) by integrating the features of PGPE and EM-based Policy Search. To establish the lower bound, we consider a new parameter distribution over hyperparameter vector ρ'. Using Jensen's inequality under the assumption that $R(h)$ is strictly positive, we have the log ration of two objective funcions

$$\begin{aligned}\log\frac{J(\rho')}{J(\rho)} &= \log\int_{\Theta}\int_{H}\frac{R(h)p(h|\theta)p(\theta|\rho)}{J(\rho)}\frac{p(\theta|\rho')}{p(\theta|\rho)}dhd\theta \\ &\geq \int_{\Theta}\int_{H}\frac{R(h)p(h|\theta)p(\theta|\rho)}{J(\rho)}\log\frac{p(\theta|\rho')}{p(\theta|\rho)}dhd\theta.\end{aligned}$$

Hence, the lower bound is defined by

$$\log J_L(\rho') \equiv \log J(\rho) + \int_{\Theta}\int_{H}\frac{R(h)p(h|\theta)p(\theta|\rho)}{J(\rho)}\log\frac{p(\theta|\rho')}{\theta|\rho}dhd\theta. \tag{3.9}$$

To maximize this lower bound, the derivative of (3.9) with respect to ρ' should equal to 0

$$\nabla_{\rho'}\log J_L(\rho') = \int_{\Theta}\int_{H}\frac{R(h)p(h|\theta)p(\theta|\rho)}{J(\rho)}\nabla_{\rho'}\log\frac{p(\theta|\rho')}{p(\theta|\rho)}dhd\theta = 0.$$

Since $J(\rho)$ is constant, this equation can be simplified as

$$\int_{\Theta}\int_{H}R(h)p(h|\theta)p(\theta|\rho)\nabla_{\rho'}\log p(\theta|\rho')dhd\theta = 0.$$

By applying sampling trick, we have

$$\frac{1}{N}\sum_{n=1}^{N}\nabla_{\rho'}\log p(\theta^n|\rho')R(h^n)=0. \tag{3.10}$$

If $p(\theta|\rho')$ is represented by an exponential family distribution, the update rule is given by a closed form. In particular, we consider that $p(\theta|\rho')$ is given by a product of independent Gaussian distribution $\mathcal{N}(\theta_i|\eta_i',\sigma_i'^2)$ for each parameter θ_i in θ. The log derivatives of $p(\theta|\rho')$ with respect to η_i' and σ_i' are computed as

$$\nabla_{\eta_i'}\log p(\theta|\rho')=\frac{\theta_i-\eta_i'}{\sigma_i'^2} \tag{3.11}$$

$$\nabla_{\sigma_i'}\log p(\theta|\rho')=\frac{(\theta_i-\eta_i')^2-\sigma_i'^2}{\sigma_i'^3} \tag{3.12}$$

Substituting (3.11) and (3.12) into (3.10) yields

$$\eta'=\frac{\sum_{n=1}^{N}[R(h^n)\theta_i^n]}{\sum_{n=1}^{N}R(h^n)}$$

$$\sigma'=\sqrt{\frac{\sum_{n=1}^{N}[R(h^n)(\theta_i^n-\eta_i')^2]}{\sum_{n=1}^{N}R(h^n)}}.$$

It should be noted that the denominator is positive because we assume that $R(h)$ is strictly positive so that it can resemble an (improper) probability distribution to weight the parameters. To obtain a good sampling performance, we only take the parameters from the best K returns in N trajectories for updating.

Algorithm 8: EM-based Policy Hyperparameter Exploration with K-elite Selection

Input : Initialize policy hyperparameter η and σ

repeat

  Perform N episodes and sample N trajectories:

    for each episode n draw $\theta_i^n \sim \mathcal{N}(\eta_i, I\sigma_i^2)$ for all i

    collect $H = \{h^n\}_{n=1...N}$

    evaluate $R(h^n)$

  Select K best trajectories from the sorted $R(h^n)$

  Update

$$\eta_{new} = \frac{\sum_{k=1}^{K}[R(h^k)\theta_i^k]}{\sum_{k=1}^{K} R(h^k)}$$

$$\sigma_{new} = \sqrt{\frac{\sum_{k=1}^{K}[R(h^k)(\theta_i^k - \eta_{new})^2]}{\sum_{k=1}^{K} R(h^k)}}$$

until *Convergence*;

3.3.2 Simulation Experiment

In this section, we compare our method with PGPE and classic policy gradient method Finite Difference. For each method we use $N = 20$ trajectories to update one set of parameters, and select $K = 10$ to obtain the elite parameters for updating. The results are taken by the average of 20 independent runs. We plot the learning curves of the average and the standard error of cumulative returns against the iterations of parameter updating.

3.3.2.1 Pendulum swing-up with limited torque

The target of this non-linear control task is to swing up the pendulum to the upright position and stay as long as possible [43]. See Appendix for the details of the simulation setup. We use 16×16 radial basis functions to represent the two-dimensional state variables, the angle, and the angular velocity of the pendulum: $\boldsymbol{x} = \{\varphi, \dot{\varphi}\}$. We use the normalized radial basis function defined by

$$\Phi_k(\boldsymbol{x}) = \frac{e^{-\|s_k^T(x-c_k)\|^2}}{\sum_{k=1}^{K} e^{-\|s_k^T(x-c_k)\|^2}}$$

where k is the index of the radial basis functions and s_k and c_k are the size and center of the k-th basis function.

The action is the torque applied to the pendulum $u = 5 \times \tanh(\theta^T \Phi(x))$ with maximum torque 5 Nm, where θ is the policy parameter and $\Phi(x)$ is the basis function vector. The system starts from an initial state $x_0 = \{\varphi_0, 0\}$, where φ_0 is randomly selected from $[-\pi, \pi]$ rad, and terminates when $|\dot{\varphi}| \geq 4\pi$ rad/s. The sampling rate is 0.02 sec for each time step and maximum time steps is 1000 (=20 sec) for one episode. The strictly positive reward for one history is given by

$$R(h) = \sum_{t=1}^{T} \exp(-x_t^T Q x_t - u_t^T R u_t) \tag{3.13}$$

where Q and R are the quadratic penalty matrix determined by users. Here we used $Q = I_2$ and $R = 1$, where I_2 is the 2-dimensional identity matrix. Note that a linear state feedback policy failed to achieve the task because the maximum torque was smaller than the maximal load torque. For Finite Difference, the initialization of policy parameters is $\theta_o = 0$, and the steps for the policy parameter update are $\delta\theta \sim U(-3.46, 3.46)$, a uniform distribution with variance 1. For PGPE and EPHE, the initial hyper parameters are $\eta_0 = 0$, $\sigma_0 = 1$.

Figure 17 shows the performance of FD, PGPE and EPHE. We hand-tune the learning rates for each method and found that separate learning rates for each parameter are required for PGPE. The optimal parameters were $\alpha = 0.1$ for FD, and $\alpha_\eta = 0.001$, $\alpha_\sigma = 0.0001$ for PGPE. We also showed the performance of PGPE with $\alpha_\eta = 0.0001$, $\alpha_\sigma = 0.0001$ to illustrate its parameter dependence. The proposed method learns faster and achieved better performance after 30 iterations without the need for tuning the learning rate.

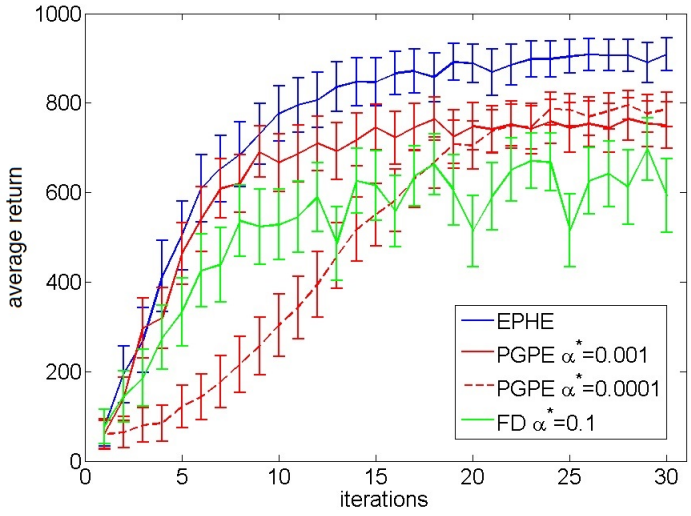


Figure 17. Learning curves of EPHE-K, PGPE, and FD in pendulum swing-up task

3.3.2.2 Cart-pole balancing

In this task, the agent aims to maximize the length of time of a movable cart balancing a pole upright in the center of a track [44]. See Appendix for the details of the simulation setup. The state variables are the position and the velocity of the cart on the track, and the angle and the angular velocity of the pole: $\boldsymbol{x} = \{x, \dot{x}, \theta, \dot{\theta}\}$. The action is the force applied to the cart given by a linear parameterized policy $u = \boldsymbol{\theta}^T \boldsymbol{x}$. We add Gaussian white noise with standard deviation of 0.001 rad/s and 0.01 m/s to the dynamics. The system starts within a random position and a random angle inside [-0.2,0.2]

rad, and [-0.5,0.5] m until it reaches the target region of [-0.005,0.05] rad and [-0.05,0.05] m, and terminates at $|x| \geq 2.4$ m, and $|\theta| \geq 0.7$ rad. The sampling rate is 0.02 sec for each time step and maximum time steps is 1000 (=20 sec) for one episode. The strictly positive reward is the same as (3.13). Here we use $Q = I_4$ and $R = 1$. The initializations of policy parameters for FD and hyperparameters for PGPE and EPHE are from a reasonable prior knowledge, which indicates certain distance from the optima. The parameter updates for the FD controller is $\delta\boldsymbol{\theta} \sim U(-3.9, 3.9)$, a uniform distribution with variance 5. We test with the same initialization as FD of $\boldsymbol{\eta}_0$ with different $\boldsymbol{\sigma}_0 = 5$ for PGPE, and $\boldsymbol{\sigma}_0 = 35$ for EPHE.

Figure 18 shows the performance of FD, PGPE ad EPHE. The best learning rates were $\alpha = 0.01$ for FD, $\alpha_\eta = 0.001$, $\alpha_\sigma = 0.0001$ for PGPE. EPHE achieved faster learning without learning rate tuning.

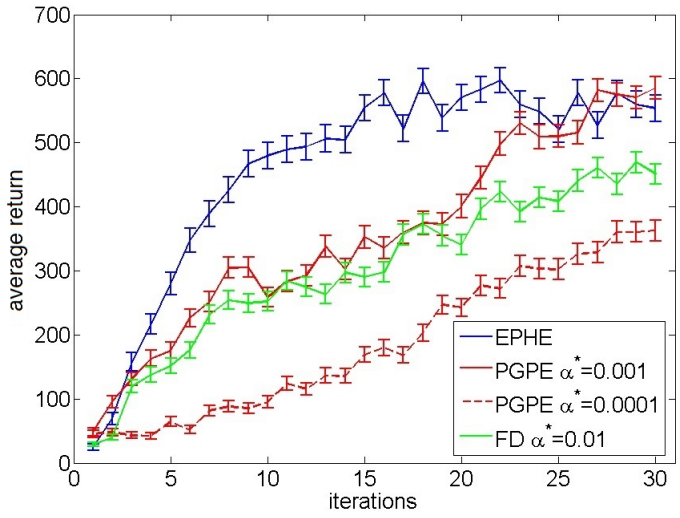


Figure 18. Learning curves of EPHE-K, PGPE, and FD in cart-pole balancing task

3.3.2.3 Smartphone balancer

Finally we test EPHE and compared it with PGPE and FD in our smartphone balancer simulator for standing-up and balancing behavior. The state variables are tilting angle and angular velocity of the body, and rotating angle and angular velocity of the wheel where $\boldsymbol{x} = \{\vartheta, \dot{\vartheta}, \varphi, \dot{\varphi}\}$. The control input u is the motor torque applied to the left and right wheel. We adopt a switching framework in which a linear feedback stabilizer is selected to achieve balancing if the tilting angle of the robot body is within the rangle of $[-\vartheta_{thre}, \vartheta_{thre}]$, otherwise the central pattern generator (CPG) based destabilizer is applied, defined by

$$\dot{x}_{CPG} = \omega y_{CPG} + \beta\dot{\vartheta}, \quad \dot{y}_{CPG} = \omega x_{CPG} \tag{3.14}$$

where x_{CPG} and y_{CPG} are the CPG state and ω and β are its parameters. The control signal is given by $u = y_{CPG}$. The policy parameters are the four-dimensional control gain vector for the linear stabilizer, the switching threshold, and two parameters of the oscillator: $\boldsymbol{\theta} = \{k_\vartheta, k_{\dot{\vartheta}}, k_\varphi, k_{\dot{\varphi}}, \vartheta_{thre}, \omega, \beta\}$. Figure 19 shows the control architecture. We also add observation Gaussian white noise with standard deviation of 0.01 to the system.

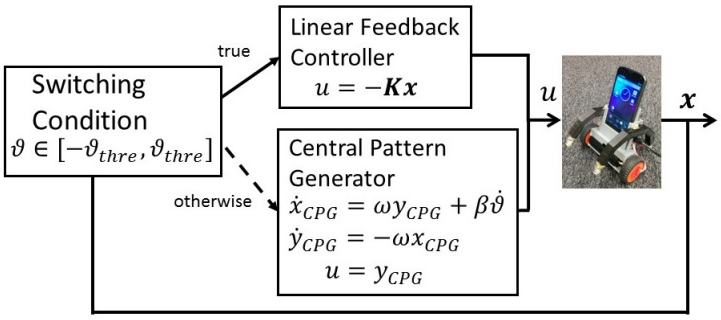


Figure 19. Switching control architecture of the smartphone balancer

The agent is required to start moving from the resting angle 60^o, bounce with the bumper to stand up and finally achieve balancing. The simulation runs with a sampling rate 0.02 sec for each step. The agent learns one episode within the maximum of 1000 steps (=20 sec). The cumulative reward is calculated as (3.13). We initialize the parameters $\boldsymbol{\theta}_0$ and the step size $\delta\boldsymbol{\theta}$ for FD, and the hyperparameters $\boldsymbol{\eta}_0$ for PGPE and EPHE with uniform distributions, and fixed $\boldsymbol{\sigma}_0$ based on the prior knowledge we obtained in previous section.

Figure 20 shows the learning performance. The best learning rates are, $\alpha_K = 0.0001$, $\alpha_{\vartheta_{thre}} = 0.00001$, $\alpha_\omega = 0.01$, $\alpha_\beta = 0.01$ for FD, $\alpha_\eta = 0.001$, $\alpha_\sigma = 0.001$ for PGPE. the success rates of each method are illustrated in Table 5. EPHE ourperforms others after 10 iterations and achieved a more reliable performance after 20 iterations.

We also plot the distribution of 20 final optimized parameters in Figure 21. FD has the most centralized distribution of final optimized parameters because it represents the policy parameters while PGPE and EPHE represent the distribution of the policy parameters. We pick up one set of the optimized parameters $\boldsymbol{\eta} = \{0.0021, 0.00982, 0.2953, 0.0651, 47^o, 11.7169, 18.7890\}$ based on EPHE and sample 10 sets of policy parameters based on the Gaussian prior distribution to illustrate the bouncing and stabilizing behaviors in Figure 22. It shows that the control signals are synchronized with the angular velocity and the switcher can successfully coordinate the two controller so achieve the expected behaviors.

EPHE	90%
PGPE	60%
Finite Difference	85%

Table 5. Successful rate of EPHE-K, PGPE and FD

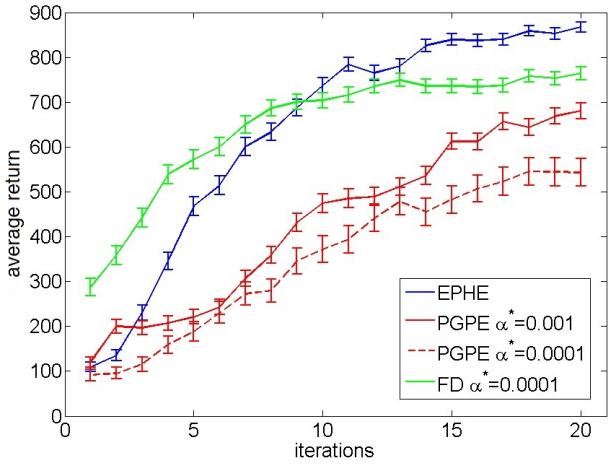


Figure 20. Learning curves of EPHE-K, PGPE, and FD in standing-up and balancing task of smartphone balancer simulator

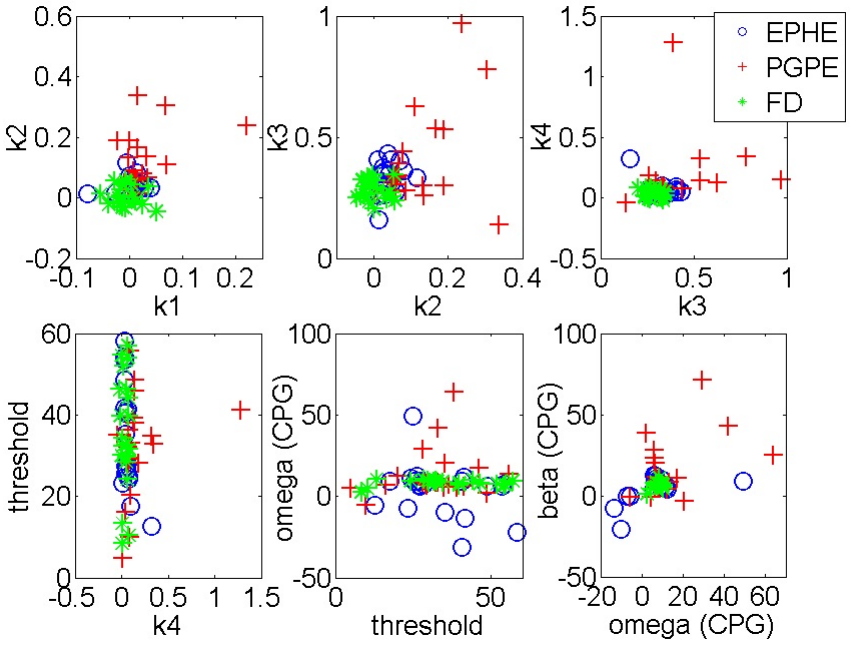


Figure 21. Distributions of the optimized parameters of EPHE-K, PGPE, and FD in standing-up and balancing task of smartphone balancer

simulator

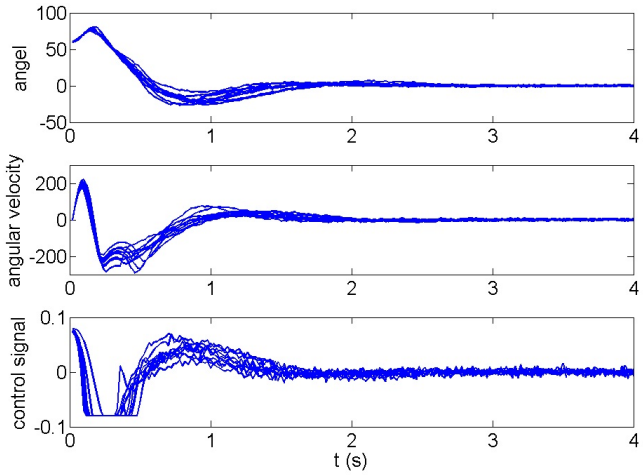


Figure 22. Trajectories realized by the policy of which the parameters are sampled from the optimized prior distributions

3.3.3 Discussion

Our method EPHE can compete the other two methods in all the tasks with a faster convergence speed and a steadily higher return (Figure 17, 18, 20). In the smartphone balancer simulator case, FD learns better in the beginning because it searches in the policy parameter space directly under one-dimensional uniform distribution domain, and it is also easy to be trapped in local optima, illustrated in Figure 21.

We also found that the difference between the initialization of the variances indicates different insights of PGPE and EPHE: PGPE optimizes the hyper-parameters by computing gradients, in which case, smaller variance leads to a more precise approximation. While EPHE computes the average of the sampled points which suggests larger initial variance explores more. The learning behavior is much improved by the K-elite selection mechanism. Because parameters are quite far away from the optima in the beginning of learning, frequent failures decelerate the learning process. Also, it is easy

to be trapped in local optima by updating with all the parameters (when $K = N$). Figure 23, 24 and 25 reveal the sensitivity of K to N in three tasks. Agents with smaller setting of K learn relatively faster but reach no better performance in the end. There is no significant difference between different settings of K, but agents without the selection mechanism achieve much worse behaviors. Another interesting finding is, in the smartphone balancer case, the threshold of switching condition seems not crucial parameters to be tuned. This saves the burden of making different arms for the robot body in real hardware construction.

3.3.4 Section Summary

In this section, we developed a new policy search algorithm, EM-based Policy Hyperparameter Exploration (EPHE). We tested it in two benchmark tasks and the smartphone balancer simulator with a non-linear, non-differentiable controller. Our method integrated PGPE with the EM-based update that maximizes a lower bound of the exptected return in each iteration of hyperparameter updating. The simulation results showed that our method outperformed other policy gradient methods such as Finite Difference and PGPE after fine tuning of the learning rates.

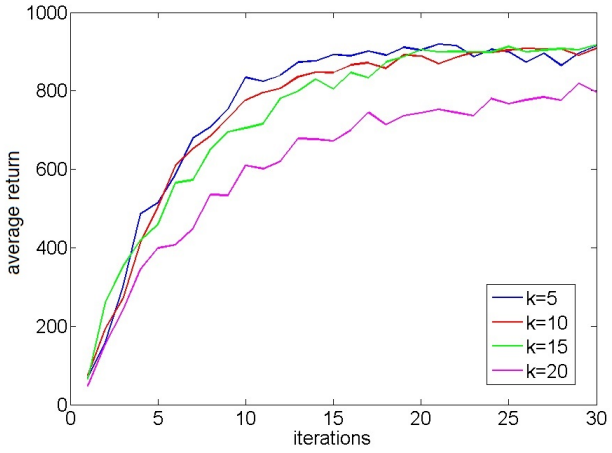


Figure 23. Sensitivity of K to N in pendulum swing-up task

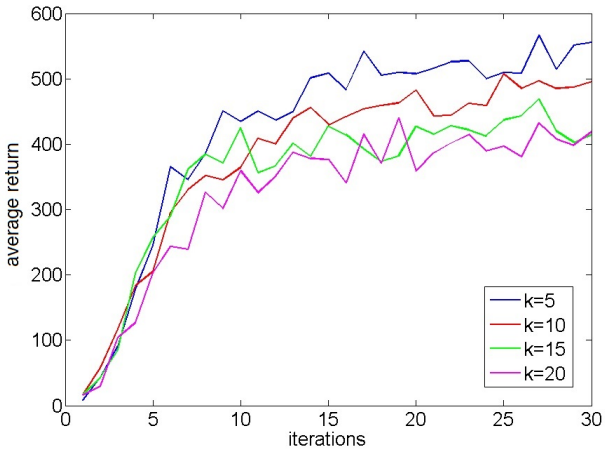


Figure 24. Sensitivity of K to N in cart-pole balancing task

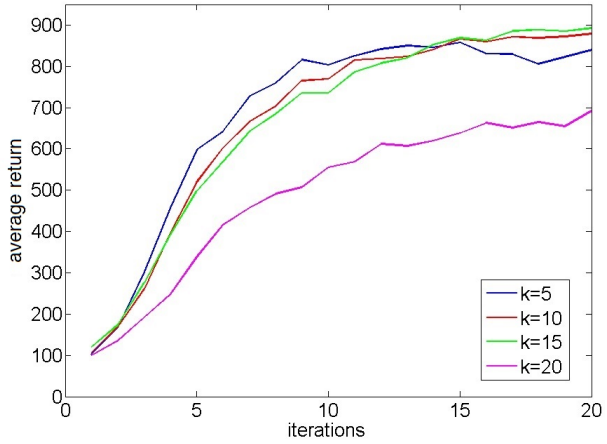


Figure 25. Sensitivity of K to N in in standing-up and balancing task of smartphone balancer simulator

3.4 EPHE-AB

3.4.1 Method

In the previous section, we showed EPHE with a K-elite mechanism (EPHE-K), in which we selected the K best parameters to discard bad samples for updating the hyperparameters and improved the learning process. However, this requires a new parameter tuning. Here we propose an adaptive baseline using the mean value of the returns:

$$R_b(h^n) = \max(0, R(h^n) - \text{mean}(\{R(h^n)\}_{n=1}^N)), \tag{3.15}$$

where we denote the returns collected at the current iteration step as $\{R(h^n)\}_{n=1}^N$ and $\text{mean}(\{R(h^n)\}_{n=1}^N) = \sum_{n=1}^{N} R(h^n)/N$. Note that $R_b(h)$ should be remained non-negative to resemble an (improper) probability distribution to weight the parameters. Therefore, we call the adaptive baseline by (3.15) the EPHE-AB-mean.

Figure 26 illustrates an example of weighting by the EPHE-AB-mean, EPHE-K and EPHE with no elitism. Suppose that we have 20 returns following a Gaussian distribution and sort them in descending order as $\{R^i\}_{i=1}^{20}$. Then, weighting coefficients $R^i / \sum_{i=1}^{20} R^i$ are computed for the EPHE-AB-mean, EPHE-K(=10), and the no-elite methods. Unlike EPHE-K(=10), the EPHE-AB-mean enhances the differences of the returns.

In the same way, we consider the adaptive baseline by replacing the mean operator with the $m-$standard deviation from the mean operator defined by

$$\text{mean} - m\text{std}(\{R(h^n)\}_{n=1}^N) = \text{mean}(\{R(h^n)\}_{n=1}^N) + m \times \text{std}(\{R(h^n)\}_{n=1}^N).$$

Note that the adaptive baseline can also be equipped to other EM-based policy search methods, like episodic PoWER.

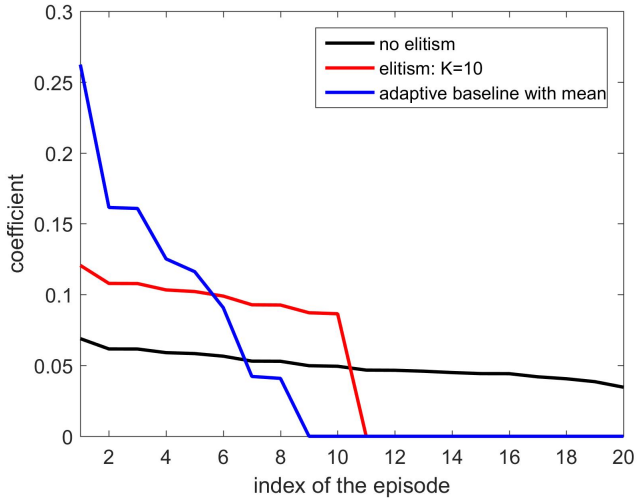


Figure 26. Weight comparison of EPHE-AB-mean, EPHE-K and no-elitism methods

Algorithm 9: EM-based Policy Hyperparameter Exploration with Adaptive Baseline

Input : Initialize policy hyperparameter η and σ

repeat

  Perform N episodes and sample N trajectories:

    for each episode n draw $\theta_i^n \sim \mathcal{N}(\eta_i, I\sigma_i^2)$ for all i

    collect $H = \{h^n\}_{n=1\ldots N}$

    evaluate $R(h^n)$

    calculate $R_b(h^n) = \max(0, R(h^n) - \text{mean}(\{R(h^n)\}_{n=1}^N))$

  Update

$$\eta_{new} = \frac{\sum_{n=1}^{N}[R_b(h^n)\theta_i^n]}{\sum_{n=1}^{N} R_b(h^n)}$$

$$\sigma_{new} = \sqrt{\frac{\sum_{n=1}^{N}[R_b(h^n)(\theta_i^n - \eta_{new})^2]}{\sum_{n=1}^{N} R_b(h^n)}}$$

until *Convergence*;

3.4.2 Simulation Experiment

In this section, we compare EPHE-AB-mean with EPHE with CMAES weighting scheme (EPHE-CW) [36], EPHE with REPS weighting scheme (EPHE-RW) [35][39], EPHE-K, PGPE, NES and FEM mentioned in Section 3.2, and compare EPHE-AB with the baselines of the mean, one and two standard deviations from the mean, and a fixed baseline in three simulation experiments same as in Section 3.3. For each method we use $N = 20$ trajectories to update one set of parameters. We select $K = 10$ to obtain the elite parameters for updating in EPHE-CW, EPHE-RW and EPHE-K. The results are taken by averaging 20 independent runs. We plot the learning curves of the average and the standard error of the cumulative returns and the number of discarded samples against the iterations of the parameter updating.

The weights of EPHE-CW are computed by

$$w_k = \ln\frac{K+1}{2} - \ln k, \text{ for } k = 1, ..., K.$$

The weights of EPHE-RW are computed by

$$w_k = \exp\frac{R(h^k)}{\eta}$$

where η is the temperature parameter obtained by optimizing the dual function $g(\eta)$, such that $\eta > 0$, [35][39][45]

$$g(\eta) = \eta\epsilon + \eta\log\sum_{k=1}^{N}\frac{1}{N}\exp\frac{R(h^k)}{\eta}$$

and ϵ is the upper bound of KL divergence set as 0.1. We used the function `fminunc` in MATLAB to obtain the optimal η. Note that EPHE with REPS weighting scheme is equivalent to episodic REPS [45].

3.4.2.1 Pendulum swing-up with limited torque

Here we use the same experimental setting as in Section 3.3.2.1. The initial hyperparameters are $\boldsymbol{\eta}_0 = 0$, $\boldsymbol{\sigma}_0 = 1$ for each algorithm. The best learning rate for PGPE and NES are $\alpha_\eta = 10^{-3}$ and $\alpha_\sigma = 10^{-4}$ which are selected, respectively from sets $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. The forgetting rate for FEM is 0.05.

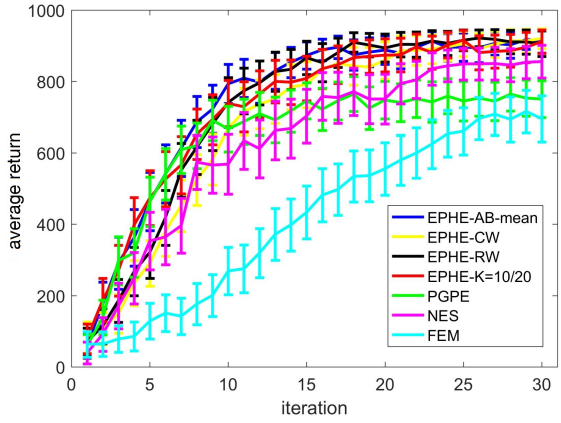


Figure 27. Learning curves of EPHE-AB with baseline of mean, EPHE-CW, EPHE-RW, EPHE-K, PGPE, NES, FEM in pendulum swing-up task

Figure 27 shows the EPHE-AB performance with the baseline of mean, EPHE-CW, EPHE-RW, EPHE-K, PGPE and NES with the best learning rate and FEM. The error bars show one standard deviation over 20 independent runs. There is no significant difference between EPHE-AB and EPHE-K, but they learn faster and achieved better performance than FEM and gradient based PGPE and NES after 10 iterations.

Figure 28 shows the EPHE-AB performances with different baselines. There are no significant differences among the fixed baseline and the baselines of mean and the one standard deviation from it. Figure 28 (b) shows the number of discarded samples against the iterations, and we found that in the early stage of learning, the baseline with mean discards most of the samples while the other three discard about 60% of them.

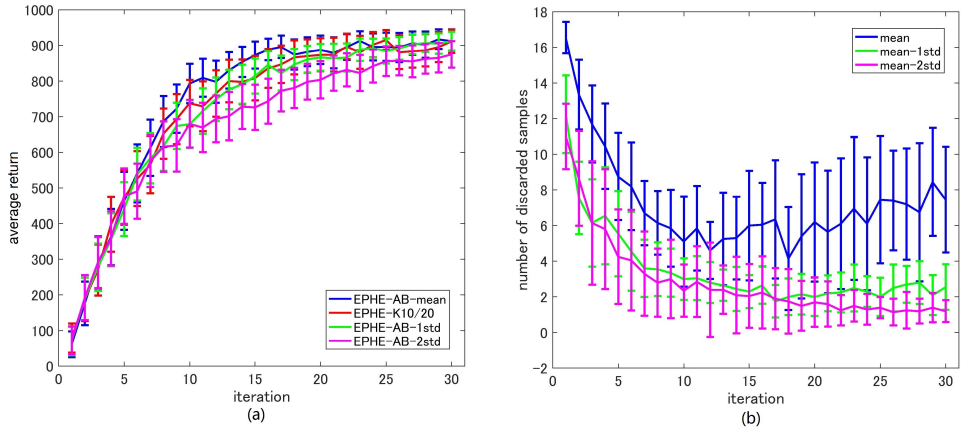


Figure 28. Effects of baseline functions in pendulum swing-up task: (a) learning curves and (b) number of discarded samples

3.4.2.2 Cart-pole balancing

Here we use the same experimental setting as in Section 3.3.2.2. The initial hyperparameters are $\boldsymbol{\eta}_0 = 0$, $\boldsymbol{\sigma}_0 = 35$ for EPHE-AB, PoWER and FEM, and $\boldsymbol{\sigma}_0 = 5$ for PGPE and NES. The best learning rate for PGPE and NES are $\alpha_\eta = 10^{-4}$ and $\alpha_\sigma = 10^{-5}$ which are selected, respectively from sets $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. The forgetting rate for FEM is 0.1.

Figure 29 shows the EPHE-AB performance with a baseline of mean, EPHE-CW, EPHE-RW, EPHE-K, PGPE and NES with the best learning rate and FEM. The error bars show one standard deviation over 20 independent runs. The proposed method learns faster and achieved better performance after 10 iterations than the other algorithms. PGPE and NES with the best learning rate learned slowly, but they achieved almost the same performance as the best performance at the end of the iterations.

Figure 30 shows the EPHE-AB performance with different baselines (a) and the number of discarded samples against the iterations (b). There are no significant differences among the baselines of mean, one and two standard deviations from it. However, the adaptive baselines perform better than the

fixed baseline. Figure 30 (b) suggests that in the early stage of learning, the baseline with mean discards most of the samples while the other three discard about 70% of them.

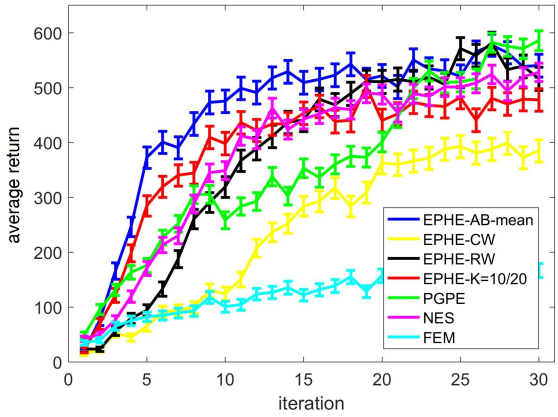


Figure 29. Learning curves of EPHE-AB with baseline of mean, EPHE-CW, EPHE-RW, EPHE-K, PGPE, NES and FEM in cart-pole balancing task.

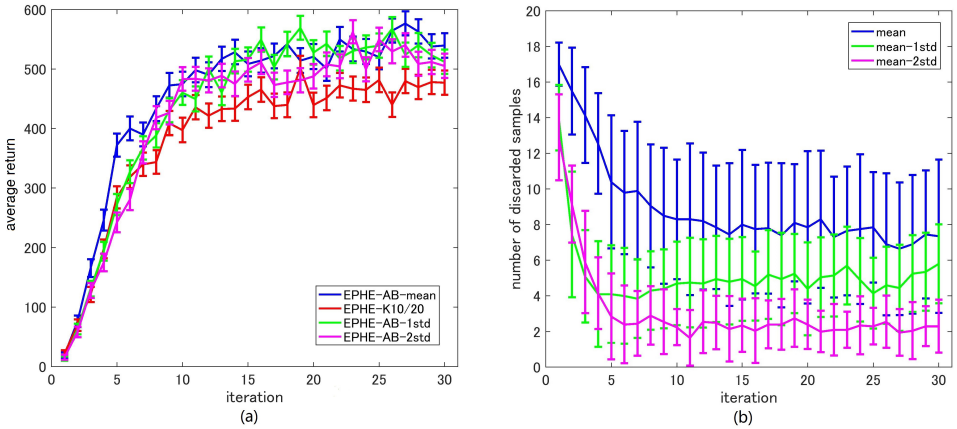


Figure 30. Effects of baseline functions in cart-pole balancing task: (a) learning curves and (b) number of discarded samples

3.4.2.3 Smartphone balancer

Here we use the same experimental setting as in Section 3.3.2.3. The initialized hyperparameters $\boldsymbol{\eta}_0$ and $\boldsymbol{\sigma}_0$ are based on the prior knowledge we obtained in Section 2. $\boldsymbol{\sigma}_0$ for PGPE and NES are smaller than EPHE, PoWER and FEM, because PGPE uses small initial variance to approximate the gradient more precisely while EPHE uses large initial variance to explore the parameter space. The forgetting rate for FEM is 0.2.

Figure 31 shows the learning performance. The error bars show one standard deviation over 20 independent runs. EPHE-AB-mean learns faster than the other methods and achieved a more reliable performance after 7 iterations.

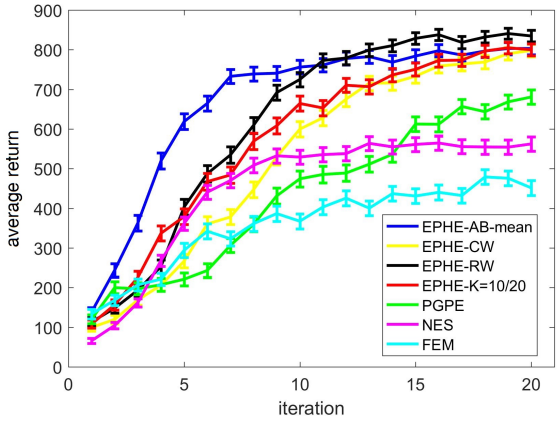


Figure 31. Learning curves of EPHE-AB-mean, EPHE-CW, EPHE-RW, EPHE-K, PGPE, NES, and FEM in standing-up and balancing task of smartphone balancer simulator

Figure 32 shows the EPHE-AB performance with different baselines (a) and the number of discarded samples against the iteration (b). The baseline of mean outperforms the others and the adaptive baselines perform better than the fixed baseline. Figure 32 (b) suggests that in the early state of learning, the baseline with mean discards most of the samples and has a steady decrease during the learning. The baseline with one and two standard deviations from the mean preserve most of the samples due to a large standard

deviation of returns based on the reward function. The reason of baselines with one and two standard deviations discarded almost all the samples in the beginning of learning is, the return distribution had a large variance but relatively low mean value.

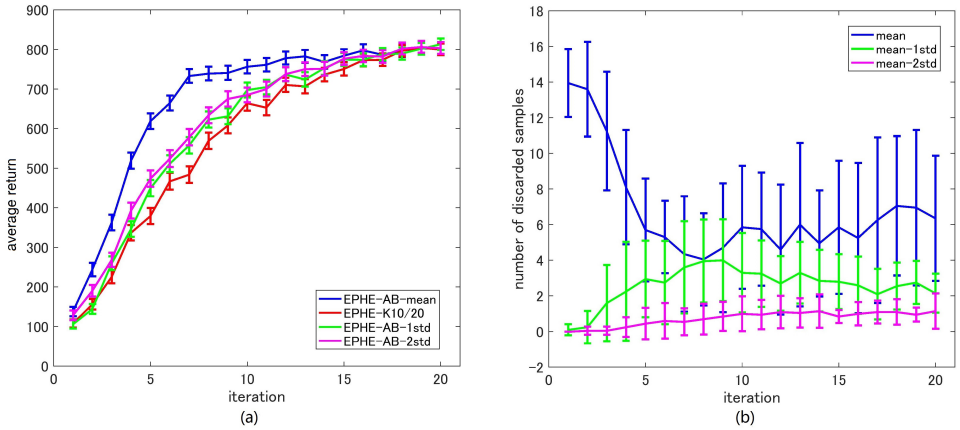


Figure 32. Effects of baseline function in standing-up and balancing task of smartphone balancer simulator (a) learning curves and (b) number of discarded samples

3.4.3 Hardware Experiment

Here we implement EPHE-AB-mean and compare it with PGPE in the real smartphone balancer system for two tasks, one is to learn standing-up and balancing, and another one is to learn approaching to a visual target while balancing.

3.4.3.1 Standing-up and balancing task

This is a real hardware implementation of the simulation task in Section 3.3.2.3 and Section 3.4.2.3. The balancer is required to learn to get off the floor and remain balancing in the upright region. It is equipped with an

elastic bumper with a 0.8869 N/mm spring coefficient and an initial resting angle: $\vartheta_0 = 25^o$. The state variables are the tilting angle and angular velocity of the body obtained from the sensor fusion of the gyroscope and accelerometer inside the smartphone, and the angular velocity of the left and right wheels from the rotary encoder inside the wheel:$\boldsymbol{x} = \{\vartheta, \dot{\vartheta}, \dot{\varphi}_L, \dot{\varphi}_R\}$. We eliminate the rotating angle of the wheels due to the accumulated error of the rotary encoder. Since we consider only the sagittal plane behaviors, the same control signal is sent to both wheels. The reward function is +1 when the tilting angle is inside $[-5^o, 5^o]$ and the angular velocity of the robot body is inside [-173,173] deg/s, and otherwise it is 0.

We use the same control architecture in Figure 19. We fix $\vartheta_{thre} = 20^o$ and the policy parameters to be optimized are the three-dimensional control gain vector and CPG parameters $\boldsymbol{\theta} = \{k_\vartheta, k_{\dot{\vartheta}}, k_{\dot{\varphi}}, \omega, \beta\}$. Note that since $\dot{\varphi}_L$ and $\dot{\varphi}_R$ share the same control gain as $k_{\dot{\varphi}}$ the feedback gain is given by $\boldsymbol{K} = \{k_\vartheta, k_{\dot{\vartheta}}, k_{\dot{\varphi}_L}, k_{\dot{\varphi}_R}\}$.

The sampling rate is 0.005 sec for each step, and one episode contains 12000 steps (=60 sec). The robot rests for 4 sec at the end of each episode to recover its resting angle. We update the hyperparameters every 10 episodes for 10 iterations, meaning 100 episodes for an entire run. The initializations of the hyperparameters are $\boldsymbol{\eta}_0 = [0, 0, 0, 0, 0]^T$, and $\boldsymbol{\sigma}_0 = [100, 200, 20, 20, 20]^T$. The best PGPE learning rates are selected, respectively from sets $\{10^{-6}, 5 \times 10^{-6}, 10^{-5}\}$ and $\{10^{-5}, 3 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}\}$. Since EPHE-AB was better than or equal to the other methods in all the three simulation results, we focus on a comparison between EPHE-AB and PGPE with the best learning rate to investigate the different updating approaches of gradient estimating and reward weighting.

We run 5 independent runs and measure the average and the standard error of the cumulative returns against the iterations of hyper parameter updating. An episode is regarded as successful if the agent bounces to the upright position and maintains its balancing until the episodes end. Failures includes when the agent couldnt stand up at all, swinging forward and backward at a relatively high speed, etc. We define a successful learning as when the agent found the hyperparameters that achieved successful episodes within 5 iterations. The successful learnings of EPHE-AB and PGPE are 5/5 and 3/5 respectively.

Figure 33 compares the learning curves of EPHE-AB and PGPE with the best learning rate ($\alpha_\eta = 5 \times 10^{-6}, \alpha_\sigma = 3 \times 10^{-5}$). The error bars show one standard deviation over 5 independent runs. Although PGPE learned successful behaviors, the successful rate was low. EPHE-AB outperforms PGPE during the whole learning process.

Figure 34 shows the distributions of the final parameters found by 5 runs of EPHE-AB and PGPE with the best learning rate. With EPHE-AB, the parameters converge to similar distributions, while with PGPE, they often stay flat or degenerated. For example, for key parameter k_ϑ, EPHE-AB and PGPE found similar parameters ranging from 150 to 350, but only three of PGPEs final distributions converged. For the second key parameter $k_{\dot{\vartheta}}$, three of the PGPE distributions were near those successfully found by EPHE-AB but with larger variance, and two converged to the values away from the parameters found by EPHE-AB.

Figure 35 shows the trajectory of a successful standing-up and balancing task with one of the final hyper parameters:

$$\boldsymbol{\theta} = \{248.66, 1007.23, -8.51, -5.85, 17.15\}$$

learned by EPHE-AB. The yellow area shows when the bumper was activated. The agent bounced three times, reached and stayed in the target position after 4 seconds.

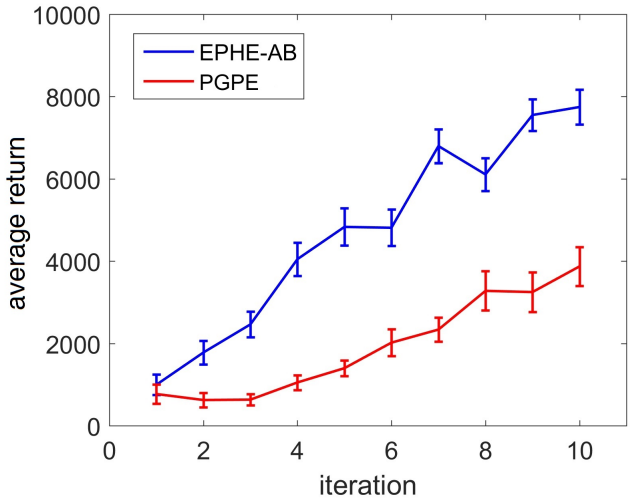


Figure 33. Learning curves of EPHE-AB and PGPE in standing-up and balancing task of real hardware

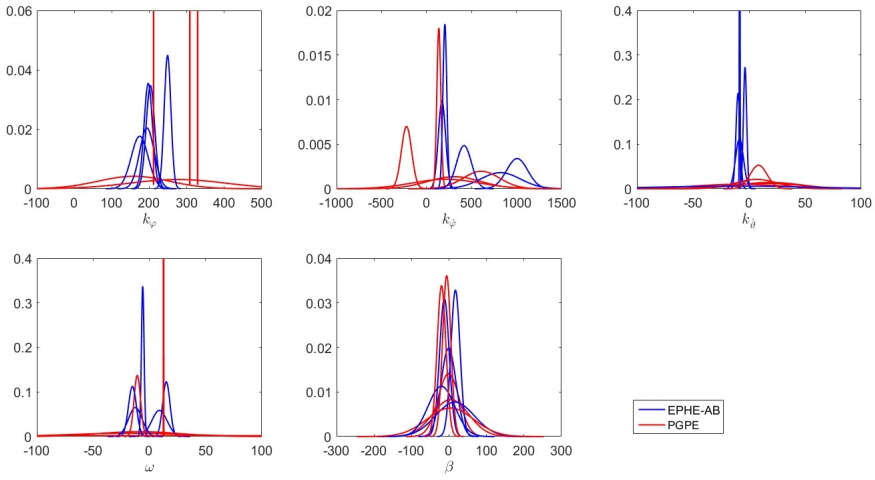


Figure 34. Distributions of final parameters found by 5 runs of EPHE-AB and PGPE in standing-up and balancing task of real hardware

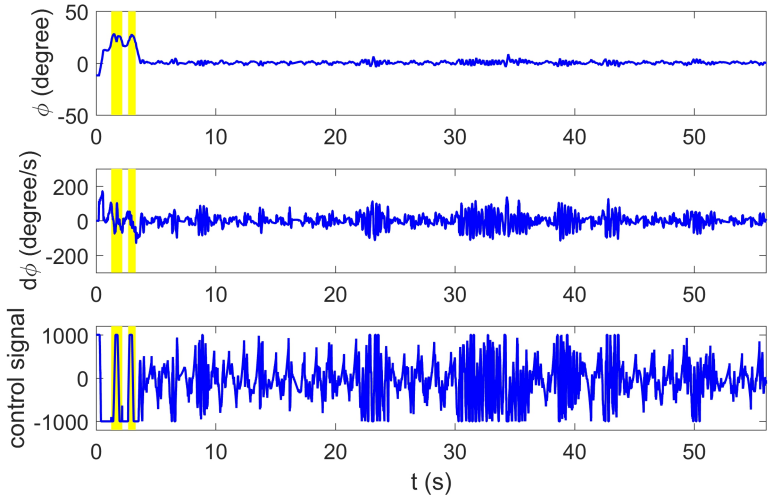


Figure 35. Typical trajectories of states and control signal with switching controller optimized by EPHE-AB

3.4.3.2 Vision-based approaching task

This task requires the smartphone balancer to learn to stand-up and approach a visual target while balancing. The basic hardware setting is the same as in Section 3.4.3.1. We use an Android phones camera and detect a blob of a specified color and computed its center position and area using the OpenCV library. The state variables are $\boldsymbol{x} = \{\vartheta, \dot{\vartheta}, \dot{\varphi}_L, \dot{\varphi}_R, c_x, D\}$, where $c_x \in [-0.3, 0.3]$ is the blob center on the X axis, and $D \in [0, 10]$ is the distance to the target calculated by the inverse of the square root of the blob area $D = \frac{a}{\sqrt{S_{blob}}}$, where parameter a is set to scale $D = 1$ to 10 dm. When the target is invisible, we set $c_x = \pm 0.4$ based on the previous sign of c_x, and $D = 15$. When the target is visible, the immediate reward at time t is given by

$$r_t = \exp\left(-w_1\left(\frac{\vartheta_t}{Z_\vartheta}\right)^2 - w_2\left(\frac{c_{x_t}}{Z_{c_x}}\right)^2 - w_3\left(\frac{D_t - 5}{Z_D}\right)^2\right),$$

where Z_ϑ, Z_{c_x}, and Z_D are respectively the constants to normalize ϑ_t, c_{x_t}, $D_t - 5$ to [-1,1]. We assign $w_1 = w_2 = 1.2$, and $w_3 = 0.6$. When the target is not visible, the reward is $r_t = 0$. The return is computed by

$$R(h) = \sum_{t=1}^{T} r_t,\ T = 12000.$$

We set up four initial positions of the robot to start, (Figure 36). The robots desired behavior is to start running from $D = 10$ (X_2, X_3, X_4) or $D = 2$ (X_1); it stops at $D = 5$ (star marks) while keeping the target in the middle and balancing. The robot lies down and faces straight ahead with initial state X_1, X_2, X_3 and faces back to the target with initial state X_4.

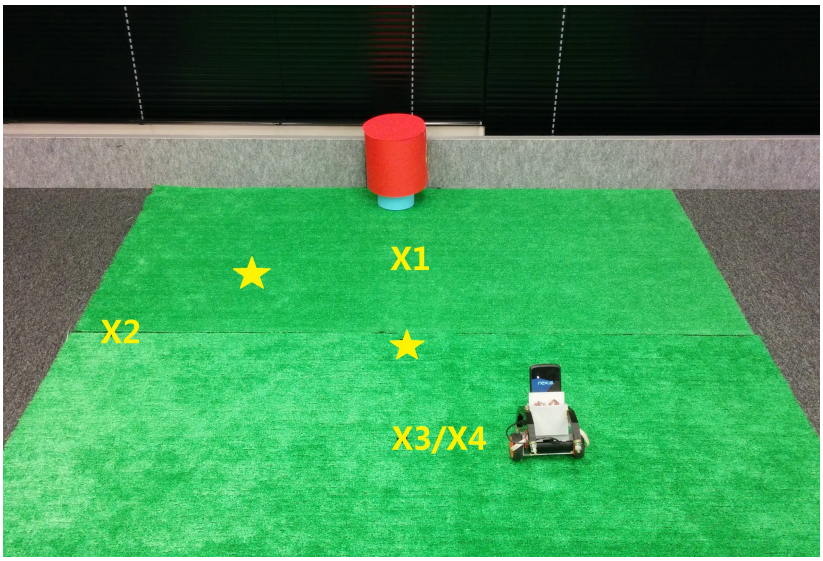


Figure 36. Landscape and initial positions of the robot in vision-based approaching task

The following is the control structure:

When ϑ is inside the threshold $\vartheta_{thre} \in [-20^o, 20^o]$, and the linear feedback stabilizer is activated:

$$u_L = -\Big(k_{\vartheta_L}\vartheta + k_{\dot{\vartheta}_L}\dot{\vartheta} + k_{\dot{\varphi}_L}\dot{\varphi}_L + \mathrm{ref}_L + k_{cL}c_x + k_{dL}(D-5)\Big)$$

$$u_R = -\Big(k_{\vartheta_R}\vartheta + k_{\dot{\vartheta}_R}\dot{\vartheta} + k_{\dot{\varphi}_R}\dot{\varphi}_R + \mathrm{ref}_R + k_{cR}c_x + k_{dR}(D-5)\Big),$$

Otherwise, the CPG-based destabilizer is activated as (3.14).

The policy parameters are 14-dimensional

$$\boldsymbol{\theta} = \{k_{\vartheta_L}, k_{\dot{\vartheta}_L}, k_{\dot{\varphi}_L}, k_{\vartheta_R}, k_{\dot{\vartheta}_R}, k_{\dot{\varphi}_R}, \mathrm{ref}_L, \mathrm{ref}_R, k_{cL}, k_{dL}, k_{cR}, k_{dR}, \omega, \beta\},$$

where $k_{\vartheta_L}, k_{\dot{\vartheta}_L}, k_{\dot{\varphi}_L}, k_{\vartheta_R}, k_{\dot{\vartheta}_R}, k_{\dot{\varphi}_R}$ are the linear feedback control gains for stabilization, ref_L and ref_R are the reference constants for the wheel to move forward and backward, $k_{cL}, k_{dL}, k_{cR}, k_{dR}$ are the control gain for the steering, and ω, β are the CPG parameters. The difference of k_{cL} and k_{cR} can cause rotation toward the target, but we set up all the parameters differently for the left and right wheels to adapt to any asymmetry in the hardware and the electronics. The sampling rate is 0.005 sec for each step, and one episode is 12000 steps (=60 sec). We change the initial position in every 5 episodes. The hyperparameters are updated every 20 episodes for 10 iterations, resulting in 200 episodes for one entire run.

The robot is forced to stop when $D \leq 1$ for safety concerns. The initial hyperparameters are set up based on the optimal parameters we obtained from the previous Section:

$$\boldsymbol{\eta}_0 = [250, 1000, -9, 250, 1000, -9, 200, 200, 0, 0, 0, 0, -6, 17]^T$$

$$\boldsymbol{\sigma}_0 = [50, 50, 10, 50, 50, 10, 500, 500, 2000, 100, 2000, 100, 10, 10]^T$$

The best learning rates for PGPE are selected from sets $\{10^{-6}, 10^{-5}, 10^{-4}\}$ and $\{10^{-6}, 10^{-5}, 10^{-4}\}$.

We run 5 independent runs and measure the average and the standard error of the cumulative returns against the iterations of hyperparameter updating. An episode is regarded as successful if the robot could stand up regardless of its initial position and facing direction, move toward the target position

and keep balancing until the end of the episode. Failures includes when the agent keeps moving around and was unable to stand up when the target was spotted, or when it takes too long to learn balancing instead of moving forward, etc. A successful learning is defined as when the agent identifies the hyperparameters that achieve the successful episodes within 5 iterations. The successful learning of EPHE-AB and PGPE are 2/5 and 0/5, respectively.

Figure 37 shows the learning curves of EPHE-AB and PGPE with the best learning rate ($\alpha_\eta = 10^{-5}, \alpha_\sigma = 10^{-5}$). The error bars show one standard deviation over five independent runs. PGPE failed to learn successful behaviors while EPHE-AB shows a clear learning curve.

Figure 38 shows the distributions of the final parameters found by EPHE-AB and PGPE with the best learning rate. Most of the parameters found by EPHE-AB converge, while with PGPE, even though each parameter has one or two convergent cases, the directions of the convergences were scattered. The key steering parameters in this task are k_{cL} and k_{cR}. The successful pairs found by EPHE-AB were $\{k_{cL}, k_{cR}\} = \{2548, -1377\}$ and $\{1092, -2230\}$, while PGPE failed to discover similar pair values. Hence, it was difficult for the agent to spot the target, not to mention standing and approaching.

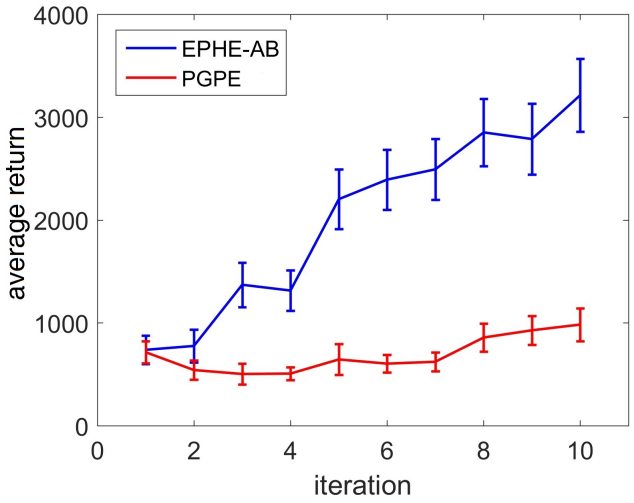


Figure 37. Learning curves of EPHE-AB and PGPE in vision-based approaching task of real hardware

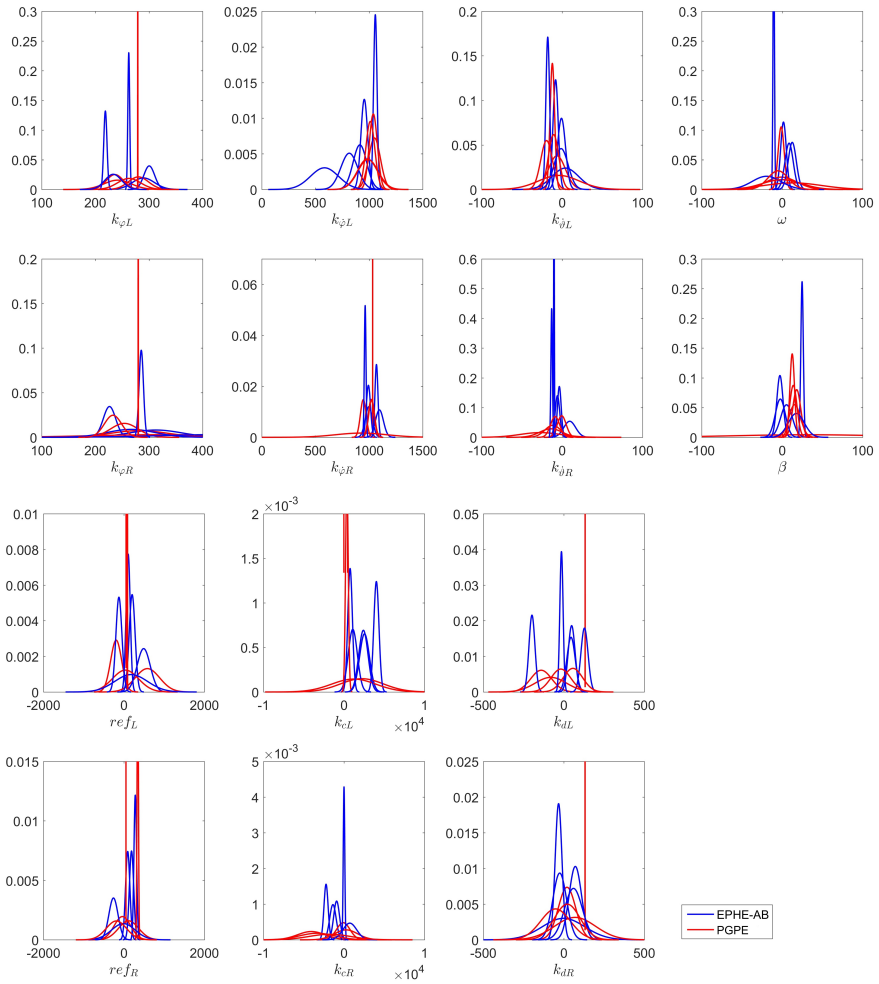


Figure 38. Distributions of final parameters found by 5 runs of EPHE-AB and PGPE in vision-based approaching task of real hardware

Figure 39, 40 show the trajectories and video snapshots of the successful episodes with the parameters [290, 955, 4, 226, 1096, 9, 204, 189, 1092, 48, -2230, -33, -11, 25] found by EPHE-AB from starting positions $X2, X4, X1$ and $X3$. Yellow area shows when the bumper is activated. From $X2$ and $X4$, the robot initially did not face the target, so it circled until it found the target. From $X2$, it took 3 seconds to navigate and 5 seconds to approach

while it bounced twice to stand up. From $X4$, the agent faced back to the target. It took around 4 seconds for navigation and around 6 seconds for approaching. It bounced four times to stand up.

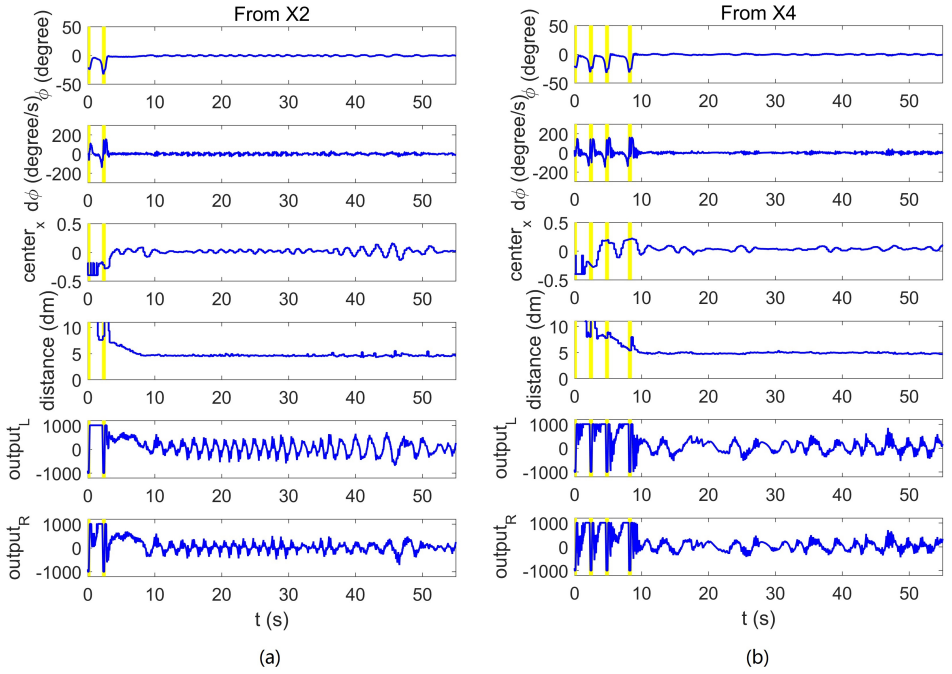
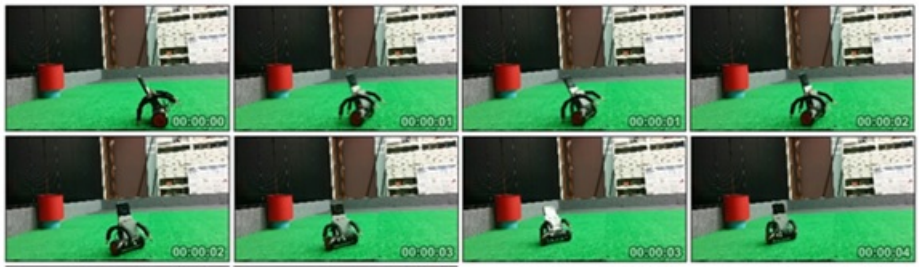
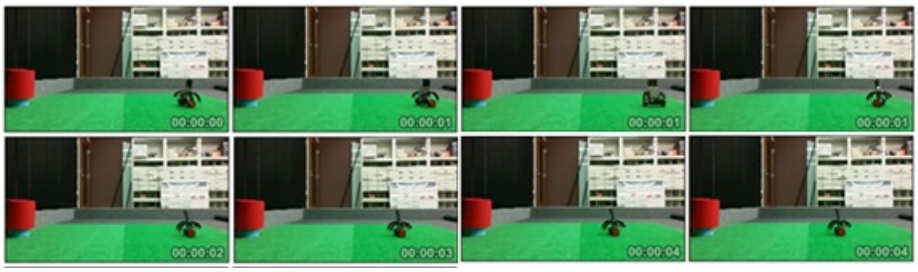


Figure 39. Typical trajectories of successful episodes from starting position $X2$ (a) and $X4$ (b)



(a)



(b)

Figure 40. Video snapshots of successful behaviors in vision-based approaching task from starting position $X2$ (a) and $X4$ (b)

3.4.4 Discussion

In Section 3.4.2 we compared EPHE-AB with the baseline of mean, EPHE-CW, EPHE-RW, EPHE-K, PGPE, NES and FEM in three simulation tasks. EPHE-AB significantly outperformed the others in the cart-pole balancing and smartphone standing-up tasks in the beginning of learning. In smartphone standing-up task, EPHE with REPS Weighting achieved higher returns in the end of learning, this is due to the convergence of η to small value when most of the returns are close to 1000. When η is small, it gives heavy weights to very small number of the highest returns.

We examined the learning behaviors of EPHE-ABs with different baselines, including the fixed baseline with half of the sample size, mean, one and two standard deviations from the mean. We found no significant performance differences among all of the baselines in the pendulum swing-up and cart-pole balancing task; however, the baseline with the mean outperformed the others in the smartphone standing-up tasks. This result suggests that the baseline with the mean discards most of the samples in the early stage of learning, and leads to a steady decrease of the number of discarded samples during learning. By examining the return distribution of different tasks, we found that the adaptive baseline plays an important role in the beginning of learning, when a relatively small number of episodes result in large returns. While in the end of learning, the mean baseline is lower than the fixed baseline

when the returns contain several value around 0. This also explains the better performance of EPHE with REPS Weighting in the smartphone standing-up task. See Figure 41.

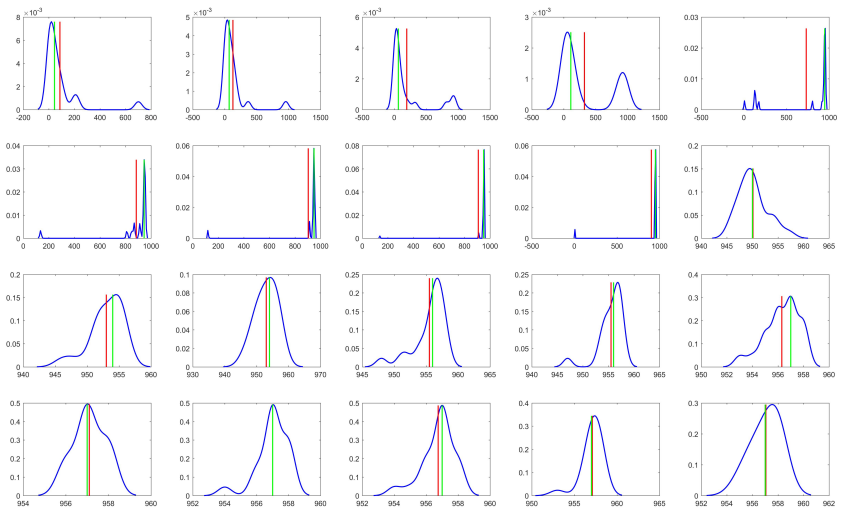


Figure 41. Return distribution of one successful learning in standing-up and balancing task of smartphone simulator

In the hardware experiments in Section 3.4.3, we compared EPHE-AB with the baseline of mean with PGPE with the best learning rate and showed that EPHE-AB learned the tasks more reliably and efficiently than PGPE.

In the Android-bot simulation task (Section 3.4.2.3), PGPE with the best learning rate performed worse than EPHEs. A possible reason is that, we used the same learning rate for updating all of the hyperparameters, even though the gradients for the mean and the variance of different parameters can be largely different. This reason might also have caused PGPE to be trapped in the local optima. In the hardware experiments, the results in Figure 33 showed that PGPE learned appropriate parameters with slow climbing. However, in a more complicated task, the results of Figure 37 showed that even with the best learning rate PGPE performed much worse

than EPHE-AB. Figures 34 and 38 showed different situations of the convergence of EPHE-AB and PGPE based on the learned hyperparameters, and provided a hint of the failures of PGPE resembled the Android-bot simulation task. PGPE might work better if we introduced different learning rates for different hyperparameters, but that would necessitate more samples for reliable estimates of the optimal learning rates. On the other hand, EPHE-AB realized adaptive step size based on simple reward-weighted mean and variances of the sampled parameters. It could almost reach the optimum in the hyper parameter space after only one or two iterations while PGPE might move slowly in the wrong direction.

EPHE-AB achieved good performances in the hardware experiments. In the standing-up and balancing tasks it achieved a 100% successful rate. The vision-based approaching task was much more challenging not only because of its additional task requirement but also due to the variable initial positions, which increased the variance in the performance evaluation. Nevertheless, EPHE-AB achieved a 40% success rate. By analyzing the learning history, we found that many of the episodes starting from the most difficult position X4 were discarded because of relatively low returns. A possible solution is to evaluate each sample of the parameters over multiple starting conditions.

PGPE worked much worse in the hardware experiments than in the simulation tasks. For the standing-up and balancing task in the hardware, the wheel angle was not usable and we used a simple binary reward function might be the reason. In the approaching while balancing task, the increased number of policy parameters indicates more difficulties to PGPE. As illustrated in Figure 38, such hyperparameters for steering the robot as $k_{cL}, k_{dL}, k_{cR}, k_{dR}$ in PGPE usually failed. Hence the robot never fairly spotted the target, so that it obtained low rewards.

3.4.5 Section Summary

In this section, we improved the updating mechanism of the EM-based Policy Hyperparameter Exploration (EPHE) method by computing an adaptive baseline (EPHE-AB) to discard inferior samples. Note that the adaptive baseline can be applied to the other methods under the EM-based policy

search framework. We verified the improvements in three simulation tasks, a pendulum swing-up with limited torque, cart-pole balancing, and a simulator of our smartphone balancer that compared to EPHE-CW, EPHE-RW, EPHE-K, PGPE and NES with the best learning rate and FEM. Our results showed that EPHE-AB-mean achieved the best performance among the other methods and a fixed baseline is inadequate to preserve informative samples. The choice of the baseline of mean is more sensitive to outliers than others in the early stage of learning, which is important for learning from primitive initial parameters and random initial states. We also implemented EPHE-AB with the baseline of mean in our real smartphone robot system to achieve two tasks, standing-up and balancing, and approaching a visual target while balancing. We compared EPHE-AB-mean with PGPE in hardware experiments. EPHE-AB-mean achieved nearly optimal behaviors in as few as 4 iterations in both tasks, demonstrating its efficient and tuning-free learning.

Note that in the episodic manner, many policy search methods derived from different principles are equivalent or similar, like EPHE-K, PoWER, and PI2, and EPHE-RW and REPS. In the future, we investigate the step-based EPHE for optimizing stochastic policies to reveal the difference between EPHE and the other algorithms. Since EPHE can use different probability distribution for policy parameters and hyperparameters, the different update rules can be derived in this case.

The EPHE methods were shown to be a promising approach in actual robotic tasks, but they discard a part of the training samples based on the return values. To overcome this data-inefficiency, a promising direction is the importance sampling technique that reuses samples over multiple iterations. Previous studies show that using an importance sampling technique achieves significant performance improvement in PGPE [46] and Reward-Weighted Regression [47]. EPHE can be straightforwardly integrated with importance sampling by considering the ratio between parameter distributions in different iterations. This extension will be left for future study.

3.5 Chapter Summary

In this Chapter, we focus on introducing intelligence into our smartphone balancer to learn basic behaviors such as standing-up and balancing and approaching to visual target, specifically under the domain of model free policy-based reinforcement learning. We first illustrated the problem setting of the framework, namely the Markov Decision Process and reviewed related policy-based reinforcement learning methods, such as Finite Difference, REINFORCE, PGPE, NES from policy gradient methods, RWR, PoWER and FEM from EM-based policy search methods. We developed a novel method EM-based Policy Hyperparameter Exploration (EPHE) combining the ideas of PGPE which evaluates a deterministic policy in each episode with the policy parameters sampled from a prior distribution given by the hyperparameters, and EM-based RWR which updates the parameters by reward-weighted averaging derived from a closed-form maximization of the lower bound of the objective function with respect to the hyperparameters of exponential family distributions. We introduced a K-elite selection mechanism to discard worse sample based on the sorted return history, and the simulation results of pendulum swing-up task, cart-pole balancing task and smartphone balancer task showed the improved performance compared to policy gradient methods like Finite Difference and PGPE. We further proposed an adaptive baseline to discard worse samples below certain baselines including mean, one and two standard deviation from the mean of a batch of collected trajectories, and examined the performances. The three simulation results showed enhancement of our method equipped with adaptive baseline with mean compared to methods like EPHE-CW, EPHE-RW, EPHE-K, PGPE, NES, and FEM. The simulation results of the comparison of three baselines suggested that the baseline with mean discarded most of the samples in the early stage of learning and the number of the discarded samples steadily decreases during learning. We then implemented our proposed method with adaptive baseline in the real smartphone balancer hardware system, and the balancer successfully achieved the required behaviors and outperformed the previous policy gradient method as PGPE, and revealed the insights between policy gradient methods and reward-weighted methods. With the learning skills, the smartphone balancer acquired basic mobilities such as standing-up, balancing and moving around based on the visual sensor, which can be regarded as a primitive autonomous individual from biological perspective.

Chapter 4

Towards High-level Behaviors

4.1 Introduction

The concept of Behavior-based Robotics (BBR) [13] [14] was introduced in the mid 1980s by Rodney Brooks. It is an approach in robotics that robots are able to exhibit complex-appearing behaviors despite internal state to model its immediate environment, mostly gradually correcting its actions via sensory-motor links. BBRs are required to be autonomous, distributed, layered on top of an existing behavioral substrate, and capable of learning in real time. It assumes a subsumption architecture of the robot behaviors, which can be visualized as a layered stack of parallel behaviors where the behaviors at the bottom deal with the survival, and the behaviors at the top achieve desirable goals if opportunity exists. Higher layers encompass lower layers, and high-level intelligent competences such as communication, and reasoning are believed emerged from the basic survival abilities such as foraging, mating and escaping. See Figure 3.

Constructing robots with the basic surviving skills can help us understand the emergence of complex behaviors, intelligence and the development of reward systems, reveal the social psychological phenomenons and gain the insights from the nature of evolution. In practice, we focus on foraging and mating behaviors. Foraging behavior is defined as an agent approaches to a food source and realizes battery charging. Mating behavior is defined as one agent approaches to another and two of them exchange genetic information through communication.

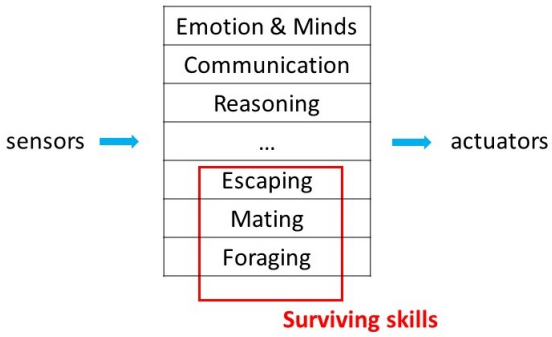


Figure 3. Subsumption architecture of robot behaviors

The first robot equipped with the foraging ability is the "Tortoise" developed by Walter in 1953 [16]. The behavior was replicated using a simple light following approach that allowed it to guide itself towards the recharge station in the arena. McFarland and Spier [17] defined a "basic cycle" consisting work - find station -recharge to investigate energy and utility in a self-sufficient autonomous mobile system [18]. Mataric proposed a reinforcement learning framework to achieve multi-robot foraging [19]. The behavior defines avoiding, dispersing, searching, homing and resting as behavior primitives. Floreano and Keller successfully achieved cooperative foraging using Khephera robots and further revealed the emergence of altruism [20]. Doya and Uchibe launched the "Cyber Rodent Project" [21], using groups of rodent like robots under the biological constraints of self-preservation (foraging) and self-reproduction (mating) to investigate the reward mechanism of robots and biological creatures. A Cyber Rodent can search for and recharge from battery packs and copy its program to another agent through infrared communication. The Cyber Rodent platform investigated different learning and evolution architectures for various behaviors [22] [23] [24], finding intrinsic and extrinsic rewards [25], examined Darwinian embodied evolution [26], and revealed the insights in the emergence of polymorphic mating strategies [27].

Our smartphone robot platform is one of the promising testbed for behavior-based and multiagent robotic research initiated by achieving foraging and mating task. In the previous chapters, the smartphone balancer realized basic behaviors such as standing-up, balancing, and approaching target while balancing, which can be regarded as lower-level fundamental behaviors of foraging and mating.

In this Chapter, we introduce a real-world environment for the smartphone balancer to habitat. We focus on vision based foraging and mating. We first show the hardware construction modified based on the previous version in Chapter 2. We then show experimental setting and results. We use the learning method consistent with the policy based reinforcement learning proposed in Chapter 3. The results show that the robots succeeded in foraging and mating task and indicating a potential in realizing higher level behaviors.

4.2 Foraging Task

4.2.1 Hardware and Environment Construction

The hardware components are consistent with Section 2.2.2 including, Nexus 4, IOIO OTG board, Connect Break Out Board and HUB-EE Wheel. We use the "Cheero Power Plus 3 mini" battery with capacity of 6700mAh and connect its four LED indicators to IOIO OTG pin 22 to 25 as digital inputs to obtain the battery level. We use Qi wireless charging sheet connected to the battery through USB micro-b, and Aukey wireless charging plate. See Figure 42 of the cheero battery, charging sheet and charging plate. Charging begins when the sheet is on the top of plate within maximum 3 mm distance.

The charging plate (radius 3.5 cm) is placed in the center of a green circle (radius 44 cm), namely the food source indicator. The battery and charging sheet (largest sensable length 5 cm) are located in the bottom of the robot. Once the agent arrives at the charging spot above the charging plate, the battery starts charging. See Figure 43 of the food source indicator.

Figure 42. Cheero battery, charging sheet and charging plate

Figure 43. Food source indicator

4.2.2 Experimental Setting

This task requires the smartphone balancer learning to approach the center of food source indicator namely the charging plate and start charging. We use the front camera of the Android phone and detect food source indicator by color blob detection using OpenCV library same as in Section 3.4.3.2.

The state variables in this task are $\boldsymbol{x} = \{\vartheta, \dot{\vartheta}, \dot{\varphi}_L, \dot{\varphi}_R, \Delta, D\}$, where Δ is the difference of extreme points of lefttop and righttop on Y axis of the blob, and D is the distance between the robot and the charger. D from 0 to 10 dm is determined by

$$D = b_0 + b_1\vartheta + b_2 c_y + b_3 \vartheta c_y$$

calculated by function `regress`() in MATLAB. Note that the robot arrives at charging plate when $D = 0$. Δ is normalized to $\Delta \in [-0.3, 0.3]$ as a variable for steering. We calculate the blob's center position c_x on X axis and c_y on Y axis. They are normalized to $c_x \in [-0.3, 0.3]$ and $c_y \in [1, 0]$. See Figure 44.

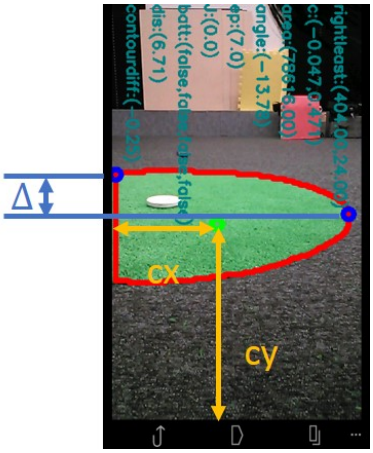


Figure 44. Illustration of state variables of foraging task

When the target is invisible, $\Delta = \pm 0.4$ and $D = 15$. When the target is visible, the immediate reward at time t is given by

$$r_t = \exp\left(-w_1\left(\frac{\vartheta_t}{Z_\vartheta}\right)^2 - w_2\left(\frac{\Delta_t}{Z_\Delta}\right)^2 - w_3\left(\frac{D_t - d_{target}}{Z_D}\right)^2\right),$$

where Z_ϑ, Z_Δ, and Z_D are respectively the constants to normalize ϑ_t, Δ, $D_t - 5$ to [-1,1] and d_{target} is 0. We assign $w_1 = w_2 = 1.2$, and $w_3 = 0.6$. When the target is not visible, the reward is $r_t = 0$. When the battery is charged, namely, the robot receives a digital signal, we add an extra reward $r_b = \frac{1}{3}T\,(4000)$ to the return, stop the robot and end episode. The cumulative return for one episode was computed by

$$R(h) = r_b + \sum_{t=1}^{T} r_t,\ T = 12000.$$

The initial position of the robot is at the edge of the circle.

The following is the control structure:

When ϑ is inside the threshold $\vartheta_{thre} \in [-20^o, 20^o]$, and the linear feedback stabilizer is activated:

$$u_L = -\Big(k_{\vartheta_L}\vartheta + k_{\dot{\vartheta}_L}\dot{\vartheta} + k_{\dot{\varphi}_L}\dot{\varphi}_L + \mathrm{ref}_L + k_{\Delta L}\Delta + k_{dL}D\Big)$$

$$u_R = -\Big(k_{\vartheta_R}\vartheta + k_{\dot{\vartheta}_R}\dot{\vartheta} + k_{\dot{\varphi}_R}\dot{\varphi}_R + \mathrm{ref}_R + k_{\Delta R}\Delta + k_{dR}D\Big),$$

Otherwise, the CPG-based destabilizer is activated as (3.14).

The policy parameters are 14-dimensional

$$\boldsymbol{\theta} = \{k_{\vartheta_L}, k_{\dot{\vartheta}_L}, k_{\dot{\varphi}_L}, k_{\vartheta_R}, k_{\dot{\vartheta}_R}, k_{\dot{\varphi}_R}, \mathrm{ref}_L, \mathrm{ref}_R, k_{\Delta L}, k_{dL}, k_{\Delta R}, k_{dR}, \omega, \beta\},$$

where $k_{\vartheta_L}, k_{\dot{\vartheta}_L}, k_{\dot{\varphi}_L}, k_{\vartheta_R}, k_{\dot{\vartheta}_R}, k_{\dot{\varphi}_R}$ are the linear feedback control gains for stabilization, ref_L and ref_R are the reference constants for the wheel to move forward and backward, $k_{\Delta L}, k_{dL}, k_{\Delta R}, k_{dR}$ are the control gain for the steering, and ω, β are the CPG parameters. $k_{\Delta L}$ and $k_{\Delta R}$ can cause rotation toward the target, but we set up all the parameters differently for the left and right wheels to adapt to any asymmetry in the hardware and the electronics.

The sampling rate is 0.005 sec for each step, and one episode is 12000 steps (=60 sec). The hyperparameters are updated every 20 episodes for 10 iterations, resulting in 200 episodes for one entire run.

We first let the robot run from the primitive initial hyperparameters based on the optimal parameters obtained from the previous section:

$$\boldsymbol{\eta}_0 = [290, 955, 4, 226, 1096, 9, 204, 189, -2000, 0, 2000, 0, -16, 38]^T$$

$$\boldsymbol{\sigma}_0 = [50, 100, 10, 50, 100, 10, 50, 50, 500, 100, 500, 100, 10, 10]^T.$$

After the robot knows the right direction to approach, namely a smoother standing-up, balancing and approaching behavior to the food indicator, we use the obtained hyperparmeters to learn foraging.

$$\boldsymbol{\eta}_0 = [255, 1147, -6, 248, 1252, 3, 216, 164, -1520, 144, 2830, 112, -14, 21]^T$$

$$\boldsymbol{\sigma}_0 = [50, 100, 10, 50, 100, 10, 50, 50, 200, 100, 200, 100, 10, 10]^T.$$

4.2.3 Results

We run 5 independent runs and measure the average and the standard error of the cumulative returns against the iterations of hyperparameter updating. An episode is regarded as successful if the robot could stand up, move smoothly toward the center of the food source indicator, stop and keep balancing above the target, receive a digital signal from the charging plate and start charging. A successful learning was defined as when the robot could find optimal policy hyperparameters within 10 iterations. The successful rate of learning is 2/5.

Since it is an integrated behavior including standing-up, balancing, navigation and approaching, many cases of successes and failures occur. The sensitivity of the charging sheet and limited sensing area require the robot slowly approaching the charging plate. The best behavior is regarded as the robot could stand up and directly approach the food source while balancing without bouncing. However, in some cases the robot passed the food source due to the fast speed and turned around to approach again. During this process, it might lose balance and bounce to stand up several times. As long as the robot can approach and keep balancing above the center, it is treated as successful. Failures include the robot spent much time on learning balancing instead of moving, the robot could not navigate the target fairly, even the target was spotted the robot could not stabilize and approach in a certain speed, even the robot could approach to the goal position smoothly it failed charging.

Figure 45 shows a clear learning curve of the foraging task.

Figure 46 shows the successful charging frequency against the policy updating iteration in each run. Run 2 and Run 3 are regarded as successful with a relatively increasing tendency for the successful charging frequencies. In the early stage of learning, some time the robot succeeded in sensing the charging plate, but it was not regarded as optimal policy due to the other behaviors were not synchronized. Most of the successful charging appears in the end of the learning.

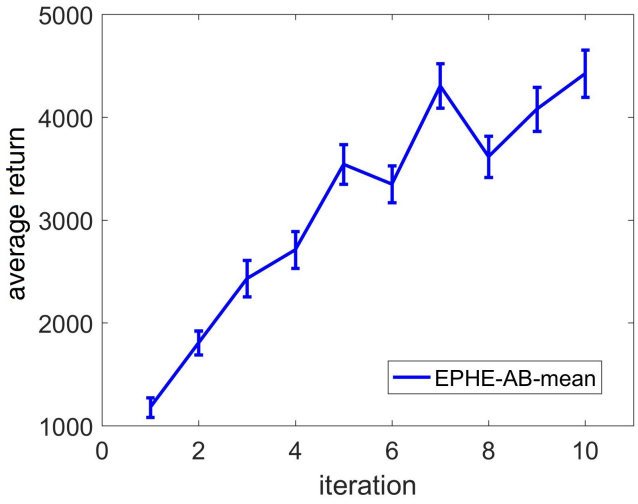


Figure 45. Learning curves of EPHE-AB-mean in vision-based foraging task

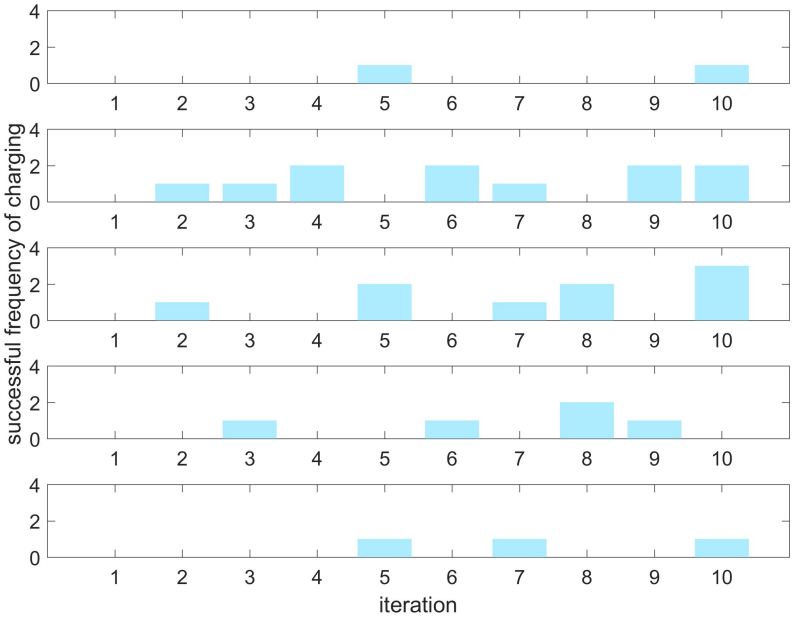


Figure 46. Successful charging frequency against the policy updating iteration in each run

4.3 Mating Task

4.3.1 Experimental Setting

This task requires a smartphone balancer (sender) learning to approach another smartphone balancer (receiver) as its mating partner. As illustrated in Figure 47, the receiver shows a heart shape figure as a visual signal to attract the sender. The sender's front camera detects the signal by color blob detection using OpenCV library same as in the previous sections. Once the sender finds its distance from the receiver is lower than a threshold, it shows a qr code incorporates its genetic information, such as policy parameters. The receiver detects the qr code also by the front camera, and shows a blue figure indicating the information received. The sender again detects the blue color blob, and confirms the success of the information exchanged.

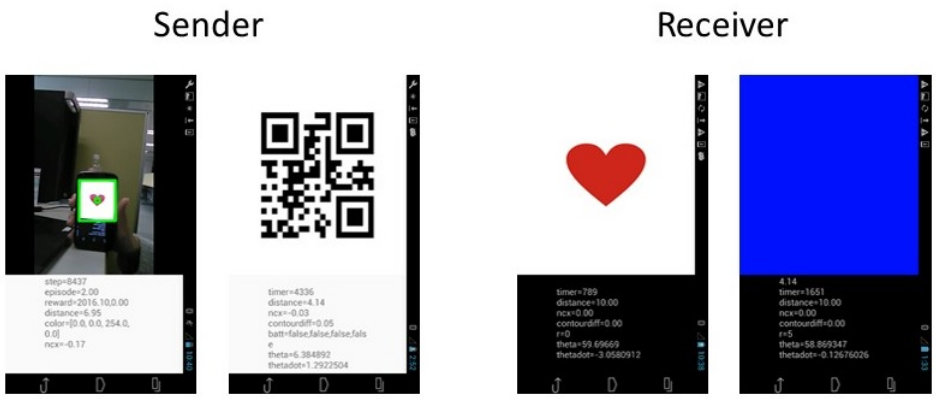


Figure 47. Sender and Receivers' visual signals for mating task

The communication between the sender and the receiver is as follows:

Sender	Receiver
	1. show heart figure
2. detect heart figure as a blob	
3. if $d \leq d_{thre}$, show qr code	
	4. if qr detected, show blue figure
5. if blue figure detected, obtain reward	
end episode	

Table 6. Communication between sender and receiver

For simplicity, we first fix the receiver and let the sender learn to mate unilaterally. The state variables of the sender are $\boldsymbol{x} = \{\vartheta, \dot{\vartheta}, \dot{\varphi}_L, \dot{\varphi}_R, c_x, D\}$, where $c_x \in [-0.3, 0.3]$ is the blob center on the X axis, and $D \in [0, 10]$ is the distance between the sender and the receiver calculated by the inverse of the square root of the blob area.

When the target is invisible we set $c_x = \pm 0.4$ based on the previous sign of c_x, and $D = 10$. When the target is visible the immediate reward at time t is given by

$$r_t = \exp\left(-w_1\left(\frac{\vartheta_t}{Z_\vartheta}\right)^2 - w_2\left(\frac{c_{xt}}{Z_{c_x}}\right)^2 - w_3\left(\frac{D_t - d_{target}}{Z_D}\right)^2\right),$$

where Z_ϑ, Z_Δ, and Z_D are respectively the constants to normalize ϑ_t, Δ, $D_t - 5$ to [-1,1], and d_{target} is 3 (30 cm). We assign $w_1 = w_2 = 1.2$, and $w_3 = 0.6$.

The sender starts running $D = 10$ (100 cm) from the receiver, when $D \leq 3$ (30 cm) it shows the qr code. The detectable range of the receiver is $D \in [1, 3]$. When mated signal is received by the sender, namely the blue figure is detected, we add an extra reward $r_m = \frac{2}{3}T$ to the return, stop the robot and end episode. When the target is not visible, the reward is $r_t = 0$. The cumulative return for one episode is computed by

$$R(h) = r_m + \sum_{t=1}^{T} r_t,\ T = 3000.$$

The following is the control structure:

When ϑ is inside the threshold $\vartheta_{thre} \in [-20^o, 20^o]$, and the linear feedback stabilizer is activated:

$$u_L = -\Big(k_{\vartheta_L}\vartheta + k_{\dot{\vartheta}_L}\dot{\vartheta} + k_{\dot{\varphi}_L}\dot{\varphi}_L + \mathrm{ref}_L + k_{cL}c_x + k_{dL}D\Big)$$

$$u_R = -\Big(k_{\vartheta_R}\vartheta + k_{\dot{\vartheta}_R}\dot{\vartheta} + k_{\dot{\varphi}_R}\dot{\varphi}_R + \mathrm{ref}_R + k_{cR}c_x + k_{dR}D\Big),$$

Otherwise, the CPG-based destabilizer is activated as (3.14).

The policy parameters are 14-dimensional

$$\boldsymbol{\theta} = \{k_{\vartheta_L}, k_{\dot{\vartheta}_L}, k_{\dot{\varphi}_L}, k_{\vartheta_R}, k_{\dot{\vartheta}_R}, k_{\dot{\varphi}_R}, \mathrm{ref}_L, \mathrm{ref}_R, k_{cL}, k_{dL}, k_{cR}, k_{dR}, \omega, \beta\},$$

where $k_{\vartheta_L}, k_{\dot{\vartheta}_L}, k_{\dot{\varphi}_L}, k_{\vartheta_R}, k_{\dot{\vartheta}_R}, k_{\dot{\varphi}_R}$ are the linear feedback control gains for stabilization, ref_L and ref_R are the reference constants for the wheel to move forward and backward, $k_{cL}, k_{dL}, k_{cR}, k_{dR}$ are the control gain for the steering, and ω, β are the CPG parameters. k_{cL} and k_{cR} can cause rotation toward the target, but we set up all the parameters differently for the left and right wheels to adapt to any asymmetry in the hardware and the electronics.

The sampling rate is 0.005 sec for each step, and one episode is 3000 steps (=15 sec). The hyperparameters are updated every 20 episodes for 10 iterations, resulting in 200 episodes for one entire run.

The initial hyperparameters were set up based on the optimal parameters we obtained from the section 3.4.3.1:

$$\boldsymbol{\eta}_0 = [290, 955, 4, 226, 1096, 9, 204, 189, -2000, 0, 2000, 0, -16, 38]^T$$

$$\boldsymbol{\sigma}_0 = [50, 100, 10, 50, 100, 10, 50, 50, 500, 100, 500, 100, 10, 10]^T$$

4.3.2 Results

We run 5 independent runs and measure the average and the standard error of the cumulative returns against the iterations of hyperparameter updating. An episode is regarded as successful if the sender could stand up, move smoothly toward the receiver and complete the information exchange successfully. A successful learning is defined as when the robot could find optimal policy hyperparameters within 10 iterations. The successful rate of learning is 3/5.

This task is relatively simple because the initial position of two mates is face to face, so we shortened the learning time as the time length T for each episode was only 3000 steps. The most observed successful behaviors were the sender bounced several times along the way to the receiver and finally kept balanced and completed the information exchange. The best behaviors were the fastest ones accomplishing the task with less bouncing. The key behavior was the sender's stability of balancing and keeping a certain distance for the receiver to scan the qr code easily. Failures include the sender could not find the right direction to the receiver, the sender could not stop in front of the receiver (bumping the receiver), and the sender could not stand-up, and keep balancing close to the receiver.

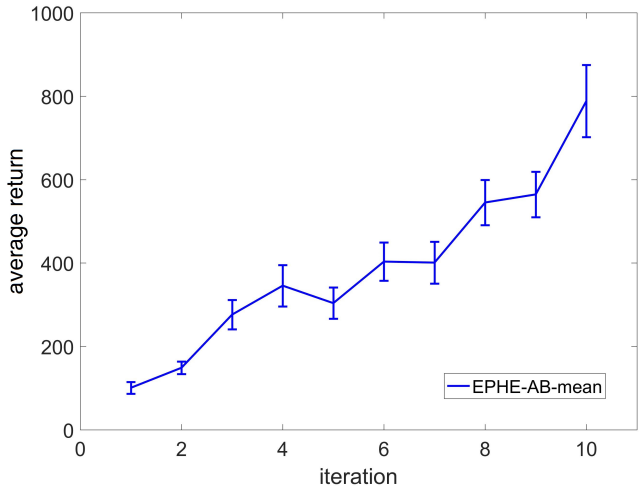


Figure 48. Learning curve of EPHE-AB-mean in vision-based mating task

Figure 48 shows a learning curve of the mating task.

Figure 49 shows the successful mating frequency against the policy updating iteration in each run. With the successful runs (Run 1, Run 3, Run 5), the successful mating frequency has a smooth increase during learning. This suggests that the setting of vision-based guiding reward is consistent with the sparse mating reward.

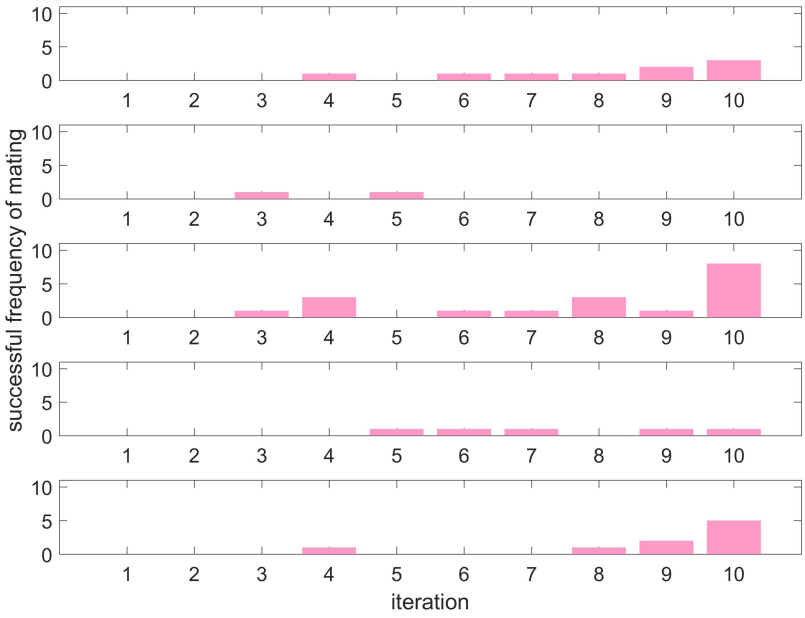


Figure 49. Successful mating frequency against the policy updating iteration in each run

4.4 Discussion

In the foraging task, we found a gap between the vision reward and the actual charging reward, even the charging behavior was achieved. Specifically, the increasing cumulative returns of vision do not correspond to successful charg-

ing. And the successful charging some time occurs with low vision returns. The reasons are:

1) The setting of reward function of vision is not accurate enough to fit the charging requirement.

2) The reward function does not describe the charging behavior appropriately. The successful charging some time requires the robot slowly passing the charging plate instead of stopping exactly above it. And if the robot passes the charging plate too fast, it would turn around and re-navigate. During this behavior, it would not gain much vision rewards.

Particularly, we need to reconsider the reward function as

$$r_t = \exp\left(-w_1\left(\frac{\vartheta_t}{Z_\vartheta}\right)^2 - w_2\left(\frac{\Delta_t - \Delta_{target}}{Z_\Delta}\right)^2 - w_3\left(\frac{D_t - d_{target}}{Z_D}\right)^2\right)$$

by treating $\Delta_{target}, d_{target}, w_1, w_2, w_3$ as tunable parameters.

3) The robot actually does not know its precise location near the optimal charging area, in which case the vision information is inadequate. We might use some hand-coded behavior for the robot to find the charging spot blindly.

The mating task has a high similarity with the vision-based approaching task in Section 3.4.3.2 if the receiver is fixed. The initial policy hyperparameters were reasonable however the distancing parameters k_{dL} and k_{dR} were learning from 0, which caused difficulties for a good braking. We still need to consider appropriate penalties as reward shaping for smoother behaviors.

Setting up the initial policy hyperparameters can be tricky and finding a good convergent direction in the early stage of learning is crucial. These two factors severely affect the entire behavior. For example, if the parameters in charge of standing-up were not synchronized well, the agent would spend most of the time learning standing up instead of navigating and approaching. Therefore, all the policy parameters are related for a smooth behavior, that is they are formed as a certain set and should be cooperative with each other.

Our next steps are investigating these factors more detailedly, and further

achieving collision avoidance and multi-agent cooperative mating.

4.5 Chapter Summary

In this chapter, we viewed our robot system under a broader spectrum of research topics, including behavior-based robotics inspired from ethology and psychology, a sustainable multi-agent society for understanding reward mechanism and emergence behaviors. Among these topics, the essential and prerequisite behaviors are foraging and mating. We implemented our proposed EPHE-AB method from Chapter 3 in the smartphone balancer system from Chapter 2 to achieve the foraging and mating behavior under a low-cost hardware environment. Results showed the success of the achievement, although the design of the reward system remains to be improved.

Chapter 5

Conclusion and Future Work

Our contributions in this thesis are twofold. First, we developed an affordable high-performance smartphone based robotic platform, namely a smartphone balancer which allows various behavioral realizations. This platform is not only a practical testbed for optimal control and reinforcement learning algorithm design, but also has high potential in the fields of behavior-based robotics and multi-agent research. It is effortless to construct and maintain, and friendly to both beginners and specialists. It is also a powerful platform that can benefit multi-disciplinary research topics, such as mimicking biological societies for ethology and psychology or understanding reward mechanisms towards animal and human's minds.

Second, we proposed a novel policy based reinforcement learning method, EM-based Policy Hyperparameter Exploration (EPHE) and an adaptive baseline which can be equipped to EPHE to realize various behaviors of the smartphone balancer. The advantages of our methods are, 1) the controller is deterministic, 2) it does not require the controller to be differentiable, 3) it avoids hand-tuning of the learning rate, which are trouble-saving and practical in real robot experiments.

We first illustrated the design and control of the smartphone balancer and verified the feasibility of proposed platform. The implementation of the proposed algorithms are first conducted in simulation tasks, including benchmark pendulum swing-up with limited torque task, and cart pole balancing task, and the standing-up and balancing task in the smartphone balancer simulator. We compared EPHE with K-elite mechanism (EPHE-K) to discard worse samples with previous algorithms like PGPE and Finite Difference. The results showed the superiority of our method and the necessity of discarding worse samples. However, selecting a fixed baseline by user might require repeated running of the experiments. To further investigate the discarding mechanism, we proposed an adaptive baseline with EPHE (EPHE-AB) to discard worse samples below the average of the return history (EPHE-AB-mean), and compared it with EPHE-K, EPHE with CMAES weighting scheme, EPHE with REPS weighting scheme, and previous methods such as PGPE, NES and FEM. We also tested EPHE-AB with different baseline, including one and two standard deviation from the mean. The simulation results showed that EPHE-AB-mean outperformed the other methods and baseline settings in the early stage of learning. The choice of baseline of mean results in a steady decrease of discarding numbers of the samples,

and suggested the competence to pick up positive outliers in the beginning of learning. We then applied the EPHE-AB-mean and compared it with PGPE in the real smartphone balancer system to achieve behaviors of standing up and balancing and approaching to a vision-based target. The results showed that our algorithms outperformed PGPE significantly, and the required behaviors were successfully achieved.

Finally, due to the potential of the smartphone robot platform in the field of behavior-based and multiagent robotics, we applied our methods in the smartphone balancer to achieve high level behaviors such as foraging and mating. The results showed the feasibility.

Currently, the EPHEs are episodic methods, and equivalent to other policy search methods like PoWER and PI^2. EPHE with REPS weighting scheme is equivalent to episodic REPS. In the future we would like to investigate the step-based EPHE for optimizing stochastic policies to reveal the difference between EPHE and the other methods. Since EPHE can use different probability distribution for policy parameters and hyperparameters, the different updating rules can be derived. On the other hand, incorporating importance sampling technique that reuses samples over multiple iterations with EPHE are also straightforward, due to its data-inefficiency of discarding samples only in every update of hyperparameters.

Appendix

The equation of motion of pendulum swing-up task in Section 3.3.2.1 and 3.4.2.1 is given by

$$ml^2\ddot{\varphi} = -\mu\dot{\varphi} + mgl\sin\varphi + u$$

where $m = 1$ [kg] is the pendulum mass, $l = 1$ [m] is the pendulum length, $= 0.01$ [kg m^2/s] is the coefficient of friction, and $g = 9.81$ [m/s^2] is the gravitational constant.

The equation of motion of the cart-pole balancing task in Section 3.3.2.2 and 3.4.2.2 is given by

$$\ddot{x} = \frac{F - m_p l(\ddot{\varphi}\cos\varphi - \dot{\varphi}^2\sin\varphi)}{m_c + m_p}$$

$$\ddot{\varphi} = \frac{g\sin\varphi(m_c + m_p) - (F + m_s l\dot{\varphi}^2\sin\varphi)\cos\varphi}{4/3l(m_c + m_p) - m_p l\cos^2\varphi}$$

where $m_c = 1$ [kg] is the cart mass, $m_p = 0.1$ [kg] is the pole mass, $l = 0.5$ [m] is the half length of the pole, $g = 9.81[m/s^s]$ is the gravitational constant and $-10N \leq F \leq +10N$ is the force applied to the cart.

Bibliography

[1] Andrew Y Ng, H Jin Kim, Michael I Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning.

[2] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research*, 32(3):263–279, 2013.

[3] Honglak Lee, Yirong Shen, Chih-Han Yu, Gurjeet Singh, and Andrew Y Ng. Quadruped robot obstacle negotiation via reinforcement learning. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3003–3010. IEEE, 2006.

[4] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56. ACM, 2008.

[5] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[6] Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[7] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, 2010.

[8] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

[9] Jan Peters and Stefan Schaal. Applying the episodic natural actor-critic architecture to motor primitive learning. In *ESANN*, pages 295–300, 2007.

[10] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. In *Advances in neural information processing systems*, pages 849–856, 2009.

[11] Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Fitness expectation maximization. In *International Conference on Parallel Problem Solving from Nature*, pages 337–346. Springer, 2008.

[12] Marin Kobilarov. Cross-entropy randomized motion planning. *Robotics: Science and Systems VII*, page 153, 2012.

[13] Rodney Allen Brooks. *Cambrian intelligence: the early history of the new AI*, volume 1. MIT press Cambridge, MA, 1999.

[14] Ronald C Arkin. *Behavior-based robotics*. MIT press, 1998.

[15] Stefano Nolfi and Dario Floreano. Evolutionary robotics, 2000.

[16] W Walter. The living brain. 1953.

[17] David McFarland and Emmet Spier. Basic cycles, utility and opportunism in self-sufficient robots. *Robotics and Autonomous Systems*, 20(2):179–190, 1997.

[18] David McFarland. Animal robotics-from self-sufficiency to autonomy. In *From Perception to Action Conference, 1994., Proceedings*, pages 47–54. IEEE, 1994.

[19] Maja J Matarić. Reinforcement learning in the multi-robot domain. In *Robot colonies*, pages 73–83. Springer, 1997.

[20] Dario Floreano and Laurent Keller. Evolution of adaptive behaviour in robots by means of darwinian selection. *PLoS Biol*, 8(1):e1000292, 2010.

[21] Kenji Doya and Eiji Uchibe. The cyber rodent project: Exploration of adaptive mechanisms for self-preservation and self-reproduction. *Adaptive Behavior*, 13(2):149–160, 2005.

[22] Eiji Uchibe and Kenji Doya. Competitive-cooperative-concurrent reinforcement learning with importance sampling. In *Proc. of International Conference on Simulation of Adaptive Behavior: From Animals and Animats*, pages 287–296, 2004.

[23] Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I Christensen. Biologically inspired embodied evolution of survival. In *2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2210–2216. IEEE, 2005.

[24] Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I Christensen. Evolutionary development of hierarchical learning structures. *IEEE transactions on evolutionary computation*, 11(2):249–264, 2007.

[25] Eiji Uchibe and Kenji Doya. Finding intrinsic rewards by embodied evolution and constrained reinforcement learning. *Neural Networks*, 21(10):1447–1455, 2008.

[26] Stefan Elfwing, Eiji Uchibe, Kenji Doya, and Henrik I Christensen. Darwinian embodied evolution of the learning ability for survival. *Adaptive Behavior*, 19(2):101–120, 2011.

[27] Stefan Elfwing and Kenji Doya. Emergence of polymorphic mating strategies in robot colonies. *PloS one*, 9(4):e93622, 2014.

[28] Francesco Mondada, Edoardo Franzi, and Andre Guignard. The development of khepera. In *Experiments with the Mini-Robot Khepera, Proceedings of the First International Khepera Workshop*, number LSRO-CONF-2006-060, pages 7–14, 1999.

[29] Yoshida Naoto. Development of robot platform with smart phone. Master's thesis, NAIST, 2012.

[30] Yoshida Naoto, Junichiro Yoshimoto, Eiji Uchibe, and Kenji Doya. Development of robot platform with smart phone. In *annual Conference on Robotics Society of Japan*, 2012.

[31] ZMP INC. Stabilization and control of stable running of the wheeled inverted pendulum development of educational wheeled inverted robot e-nuvo wheel ver.1.0. Technical report.

[32] Zhijun Li, Chenguang Yang, and Liping Fan. *Advanced control of wheeled inverted pendulum systems*. Springer Science & Business Media, 2012.

[33] Segway. `http://www.segway.com`.

[34] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, pages 262–270, 2011.

[35] Jan Peters and Katharina Mülling. Relative entropy policy search. 2010.

[36] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[37] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11(Nov):3137–3181, 2010.

[38] Freek Stulp and Olivier Sigaud. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.

[39] Abbas Abdolmaleki, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. Regularized covariance estimation for weighted maximum likelihood policy search methods. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 154–159. IEEE, 2015.

[40] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[41] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[42] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[43] Kenji Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245, 2000.

[44] Martin Riedmiller, Jan Peters, and Stefan Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 254–261. IEEE, 2007.

[45] Andras Gabor Kupcsik, Marc Peter Deisenroth, Jan Peters, and Gerhard Neumann. Data-efficient generalization of robot skills with contextual policy search. In *AAAI*, 2013.

[46] Tingting Zhao, Hirotaka Hachiya, Voot Tangkaratt, Jun Morimoto, and Masashi Sugiyama. Efficient sample reuse in policy gradients with parameter-based exploration. *Neural computation*, 25(6):1512–1547, 2013.

[47] Hirotaka Hachiya, Jan Peters, and Masashi Sugiyama. Reward-weighted regression with sample reuse for direct policy search in reinforcement learning. *Neural Computation*, 23(11):2798–2832, 2011.