MODELLING HUMAN TARGET REACHING USING A NOVEL PREDICTIVE DEEP REINFORCEMENT LEARNING TECHNIQUE

by

Farzaneh Sheikhnezhad Fard

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
March 2018

To my beloved husband, my best friend, and my everything,
who always makes me the happiest and luckiest person in the world,
HOSSEIN,

To my amazing father, who is my first and my strongest motivation to
pursue my dreams, and taught me honesty and integrity,
ALI,

To my awesome mother, who taught me hard work and patience
always pays off,
MAHROKH,

and last but not least, to my lovely in-laws, who always support and
love me thoroughly with their endless love,
BELGHIS and MOHAMMAD EBRAHIM.

Table of Contents

List of Tables

List of Figures

Abstract

It is hypothesized that the brain builds an internal representation of the world and its body. Moreover, it is well established that human decision making and instrumental control uses multiple systems, some which are habitual and some which require planning.

In this thesis, we proposed a novel model called **adaptive observer**[1] that learns the internal representation of the world using dynamic neural fields (DNF). DNF is a well-known model that simulates brain activity in cortical tissues. By DNF the activity of the population of neurons is being considered instead of the activity of only one single neuron. Later, we introduce a model called **arbitrated predictive actor-critic (APAC)** [2, 4]. In APAC, we proposed a general architecture comprising both habitual and planning control paradigms by introducing an arbitrator that controls which subsystem is used at any time.

Both adaptive observer and APAC imply the internal model, however, they are different in some aspects. For example, the adaptive observer, unlike APAC, uses DNFs to represent neural activities. While, APAC, unlike the adaptive observer, can learn the kinematics of the system without a prior knowledge and combines two control systems for decision making. APAC model takes advantage of a fast habitual controller when it is reliable enough.

Both models are studied and tested under different conditions on a target reaching task. The adaptive observer was tested with a real robotic arm, while the APAC was examined with a simulated robot arm. In adaptive observer, a path integration technique is implied to reach the target. Such adaptive observer can also explain some interesting features and behaviours in the brain, namely moving with impaired sensory input, and motor adaptation. Through permutation of target-reaching conditions, we also demonstrate that APAC is capable of learning kinematics of the system rapidly without a priori knowledge and is robust to (A) changing environmental reward and kinematics, and (B) occluded vision. The arbitrator model is compared to pure planning and pure habitual instances of the model.

Acknowledgements

On completing my thesis, I owe a debt of gratitude to a number of people for their support.

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Thomas Trappenberg, for his tirelessness, enthusiasm, motivation, and immense knowledge. I owe him for all the opportunities he provided for me. I was lucky to have him as a supervisor and an advisor who walked me through all difficulties both in academic and non-academic issues.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. David Westwood, and Dr. Malcolm Heywood, and Dr. Sageev Oore, for their insightful comments and encouragement. I would also thank Dr. Mae Seto and Dr. Evangelos Milios for their help and support during my aptitude and proposal defence. I would like to thank my external committee Dr. Jeff Krichmar. It is with great honour that Dr. Krichmar accepted to be my committee member.

I thank my smart and hard-working colleague Paul Hollensen. I cannot imagine being able to finish my degree without his help and support in first years of my PhD. I learned a lot from all interesting discussions we had. I would also like to thank my dear friend, Pam Williams, former manager of the international office at Dalhousie, who is my great support and source of positive energy since the first day that I met her in the orientation day almost 5 years ago.

I thank all my family and friends for supporting me spiritually. To name a few, I thank Majid, Misagh, Farid, Parvaneh, Fariborz, Hossein, Ehsan, Farzad, Sara, Sahar, Setareh, and many others who became my new family and make me feel at home where I am far far away from home.

And last, but by no means least, I thank my husband, Dr. Hossein Parvar, for his patience, endless support, and all sacrifices he did and always does for me. Thanks for always being there for me.

Chapter 1

Introduction

Arthur Samuel [5] defined the *machine learning* (ML) term for the first time in 1959 which gives *"computers the ability to learn without being explicitly programmed"*. Many algorithms and methods introduced to obtain this ability for machines. For example TD-Gammon [6] to play back gammon, or AlphaGo [7] that plays Go at a championship level. Despite all efforts and attempts, still, there are many challenging problems that machine learning has not been successful in learning.

The concept of machine learning is learning from training data to make a better prediction of unseen data. ML tasks are divided into three categories, namely 1) unsupervised learning, 2) supervised learning, and 3) reinforcement learning (RL).

In *supervised learning*, the input is provided with labels and the actual prediction error can be computed. The goal of supervised learning is to map the input to the given labels. In *unsupervised learning*, input has no label and the agent learns patterns of the input or infers a function to describe a hidden structure of the input. In *reinforcement learning*, the agent learns to maximize received reward from the environment by taking a series of actions to reach a certain goal.

The solution for any particular problem is defined based on the type of the learning *signal* or *feedback* available to a learning system [8]. The type of feedback or learning signal changes from case to case which is very important in order to solve the problem. In some cases, the feedback is accurate and is available in form of the actual error signal, while in some other cases, the feedback is a rough or even a binary reward signal. Moreover, sometimes the feedback signal is available immediately, whereas in some cases the feedback of the taken action will be delayed.

Much of the current reinforcement learning (RL) literature is in the domain of model-free control. Such a learning agent learns a value function from interacting with the environment, usually updating a proposed value function from a temporal difference between the previous expectation and a new experience [9, 10]. The value function is like a big lookup-table that can quickly supply evaluations for possible actions and hence provides guidance for actions in a fast and somewhat automated way which is characterized as habitual since it learns by repeating some actions. The habitual action selection, however, takes time to learn and requires that similar previous situations have been encountered sufficiently, the advantage is that after learning, decisions and correspondingly actions can be generated very fast.

In contrast, a system that has some internal models of the environment can be used to derive a value function on demand for a specific situation. A prime example is a Markov decision problem where the reward function and transition function of the agent are known so that the Bellman equations can be used to calculate the optimal value function for every state action pair without interacting with the environment. Of course, this system requires learning of the internal models, which requires previous interactions with the environment. The learning of internal models can be achieved through some form of supervised learning. Once the models have been learned, the model-based system is able to calculate a value function on the fly. This resembles some form of internal deliberation. The advantage of such a system is its flexibility to new situations. However, deliberations take time so that a habitual system is preferable when it comes to situations that benefit from a higher degree of automation.

1.1 Task and Problem

Human target reaching is an interesting problem that has been studied for a long time. Although there are various types of robotic arms that have been used in industries and manufacturing, there is still no good model to simulate how our brain controls arm movements and reaching. In industrial robotics, robots are used to perform dull, dirty and dangerous tasks instead of humans. They have been programmed very accurately and precisely to do a certain task under certain circumstances [11]. In contrast, in cognitive robotics, robots deal with cognitive phenomena such as perception, attention, anticipation, planning, memory, learning, and reasoning [12]. Cognitive robotics is an interdisciplinary area for robotics and cognitive science.

Cognitive robotics has two major aspects. One aspect is developing robots that can mimic human behavior, which leads us to a more flexible and adaptive robots which are safe to interact and cooperate with human. The other aspect is obtaining better understanding of the brain and try to help people with brain disorders, by studying and testing the theories of how the brain works onto robots and analyze robot's behavior [13]. Indeed similar to *animal model* in psychology, which is used to study psychological or psycho-pathological process that is similar to a human condition, *robot model* is used in cognitive robotics. For example, industrial robots vastly relies on visual information (for instance, the KUKA arm that plays ping-pong), however, humans are able to reach and grasp a target even with occluded vision. Moreover, humans can decide and arbitrate between different tasks and decide on what to do next. These concepts were initial incentives for us to look for a new model to simulate these features of the brain onto robots.

In this dissertation, the focus is on the target reaching problem which is a complex problem with a continuous state/space characteristics. In the reaching task, there are three ill-posed problems. First, ill-posed problem is due to an existence of an infinite number of trajectories between two points. The second ill-posed problem relates to the inverse kinematic of the plant (e.g arm). And the third ill-posed problem is inverse dynamics problem in which motor commands should map onto actuators (e.g muscles) [14]. Moreover, the complexity of the problem grows dramatically with an increase in the degree of freedom. Learning the reaching task in this 2D environment is learning a non-linear mapping function that maps joint angles of the robot arm onto a location of the end-effector in the environment.

Target reaching problems can be divided into two types, either one-step mapping or multi-step reaching. In one-step mapping, the task is learning the mapping between angles of the joint to x-y coordinate where the target is located. However, in multi-step reaching the task becomes more like a path-planning problem where the trajectory of the movement is important.

In this dissertation,we introduce two learning systems for target reaching using different approaches. One learning system is called the **Adaptive Observer** that learns the internal model of the agent using dynamic neural fields (DNFs) [1]. DNF is a successful model to simulate brain activities in cortical tissues. Strauss and Heinke [3] implemented a robotics-based approach for target reaching with DNFs to investigate current theories of human behaviour. Their main goal was investigating the dynamic interaction between a decision process and the control of the arm movement. The task was multi-step reaching and path planning was used to demonstrate the solution.We extended their work by learning an adaptive observer which learns the internal model of the robotic arm. We also illustrated that a path integration mechanism can be trained from few examples that enables the robotic arm to move in arbitrary directions and velocities. The model also learned to compensate the delay in sensory feedback as well as getting adapted to changes in the environment (e.g. the stiffness of motors). The implementation also illustrates how DNF can integrate expectation and actual sensory inputs to make a new sensory perception for the robot arm, by which the arm is able to reach the target even with occluded vision.

The adaptive observer like many other solutions cannot explain the competition or cooperation between different existence controllers in the brain. Therefore, we introduced a new structure that we call the **Arbitrated Predictive Actor-Critic (APAC)**. The APAC combines a habitual reinforcement learning system with a supervised learning system of an internal model to learn both one-step mapping and multi-step reaching tasks. Most importantly, we introduced an arbitration system that mediates between their usage. We specifically discuss a situation in which both systems alone can solve an exemplary task so that we can study the consequences of their direct interactions in relation to their exclusive use. We show that this system is responsive to changes in the environment and that it can learn the reward function very fast. Our results demonstrate how the learning paradigm tend to rely on habits after learning the reward function. Our results are in line with evidence of human behaviour mentioned above. Although APAC does not take advantage of DNFs, it is possible to implement the APAC model with DNFs as a more biologically plausible model of connections between neurons.

1.2 Human Target Reaching

Reaching can be described as the transportation of the hand to the object by the upper limb. Reaching a target in the form of pointing is different from grasping. Grasping requires processing of object shape, size and orientation to preshape the hand and fingers. However, in pointing relocating the hand at the target location is unnecessary. Indeed finger pointing requires a small movement and may involve only small areas of reaching [15]. Basically, reaching can be either toward a visual target or toward a non-visual target using memory-guided movements.

1.2.1 Behavioural Analysis of Human Target Reaching

Motor control theorists have always recognized that information about target location in the retina, eye positions in the head, head position on the trunk, as well as arm position need to be integrated to make a successful reaching. Because of inertia, the eye moves first, followed by the head, and then by the hand. Normally hand starts the movement when the eye has landed on the target [16].

A target reaching task is a consequence of moving joints like the shoulder and the elbow. During the movement, that brings the arm closer to the object, corrective adjustments also happen, mostly to preshape fingers and wrist, to make the reaching more accurate [16]. Interestingly, trajectories of arm movements in three-dimensional space make curvilinear independent of its speed, orientation, or external loads [17]. Moreover, Lacquaniti et al [17] showed that some angles of arm and forearm movements are linearly related to any given movement that means they are tightly coupled independent of wrist movements in 3-dimensional space.

Another interesting feature of reaching movements is that they display a typical bell-shaped velocity profile [18] shown in Figure 1.1, with an accelerating and a decelerating phase. Therefore, during movement, hand velocity never remains constant. As Martenuik et al. suggest that one reason to have the different decreasing duration for grasping versus pointing might be for precision. For example, when the object is fragile and wrist and fingers need to form precisely to grasp it compared to when it wants to point to an object. On the other hand, a reason to have such slowly increasing velocity at the beginning of the movement might be the joints' inertia.

The accuracy of reaching movements is maximal when the hand starting position is located in front and around the reachers midline when movement is of small amplitude, and when it is directed straight ahead rather than in other directions. The faster the movement, the less it is accurate, a principle is known as speed-accuracy trade-off. Changing speed also affects joint movement dynamics and requires a rescaling of the joint torques [16].

Visual reaches are more accurate than non-visual or memory guided reaches [19].

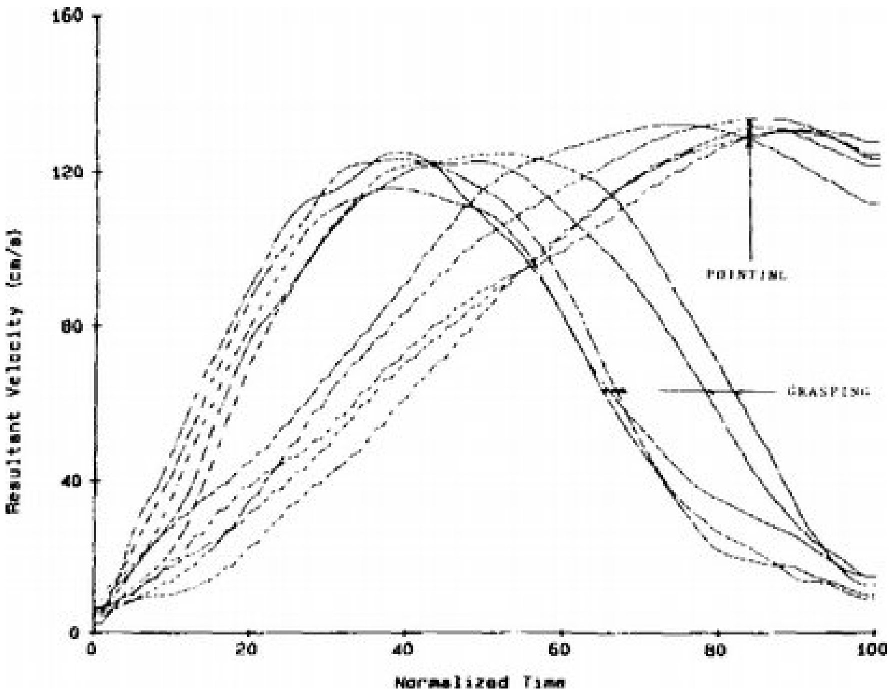


Figure 1.1: The normalized velocity of movements for pointing and grasping. Image from Marteniuk et al [18].

Under occluded vision, although highly accurate information is unavailable to the motor system following the visual occlusion, reaching is still possible which is called memory-guided reaching [20]. The main difference between the two type of reaching is that in visual reaching online adjustments are possible, however in memory-guided reaches there is no visual information to adjust the movement.

1.2.2 Neurophysiology of Human Target Reaching

Figure 1.2 shows different areas of the human brain, while Figure 1.3 represent different regions of the brain that are involved and activated in the target reaching. To reach a visual target, the visual information projects from the retina to a visual cortex in the occipital lobe. The gaze-centred information needs to be transformed into shoulder-centred coordinates by considering the current position of the eye in the head and head on the trunk [21, 22]. Even with occluded vision, hand position signals are represented in gaze-centred coordinates within PPC [23], this evidence suggests that hand-target comparison happens in PPC [24, 23].

Brain activities are different for visual and non-visual reaching. A reach-dominant

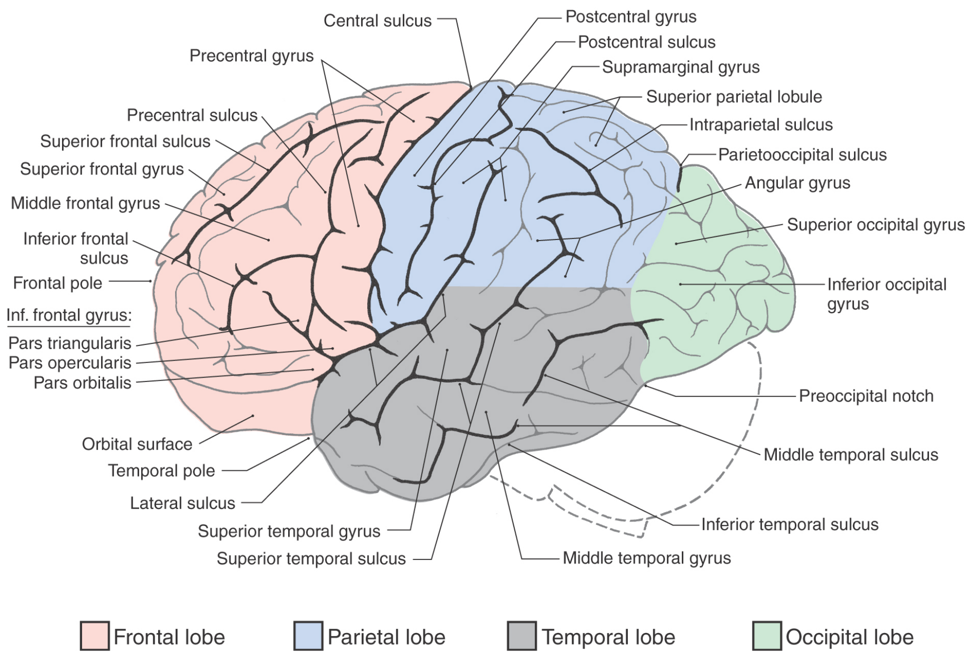


Figure 1.2: Different areas of human brain: Image from `https://clinicalgate.com/the-telencephalon/`

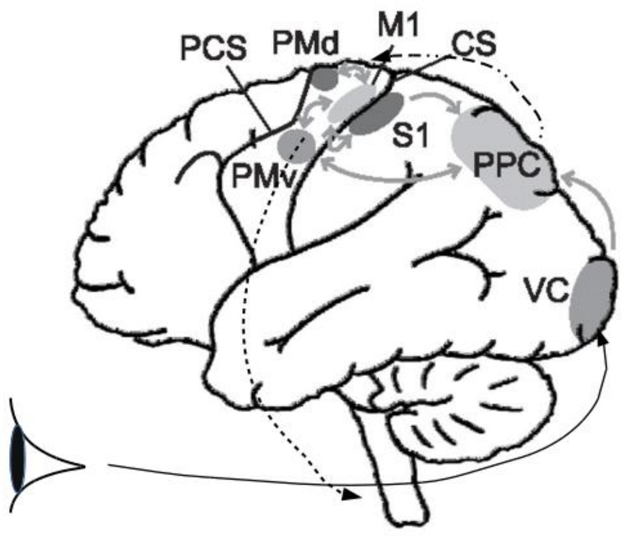


Figure 1.3: Brain area involve in a visual target reaching including visual cortex(VC), posterior parietal cortex (PPC), central sulcus (CS), precentral sulcus (PCS), primary somatosensory area for arm movements (proprioception) (S1), primary motor cortex area for arm movements (M1), ventral premotor cortex (PMv), and dorsal premotor cortex (PMd). Figure modified from [25].

region in the anterior precuneus (aPCu), extending into the medial intraparietal sulcus, is equally active in visual and non-visual reaching [15]. Indeed, these results suggest that the aPCu is a sensorimotor area with proprioceptive inputs. However, a second region, at the superior end of the parieto-occipital sulcus (sPOS), is more active for visual than for non-visual reaching [15], which shows sPOS is a visuomotor area with visual input. As Filimon et al. [15] showed, other regions of the brain including medial, anterior intraparietal, and superior parietal cortex was also activated during both visual and nonvisual reaching. They also showed that anterior areas of the parietal cortex are more active responding to hand movements, while posterior areas are more active responding to both hand and eye movements. Filimon et al. [26] showed that sPOS has motor, not purely visual, properties. For non-visual reaching, Left sPOS responds strongly during imagined right-handed reaches [26]. Since visual cortex is not activated relative under non-visual reaching, Filimon et al. suggested that sPOS activation is not due to visual imagery of a moving hand and instead, it indicates motor planning [26]. Since, left sPOS responses robustly during imagined and executed reaches, but more weakly during passive observation of a moving right hand, therefore, it seems to play a visuomotor, not just visual, role in reaching [26].

According to Crawford et al. [27], hand and target location get updated in parietal reach region (PRR) during the movement. PRR includes various precuneus regions, including anterior precuneus (aPCu), superior parieto-occipital sulcus (sPOS), the medial intra-parietal (MIP) cortex and visual area V6A which both are activated for reaching movements [23, 28, 15]. The Cerebellum also predicts and updates these predictions based on visual information and that is why human is able to perform precise and percept actions [29, 30, 31].

At least two sources of information are needed to make a reaching toward a visual target. One is the visual feedback from hand position, and another one is proprioception from a moving limb [15]. In visually guided actions, the dorsal stream plays a critical role [32]. Neuropsychological evidence has also demonstrated that the brain uses a unique representation of the space immediately surrounding the body that is called "Peripersonal space" [33, 34, 35].

Researchers found the superior parieto-occipital cortex (SPOC) [36, 28] and the area in the anterior part of the PPC (medial IPS, mIPS) [37] are general regions implicated in reaching movements. Prado et al. [38] showed that mIPS and SPOC regions have different functional properties. For example, mIPS, responds during reaching movements regardless of where the eyes are directed, while the area within SPOC responds during reaching movements to peripheral but not foveated targets [37].

Anderson et al. [39, 40] showed that posterior parietal cortex (PPC) has both sensory and motor properties and it involved in sensory-motor transformations. Moreover, PPC subserves cognitive functions related to action such as planning of saccades, reaches, and grasps [41]. For example, patients with PPC lesions have difficulty in estimating the location of stimuli in 3D space [42] or planning their movements [43].

Buneo et al. [44] also showed that some PPC neurons in PRR encode both target position and current hand position in eye-centred coordinates that builds a displacement vector in eye coordinates. Other PPC neurons encode reach-related variables without reference to the eye that makes a target position vector in hand coordinates [41]. In particular, Buneo et al. [44, 23] discuss in some length how a hand centred target map could be calculated with the appropriate coordinate transformation of eye-centred hand and target maps. Moreover, Averbeck reported a linear relationship between the velocity of hand movement and neural activities in areas 5 and 2 of parietal cortex in monkeys [45].

1.2.3 Some Other Neurobiological Findings

Since PPC is considered to be crucial for high-level cognitive plans of movements, forming intentions, and for decision making [46, 47, 48, 49, 50, 44, 41, 51], it would be an ideal place for the adaptive system of internal representation of the world that is called internal models. It would be very informative to study specifically if the representations of the target and hand locations reflect anticipated locations. Also, the PPC plays an important role in converting sensory information to proper motor control, that is from visual to motor coordinates [44].

The cerebellum is another area that has been discussed as a structure that can implement internal models [52, 31]. Indeed, the cerebellum is known to integrate motor and sensory information in order to predict the consequences of behaviour [29, 30], and it has been shown that the cerebellum helps to adapt in the compensation of force field [53, 54, 55, 30]. Shadmehr showed that the motor controller during force field learning was gradually adapting a model of the force field, to predict and compensate for the forces imposed on the arm [55]. Therefore, an adaptive forward model which is possibly in the cerebellum, can overcome noisy and delayed sensory feedback as well as variable dynamics and kinematics of the body and the environment. This adaptive forward model predicts the future state of the body and will optimize its prediction using true or estimated sensory feedbacks [56, 57, 58]. Note that due to having delayed sensory feedback, it is essential to have a predictive internal model [56, 57, 44, 58].

Moreover, since PPC builds an internal representation of the body including hand location, it can also control and adjust the reaching [59].

Daniel Wolpert [60] says that the only reason that human and other animals have the brain is: "to produce adaptable and complex movements". He emphasizes that "movement is the only way to interact with the world". Therefore in order to learn more about the brain, we need to expand our knowledge of how the brain produces and controls the movements.

Human and animals usually learn a new task only in a few trials [61, 62]. In fact, it is hypothesized that the brain builds an internal representation of the world and its body [56, 63, 64, 65]. Other evidence shows that **forward** and **inverse models** exist in the brain too [65, 56]. A forward model predicts the future position of the arm based on the current state and the taken action, whereas an inverse model estimates proper motor commands that transfer the agent from current state to the desired state. The internal model is used to perform in the environment and learn a new task. Flanagan et al. [66] showed that the internal model can predict the load force and the kinematics of a hand movement that depends on the load. Moreover, when learning how to use a new tool, humans make a transient change in the internal model of the arm as well as making an internal model of the tool [67]. Furthermore, imitation experiments show that a direct mapping happens between observation and the internal model [68]. Another advantage of having an internal representation is obtaining a reliable source of information for the agent to perform accurately even if there are no other sources of information (e.g. visual information) available [64, 65].

Adaptive internal model has long been considered to deal with noisy and delayed sensory feedback as well as changing kinematics of the body and environment [58]. Internal model uses forward model to predict future state of the system according to current state and applied motor control [57, 58]. Since PPC is considered to be crucial for high-level cognitive plans of movements, forming intentions, and for decision making [41, 47, 48, 44, 49, 50, 46, 51], it would be an ideal place for the adaptive internal model. The realization of our internal model within the PPC is hence quite possible.

Researchers also found evidence that shows the existence of two control systems in the brain [69, 70, 71]. Daw et al. [61] mentioned there are more than one control systems in the brain. One is in the prefrontal cortex [72], and the other one is in the Basal Ganglia [73]. These systems are usually known as model-based (MB) and model-free (MF) respectively. The former predicts action-outcomes using an internal model of the agent's environment, while the latter learns to repeat previously rewarded actions. Moreover, the model-based control system is a goal-directed controller, while the model-free behaves as a habitual system. Lengyel et al. [74] also showed that MB learning is very fast and performs better compared to MF, however, MF takes over the MB controller after limited experiences.

More importantly, research showed the two different control systems are used in different situations and can be simultaneously active [74, 75]. For example, in the brain, the cortical system represents a generalized mapping of input distributions while hippocampal learning is an instance-based system [74]. Moreover, when the model of the environment is known and there is sufficient time to plan, the best strategy is a model-based planning [61], but when the decision should be taken very fast the model-free planning is used [76].

1.3 Combined Planning and Habits - A General View

Not only have human decision-making studies supported the notion that both habitual and planning controls are used during decision-making [77, 71], there is evidence that arbitration may be a dynamic process involving specific brain regions. In fact, Lee [78] showed that the inferior lateral prefrontal and frontopolar cortex are involve in the arbitration process by encoding both reliability signals and the output of a comparison between those signals.

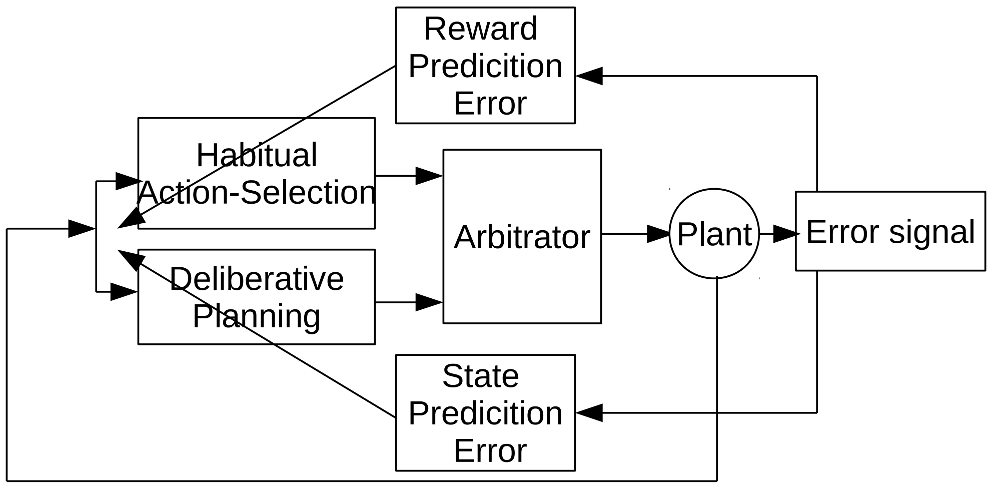


Figure 1.4: General view of APAC model.

Our APAC model results suggest that such an arbitration strategy, wherein the planning paradigm is used until the habitual system's predictions become reliable can result in performance that is non-inferior to exclusive planning control in most cases. Thus, our APAC model supports (A) the importance and value of implementing predominantly planning control early in behavioural learning and (B) the diminishing importance of planning control with greater experience in a relatively static environment. Moreover, since feedback to learning systems can differ in different situations and can be provided from different modalities such as vision or auditory input, therefore combination and integration of different controllers should be possible.

The general architecture of the proposed *Arbitrated Predictive Actor-Critic* (APAC) is shown in Figure 1.4. In this model, each control paradigm implies a specific type of teaching feedback. The deliberative planning controller incorporates internal models (e.g. adaptive observer) that are usually trained with supervised errors to form the adaptive internal model so that we considered here an explicit state predictions error. In contrast, deep reinforcement learning is a very specific and relatively recent form of habitual action selection which learns from reward prediction errors. The new component here is an arbitrator that mediates between these systems that can select the command given to the controlled system, the agent or in the plant in the common language of control theory. Of course, it is possible that both decision systems are trained with a combination of supervised and reinforcement learning, but this is not the crucial point in this work. The model is designed to study how a combined control system behaves in different environmental situations.

1.4 Related Works

TD learning [79] is a successful model that is undoubtedly a central idea of the reinforcement learning. In classical conditioning the learning is conceptualized in conditioned and unconditioned stimuli [80]. TD is the extension of the Rescorla-Wagner model which is a model of classical conditioning. TD learning simulates and relates to temporal difference model of animal learning. TD learning also models dopamine-based learning in the brain [81, 82].

Using neural network as a function approximator was another big step that broadened the range of possible machine learning problems to be solved, especially for control problems that deal with continuous states/actions spaces [83, 84]. Barto, Sutton and Anderson proposed a new structure that is called Actor-Critic (AC) that was implemented by neural networks [85].

Later, Barto [81] represented an adaptive critic which has similar behaviour to the dopamine neurons projection to the Striatum and frontal cortex. The adaptive critic uses the internal sensory information to learn an effective reinforcement signal.

Riedmiller [86] used a multilayer perceptron to learn a Q-value function for Q-learning algorithm which is called Neural Fitted Q-learning (NFQ). He also introduced experience replay memory to restore and reuse past experiences that obtained a model-free solution. Later they expanded their solution to fit for continuous actions that is called NFQCA [87].

Mnih et al. [9, 10] took advantage of deep neural networks and introduced Deep Q-Network (DQN) algorithm. Q-learning is unstable and difficult to learn large and non-linear function approximators. Therefore another network, named target network, is used in DQN that avoids oscillations [9]. Indeed, the target network is a copy of the main neural network that learns the Q-learning algorithm. The main network is trained in every step, while the target network is updated less frequently. DQN performed better than all previous algorithms; however, since it is designed to play Atari games, it has no need to learn continuous actions or states.

Silver et al. [88] proposed a Deterministic Policy Gradient method (DPG) for RL problems with continuous actions. DPG is an actor-critic network that uses TD rule to train the critic network and policy gradient method to train the actor network. In DPG, the policy is updated in the direction that most improves the Q-function. Silver et al. also proved that gradient of the actor is equivalent to a gradient of the Q-function provided by the critic multiplied by the gradient of the actions obtained by the actor with respect to their parameters [88].

Deep Deterministic Policy Gradient (DDPG) [89] is implemented based on NFQCA [86] and DPG [88]. DDPG also takes advantage of batch normalization technique [90] which normalizes each dimension and provides unit mean and variance across samples. Using batch normalization, DDPG learns different tasks both with high and low dimensional observations using the same network and hyperparameters.

The DDPG is one of MF solutions which have demonstrated success in recent years by learning to perform complex tasks without *a priori* knowledge about system dynamics [91, 9, 89, 92, 93]. However, MF solutions are not data efficient and they need a long time for training, therefore they are mostly inapplicable to motor control and robotics domain. In contrast, model-based solutions have better performance and even faster convergence [94, 95].

To improve learning process in the actor-critic, Rosenstein and Barto [96, 97] combined supervised learning and reinforcement learning scheme in the actor and built a supervised actor-critic. This solution was able to tune the actor manually and very fast. This solution is beneficial when dynamics of the system changes dramatically or a new policy is needed to be learned in a very short time. These authors used a gain scheduler that weights the control signal provided by the actor from the reinforcement learner and the supervised actor. In contrast to supervised actor-critic, our proposed model autonomously learns the internal model and arbitrates between the two controllers automatically and not manually. The intention of our model is to study the interaction of habitual and planning systems in form of an arbitrator and ultimately to understanding human behaviour.

Sutton proposed Dyna-Q that is an integrated model for learning and planning

[98]. Dyna-Q is a blend of model-free and model-based reinforcement learning algorithm. Dyna-Q can build a transition function and the reward function by hallucinating random samples. Therefore, although it uses a model-free paradigm at the beginning it becomes a model-based solution by learning the model of the world using the hallucination. Moreover, despite success in learning the value function very fast, Dyna-Q still lacks the explanation of cooperation and competition between different controllers. In our model, the internal model is used to predict the future state of the agent, unlike Dyna-Q that uses the model to train the critic and anticipate the future reward. Moreover, Dyna-Q starts from the model-free controller and becomes a model-based controller. Hence, while Dyna-Q has focused on the utilization of internal models to learn a reinforcement controller, our study here is concerned with the arbitration of two control systems.

All in all, despite all former models, APAC learns both model-based and model-free controllers autonomously. It also can arbitrate between the two controllers. And more importantly, although it is faster because it uses habitual action selection, it performs as accurate as planning controller.

1.5 Chapter by Chapter Thesis Outline

In the present work, we aim to introduce two main models that we have developed to simulate human target reaching under various conditions. First, some basic concepts will be explained very briefly in three sections namely: Neural Networks, Supervised Learning, and Reinforcement learning. Dynamic neural fields and path integration in DNFs are explained and discussed under neural network section. Internal models are also described under the supervised learning section in this chapter as well as temporal difference learning and the basic DDPG model. The first proposed model of target reaching, the adaptive observer implemented by DNFs is explained in Chapter 3. Thereafter, in Chapter 4, the arbitrated predictive actor-critic is introduced in more detail for target reaching. In this chapter, we report the performance of APAC under two different target reaching namely 1) One-step mapping and 2) Multi-step reaching. Results and definition of the environment and problem conditions will be explained under related chapters for each model. Finally, we conclude with the conclusion and future works in Chapter 5.

Chapter 2

Basic concepts

Machine learning is divided into three categories, namely: 1) unsupervised, 2) supervised, and 3) reinforcement learning techniques. There are many different algorithms in each category which have been developed and proposed so far. Type of the application and the task, as well as the feedback signal, are important parameters that indicate which solution should be chosen. In this dissertation, supervised and reinforcement learning methods are discussed.

In this chapter, we first talk about neural networks (NN) as a very useful tool for solving problems in the machine learning field. Then supervised and reinforcement learning problems with neural networks implementation will be discussed. These basic concepts make the skeleton of this dissertation and later in following chapters we show how our proposed models are built upon these concepts.

2.1 Neural Networks

A neural network is a biologically-inspired programming paradigm which has been successfully used to solve various problems. Warren McCulloch, Walter Pitts, D.O. Hebb, B. G. Farely, W. A. Clark were pioneers who paved the way for neural network computational machines in 1940 [99, 100, 101]. A neural network is a connected network of neurons that can observe the input data and alter weights of each connection in order to obtain desired output [102]. A neural network can demonstrate leaky integrate and fire neuron scheme, however, other activation functions are applicable. A very simple neural network with only one layer called **a perceptron** introduced by Frank Rosenblatt in 1950. Figure 2.1 illustrates a perceptron and a multi-layer perceptron with one hidden layer.

Weights demonstrate the strength of the connection between two nodes. Learning process happens through altering these weights. The outputs are produced by applying an activation function on the $\Sigma_i x_i w_i$, where x_i is an input and w_i is the weight

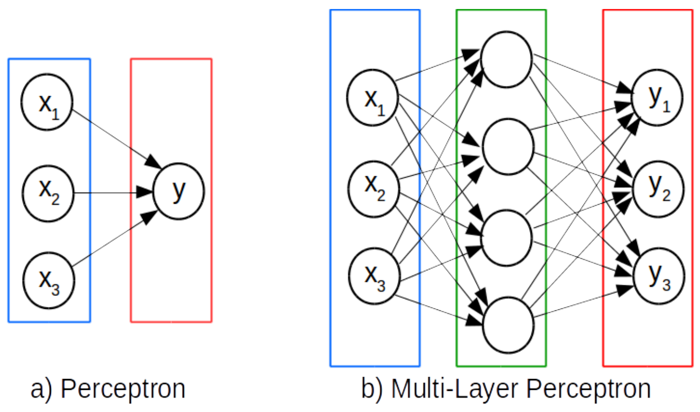


Figure 2.1: A perceptron (left) and A multi-layer perceptron (MLP) with one hidden layer (right). Nodes inside blue, red, and green boxes indicate input nodes, output nodes, and hidden nodes respectively.

of its corresponding connection. An activation function can be a linear, sigmoid, hyperbolic tangent, rectified linear, or any other functions that can be used for decision making. An objective function such as mean-square-error or cross-entropy of the desired output and calculated value by the network is also used to calculate the error between desired output and prediction of the network. Then usually the calculated error back-propagates through the network and adjusts the weights, however, other learning paradigms are also applicable.

Deep neural networks are neural networks usually with several hidden layers. Having more layer enables the network to learn more complex functions. Deep learning uses deep neural networks and applies powerful tools and techniques such as dropout, batch normalization, convolution, and many others. Using these methods, deep learning provides successful solutions for wide range of applications and problems. Deep learning has shown huge success especially in classification, image recognition, and also for natural language processing [103].

2.1.1 Dynamic Neural Fields

A dynamic neural field (DNF) is a long established model of a cortical sheet [104, 105, 106, 105]. DNF models have been prominent in modelling brain functions and behaviour [107, 108, 109, 110], and they are often an important ingredient in cognitive robotics [108, 111, 112, 113, 114, 3]. Dynamic neural field models are also a general biologically plausible implementations of a Kalman filter [115]. Moreover, DNFs are discussed frequently in biological motor control [116, 117, 118, 119, 120, 108] and biologically inspired robotics systems [108, 111, 121, 113, 114, 3, 122, 123].

More specifically, a DNF describes the time evolution of a cortical map that can encode a continuous feature value [124]. The time evolution of the neural field is governed by

$$\tau \frac{\partial u(x,t)}{\partial t} = -u(x,t) + \int w(x-x')r(x',t)dx' + I^{ext}(x,t) + h, \tag{2.1}$$

where τ is a time constant and $u(x,t)$ describes the field activity at time t and location x, while $I^{ext}(x,t)$ denotes the external input to the field and x' indicates source locations of recurrent connections. The parameter $h <= 0$ is the resting level of the field.

The rate $r(x,t)$ has a monotonic relation to the field activation,

$$r = \frac{1}{1+e^{-\beta(u-u_0)}}, \tag{2.2}$$

with slope parameter β and threshold u_0.

The kernel function $w(x,x')$ of the integral transformation in the second term on the right hand site of Equation 2.1 specifies the interaction strength (weight) within the field that is most often chosen as a Mexican-hat function or a difference of Gaussians [106]. We can split the integral term into components describing local excitation, local inhibition, and global inhibition,

$$\begin{aligned} \int w(x-x')r(x',t)dx' &= \int_{x'} w_{exc}(x-x')r(x',t)dx' \\ &+ \int_{x'} w_{inh}(x-x')r(x',t)dx' \\ &+ g_{inh} \times \int_{x'} r(x',t)dx'. \end{aligned} \tag{2.3}$$

The weight kernels w_{exc} and w_{inh} are considered to have a Gaussian form

$$w \propto= \frac{a}{\sigma\sqrt{2\pi}} exp(-\frac{|x|^2}{2\sigma^2}), \tag{2.4}$$

with different parameters for the width σ and amplitude a for the excitatory and inhibitory kernels. The parameter g_{inh} is a constant value for global inhibition.

To implement DNFs in computer simulations one need to discretize the model both in time and space. The spatial discretization is achieved by

$$\begin{aligned} x \quad &\to i\Delta x && (2.5) \\ \int \mathrm{d}x \quad &\to \Delta x \textstyle\sum, && (2.6) \end{aligned}$$

where the discrete spatial locations are indexed with $i = 1, ..., N$, and $\Delta x = \frac{2\pi}{N}$ is the spatial resolution of the space that represents feature values in the range $[0, 2\pi]$. The straightforward spatial discretization results in the continuous-time recurrent neural network of the form

$$\tau \frac{\mathrm{d}u_i(t)}{\mathrm{d}t} = -u_i(t) + \frac{2\pi}{N} \sum_j w_{ij} r_j(t) + I_i^{\mathrm{ext}}(t) + h, \tag{2.7}$$

where w_{ij} is the connection strength between source location j and destination i. This continuous time ordinary differential equation can then be discretized in time in various ways for numerical integration. A linear approximation with the simple Euler method that considers a small increment time steps Δt,

$$u_i(t + \Delta t) = (1 - \frac{\Delta t}{\tau})u_i(t) + \frac{\Delta t}{\tau} \left(\frac{2\pi}{N} \sum_j w_{ij} r_j(t) + I_i^{\mathrm{ext}}(t) + h \right) \tag{2.8}$$

is sufficient for our studies here.

The DNF has been introduced above as a one-dimensional feature map for feature x. However, the generalization to higher-dimensional feature maps is straightforward [125]. The dynamic of an N-dimensional feature vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ would be governed by the dynamic field equation [126]:

$$\begin{aligned} \tau \frac{\partial u(\mathbf{x}, t)}{\partial t} \quad = \quad & -u(\mathbf{x}, t) + \int_{x'_1} \int_{x'_2} ... \int_{x'_N} w(\mathbf{x}, \mathbf{x}') r(\mathbf{x}', t) dx'_1 dx'_2 ... dx'_N \\ & + I^{ext}(\mathbf{x}, t) + h \end{aligned} \tag{2.9}$$

with an appropriate N×N-dimensional weight kernel $w(\mathbf{x}, \mathbf{x}')$. Note that when the weight kernel is shift invariant, which means that the weight values are the same

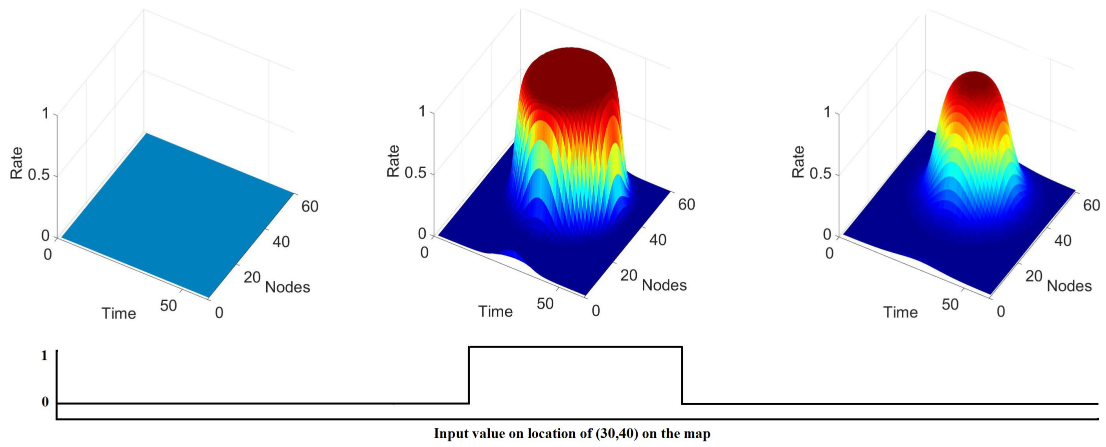


Figure 2.2: Activity of a DNF at different times, before (left), during (middle), and after (right) an external input was applied to the location $(\pi, 2/3\pi)$ of the map. The simulation was made with $N = 60 \times 60$ nodes. The activity is stable even after removing the external input.

between all pairs of locations having the same distance between them, the integral is equivalent to a convolution. For the implementation of the two-dimensional convolution it is advisable to apply the convolution theorem that calculates the convolution as a product in Fourier space. Note that fast Fourier transforms (FFTs) implement circular convolution, implying periodic boundary conditions. Thus if the environment is non-periodic, as in our case, the DNF maps should be larger than the environment such that boundary regions are not activated.

An important feature of a DNF map with weight kernels that mediate local excitation and long distance inhibition is that they can develop an activity packet that is stable even after removing an external stimuli that triggered it in the first place [106]. Figure 2.2 illustrates the activity in a two-dimensional DNF map at three different simulation times, before an external input was applied to the centre of the map, during the period when a central input was applied, and after the external input was removed. The important result here is that there is a localized activity zone, often called an activity packet or a bubble or a bump, which is triggered by external input and persists after the external input is removed. This, in essence, describes how persistent activity underlying working memory can be attained in recurrent networks.

2.1.2 Path Integration in DNF models

Humans are able to update internal representation such as the sense of direction in the dark using velocity signals from our vestibular system in the ears. More generally, an observer that we introduce below needs the ability to add up (integrate) changes of locations from movement information. For example, the system could use copies of internal movement commands or velocity information to update its state approximation. This update of the state representation from velocity information is commonly termed path integration (PI) [127] in biological systems or dead reckoning in robotics.

There are several possible implementation strategies of PI in a DNF framework [128, 129, 130, 131, 132, 133, 134]. We will here utilize the implementation proposed by Connors and Trappenberg [135], although other implementations are possible [134]. The PI mechanism followed here was first suggested by Stringer et al. [129] and is based on learned modulators of the weight matrix that make the effective weight matrix asymmetric. Note that a symmetric weight kernel is essential for having a stable, non-moving activity packet in a DNF. An asymmetric weight kernel can be used to move the activity packet in the direction of the asymmetry. The orientation and speed of the movement depends thereby in a non-linear way on the strength and amount of asymmetries in the weight kernel. A specific model proposed by Stringer et al. [129, 130, 131] is given by

$$\tau \frac{\partial u(x,t)}{\partial t} = -u(x,t) + \int w^{\text{eff}}(x, x', r^{\text{mov}}) r(x',t) dx', \tag{2.10}$$

This model moves the activity packet based on input from the activity of movement indicator cells, r^{mov}, such as vestibular cells in humans which carry velocity information of the head. The effective weight matrix at time t is calculated here according to [135]

$$w^{\text{eff}}_{ij} = w_{ij} \sum_k (1 + w^{\text{mov}}_{ijk} r^{\text{mov}}_k), \tag{2.11}$$

which specifies a specific form of combining the movement tensor with the activity of a movement indicator cell r^{mov}_k. The movement tensor is incrementally learned with

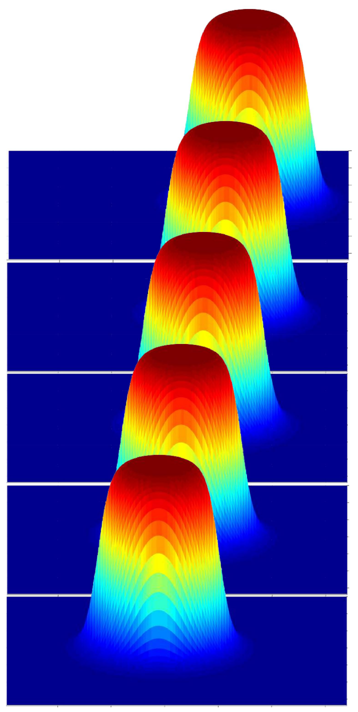


Figure 2.3: Step by step path integration. Applying path integration on 2D DNF move the activity packet which is shown in every 5 steps in the Figure. This particular movement has conveyed the bump by amount of 25 nodes on the map.

Hebbian-style learning between three factors,

$$\Delta w_{ijk}^{\mathrm{mov}} = \epsilon r_i \overline{r}_j r_k^{\mathrm{mov}}, \tag{2.12}$$

where ϵ is a learning rate. This Hebbian learning takes the history of the movement into account by means of a trace term $\overline{r}_j$ over a temporal window size characterized with the parameter η,

$$\frac{1}{\eta}\frac{\partial \overline{r}_j}{\partial t} = r_j(t) - \overline{r}_j(t) \tag{2.13}$$

This movement weight tensor is hence based on the experience of movements and the co-activation of movement indicator cells r^{mov}, which in this model indicates the velocity of the movement. Such a learning rule can be used in an on-line system and only requires a short term trace memory which can be implemented by a simple recurrent delay line. Figure 2.3 shows a time series of two-dimensional map activities that are the result of applying the PI mechanism for 5 time steps.

2.2 Supervised learning

If data is in form of input and its desired output, a supervised learning approach can be used to learn the pattern that maps the input data to the output data [102]. Since the learned pattern can predict the output of an unseen data, therefore it is sometimes called predictive learning.

Supervised learning is used for two tasks, 1) regression, or 2) classification. If the prediction is in form of a continuous numerical value, the problem is a regression problem. In the classification problem, the task is predicting a label to an unseen data.

If we want to solve a supervised learning problem using linear regression, a loss function needs to be defined and then minimized (or maximized). A very successful method is **Gradient Descent**, in which partial derivatives of the loss function respect to each parameter need to be calculated. Indeed by changing parameters along the gradient of the loss function, this method minimizes the error.

$$\theta_i \leftarrow \theta_i - \alpha \nabla L, \tag{2.14}$$

where θ_i is a weight, α is a learning rate, and ∇L is the loss function changes.

2.2.1 Internal models: Observer

The **internal model** or the **observer** is a supervised learner that can simulate and mimic the behaviour of a system [136, 137]. Internal models can be divided into two categories, namely **forward models** and **inverse models** [64, 138].

According to Jordan [136], an observer is a dynamic system that runs in parallel with the actual plant and estimates the current state of the system based on observation of the inputs and outputs of the system (see Figure 2.4). Using different sources of information namely external sensory feedback and motor commands along with an internal copy of them, the observer integrates information and provide an estimation over time [139, 63]. Kalman-filter in robotics system is an example of such an observer [140].

Observers are a proven concept in neuroscience to discuss motor control in animals and humans [141, 139] like biological observers implanted in the central nervous

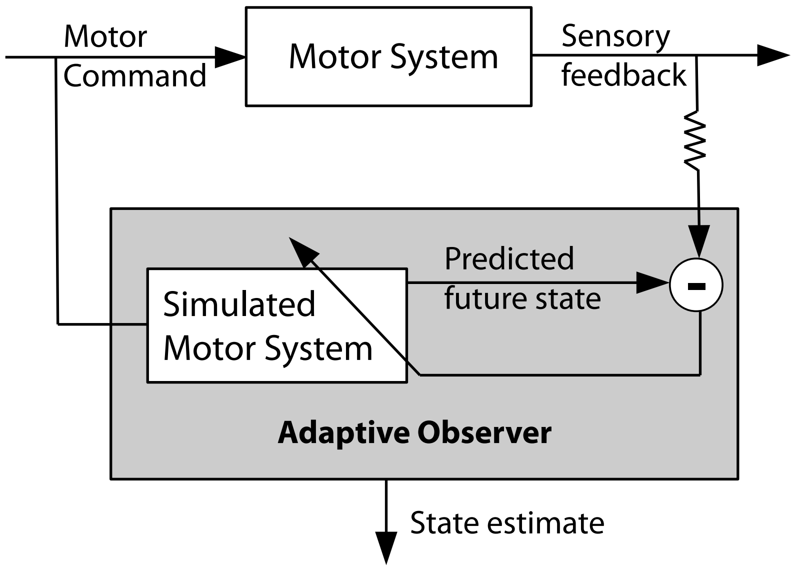


Figure 2.4: Illustration of how an observer runs in parallel with a motor plant. It integrates measurements of the system to have a better estimation of current state of the system. Sensory feedback is typically delayed in time compared to much faster efferent copies, so that appropriate delays have to be taken into account. (adapted from [63]).

system (CNS). These observers are considered to be the internal models of the agent as well. This observer integrates various information to produce a behaviour. Efferent copies (e.g. internally generated motor commands), afferent signals (e.g. proprioceptive feedback from muscles), or re-afferent signals (e.g. visual input of arm positions) are some of the information available to the observer.

Since the internal model is used for prediction of the future state of the agent (e.g. our body position) according to its current state and taken action, a supervised solution can be implemented to simulate the internal model's behaviour. A supervised method can learn the pattern from collected samples during random or initial movements. Assume visual information is available during sampling. Therefore each sample contains its current state (s_t), set of taken actions (a_t), and future state of the body after taking these actions (s_{t+1}). A supervised model can interpret current state and taken actions as an input, while the future state is the desired output of the system. Note that, human (or even an animal) has a different source of perception and one is the visual information. Therefore, if the visual information is unavailable (when the creature is blind), other sources can be used to collect samples and learn from.

If we assume the visual information is available during training, we can use a supervised method implemented by the neural networks to learn the pattern and behave as the observer for the agent.

$$y_t = f(s_t, a_t; \theta_t), \tag{2.15}$$

where θ_t is the network parameter at time t, and y_t is the output of the network. To learn the pattern of the sampled data, the loss function can be defined as mean squared error (MSE) of actual future state and the predicted value by the network:

$$L = \frac{\sum_1^n (y_t - s_{t+1})^2}{n} \tag{2.16}$$

Gradient descent is used to tune the weights of the network.

Machine learning is all about modelling the data. However, learning the function (or the model) that perfectly explain the training data is not enough and may cause a problem that is called **overfitting**. Overfitting happens when the error on the training data is decreasing but the error on unseen data or test data is not getting any better and even sometimes increasing. **Underfitting** also happens when a model is good neither on the training data nor on the test data. The goal of machine learning indeed is providing good results on both seen and unseen data therefore, the learned function have to generalize well on the unseen data as well as the training data. For more complex world model, more parameters should be considered to learn various features of the data set which sometimes lead to overfitting. To control impact of each parameter on the loss function, regularization can be used. For example, assume that the cost function is in form of:

$$L = \lambda_1\beta_1 + \lambda_2\beta_2^2 + \lambda_3\beta_3^2, \tag{2.17}$$

where λ_1, λ_2, and λ_3 are regularization hyper-parameters and β_1,β_2, and β_3 indicate parameters of the loss function. Error on the training set and on the test set are called **Bias** and **Variance** respectively. Finding a right balance between these two is challenging and is also called **bias-variance tradeoff**. Figure 2.5 shows 3 different

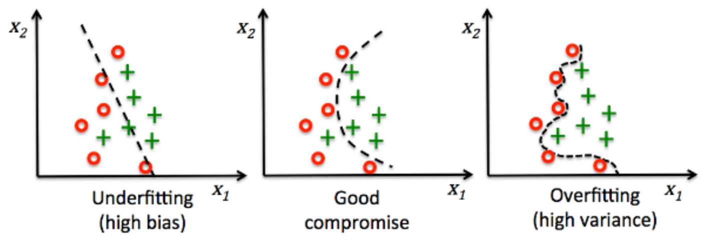


Figure 2.5: Bias-Variance Tradeoff [142]

models on one data set. Underfitting may occur when the model is too simple and it is not complex enough to fit the training data. Overfitting also may happen when the model is too complex and is only good on the training data. However, a good model is sufficiently good for both the training and the testing data sets.

2.3 Reinforcement Learning

Reinforcement learning is a very well known learning paradigm in animal learning. Typical RL problems involve four key quantities: (1) states, which can be thought of as contexts or stimuli; (2) actions that are available at or given by these states; (3) transitions between states by actions; and (4) rewards, which quantify the immediate worth of states in terms of reward or punishments [143]. Reinforcement learning (RL) is about learning what to do through interacting with the environment. The goal of reinforcement learning is how to maximize the reward received from the environment through series of actions [144]. A RL problem is a **credit assignment problem**. Credit assignment is assigning a credit (or blame) to the taken actions that made the overall outcome of the system [145, 79].

To collect maximum reward the agent needs to learn an optimal policy (decision-making rule) or an optimal value function. A policy defines behaviour of the agent at a given time, while value function indicates what is good in a long run [144]. The optimal policy can be a sequence of actions at each state that maximizes the reward received from the environment. An agent needs to explore the environment to learn more about the environment. The agent also needs to exploit its current information to make the best decision.

State of an agent transfers to the new state based on its dynamics. If the agent knows the *transition function* or the *internal representation* of the world, it can plan to take the action which provides more reward. However, this prior knowledge is not available in many cases, and providing such prior knowledge is also impossible in some cases. The transition function or internal representation of the world is called *model* or *internal model*.

There is a trade-off between exploration and exploitation. **ϵ-greedy** is a very common strategy by which the agent chooses between exploration and exploitation. Typically, when it is going to explore the environment (ϵ probability) it randomly selects an action to take, while for exploitation ($1 - \epsilon$ probability), the action that maximizes the reward received from the environment is chosen.

Different strategies may use to select the action. In the other words, the policy of selecting the action is a very important feature of a solution. If the method attempts to evaluate or improve the policy that is used to make decisions is called an **On-policy** method. Whereas, by an **off-policy** strategy, the agent evaluates one policy while following another [144].

2.3.1 Temporal difference (TD) learning

TD learning is undoubtedly a central idea of the reinforcement learning that simulates and relates to temporal difference model of animal learning. TD learning also models dopamine-based learning in the brain [81, 82]. TD combines dynamic programming (DP) and Monte Carlo (MC) ideas [144]. Similar to MC, TD learns by sampling from the environment. TD also approximates its current estimate ($V(s_t)$) based on its previously learned estimates, therefore it resembles the DP. The simplest version of TD rule is formulated as below:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)], \tag{2.18}$$

where r_{t+1} is the actual return received from the environment at next time step, and α is a learning rate. Here γ is a number between 0 and 1 ($0 \leq \gamma \leq 1$) and is called discount factor. The term $r_{t+1} + \gamma V(s_{t+1})$ is called target and is estimated value of the future state [144]. Since this rule only considers the nest step it is called TD(0).

In general TD(λ) can be defined as following [144]:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t^\lambda - V(s_t)], \tag{2.19}$$

where R_t^λ is given by:

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R^{(n)} + \lambda^{T-t-1}R_t, \tag{2.20}$$

where $0 \leq \lambda \leq 1$, $R_t^{(n)}$ is n-step return at time t, T is number of steps in th episode. If $\lambda = 1$ then $R_t^\lambda = R_t$ which is the total return same as MC. If $\lambda = 0$ then it depends only on the immediate reward which is shown in TD(0) (see Equation 2.18)

This update rule can be updated in an on-policy or an off-policy manner. Q-learning and SARSA are two algorithms that have been used in reinforcement learning problems successfully. Q-learning is an off-policy TD learner by which the learner learns the value of the optimal policy independently of the agent's actions. SARSA instead is an on-policy TD learner.

Q-learning and SARSA algorithms

One way to solve the RL problem is learning the value function. The value function can be interpreted as a general value function that describes the value of each state independently from the range of actions that can be taken. This value function is shown by $V(s_t)$ and is called the state-value function. If we need to learn the value function of each state depend on each available action, $Q(s_t, a_t)$ is used that is called state-value function.

Algorithms 1, and 2 show SARSA and Q-learning solutions respectively. Both of these algorithms use $Q(s_t, a_t)$ to update TD rule. The main difference between these algorithms is that in SARSA the action to be taken is selected according to the policy, whereas in Q-learning the action that provides maximum reward is chosen in next step. Therefore, SARSA always relies on the policy that it learns, while in Q-learning the policy that is learned differs from the one that chooses the actions to be taken.

Algorithm 1 SARSA: An on-policy TD learning [144]

```
Randomly initialize $Q(s,a)$, $\forall s \in S, a \in A(s)$,
for episode=1,M do
    Initialize $s_t$
    Choose $a_t$ from $s_t$ using policy derived from $Q$ (e.g. $\epsilon$-greedy)
    repeat
        Select action $a_t$, Observe $r_t$, $s_{t+1}$
        Choose $a_{t+1}$ from $s_{t+1}$ using policy derived from $Q$ (e.g. $\epsilon$-greedy)
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
        $s_t \leftarrow s_{t+1}; a_t \leftarrow a_{t+1}$
    until $s_t$ is Terminal
end for
```

Algorithm 2 Q-learning: An off-policy TD learning [144]

```
Randomly initialize $Q(s,a)$, $\forall s \in S, a \in A(s)$,
for episode=1,M do
    Initialize $s_t$
    Choose $a_t$ from $s_t$ using policy derived from $Q$ (e.g. $\epsilon$-greedy)
    repeat
        Select action $a_t$, Observe $r_t$, $s_{t+1}$
        $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$
        $s_t \leftarrow s_{t+1}$;
    until $s_t$ is Terminal
end for
```

2.3.2 Model-free vs. Model-based Solutions

Reinforcement learning solutions can also be categorized into two main approaches, namely model-based (MB), and model-free (MF) solutions. MB and MF solutions in reinforcement learning (RL) systems are two ends of the spectrum of RL methods [61, 70, 71, 146]. A model is the transition function that mimics the behaviour of the environment. Indeed, the model can predict and produce the future state of the agent with a given state and action. Therefore it can be used for planning [144, 147].

MF solutions are not data efficient and they need a long time for training, therefore they are mostly inapplicable to robotics domain. In contrast, MB solutions have a better performance and even faster convergence [94, 95]. Therefore, an MB solution is applicable when dynamics of the system is available. Whereas, MF solutions do not need to have prior knowledge about the dynamics or transition function of the agent

[144, 148]. In some cases the model of the environment can be learned easily, however, in many cases obtaining the model or the transition function is somehow too expensive or impossible to be obtained. For example, a robot with changing configuration that interacts with a dynamic environment has a changing model. Learning the model of such changing environment is very challenging.

In an MB solution, either the model is available as a prior knowledge or it is easy to be learned. Therefore the model can be given as the prior knowledge or the agent learns it by exploring the environment. However, in MF, the agent learns the value function or policy without learning the model.

2.3.3 Actor-critic

RL solutions usually are divided into 3 groups, namely: actor-only, critic-only, and actor-critic. Actor-only solutions work with parametrized policies, while the critic-only solutions rely on the value function approximations, while actor-critic methods combine the strength of both actor- and critic-only techniques [149]. Actor-Critic methods are a model-free on-policy reinforcement solution that has shown a huge success in solving wide range of problems. The actor-critic method is a TD learning that uses separate memories to learn the policy and the value function independently.

The actor-critic has two main components, one to learn the policy which is called the actor since it selects the action. The other one is called the critic that criticizes the actions made by the actor to learn the value function. Therefore the learning is always on-policy [144]. Moreover, the Actor-critic framework has many links with neuroscience and animal learning, in particular with models of basal ganglia [150]. Figure 2.6 shows an actor-critic structure with some neuro-psychological details. Dopamine is a neuromodulator that plays an important role in the human and animal brain such as learning, motivation, and addiction [151, 152, 153, 154]. Dopamine is produced in either the substantia nigra pars compacta or in the ventral tegmental area. These two areas directly project to striatum that is divided into ventral and dorsal areas [155]. The striatum itself is connected to the motor areas and prefrontal cortex, which are active in motor control and planning [72].

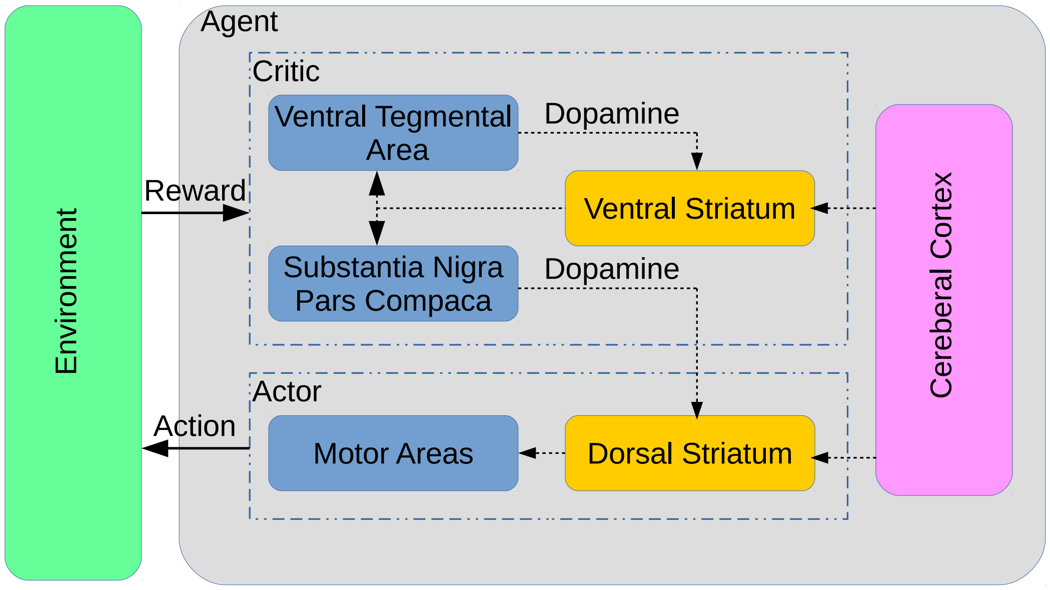


Figure 2.6: An Actor-Critic structure with neuro-pscychological details. Figure modified from: `https://mpatacchiola.github.io/blog/2017/02/11/dissecting-reinforcement-learning-4.html`

2.3.4 Deep Deterministic Policy Gradient Method

One of the very successful models of the actor-critic is the deep deterministic policy gradient model (DDPG) [89]. The actor-critic requires minimal computation to predict and select the action for the next step, therefore it is good for problems with a large state/action space (e.g. continuous state/action spaces) [144, 156, 89]. Using function approximators like the neural networks and cutting-edge techniques of deep learning enabled the actor-critic to solve wide range of problems recently. Here, we will explain the DDPG method which is a basic component of the proposed APAC. We will also explain some techniques that make a deep reinforcement learning very successful, specifically target networks and experience replay memory.

Target networks

Mnih et al. [9, 10] took advantage of deep neural networks and introduced deep Q-learning algorithm (DQN). Q-learning is unstable and difficult to learn large and non-linear function approximators. Therefore another network, named target network, is used in DQN that avoids oscillations [9]. Indeed, the target network is a copy of the main neural network that learns the Q-learning algorithm. In every step, the main network is trained. There are two methods to update the target network. A) After every few steps the main network is copied onto the target network, or B) The target network is updated in every step but proportionally to the main network which is called *smooth update* (Equation 2.21).

$$\theta' = \theta' \times (1 - \tau) + \theta \times \tau, \tag{2.21}$$

with changing parameter $\tau \ll 1$, and where θ is the target network parameter, and θ is the considered network (either the actor or the critic). Note that the main actor network is used for training, however, the target network of the actor is used for the action prediction. The target network is used for prediction and since it changes slightly w.r.t to the main network, it helps to avoid oscillation and made DQN perform better than all previous algorithms. DDPG also uses the main actor and the critic network for training, however, it uses two target networks for prediction. DDPG actually has two target networks, one for the critic network and one for the actor network that are updated using the smooth update [89].

Experience Replay Memory

Riedmiller [86] used a multilayer perceptron to learn a Q-value function for Q-learning algorithm which is called neural-fitted q-iteration (NFQ). He also introduced experience replay memory, $R(s_t, a_t, r_t, s_{t+1}, T_t)$, where s_t is the current state at time t, a_t is the action taken at time t, s_{t+1} is the next state, r_t is the reward received at time t, and T_t indicates whether the state at time $t+1$ is a terminal or not. The replay memory is a queue-like buffer with a finite size. The agent will forget older experiences and it will update its parameters based on its recent experiences.

The replay memory is used to restore and reuse past experiences to obtain a model-free solution. Later they expand their solution to fit for continuous actions (NFQCA) [87]. Since then, the experience replay plays a significant role in the reinforcement learning solutions and becomes a norm. The replay technique allows the methods to reuse the past experiences which makes a better generalization on the learning.

Replaying past experiences is similar t exploiting the model. Moreover, Using the experience replay memory is like planning using a learned model which can be useful in learning the task [157]. Somehow, using the experience replay is equivalent to learning by TD(λ), however, the former is easier to implement and use [158].

The size of the experience replay memory is also important. It needs to be big enough to produce a better generalization [159], however, in the changing environment, using a very big memory replay leads to slower adaptation to the changes in the environment.

DDPG like many NFQCA take advantage of the experience memory replay. At each time step, a random batch of N samples is selected from the experience replay memory, which is used to train both the actor and the critic.

Training the Critic

The critic $\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}})$ is implemented as a deep neural network, where s_t is the current state at time t, a_t is the action taken at time t, and $\theta^{\mathcal{Q}}$ are the parameters of the critic network. The goal of the critic is to approximate the accumulation of the environmental reward (sometimes called return) that can be expected from a certain state action combination. The critic is learned through temporal difference (TD) learning [147, 82].

$$\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}}) \leftarrow \mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}}) + l_1\delta, \tag{2.22}$$

$$\delta = \underbrace{r_t + \gamma \max_{a'} \mathcal{Q}'(s_{t+1}, a'; \theta^{\mathcal{Q}})}_{\text{estimated reward}} - \underbrace{\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}})}_{\text{actual reward}}, \tag{2.23}$$

where l_1 is the learning rate of the critic network, r_t is the actual immediate reward received from the environment at time t, γ is a discount factor, and $\mathcal{Q}'$ represents the estimation of the value of a state-action pair.

The computation in Equation 2.23 is done in the TD component. To train the critic using the TD rule, the error needs to be back propagated through the critic. The error between estimated value and the actual value is used to compute the loss function of the critic (Eq. 2.24),

$$L^Q = 1/N \sum (\delta)^2. \tag{2.24}$$

Training the Actor

The actor (π) receives the current state (s_t) and predicts future actions to be taken (a_t).

$$\pi(s_t; \theta^\pi) = a_t, \tag{2.25}$$

where a_t, is the taken action at time t. The actor is implemented as a deep network where θ^π indicates the parameters of the actor network and is trained using the deterministic policy gradient method [88].

One method to train the actor is policy gradient method if the policy $\pi(s_t; \theta)$ is differentiable. Plus, the actor needs to be differentiable in order to learn continuous state/action spaces. The policy gradient methods search for a minimum of $J(\theta)$ w.r.t parameter θ, where $J(\theta)$ is any policy loss function and l_2 is a learning rate. In general, the actor alters the current policy and updates its weights in order to maximize the value function provided by the critic. The update rule is in general in form of:

$$\theta_t^\pi \leftarrow \theta_t^\pi + l_2 \nabla_\theta J, \tag{2.26}$$

Note that for the actor the loss function is $\nabla_\theta J$ which is the critic because the value of the taken action is evaluated by the critic. Therefore,

$$\nabla_\theta \mathsf{J}(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{pmatrix} \tag{2.27}$$

$$\theta_t^\pi \leftarrow \theta_t^\pi + l_2 \frac{\partial Q(s_t, a_t; \theta_t^Q)}{\partial \theta_t^\pi} \tag{2.28}$$

The changes of the weights of the actor are correspond to the changes in expected reward with respect to the actor's parameters. By applying chain rule to 2.28 we will have new update rule for the actor [88]:

$$\theta_t^\pi \leftarrow \theta_t^\pi + l_2 \frac{\partial \mathcal{Q}(s_t, a_t; \theta^\mathcal{Q})}{\partial \pi(s_t; \theta_t^\pi)} \frac{\partial \pi(s_t; \theta_t^\pi)}{\partial \theta_t^\pi}, \tag{2.29}$$

where l_2 here is the learning rate of the actor. The plant takes the action and transfers to its new position which forms the new state s_{t+1}.

Silver et al. [88] also proved that stochastic policy gradient, which is the gradient of the policy's performance ($\nabla_\theta J$), is equivalent to the deterministic policy gradient which is given by:

$$\nabla_\theta J \approx 1/N \sum_i [\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\pi(s_t;\theta^\pi)} \times \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_t} \tag{2.30}$$

where $\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\pi(s_t;\theta^\pi)}$ is the gradient of the output of the critic with respect to its parameters, $\nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_t}$ is the output of the actor network with respect to its parameters and N is the size of mini-batches.

Ornstein-Uhlenbeck process

DDPG also uses an Ornstein-Uhlenbeck process [160] to add noise to the environment that results in new samples. Ornstein-Uhlenbeck process is a Markovian and a Gaussian process that is usually used as a noise process in control theory. Having such noise process in DDPG acts as the exploration process that makes new samples for training. The actor in DDPG, under the Ornstein-Uhlenbeck process, is in form of:

$$a_t = g(s_t; \theta^{\pi'}) + \mathcal{N}, \tag{2.31}$$

where $\mathcal{N}$ indicates the noise obtained from the Process.

Batch-Normalization

Another technique that DDPG uses is batch-normalization. Batch-normalization normalizes parameters of each layer and avoids diminishing of learning rates during training at different layers. Without batch-normalization the initialization of parameters should be done very carefully, however, using batch-normalization overcomes this issue and make it easier to initialize the parameters [90]. By batch-normalization, DDPG is able to learn different tasks without changing initialized parameters [89].

Chapter 3

Modelling Human Target Reaching using Dynamic Neural Fields [1]

3.1 Motivation

In this chapter, a model implemented using Dynamic Neural Fields (DNFs) for target reaching is introduced that is called the adaptive observer. DNF is a well-known model that simulates brain activity in cortical tissues. By DNF the activity of the population of neurons is being considered instead of the activity of only one single neuron. We developed a biologically motivated implementation of an arm controller that includes an adaptive observer. We also illustrated that a path integration mechanism can be trained from few examples that enables the robotic arm to move in arbitrary directions and velocities. The model also learned to compensate the delay in sensory feedback as well as getting adapted to changes in the environment (e.g. the stiffness of motors). The implementation also illustrates how DNF can integrate expectation and actual sensory inputs to make a new sensory perception for the robot arm, by which the arm is able to reach the target even with occluded vision. Materials and results of this chapter have been published in Journal of Neural Network [1].

Here, we extended the model by Strauss and Heinke [3] and developed specifically a system that incorporates an adaptive observer and that can learn delays. The task here is to reach the target while moving on an approximately straight line toward the target. We hence show how our system can learn accurate arm movements from few example trajectories, and that the model generalizes well to different arm trajectories.

Indeed we wanted to develop an adaptive model that can learn delays and have an internal model that predicts the future state of the agent using DNFs [1]. Each information provided to the agent such as vision and sensory information is integrated using neural field framework as a biologically plausible solution of the Kalman-filter. Path integration mechanisms (PI) is also implemented by DNFs to predict future locations of the arm from a current location of the arm and a movement estimate.

An important part of our model is the existence of internal maps to represent target location and hand locations as well as a map that represents their relative position. Neurons that respond to such stimuli have been reported in the Posterior Parietal Cortex (PPC) [161, 23]. In particular, Buneo and colleagues [23, 44] discuss in some length how a hand centred target map could be calculated with the appropriate coordinate transformation of eye-centred hand and target maps.

An important behavioural finding that we use in our model is that the velocity profile of the movement is bell shaped for both pointing and grasping the target [18]. Averbeck reported a linear relation between velocity of hand movement and neural activity in areas 5 and 2 of parietal cortex in monkeys [45]. Although the reason for having such bell shape velocity profile remain unanswered, in the proposed model, we obtained this velocity profile by combining two DNF maps which performs a bell shape velocity profile as found in human subjects.

According to Crawford et al. [27], hand and target location get updated in parietal reach region (PRR) during the movement. PRR includes the medial intra-parietal (MIP) cortex and visual area V6A which both are activated for reaching movements [23, 28]. The Cerebellum also predicts and updates these predictions based on visual information and that is why human is able to perform precise and percept actions [29, 30, 31].

3.2 Related work

Strauss and Heinke [3] implemented a robotics-based approach for target reaching to investigate current theories of human behaviour. The model implemented by Strauss and Heinke use a two-joint arm with LEGO components. The visual input is provided by a fast Bumble-Bee camera. Basic image processing techniques from computer vision are used to detect specific colour tags with which the components in the environment are labelled. This simplifies the object recognition component of the model that is not the focus of the studies.

Their main goal was investigating the dynamic interaction between a decision process and the control of the arm movement. They also wanted to illustrate the serial order of action planning and subsequent execution that dominated most of current thinking. They also implemented the attentive decision system and the arm controller by the DNFs to implement the biased competition [162].

They used an internal velocity signal to calculate analytically the anticipated future location of the end effector, however in this study we are asking how such a system could be augmented with an adaptive observer so that the system could function with changing environments and slow or missing sensory feedback.

Since our focus in this study is on the control dynamics of the motor arm and not the decision component of the experiments, we only summarize the control architecture of the model by Strauss and Heinke in Figure 3.1. The components of the engineering framework that have no direct biologically motivated implementation are shown with square boxes which includes the basic image processing to find the locations of the colour tags in the image and also the calculation of the motor command from the current location of the end effector to the desired location calculated from an internal velocity signal. The main biologically motivated components are the representation of the end effector location within a DNF (end-effector map) and the location of the target in a DNF (Target map). From these maps, the model computes a hand-centred target map (HCT map) by convolving the Target map with the end-effector map. The activity packet in the HCT map specifies the orientation and distance between the arm and the target. The aim of reaching a target translates to moving the activity packet to the centre of the HCT map.

One important contribution of the work by Strauss and Heinke is that they were able to produce bell-shape velocity profiles that are observed in human motor control. Such a velocity profile means that the movement speed is not static, but that the speed of the movement varies with the location of the arm relative to the target. Martenuik et al. [18] illustrated this bell-shape profile for arm movements in human behaviour experiments. At the beginning of the movement, the arm starts slowly then reaches its highest speed proportional to the distance from the target. The peak of the speed profile for grasping the target occurs around the middle of the movement. In contrast, the peak occurs later for pointing to a target instead of grasping an object at this position. Speed decreases when the arm is close to the target in order to produce a more accurate movement [18].

In Strauss and Heinke's work, after obtaining a velocity value, the future location

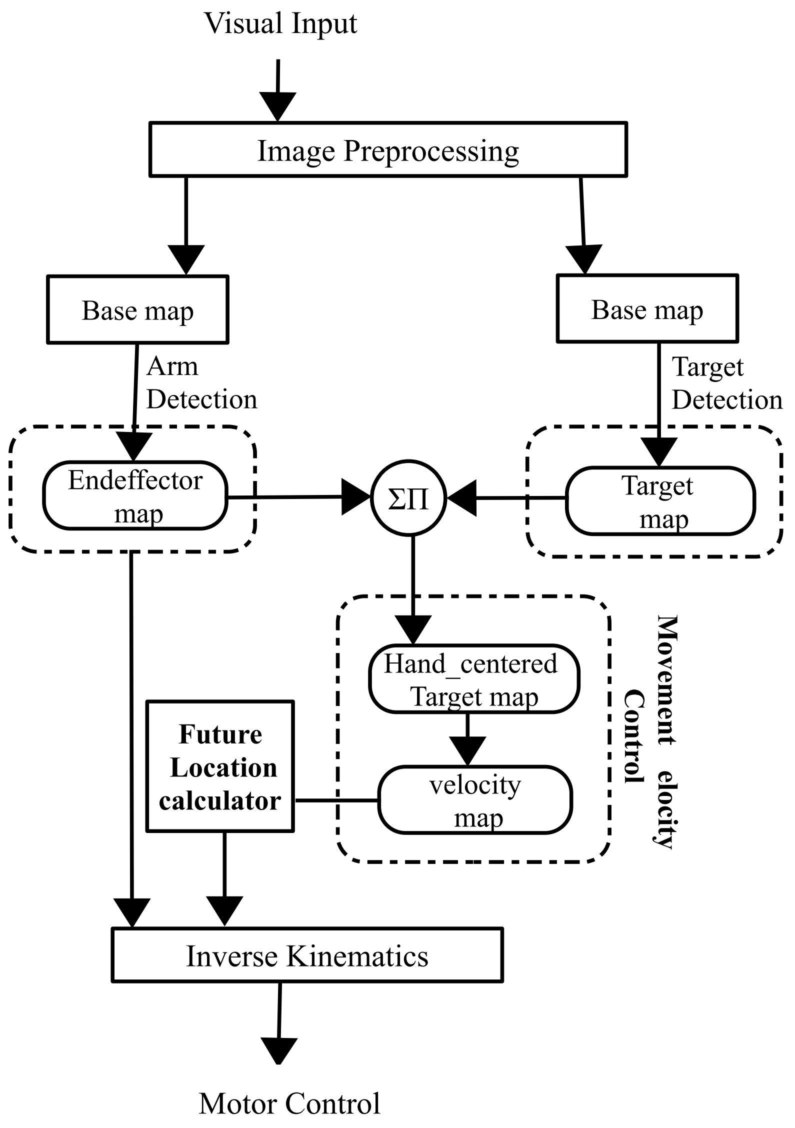


Figure 3.1: Simplified version of the model by Strauss and Heinke [3]. This figure forms the basis of the proposed model. An internal model (including the observer and motor adaptation) has been implemented on top of this model in our proposed model. All rounded rectangles are DNFs and $\sum\prod$ is convolution of two maps.

of the arm is analytically calculated by adding the linear change for a specific time. In addition, an inverse kinematics model is applied to determine the corresponding turning value for each motor joint as specified in the appendix of Strauss and Heinke [163].

I re-implemented the relevant components of the model by Strauss and Heinke including the mechanisms for generating the velocity map only with slight changes as specified below. Our main contribution on top of the previous model is the addition of an adaptive internal model that replaces the analytical calculation of the next position and also enables movements with delayed camera input and predictive blind movements. We implemented an adaptive observer that learns how to move in any desired direction and speed based on few training examples. Moreover, the observer adjusts itself and learns to compensate the delay in the system. We also show that the model can produce velocity profiles that are consistent with both, grasping and pointing movements.

3.3 Proposed Model

Delays in the system such as those due to sensory transduction, central processing, and in the motor output, have a significant effect on motor behaviour. Delays in the arm control systems can be of the order of 30 ms to 300 ms, where the lower bound is typical for spinal reflex and the upper bound reflects limits when a highly processed visually guided response is important. To overcome these delays, some engineering approaches such as the Smith-predictor and the model predictive control (MPC) have been proposed [56, 139, 63]. Wolpert et al. [139] discussed how far a Smith predictor can model many experiments with delay, and there have been extensive discussions of the possible role of the cerebellum and its possible implementation of a Smith-predictor. Moreover, Weir et al. [164] showed that humans can achieve zero-lag in tracking predictable targets which could be due to the anticipation of the target by model predictive control. In MPC the delay is anticipated before it occurs and is hence known in advance [165]. MPC utilizes an inverse model of the plant. Thus it knows how to control commands and translate it into behaviour by the plant. Our model works with delays in the sensory and motor system by adjusting the motor strength, and we discuss how our implementations compare to the control architecture of a predictive observer.

3.3.1 Model overview

Figure 3.2 shows a systems level overview of our proposed architecture to control an arm movement so that the arm can point to arbitrary targets. In this model, the main goal is to learn to predict the future location of an intended arm movement within the

DNF framework. In other words, instead of arithmetically solving the path integration to guide the arm toward the target, we introduce an adaptive observer that *learns to predict* the consequences of motor commands based on previous observations.

DNF models, as discussed in chapter 2, can be used to memorize feature values such as the pose of a robot arm. While there are additional solutions of the neural field equations [106, 166], our applications of DNFs only require the basic existence of localized bump solutions as illustrated here. Since DNF models can store current feature values, they have been proposed as biological mechanisms to internally represent features of an organism such as orientation [167, 129] or space [128, 130, 168]. In the application described in this chapter, the activity in the map encodes locations such as the position of a target or the end effector of a robot arm. Such positions can then be updated on a continual basis from visual cues.

The new model is still based on DNF maps to represent the target and the end-effector location. The basic difference is that we apply a path integration mechanism in the end-effector map to move the packet of activity to a new location corresponding to the future location of the arm. Such a module, which can integrate sensory information with predictions based on an efferent copy of a motor command, corresponds to the notion of an observer. In our case, we include learning components, detailed further below so that the observer is itself an adaptive system. As an added benefit, the observer can combine internal expectations of arm states with possible sensory feedback in a flexible manner, even letting the system function without timely visual feedback since it learned to anticipate certain consequences of internal motor commands.

In order to make this system possible we needed to consider two major issues. The first is to build an internal model of arm movements in any direction. Specifically, this includes learning appropriate asymmetric weights from few examples and to show that they can generalize. We will discuss this first in computer simulations that allow us to control the conditions without the added dynamic of motor adaptation. The second issue is that the observer needs to be able to cope with delays and dead-times in the system. We show that this can be provided by motor adaptation. In the end, we show that the system can learn in the real robot environment with sensory samples and integrated dynamics.

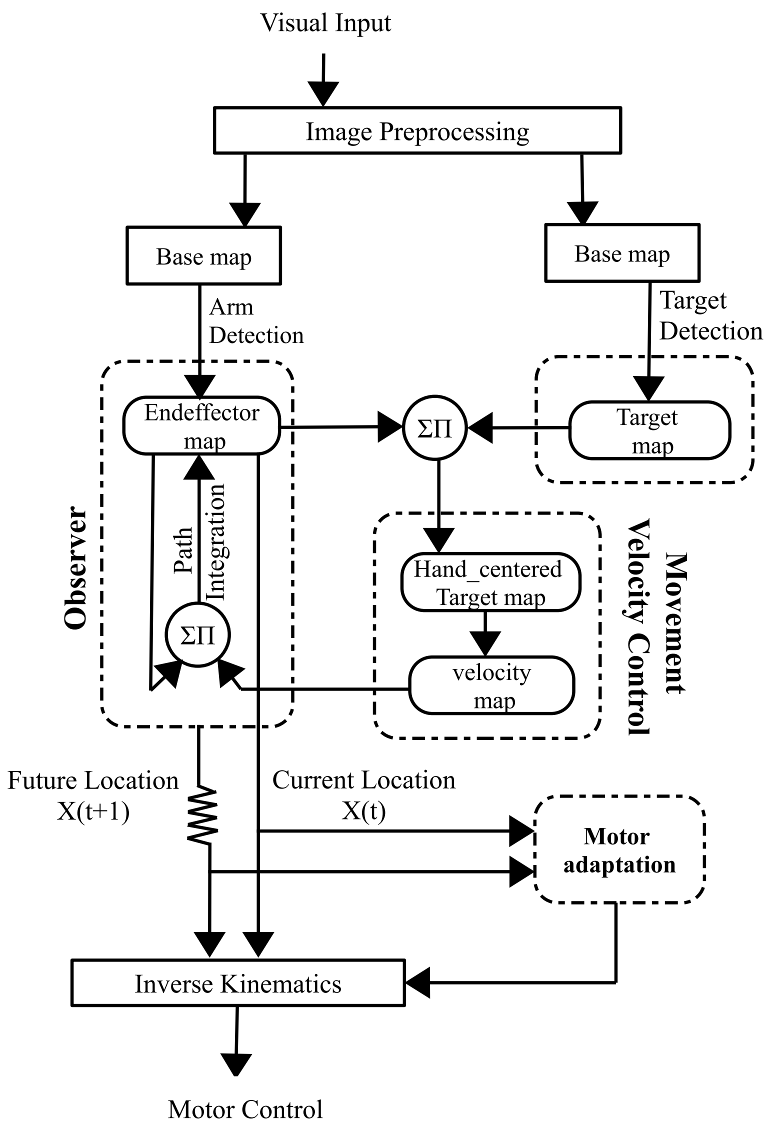


Figure 3.2: The proposed architecture. The observer integrates internal and external sensory signals to estimate current location of the robot arm. Integration of information and finding the future location of the arm is implemented by DNFs. The observer is able to guide the arm toward the target when the visual information is removed during the movement. The motor adaptation also learns motor parameters that compensate for the error between the observer's prediction and the actual, observed arm locations. This implicitly compensates for the dead time in the system. All rounded rectangles are DNFs and $\sum\prod$ is convolution. Dashed rectangles indicate main sections of the model.

3.3.2 Visual target and end-effector detection

Figure 3.3 shows the configuration of the robot arm that is built with a Lego Mindstorms kit. It has two joints and is intended to simulate a human's right arm. In contrast to the original implementation by Strauss and Heinke, who used Java programming and a Bumble-bee camera, we used Matlab and a simple web camera. This demonstrates that the schema also works in a less efficient implementation. Red Lego pieces were used to indicate the target, while blue Lego pieces indicate the end of the arm. The length of the arm is approximately 30 cm (upper arm 15 cm, forearm 15 cm) and the upper arm can move from -45 to 225 degrees and the forearm from -80 to 260 degrees. However, we restrict the elbow to a 180-degree range such that the mapping from shoulder and elbow angle to end-effector location is unique. The camera is positioned approximately 90 cm above the middle of the table, directly facing it. After capturing the environment by the overhead camera, the current locations of the arm and target are detected by blue and red colour filters, respectively.

Specifically, an RGB image of size 120×160 is captured by the webcam. In order to reduce processing time, images were re-sampled to 60×60 images. For detecting blue and red coloured objects, we used a difference of Gaussians filter on a colour opponent image. For example, to detect a blue object the green and red channels are subtracted from the blue channel of an RGB image. These values are rectified (clamped to be positive), and normalized by the blue channel (or a minimum brightness threshold, whichever is larger). This is passed to the difference of Gaussians filter which is positive in a small region corresponding roughly to the expected size of the object, and negative in a larger, surrounding region. In this way the system is selective to the conjunction of size and colour, and robust to changing lighting conditions.

The camera obtains visual information in pixel space. In order to show movements in the physical environment of the movement plane, we provided a mapping function to map camera pixels into Cartesian coordinates of the workspace. The parameters of the mapping function were adjusted by supervised learning on about 30 examples of known marker positions in a 75×55 cm wide area. We used a parameterized quadratic polynomial as the mapping function and applied maximum likelihood estimation with

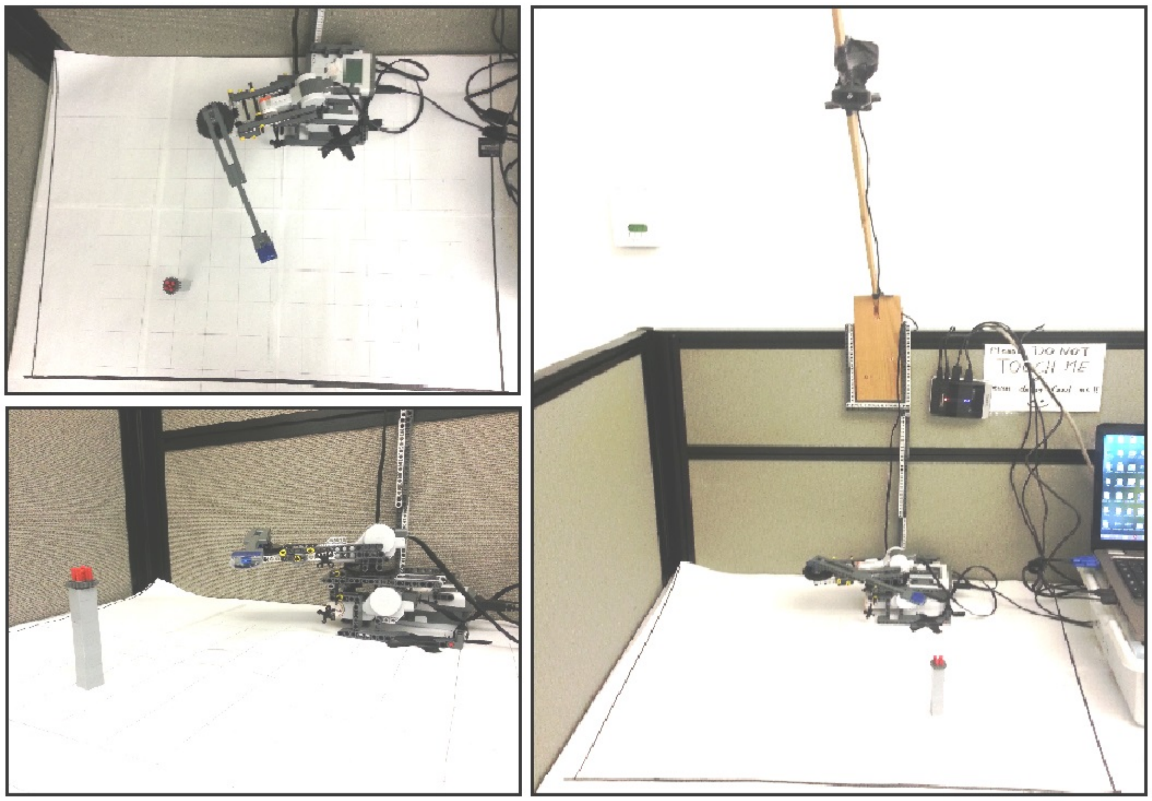


Figure 3.3: Configuration of the robot arm used in the current experiments. The Lego arm uses gears to provide greater accuracy to its movements. A simple webcam is mounted overhead to provide visual feedback.

the samples to estimate the parameters (shown in Eq. 3.1).

$$\begin{bmatrix} X_p \\ Y_p \end{bmatrix} = \begin{bmatrix} A_1 & A_2 & A_3 & A_4 \\ B_1 & B_2 & B_3 & B_4 \end{bmatrix} \times \begin{bmatrix} 1 \\ X_c \\ Y_c \\ X_c * Y_c \end{bmatrix} \tag{3.1}$$

where $\left[\begin{smallmatrix} X_p \\ Y_p \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} X_c \\ Y_c \end{smallmatrix}\right]$ represent locations in pixel and Cartesian coordinates, respectively. As the acquired images of the samples showed Gaussian variations of the locations, least mean squares regression can be used to find the mapping function. This mapping function was then used to convert the locations of the colour spots (blue for end of the arm, and red for the target location) in the image to Cartesian coordinates of the workspace. Note that this procedure is only to provide graphs in a more meaningful workspace coordinate system. Since the system is adaptive it could learn to handle movements in other coordinate frameworks.

3.3.3 Hand-centred target map and velocity map

The visual information of target and end-effector locations become inputs to the target and end-effector map, which in turn are used to calculate the hand-centred target (HCT) map. To do this we convolved the target map and the end-effector map with each other,

$$S_{\mathrm{HCT}}(x_1, x_2, t) = \int_{x_1'} \int_{x_2'} r_t(x_1', x_2', t) r_e(x_1 - x_1', x_2 - x_2', t) \mathrm{d}x_1' \mathrm{d}x_2' \tag{3.2}$$

where r_t and r_e are the firing rates of the target and end-effector maps, x_1 and x_2 specify the location in the hand-centred map, and x_1' and x_2' the location in the target map.

While we follow here the model of Strauss and Heinke, it would be interesting to think about more direct neural implementations. This is specifically desirable in light of the recent success of convolutional neural networks [169]. Convolutions could be achieved with synaptic connections that are repeated throughout the map, but it is an open question if other forms of more efficient neural computations could achieve such functionality.

This signal S_{HCT} is used as the input to the HCT map. The activity of the HCT map corresponds to the difference in target locations and end-effector location. Hence, the packet in this map is centred when the hand reaches the target. Following [3], the activity of the HCT map, f_{HCT} is used to produce the input to the velocity map,

$$\begin{aligned} S_v(x_1, x_2, t) &= M \times \frac{c_{\mathrm{zero}}}{\sigma_{\mathrm{zero}}\sqrt{2\pi}} exp(-\frac{x_1^2 + x_2^2}{2\sigma_{\mathrm{zero}}^2}) + \\ & N \times \int_{x_1'} \int_{x_2'} w_{\mathrm{v}}(x_1 - x_1', x_2 - x_2', t) f_{\mathrm{HCT}}(x_1', x_2', t) dx_1', x_2'. \end{aligned} \tag{3.3}$$

The velocity map has two inputs. One of the inputs is a predefined Gaussian activation aligned with the centre of the map. The width of this Gaussian is σ_{zero} and the amplitude c_{zero}. This input represents the resting hand or zero velocity. The second input is the broadened activity of the HCT map, where the broadening is performed by convolving with a Gaussian filter. The parameters M and N are coefficients to adjust the strength of the central input and the input from the HCT map. In the presence of the broadened activity from the HCT map, the final activity is created by combining these two inputs. The distance between this new activity packet and the centre of the map is translated into the speed of the movement at each step. The location of the activity packet is defined by the centre of mass of the field.

When the movement starts to reach the target, the location of the broad velocity packet will be close to the centre. The packet will then gradually shift away from the centre, but since the distance between hand and target will also decrease, the packet will later also move back toward the centre. This creates a bell-shaped velocity profile as shown further in the results section.

3.3.4 Learning path integration

To implement an observer (the internal model) that predicts future arm positions from efferent copies of the motor command, we needed a mechanism to update the end-effector map from the information in the velocity map. This is the essence of a path integration mechanism that was introduced in the neural field formalism in various ways [167, 128, 129, 130]. As outlined in section 2.1.2, to update the activity packet we need to apply an appropriate asymmetric weight kernel to the end-effector map. To provide such asymmetric weight kernels, movement examples are needed.

I first learn a movement in a specific direction by either simulating input to the end-effector map or by having a teacher move a hand target through the visual field in a specific direction. During this movement, we built up a trace term, $\overline{r}$, and we used this trace term with the input of the last frame to calculate the asymmetric weight kernel. Since this is assumed to be the same at each location in the visual field, we used again a convolution to move an activity packet from each possible location in this learned direction.

$$\hat{w}^{\mathrm{mov}}(x_1, x_2) = \int_0^{2\pi} \overline{r}_i(x_1 - x')r_{\mathrm{i}}(x_2 - x')dx' \tag{3.4}$$

where x_1 and x_2 are locations in DNF map while x' is every other possible location on the map.

Finally, the asymmetric weights are normalized by Equation 3.5 and adjusted by a global inhibition constant C.

$$w^{mov}(x_1, x_2) = \frac{\hat{w}^{mov}(x_1, x_2)}{\max_{x_1', x_2'}(\hat{w}^{mov}(x_1, x_2))} - C \tag{3.5}$$

In order to provide movements in all possible directions, arm movements for all these directions have to be provided. This seems not very realistic or practical due to the continuous nature of possible movement directions. I, therefore, explore three methods that we believe have not been studied before. The first one is the generalization of the learned movement in one specific direction to all other possible directions with a rotation operator. In this case, we implemented this with an algebraic solution of a rotation matrix, although it is possible that such a rotation can be learned in a neural system itself. Such an approach would then correspond to an analytic rotation.

To do this, we generated some movement examples in one direction, from the bottom centre toward the top centre of the map to learn the movement kernel w^{mov}. We then developed a program to rotate the obtained asymmetric weight kernel for this direction by a desired rotation amount. More specifically, to accurately rotate the weight tensor in the discretized form, We use the function `imrotate()` in Matlab with `bicubic` and `crop` options to obtain the same size image as the original one. A specific problem is that `imrotate()` fills the corners of the rotated image with zeros while we needed to fill them with more appropriate values. As the DNF map that we wanted to rotate has periodic borders, we create a larger version with four tiled weight kernels and after rotation, we extract the rotated asymmetric weight kernel from the centre of the image. The rotation has to be done carefully since unwanted asymmetries will cause considerable inaccuracies in the predicted movement.

The other two methods that we explored here is to learn movements in a few different directions and to interpolate between them in two different ways. In one we took the movement tensor for the closest learned position to see if this is sufficient for control purposes. The second scheme is a simple population decoding scheme where we interpolated between the two closest learned directions. Several different interpolation functions could be considered, and we chose here a simple triangular window function. Movement examples are thereby made for every $\Delta\alpha$ degrees, and we discuss below how different resolutions will influence the movement accuracy and the generalization in different directions. This population decoding dynamically forms a kernel for the current direction. More specifically, let the desired movement angle be α and we used training examples every $\Delta\alpha$ degrees. The closest training angles are therefore at

$$\alpha_1 = \lfloor \frac{2\pi}{\Delta\alpha} \rfloor \Delta\alpha \tag{3.6}$$

$$\alpha_2 = (\lfloor \frac{2\pi}{\Delta\alpha} \rfloor + 1)\Delta\alpha, \tag{3.7}$$

where $\lfloor\ \rfloor$ indicates the truncation operator. The weight matrix for the desired direction is then

$$w_{\alpha}^{\text{mov}} = \frac{\alpha - \alpha_1}{\Delta\alpha} w_{\alpha_1}^{\text{mov}} + \frac{\alpha 2 - \alpha}{\Delta\alpha} w_{\alpha_2}^{\text{mov}}. \tag{3.8}$$

In the next section, we discuss the performance of these path integration methods.

3.3.5 Motor adaptation

Testing reaction time of humans reveals that delays occur for two reasons [170], both of which are present in the proposed system. One is the time taken by the visual system to detect and localize objects. The second occurs while transmitting motor commands to the actuators. During movement, this delay causes error between the actual location of the arm and the sensed location (and thus the internal model's interpretation of where the arm is). This discrepancy has to be compensated to obtain accurate reaching. The system does, therefore, include motor parameters when translating the motor commands to the appropriate power of the motors that are adaptable as specified below.

To move the arm physically, an appropriate power for the motors, both for the shoulder and the elbow, must be provided. This is done similar to the original work by Strauss and Heinke through inverse kinematics [3]. However, we also included two additional motor command parameters that are modifiable from training examples. The velocity of the movement in each step is provided by the velocity map, but these parameters are adjusted by actual movements using the discrepancy of the predicted and actual movement.

More specifically, to be able to alter the power of each motor, we defined the modulation amplitude a_{motor}. Each parameter is a coefficient that is multiplied by the value of the power that is sent to the motor. Since we have a mapping function that obtains corresponding angles for the shoulder and the elbow corresponding to a certain location, We can tune the two motors independently. At each step, the error between the angles corresponding to the observer-predicted location and actual location of the arm is obtained. The parameters are then adapted based on the direction of the error and the direction of the movement according to Equation 3.9.

$$a_{\text{motor}} = a_{\text{motor}} - \alpha \text{ sign}(\theta_{\text{predicted}} - \theta_{\text{actual}}) \text{ sign(u)} \tag{3.9}$$

where a_{motor} indicates either the shoulder or elbow motor parameter, α is the learning rate, $\theta_{\text{predicted}}$ is the angle corresponding to the location the observer predicted, θ_{actual} is the angle of the actual resulting location of the robot arm, and u is the motor command which is positive for clockwise rotation and negative for counterclockwise. In the other words, when the predicted location by the observer is behind the actual location of the arm, the corresponding motor parameter should be decreased to slow down the speed of the movement and vice versa. Only the sign of the gradient is used for learning as this more closely mirrors the *relative* error, i.e. small movements should be correspondingly more accurate than large ones.

The essence of this adaptation algorithm is that when the turning angle of each joint of the robot arm is less than what the observer has predicted, then the motor strength is increased and vice versa. Since this is learned from examples with actual camera inputs, this adaptation also adjusts indirectly for any delay between the camera input and the actual location. As the motor parameters in our system have been initialized with random numbers at the beginning of the simulations, it takes some time to find proper values for these parameters. Note, however, that the system never stops adjusting these motor parameters, although these parameters are not stable in stable circumstances after an initial learning phase. While we use here a Cartesian coordinate system to demonstrate the principle, this should also be possible in a retinal map since the learning algorithm only depends on over- or under-shooting of an actual movement.

It would be very informative to study specifically if the representations of the target and hand locations reflect anticipated locations. Also, the PPC plays an important role in converting sensory information to proper motor control, that is from visual to motor coordinates [44]. In our model, we used a mathematical solution for this inverse transformation and we assume that it occurs in PPC.

For future work, this transformation could be learned from an initial motor babbling stage, i.e. from observing the change in end-effector location resulting from random motor commands, and continuously fine-tuned to adapt to changing kinematics of the body and environment.

3.4 Results

Learning path integration in different directions I start by discussing the performance of the different parts of the system before demonstrating the performance of the whole system. In this subsection, we discuss first the different schemes to generalize the learned movement tensors. This part is first done by computer simulations only to have more flexibility in the control of the system. We do this by directly providing Gaussian input to the end-effector map. The only concern doing this in simulations is that the delay in video input is ignored. However, this will not effect the trace term (Equation 2.13) since we assumed the delay to be constant. This trace term has to be correlated with a movement indicator cell that we assumed to be active in a sufficiently long time interval during the movement. Below, we also demonstrate that the whole system can function in a robotics environment with learning from camera input.

To demonstrate how the different methods of learning the asymmetric movement kernels effect the resulting movements, we show the trajectories generated by the internal observer without sensory feedback in a centre-out reaching task. In a typical centre-out task, the goal is moving from a centre of a circle to a point on the edge of the circle. In our experiments reported here we move an activity packet located at the centre of the map for 150 time steps outwards in different directions. The results are shown in Figure 3.4. Results are shown in three different rows, where we used training examples every 45 degrees in the first row, every 30 degrees in the middle row and every 15 degrees in the bottom row. For all these cases we tested movements for every 15 degrees with three different methods. In the left column we just used the nearest movement kernel to generate the movement. Hence, only the movements in the learned directions can be represented. In the second column, we used a triangular window function, resulting in a weighted average of the two closest movement kernels. This simple form of a population decoding produces already quite good generalized movements even with a small number of training directions. The last column actually only uses training examples from one direction and rotates the corresponding movement kernel by the appropriate degree. It is feasible that such a computation is learned by a neural system itself, and we will refer to this methods simply as analytic rotation mainly to emphasize that an internal calculation is made [171].

It is clear that the analytic rotation which produces correct rotations of the movement kernel in this task could be expected to perform accurately. However, since movements in different directions could have different effects, this method is also limited. What is more interesting in these results is that even a very rudimentary population decoding of weight kernels for a few movement directions can lead to very good movement predictions in directions that have not been part of the training set.

In the next experiment, we simulated reaching movements from various starting positions to various end positions using the kernels learned in the previous experiments. The results are shown in Figure 3.5. The figure has a similar outline to the previous figure, with different learned directions in the three rows (45 degrees top, 30 degrees middle, and 15 degrees bottom), as well as columns for the three generalization methods. The arm was always able to reach the target within a few steps, even when using only 45 degrees movements. In these experiments, visual information during the movements is available, thus the knowledge of the true position is always available. As population decoding can interpolate between learned directions, it produces smoother trajectories that closely approximate analytic rotation.

While the results above were reported for simulated input, this can also be done with camera input. Figure 3.6 shows a trajectory which has been captured by the camera. To make this trajectory the hand tag (end effector indicator) was moved across the field of view by a human. Each ”+” shows the location of the end effector at one time during the movement.

Based on the locations within the trajectory of the arm, in each step an activity

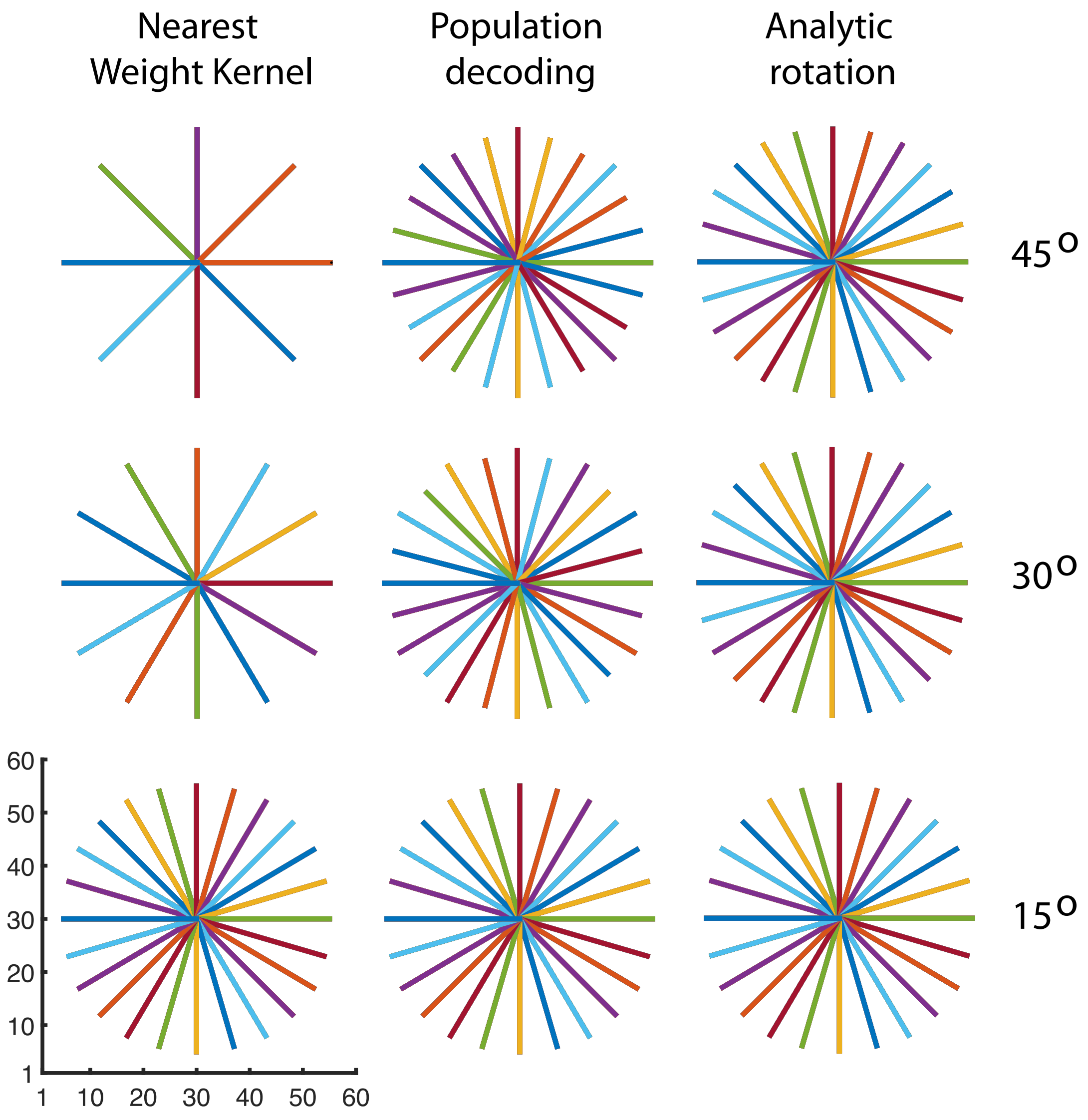


Figure 3.4: Trajectories of moving the end-effector from the centre outwards for 150 time steps in different directions. The movement weight kernels where thereby learned for a different number of directions, every 45 degree in the top row, every 30 degrees in the middle row, and every 15 degrees in the bottom row. Shown are the trajectories for the test movements every 15 degrees. In the left column, the nearest movement kernel was used. The middle column used the triangular window function from the nearest kernels, and the right column shows the performance of the calculated (analytic) rotation. Different colors are only for the beauty.

packet is made in the end-effector map and a trace rate gets updated according to 2.13. At the end of the movement, we convolved the latest activity on the map with the obtained $\overline{r}_i$ according to Equation 3.4 and normalize. This is equivalent

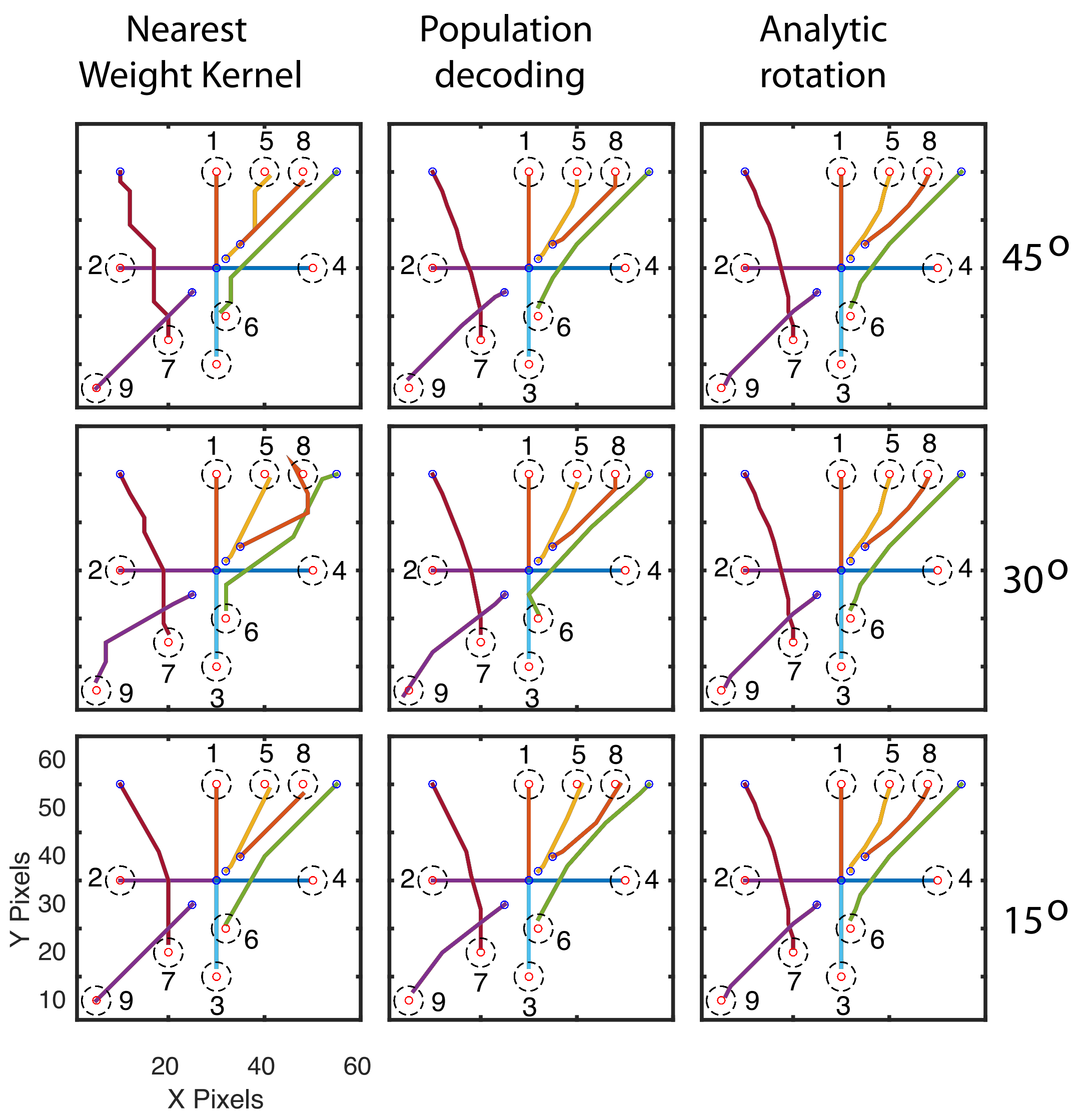


Figure 3.5: Trajectories of 9 movement examples generated from movement tensors with the different generalization strategies. Applying training movements on smaller degrees result in somewhat straighter and smother trajectories. Blue dots represent start point of each movement. Red dots are target locations and black dashed open circles show target zone for each destination. As it is shown in all examples, the arm stops inside the target zone.

to averaging across the whole movement in the limit of moving forever because the activity spends most of its time with a constant shape and rate of change of location, with only the beginning of the movement being slightly different. We also checked that the results are consistent with averaging over all steps.

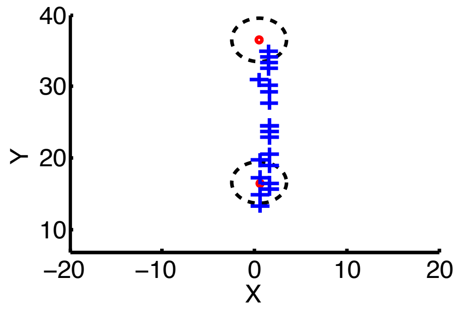


Figure 3.6: Input captured by camera for learning path integration. Here instead of simulating the training data, real movement has been done which has been captured by camera. Each blue cross indicates location of the arm during the movement at a time. Bottom red dot inside a dashed circle is start point and the top one is the end point of the movement.

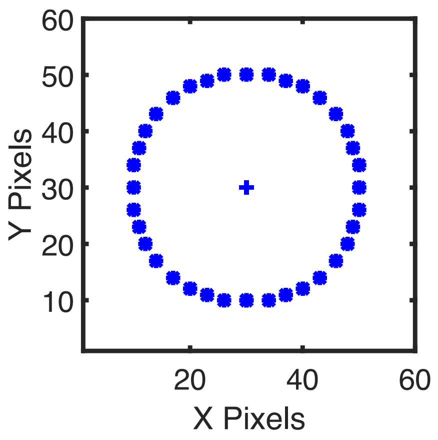


Figure 3.7: centre-out task with asymmetry obtained by a real movement and analytic rotation for every 10 degrees on the circle. The Figure shows that performance of the movement does not change in different directions. Moreover, it shows using simulated movement and the real movement have the same performance.

If this asymmetric weight kernel is applied on the end-effector map the activity will move to the top. To move in a particular direction, the analytic rotation strategy is used to dynamically make an appropriate rotating weight from the trained asymmetry. Figure 3.7 illustrates the centre-out task using the asymmetry obtained by this real movement and analytic rotation. As was seen earlier, with more training examples population decoding would give comparable results.

Adapting motor parameters and compensating delay time In this model, we used two parameters to adjust the speed of the movement for motors, one for the

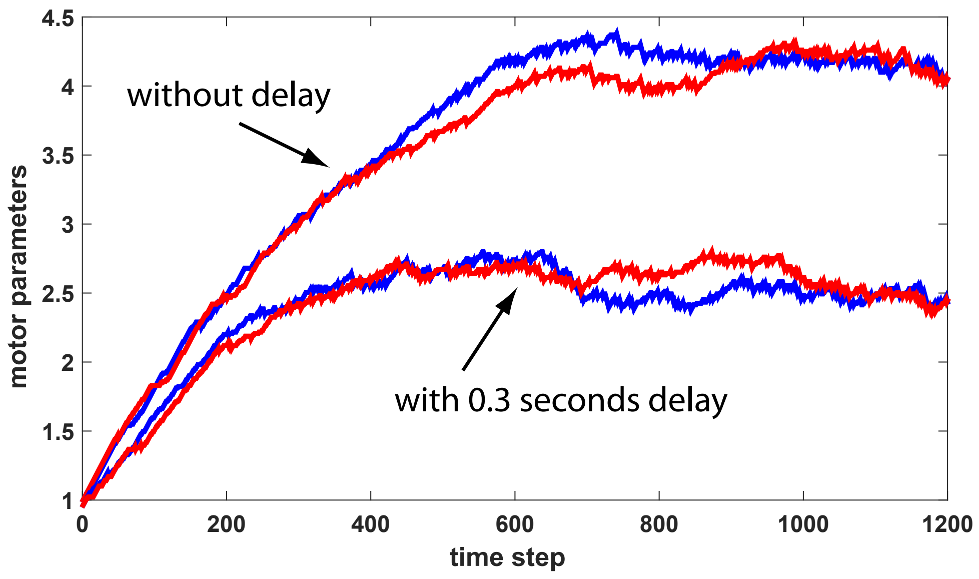


Figure 3.8: Adaptation of the motor strength parameters for the shoulder (blue curve) and elbow (red curve) with and without artificial delay. The artificial delay made by pausing the recieving sensory feedback.

shoulder and the other one for the elbow. The motor strength modulation parameters a_{shoulder} and a_{elbow} are initialized to 1. Figure 3.8 illustrates the evolution of the motor parameters over the course of learning according to Equation 3.9. Virtual target locations were randomly generated uniformly over the region of the workspace at least 10 pixels from the arm, resulting in different directions and distances relative to the current location of the arm but emphasizing larger movements. Errors of ≤ 2 degrees were treated as if they were 0, i.e. caused no change in the parameters.

Figure 3.8 shows the change of the strength parameters for the shoulder (blue curve) and the elbow (red curve). After some training examples without any artificial delay applied, both values settle in a regime around 4.7, though the shoulder parameter is slightly larger than the elbow. The ongoing fluctuations are due to many different movements with small and large amplitudes while the systems also exhibit noise.

Figure 3.9 shows the system performance after a different number of learning steps. In this figure, crosses show predicted locations by the observer, while the open circles illustrate the actual locations of the robot arm. Motor parameters are initialized to arbitrary values (here 1), thus at the first trial there is a significant error between the

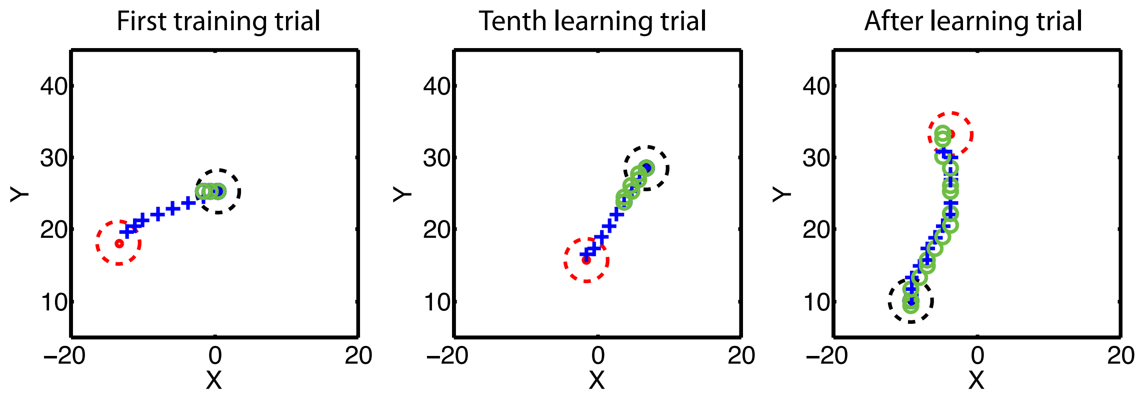


Figure 3.9: Performance of observer after adjusting motor strength parameters at early stage of learning, during learning, and after learning. In early stage of learning, motor parameters are too small which make big error between where the actual location of the arm is and where the observer senses. After learning phase, motor parameters get tunned and as it is shown, the is very small error between sensed location and actual location.

predicted and actual location of the arm. However, over the course of learning the discrepancy gradually decreases.

The tuning of the motor response compensates implicitly for the sensory and response delay. Initially, this delay and the low motor speed cause motor responses that fall short of the necessary movement. By learning from examples, this is countered by increases in the motor parameters.

Several forms of delay, dead time, or slowness in different parts of the system cause inaccuracies in the internal model. There is latency in the image acquisition as well as significant processing time to execute the model. In our installation (4-core processor running Matlab 2015a) it takes approximately 0.05 second to capture the image from the camera. Computing the activation of the DNF maps requires 0.45 second on average, with the velocity map taking up the bulk of that time. The last component includes path integration, sending commands to the motors, and the adaptation, which together takes approximately 0.01 seconds.

In the model, we make it now clear that we specifically look at a form of image acquisition delay, that is, the time delay by which we provide the image to the internal model. We now vary, as suggested, this time to show how the adaptation mechanisms perform with such changing parameters. In particular, we introduced an artificial pause after the image acquisition from 0.1 to 0.9 seconds in 0.1 second increments.

The delay was added after capturing an image, resulting in longer dead time for processing the model. In this way of implementing acquisition delay we also delay the computation of the internal model while the arm is still moving. There are alternatives to consider such as the internal model running independently (or in parallel) during the delay.

Figure 3.8) illustrates the evolution of the motor parameters with an artificial delay of 0.3 seconds. As the delay increases, the value of the motor parameters learned by the system decrease (see Figure 3.10). Longer delay causes longer movements in each step for the robot arm, thus decreasing the motor parameters reduces the discrepancy between the actual location of the arm and that predicted by PI. For very long delays, the motor parameters cannot be learned on the current hardware platform, as either they are so small that the motors do not move at all, or large enough that the robot arm passes the predicted location.

Moving in the dark During the learning phase, visual information is essential. However, after the motor parameters and the path integration are tuned, the robot arm is able to move accurately even in a dark environment (without visual feedback). Three example trajectories from an actual robotic arm movement are shown in Figure 3.11. The arm starts the movement with visual feedback, but the vision is removed by covering the camera during a number of subsequent time steps. The time steps without visual feedback have been marked only with green "+". Even without vision there is still a fairly good estimation of the true location, and new visual input is then used to adjust the prediction further to reach the target.

This capability is most clearly demonstrated in Supplementary Video 1:

https://projects.cs.dal.ca/hallab/wiki/images/4/40/Supplementary1.mp4

and Video 2:

https://projects.cs.dal.ca/hallab/wiki/images/6/61/Supplementary2.mp4

Both videos record the robot arm pointing to a target but experiencing occluded vision partway through the movement. In Supplementary Video 1, a simplified version of the Strauss and Heinke model [3], which has no internal model, is controlling the arm. While the camera is covered the arm moves with the same velocity as it did just before losing the visual signal. This occurs since the activity of the end-effector map

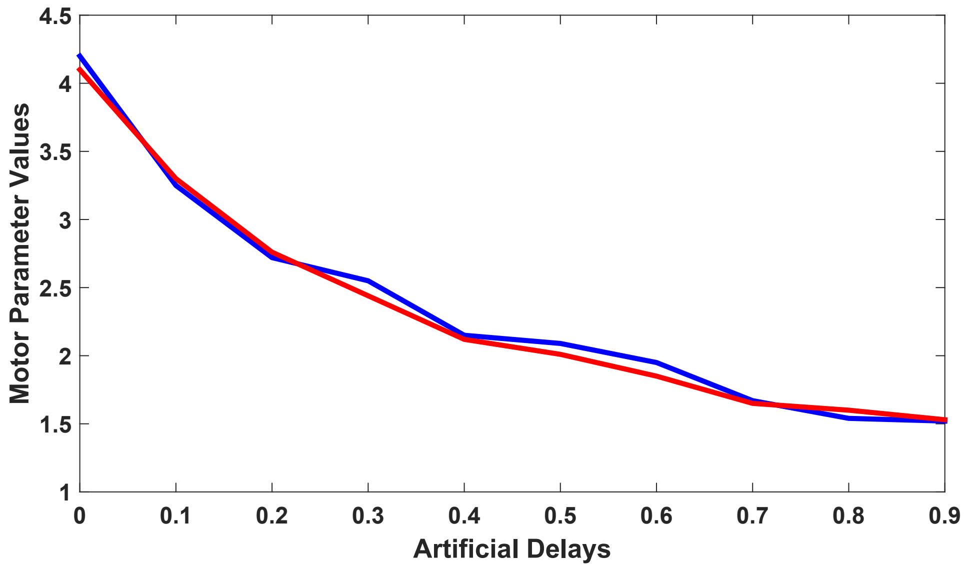


Figure 3.10: Shoulder (blue) and elbow (red) motor parameter values learned for varying image acquisition delay time. Increased delay leads to decreased motor parameter values such that the mapping from motor command to distance travelled remains the same. This minimizes the discrepency between the actual movement and the internal model's prediction.

is unchanged during this period, leading to the same activity throughout the model and thus the same generated motor commands. When the camera is uncovered the maps integrate the new information and alter the arm's trajectory to reach the target. In Supplementary Video 2, the model proposed in this work is operating the arm. Here, when the camera is covered the internal model continues to alter the activity of the end-effector map via the learned, asymmetric weights. Thus the arm performs essentially the same movement as it would with non-occluded vision. Once vision is restored, it uses this information to make any small corrections necessary to reach the target.

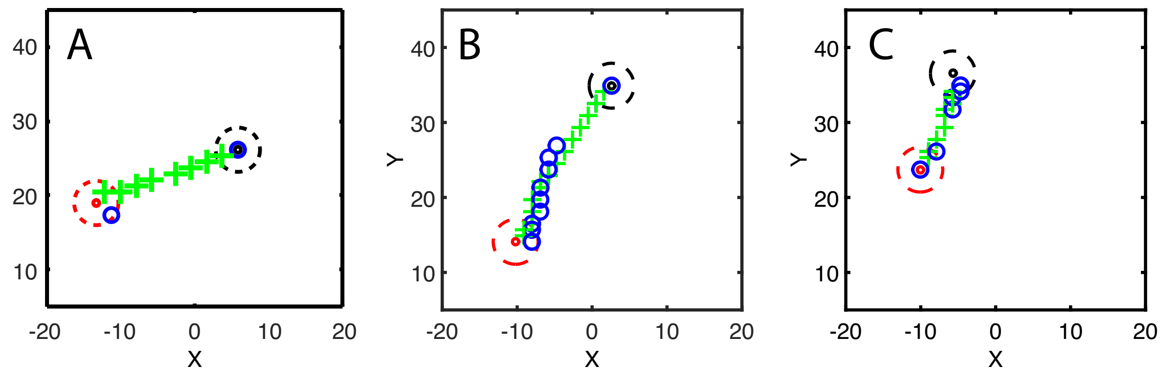


Figure 3.11: Trajectory of the arm movements with partial and full occluded vision. The green crosses show the predicted position of the arm while the open circles are the actually position extracted from the camera when vision was available. Locations with only green crosses indicate that there is no visual information. This experiments represent performance internal model with fully occluded (A), partially occluded (B and C) vision. Red dashed circle indicates target zone and black dashed circles represents start point of each movement. In fully occluded vision (A), the robot arm would have reached the target zone without restoring the vision at the end of the movement, but we wanted to demonstrate how restored vision will be taken into account to correct predictions. In partially occluded vision (B and C) visual information has been provided by camera at the end.

Table 3.1 shows performance of the observer in different cases with full and partially occluded vision. In this experiment, 10 movements of random length and direction of movement in 3 different visual circumstances have been run. Unsurprisingly, the observer has the highest error when the camera is covered for the entire path. Note that the error is shown here the indicated distance between the stopping point of the arm and the target zone and not to the target itself. If the robot arm stops inside the target zone it is considered to have reached the target and the error is zero.

I also examined the model with a range of artificial delays applied. Figure 3.12 illustrates the effect of this additional delay on the relative error of the model. In this experiment, we set the artificial delay and train the system to adapt to it. Then, using the learned motor parameters, we moved the robot arm for 10 random movements under fully occluded vision. When the arm stopped, we calculated the error between the location of the end effector and the target zone. If the arm has stopped inside the target zone (within 3 cm of the target) the error distance is considered to be zero. Figure 3.12 shows the average error distances (red line) in the range of minimum

Vision conditions	Avg. moving dist.	Avg. dist. to target zone	Error percentage
Full vision	12.06016	0	0
First half occluded	12.5987	0.3098	2.5
Fully occluded	12.6444	0.5766	4.5

Table 3.1: Performance of the observer for 10 random movements. Full vision: camera is on during the movement. The robot arm will reach the target and it always stops inside the target zone. First half occluded: after moving first half way from initial location of the arm to the target vision will be occluded. When the observer predicts that it has been reached the target, camera captures the current location of the robot arm and the error is calculated. In average for a movement about 12.6 cm it has error about 0.3 cm. This means the arm has been stopped 0.3cm outside the target zone. Fully occluded: Except for the first sight, vision is occluded for entire movement. After stopping, camera captures the actual location of the robot arm. In average for a movement about 12.7 cm the robot arm stops 0.5 cm outside the target zone which is about 4.5 percent of the movement.

and maximum error distances (gray area) for 10 random movements for each delay. The accuracy of the system does decrease with increasing acquisition delay as can be expected, but the effect would be much more drastic without the compensation mechanism.

Tracking a moving target The model is able to track moving targets following complex trajectories. Four such trajectories are shown in Figure 3.13. In three of them (top left, and bottom left/right) the target was moved through the workspace, and the arm closely tracked its position. In the fourth trajectory (top right), the object's position was shifted in sudden, large steps and then was stationary for short periods. As seen in the figure, while the arm makes some brief errors, it recovers and executes the trajectory connecting the few target locations. This experiment demonstrates that the consciously running control loop is able to follow changes in the target's position.

The difference between grasping and pointing As mentioned before, Stauss and Heinke showed already that the method of calculating velocity maps supports a bell-shaped velocity profile. However, as pointed out in Martenuik et al. [18], the peak of the velocity is different when pointing at an object or if the instruction is to pick up an object. As can be expected, grasping an object requires slower movements

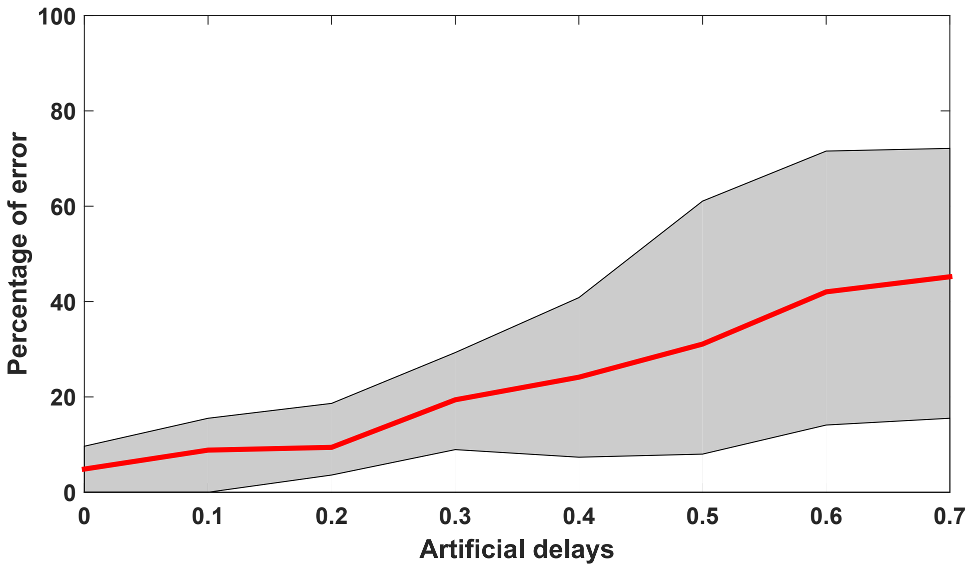


Figure 3.12: Relative error for increasing image acquisition delay time for 10 random movements with occluded vision. Artificial delays are in millisecond. Error is measured as the distance between the final location of the end effector and the target zone, proportional to the length of the movement.

close to the objects so that more carful hand adjustments can be made.

While our model has been developed for pointing movements, it is interesting to note that our model is also able to show velocity profiles more consistent with grasping. Since grasping requires slower speeds to leave time to plan hand adjustments, we lowered the relative strength of the input to the velocity map, which is determined by the parameter N in Equation 3.3. It is conceivable that such a modulation is provided by an intentional motor planning system. Figure 3.14 illustrates the velocity profile for 10 different movements for grasping and pointing to the target where the $N = 0.7$ for pointing and $N = 0.36$ for grasping. The dotted red line shows the average of the velocities which is bell shape consistent with human behaviour data [18].

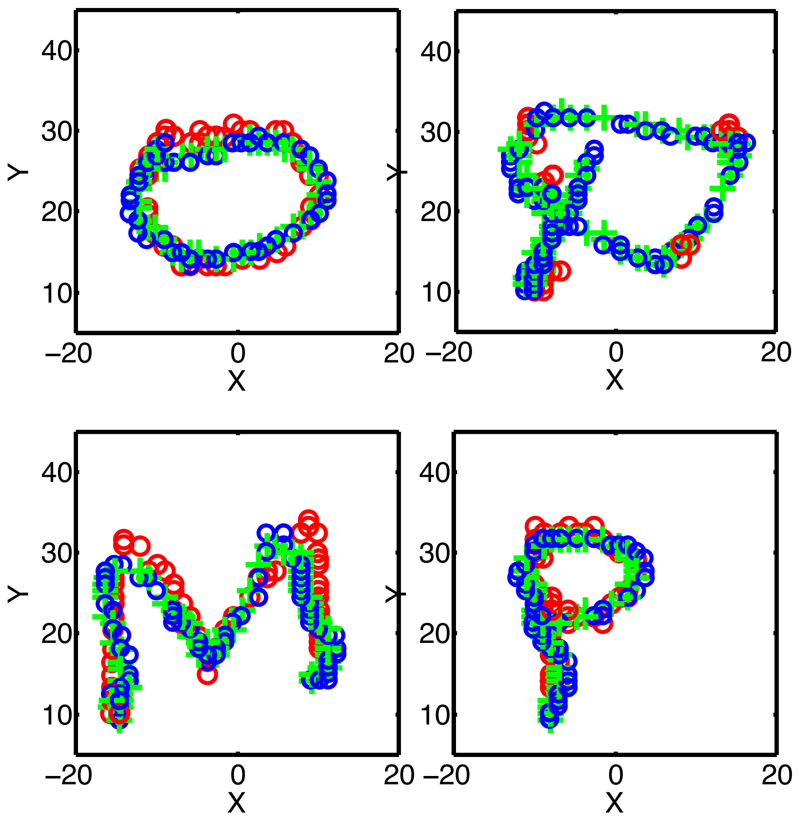
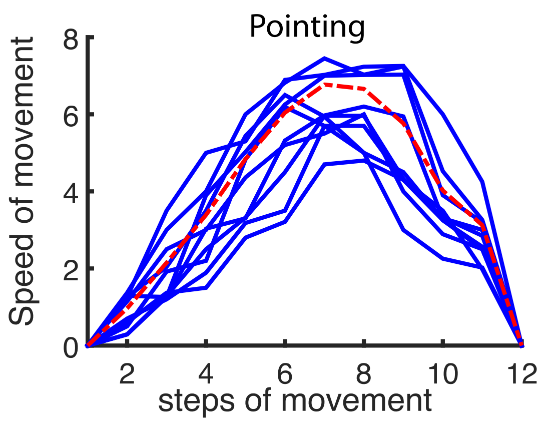


Figure 3.13: Following moving targets. Red circles: target locations; green circles: the observer's predicted location of the arm; blue circles: actual arm locations. In the top left and bottom two trajectories, the target was moved through the workspace and closely tracked by the arm. In the top right, the target experienced sudden, large changes in position.

DNFs	τ	β	h	g_{inh}	a_{exc}	σ_{exc}	a_{inh}	σ_{inh}
Target	2	12	-.5	.2	25	3	1	5
Arm	2	12	-.5	.2	25	3	1	5
HCT	35	2	-.3	.2	25	3	20	10
Velocity	30	12	-1	0.2	10	5	0	1
Path integration	4	0	0	0.25	3	$\frac{\pi}{6}$	0	0

Table 3.2: Parameters of each DNF map



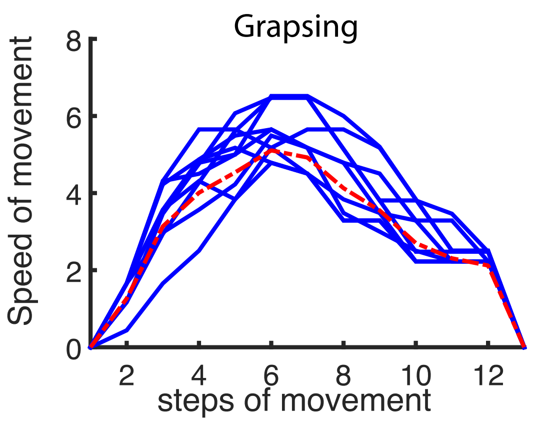


Figure 3.14: Velocity profiles for grasping and pointing an object. Each solid blue line represents one movement with 11 time steps. The velocity indicated by this curve is then applied to the Lego motors. The dotted red line represents an average of these 10 velocity profiles.

Params	Description	Value
c_{zero}	strength of zero activation in velocity map	250
σ_{zero}	width of zero activation in velocity map	50
c_{v}	strength of v_{inp} in velocity map	3
σ_{v}	width of v_{inp} in velocity map	30
M	coefficient to scale strength of V_{zero} for grasping	1
N	coefficient to scale strength of $V_{inp}$for grasping	.36
M	coefficient to scale strength of V_{zero} for pointing	1
N	coefficient to scale strength of $V_{inp}$for pointing	.7
η	Window size	.3

Table 3.3: Other parameters used in the program

Chapter 4

Arbitrated Predictive Actor-Critic (APAC) [2]

4.1 Motivation

The intention of the APAC is to study the interaction of habitual and planning systems in form of an arbitrator and ultimately to understanding human behaviour. Not only have human decision-making studies supported the notion that both habitual and planning controls are used during decision-making [71, 77], there is evidence that arbitration may be a dynamic process involving specific brain regions [78]. Our APAC model results suggest that such an arbitration strategy, wherein the planning paradigm is used until the habitual system's predictions become reliable can result in performance that is non-inferior to exclusive planning control in most cases. Thus, our APAC model supports (A) the importance and value of implementing predominantly planning control early in behavioural learning and (B) the diminishing importance of planning control with greater experience in a relatively static environment.

General structure of APAC introduced in the introduction chapter, however, in this section, we apply the APAC model to the motor control task of target reaching. We choose this task as it is a good example of a minimal control task while being complex enough to highlight the advantages and disadvantages of the two principle control architectures discussed in this chapter. Target reaching lives in a continuous state and action space with 6 degrees of freedom when considering a shoulder and elbow yaw, pitch and roll, although we simplify this here even more to a 2-dimensional system with only one angle for the elbow and one the shoulder. Learning the reaching task in this 2D environment is learning a non-linear mapping function that maps joint angles of the robot arm onto a location of the end-effector in the environment.

I applied APAC on a two joint robotic arm to learn the target reaching task. Here, we considered two types of reaching tasks:

- **One-step mapping:** The goal is learning the mapping function from desired angles of joints to the desired Cartesian coordinates of the target location. In this task, the robot arm is assumed to have no limitation in movements, therefore it can reach the target in one step.

- **Multi-step reaching:** The task is reaching the target while there is a limitation of movement assumed on the robotic arm. For example, each joint can move maximum for the amount of 30 degrees at each time step.

Materials and results reported related to one-step mapping is under review for publication in the Journal of Neural Networks, although it is uploaded as preprint on arXiv [2]. Some part of the chapter has also been published in the annual conference on Cognitive Computational Neuroscience [4]. Moreover, results for multi-step reaching has been accepted and under publication for the International Joint Conference of Neural Networks (IJCNN 2018).

4.2 One-step mapping with APAC

An example image of our simulated robot arm is shown in Figure 4.1 with the black line. The contour plot shows the distance to the target while the dotted green line shows an internal model of the robot arm early in learning.

The refined control architecture of our APAC model for the reaching tasks is shown in Figure 4.2. For this application, the state is now defined as the position of the elbow, the end-effector, and the target. The planning component is now made out of a combination of deep forward and inverse models, while the habitual system is implemented as a deep actor-critic model. An integrator is used to derive the training signals that are used for the feedback. In the following, we specify each subsystem in detail.

4.2.1 Habit learning control system

The habitual controller is implemented as a deep deterministic policy gradient model (DDPG) following the work of Lillicrap et al. [89]. Although DDPG structure explained in chapter 2, herein, we explain each component with details of the implementation for the target reaching.

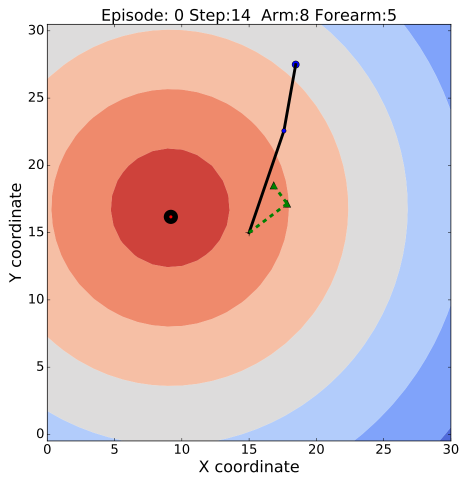


Figure 4.1: The robotic arm set up in the environment. The solid black line indicates the robot arm. Blue and red circles display the end-effector and the target locations respectively. coloured contours illustrates reward function. Black circle shows the target zone. The dashed green line represents the internal model of the arm at very early stages of learning.

DDPG takes advantage of the experience memory replay, batch normalization, target networks and Ornstein-Uhlenbeck process, which are all described in chapter 2.

The arm position is given by the vector X with two vector components, the position of the end-effector X^{end} and the position of the elbow X^{elbow}. The arm position together with the target location X_{target} defines the current state $s_t = [X^{end}, X^{elbow}, X_{target}]$ of the agent.

In DDPG, t he critic $\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}})$ is implemented as a deep neural network, where s_t is the current state at time t, a_t is the action taken at time t, and $\theta^{\mathcal{Q}}$ are the parameters of the critic network.

The critic is learned through temporal difference (TD) learning [147, 82].

$$\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}}) \leftarrow \mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}}) + l_1 \delta, \tag{4.1}$$

$$\delta = \underbrace{r_t + \gamma \max_{a'} \mathcal{Q}'(s_{t+1}, a'; \theta^{\mathcal{Q}})}_{\text{estimated reward}} - \underbrace{\mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}})}_{\text{actual reward}}, \tag{4.2}$$

where l_1 is the learning rate of the critic network, r_t is the actual immediate reward received from the environment at time t, γ is a discount factor, and $\mathcal{Q}'$ represents the

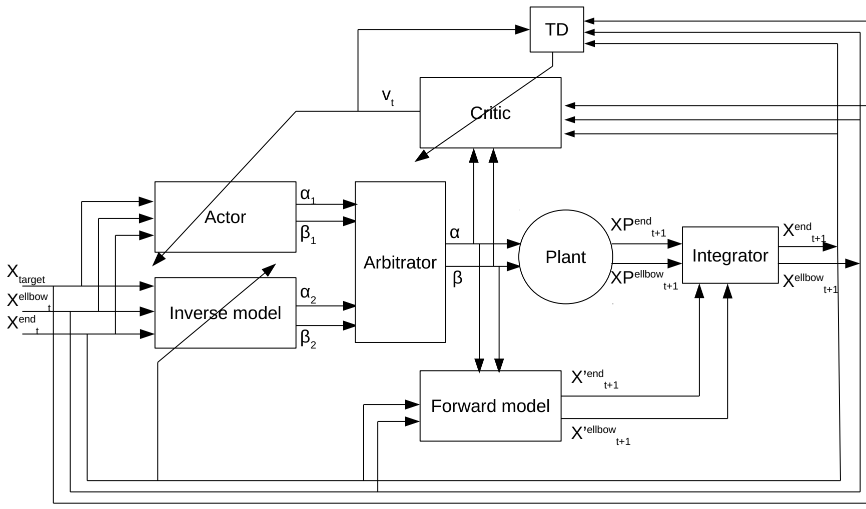


Figure 4.2: Arbitrated Predictive Actor Critic: The actor receives the current state defined as the current location of the robot arm and the target location, and provides an action that is the change in angles for shoulder and elbow. The inverse model takes the current state and predicts another action. The output of the inverse model and the actor goes to the arbitrator. Then the arbitrator selects one of these actions. The output of the arbitrator along with the current state is the input to the critic from which the critic predicts a reward. The forward model receives the selected action and the current position of the agent and predicts the future state of the agent. The agent takes the selected actions and transfers to the next state. The predicted future state from the forward model is integrated with the estimation of the actual state of the plant after taking the action and obtains a new current state for the system.

estimation of the value of a state-action pair.

The computation in Equation 4.2 is done in the TD component. To train the critic using the TD rule, the error needs to be back propagated through the critic. The error between estimated value and the actual value is used to compute the loss function of the critic (Eq. 4.3),

$$L^{\mathcal{Q}} = 1/N \sum (\delta)^2. \tag{4.3}$$

The actor (π) receives the current state (s_t) and predicts future actions to be taken (a_t).

$$\pi(s_t; \theta^{\pi}) = a_t, \tag{4.4}$$

where $a_t = [\alpha_1, \beta_1]$, where α_1 and β_1 are motor commands sent to the shoulder and elbow, respectively. The actor is implemented as a deep network where θ^{π} indicates the parameter of the actor network and is trained using the deterministic policy gradient method [88].

The changes of the weights of the actor corresponded to the changes in expected reward with respect to the actor's parameters,

$$\theta_t^{\pi} \leftarrow \theta_t^{\pi} + l_2 \frac{\partial \mathcal{Q}(s_t, a_t; \theta^{\mathcal{Q}})}{\partial \pi(s_t; \theta_t^{\pi})} \frac{\partial \pi(s_t; \theta_t^{\pi})}{\partial \theta_t^{\pi}}, \tag{4.5}$$

where l_2 here is the learning rate of the actor. The plant, which is the simulated arm in our example, takes the action and transitions to its new position $[Xp_{t+1}^{end}, Xp_{t+1}^{elbow}]$, which forms the new state s_{t+1} when combined with the target location. Like DDPG we apply noise to the environment using an Ornstein-Uhlenbeck process [160] that results in new samples.

4.2.2 Internal models for planning

For the planning controller, we need to learn the transition function of the plant to build the model of the environment. Here we use supervised learning to learn the internal representation of the agent. More specifically, we used a supervised learning controller that uses past experiences to generalize an inverse model of the arm and a forward model of the arm. The training examples used in our implementation are obtained from the same experience replay memory that is used for the habitual controller.

A combination of a forward and an inverse model is used for planning the next actions. The forward model f_F is a neural network that receives the current position of the arm $[X_t^{end}, X_t^{elbow}]$ and the action a_t and predicts the future position of the arm $[X'^{end}_{t+1}, X'^{elbow}_{t+1}]$. We can train the network from the discrepancy between the predicted future position $[XP^{end}_{t+1}, XP^{ellbow}_{t+1}]$ and the actual position from visual feedback. For the training we use the loss function

$$L^{f_F} = \frac{1}{N}\sum_t([X'^{end}_{t+1}, X'^{ellbow}_{t+1}] - [XP^{end}_{t+1}, XP^{ellbow}_{t+1}])^2, \tag{4.6}$$

where N is the number of selected samples in a batch of experiences stored in the replay memory. The size of the batch to train the forward model and the inverse model is the same as the one used for the actor and the critic.

An inverse model is another deep network, $f_I(s_t; \theta^{f_I})$. The aim of the inverse model is to provide a proper action to reach the target by minimizing the error between predicted action (a'_t) with the actual action taken (a_t) that transfers the agent from the current position to its next position. This network is then trained on the loss function:

$$L^{f_I} = \frac{1}{N}\sum_t(a_t - a'_t)^2. \tag{4.7}$$

The aim of having the forward model is learning to predict future positions of the agent by taking specific actions. Such a model enables the agent to perform the task even with occluded vision. When the inverse model has been trained well, it can be used to produce a suitable action to transfer the agent from its current state to the target location by replacing X_{target} with XP_t^{end}. Hence, the inverse model can be trained with the input $[X^{end}_{t-1}, X^{elbow}_{t-1}, XP^{end}_t]$ and predicting the proper actions on $[X^{end}_{t-1}, X^{elbow}_{t-1}, X_{target}]$. Note that $[X^{end}_{t-1}, X^{elbow}_{t-1}]$ are part of states s_t in the replay memory while XP^{end}_t is taken from s_{t+1} in the replay memory.

Another component of the overall system is "the integrator" module. In general, the integrator could be a Bayes filter such as a Kalman filter which estimates the best-estimated position from the available information that combines an internal model prediction with external sensory feedback. Since we use a reliable visual feedback in our case study we simplify this to an integrator that passes the actual location of the plant in case visual information is available. With occluded vision, the prediction of the forward model is used as the estimated actual position of the agent. In our previous work [1], we showed how to implement a Kalman Filter with Dynamic Neural Fields [106]. The integrator is the explicit critic in this example, which provides the state prediction error for the forward model (see Figure 1.4).

A training session of the system includes an infant phase that uses "motor babbling" [172, 173, 174, 175, 176]. During the babbling phase, the plant produces random movements with random actions to produce actual samples to be stored in the experience memory. In the babbling phase, the actual position of the arm after taking an action is considered the target location. Therefore, all samples, in this case, reach the terminal state and will gain the maximum reward value. The babbling phase is used to provide valid examples to train both control systems.

4.2.3 Arbitration between habitual and planning controllers

A novel component of APAC is an arbitrator. The arbitrator receives action predictions from the deliberative planning module (the inverse model), and the habitual action selection module (the actor), and makes the final decision of which action to use. This selected action is transferred to the actuators of the plant to bring the agent into its new position resulting in a new state when combined with the target location. The arbitrator's decision is also fed into the forward model and the critic for training purposes. As in DDPG, noise from an Ornstein-Uhlenbeck process is added to both proposal actions provided by the inverse model and the actor.

In our implementation of the APAC, we considered discrete action steps so that both controllers (habitual and planning) create actions at each step. However, it is known that the habitual controller is faster than deliberative planning. Therefore to imply the time constraint we set the arbitrator to give priority to the habitual controller. There are two conditions in which the arbitrator is set to select the action from the actor even if the RPE value is not under the threshold. One condition is the first episode of each experiment. The other condition is first two steps of every episode. However, from the third step on, the actor's prediction is taken if the habitual controller is reliable, meaning that the reward prediction error for the last experience is smaller than a threshold. We use $\delta < 1$ in the following experiments. Otherwise, the action from the inverse model is selected. It is, of course, possible to implement a more dynamic realization of such an arbitrator. For example, the threshold could itself be modulated according to the tasks and in this way produces a more rich speed-accuracy trade-off [177]. However, the simple implementation discussed in this work captures the minimal assumptions as outlined above and is sufficient for the following simulations.

4.2.4 Experimental Conditions and Environment

To test the APAC model on a simulated robot arm with a target reaching task (Figure 4.1), we simulated a two-joint robotic arm whose range of motion at each joint was constrained to 180 degrees. The arm motion was limited to a 2D plane of width 30 and height 30, upon which the arm "shoulder" was fixed in the centre (15,15). The initial length segment from the shoulder to the elbow was set to $l_1 = 5$, and the initial length of the lower segment ("hand" to "elbow") is $l_2 = 8$.

All experiments described herein had an episodic trial structure. At the beginning, the arms position was set to zero-degree angle at the shoulder and 180-degree angle at the elbow. Time was discretized in the simulations, and the learning agent was given only 30 action-steps per trial to achieve the designated goal. We define a "target zone" as a circle centred at the target location with a radius $r_{\text{target}} = 0.5$. The target is defined as "reached" once the robot arm is inside the target zone. If the goal was not reached within 30 time-steps, the trial was aborted and a new trial was started.

Importantly, the reward function is defined as the negative Euclidean distance between the end-effector and the centre of the target area. For this example task, the same information as is given to the supervised learner. This was deliberatively done so that the different systems are compared on the same feedback situation. It is possible to learn this task from much simpler feedback such as some reward if the target area is reached versus no reward otherwise, although this would then also need more time to train the habitual system. This point of our study here is rather the direct comparison of decision components based on a value lookup versus learning internal models.

Within this environment, we define several conditions that test a variety of target-reaching tasks studied here. These conditions include the target position (**static target** vs. **changing target** at each episode), kinematics (arm dimensions as **static kinematics** vs. **changing kinematics**), and vision (**occluded vision** vs. **perfect vision**). We tested all combinations of these factors.

Each experiment consisted of 1000 episodes of maximal 30 action steps each. In the static target condition, the target is initialized randomly and stays unchanged for all 1000 episodes. For the changing target condition, the target is located at a random location at the beginning of every episode. As discussed above, each episode was terminated when either (A) the target was reached, or (B) 30 time steps had elapsed. Targets were only placed within a reachable distance for the arm. The arm dimensions were kept fixed in the case of static kinematics; however, the length of both the upper and lower arm segments were increased by 0.001 at each time step for the changing kinematics condition. These changes were only started after the 100th episode of target reaching to provide some time for basic training. As already mentioned, an environmental noise was included in all experiments. In the occluded vision condition, the location of the arm and the target was unavailable for the agent during the movement. This task is also known as memory-guided target reaching [20, 19]. We repeated all static/changing kinematics and static/changing target conditions with our proposed models for the reaching-target task in the occluded vision condition. To examined the generalization of the models under each condition, we trained each model when targets are located only in a specific area that represents 2/3 of whole reaching area and tested with targets located in the other part of the environment, which is the rest 1/3 of the reaching area.

To move the arm, a motion function is used that is described in Algorithm 3. Algorithm APAC is also shown in Algorithm 4. This algorithm shows how APAC combines the habitual and planning through arbitration.

It should be again noted that APAC and our proposed predictive actor-critic framework is the extension of DDPG [89]. While preserving almost all parameters and features of the base DDPG model, APAC has added additional networks, which include the forward and inverse models.

The actor has two fully connected layers with *relu* activation function. These

Algorithm 3 Motion function: function that calculates location of the simulated arm with respect to the applied angles to shoulder (α) and elbow (β). l_1 and l_2 are current length of forearm and arm while O indicates the origin of the plane, where the arm is attached.

```
set up an input vector including $(\alpha, \beta, l_1, l_2, O)$
$\alpha_1 = \alpha \times \frac{\pi}{180}$
$\beta_1 = \beta \times \frac{\pi}{180}$
$elbow_x = \cos(\alpha_1) \times l_1$
$elbow_y = \sin(\alpha_1) \times l_1$
$end_x = elbow_x + \cos(\beta_1 + \alpha_1) \times l_2$
$end_y = elbow_y + \sin(\beta_1 + \alpha_1) \times l_2$
return$([end_x + O[0], end_y + O[1]], [elbow_x + O[0], elbow_y + O[1]])$
```

layers are connected to a fully connected output layer with the size of action dimension (here only two neurons). Each of neurons produces a continuous value for either the shoulder or the elbow. Since we need to learn to move the shoulder and elbow in both directions, we selected output activation function to be *tanh* which is multiplied by 180 degrees to obtain -180 to 180 degrees of movements. Note that each joint only moves between 0 to 180 degrees but it can move in two different directions. Actor learning rate is equal to 0.0001.

The critic has two hidden layers too. The first layer includes 400 fully connected neurons. In the second layer, the output of the actor (300 fully connected neurons) is combined with the output of the first layer (300 fully connected neurons) from the critic to build a fully connected layer with 600 neurons. The output of the critic is only one node since it will learn the value of taken action at each state. All activation functions selected to be *relu*, since we wanted to obtain negative and positives values in a range of [-1, 1], however, in some cases we also wanted to scale this to different ranges. For example, to learn the actions between [-30, 30], the output layer of the inverse model scaled by multiplying the relu output by 30. Learning rate is 0.001. Gamma value to update the TD rule is set to 0.99.

The forward model has three layers of 400, 300, and 4 fully connected neurons. Output layer of this network has 4 neurons to predict the future location of the robot arm (end-effector location and elbow location). All activation functions are *sigmoid*. The output of the network is multiplied by the dimension of the environment which is

Algorithm 4 Arbitrator Predictive Actor Critic

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\pi(s|\theta^\pi)$ with weights $\theta^Q$ and $\theta^\pi$
Initialize target network $Q'$ and $\pi'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^\pi$
Randomly initialize forward learner network $fl(s,a|\theta^{f_F})$ and inverse learner network $f_I(s|\theta^{f_I})$ with weights $\theta^{f_L}$ and $\theta^{f_I}$
Initialize target network $Q'$ and $\pi'$ with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\pi'} \leftarrow \theta^\pi$
Initialize replay buffer $R$
for episode=1,M do
  Initialize a random process $N$ for action exploration
  Receive initial observation state $s_1 = [X_t^{end}, X_t^{ellbow}, X^{target}]$
  for t=1,T do
    if episode<100 then
      Select action $a_t = f_I(s_t|\theta^{f_I}) + N_t$ by the inverse learner
    else
      Compute RPE
      if RPE<1 then
        Select action $a_t = \pi([X_t^{end}, X_t^{ellbow}, X^{target}]|\theta^\pi) + N_t$ by the actor
      else
        Select action $a_t = f_I([X_t^{end}, X_t^{ellbow}, X^{target}]|\theta^{f_I}) + N_t$ by the inverse model
      end if
    end if
    Feed $a_t$ and current location of the arm to the forward model and observe predicted location of the arm $[X'^{end}_{t+1}, X'^{ellbow}_{t+1}]$
    Execute action $a_t$ and observe reward $r_t$ and observe new location of the arm $[XP^{end}_{t+1}, XP^{ellbow}_{t+1}]$
    Integrate predicted location from forward model and real location and build $s_{t+1}$ along with target location
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$
    Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $R$
    Set $y_i = r_i + \gamma Q'(S_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{Q'})$
    Update forward model by back propagating the error between predicted location and real location of the arm
    Update inverse model by back propagating the error between predicted action and real taken action by the arm
    Update critic by minimizing the loss: $L = 1/N \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor using policy gradient method:
    $\nabla\theta^\pi J \approx 1/N \sum_i [\nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\pi(s_i)} \nabla_{\theta^\pi} \pi(s|\theta^\pi)|_{s=s_i}]$
    Update the inverse model by minimizing difference between action predicted by the inverse model to move from $X_t$ to $X_{t+1}$ with the actual action that transfered the plant from $X_t$ to $X_{t+1}$
    Update the target networks:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\pi'} \leftarrow \tau\theta^\pi + (1-\tau)\theta^{\pi'}$
  end for
end for
```

30 centimetres to produce location of the end-effector and elbow in a Cartesian coordinate. Learning rate is set to 0.1, and the optimization function is "Adamoptimizer" like all other networks.

The inverse model, similar to the actor, has three layers of 400, 300, and 2 fully connected neurons. Output layer of this network has 2 neurons to predict proper angles for elbow and shoulder. Activation functions for the first and the second layers are *relu*, and *tanh* is used at the output layer. The output of the network is multiplied by 180 degrees to obtain -180 to 180 degrees of movements similar to the actor.

An experience replay memory of size 1000 samples is used. Each time we train networks using a mini-batch of size 500. The smooth update parameter to update target networks is 0.001, and Adamoptimizer is used as an optimization function for actor and critic networks. All the networks use 'L2' regularization with a weight decay by the amount of 0.001.

List of all parameters and variables of all three models (DDPG, SPAC and APAC) are listed here in tables 4.1 and 4.2.

Table 4.1: Common parameters for all networks and models

Parameter name	**Value**
Maximum number of trials	1000
Maximum number of steps in each trial	30
Discount factor (γ)	0.99
Soft target network update (τ)	0.001
Random seed for randomizing	None
Buffer size for experience replay memory	1000
Minibatch size	500
Size of a 2D planar space	30×30 cm
Origin of the arm fixed on 2D planar space	[15,15]
Initial length of forearm	8 cm
Initial length of arm	5 cm
Target zone	area with radius 0.5cm around target

4.3 Multi-step reaching with APAC

In this section, we report the ability of APAC model to control a simulated two-joint robotic arm during multiple reaching tasks with movement limitations that require

Table 4.2: Network parameters and variables

Parameters	**APAC**
Input dimension to actor	6
Neurons in the first layer of actor	400
Activation func. in first layer of actor	relu
Neurons in the second layer of actor	300
Activation func. in second layer of actor	relu
Neurons in the output layer of actor	2
Scale of output of actor	180
Activation func. in output layer of actor	tanh
Actor optimization function	Adam optimizer
Actor learning rate	0.0001
Input dimension to critic	8
Neurons in the first layer of critic	400
Activation func. in first layer of critic	relu
Neurons in the second layer of critic	600
Activation func. in second layer of critic	relu
Neurons in the output layer of critic	1
Critic optimization function	Adam optimizer
Critic learning rate	0.001
Input dimension to forward model	6
Neurons in the first layer of forward model	400
Activation func. in first layer of forward model	sigmoid
Neurons in the second layer of forward model	300
Activation func. in second layer of forward model	sigmoid
Neurons in the output layer of forward model	4
Scale of output of forward model	30
Activation func. in output layer of forward model	sigmoid
Forward model optimization function	Adam optimizer
Forward model learning rate	0.01
Input dimension to inverse model	6
Neurons in the first layer of inverse model	400
Activation func. in first layer of inverse model	relu
Neurons in the second layer of inverse model	300
Activation func. in second layer of inverse model	relu
Neurons in the output layer of inverse model	2
Scale of output of inverse model	180
Activation func. in output layer of inverse model	tanh
Inverse model optimization function	Adam optimizer
Inverse model learning rate	0.01

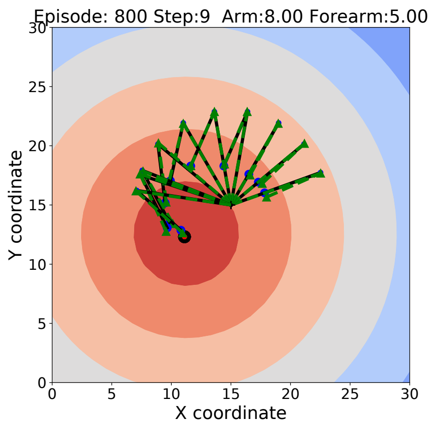


Figure 4.3: Simulated Robotic Arm: The task space is shown here as a 30cm by 30cm grid. The robotic arm is represented as the black line anchored at (15, 15). The green dashed line represents the internal representation of the arm. The red dot (with black outline) denotes the target. Contours depict the reward surface. Note the image was taken late in training, when the forward model's prediction of arm position does accurately represent true arm position.

multiple steps to solve the task.

In one-step mapping, the arm learns the mapping function from angles to Cartesian coordinates. In multi-step reaching instead, the agent learns path planning and trajectory of reaching.

Indeed, a critical test for reinforcement learning methods is if they can be generalized to sequences of actions that are necessary to learn a task. Here, we examined the ability of APAC to generate such sequences due to movement limitations of the arm. In this study there is no particular path to follow, however, the agent itself should learn the best trajectory to reach the target. To do so, the agent is forced to move only for limited angles, therefore it may need to take more than one step to reach the target at the desired location. For the multi-step reaching, we defined a limitation on movements and actions in a way that no movements greater than $\phi < 180$ degrees in either direction are allowed. Basically, the output of the actor and the inverse model is set to be in the range of $[-\phi, \phi]$. Therefore, the agent needs to take more than one step to reach the desired location. At each step, the predictions for the future state and the future reward need to mimic the applied limitation on movements. The environment is shown in Figure 4.3.

Basically, almost all details of the APAC model including parameter values, training conditions, and testing conditions remains the same for both one-step mapping and multi-step reaching. Only action range differs from the multi-step reaching which is controlled by changing the output range for the inverse model and the actor.

Similar to one-step mapping, the arbitration in APAC starts from the 2nd episode, which means that all actions in the first episode come from the actor. Another assumption is that the arbitrator always selects the actions for the first two steps at each episode from the actor. The reason is that we considered the planning takes longer time than the habit, therefore at the time that the action is predicted by the actor, the action from the planning has not predicted yet. Hence, the arbitrator only have access to actor's prediction in first two steps of each episode.

Beside above-mentioned testing conditions, we also tested two other aspects under multi-step reaching. One is RPE calculation with the actual taken action or action's prediction. As in one-step mapping, RPE is calculated in arbitrator component. The RPE is usually calculated as the difference between received and predicted reward in the previous step,

$$RPE = r_{t-1} + \gamma Q(s_t, \pi(s_t)) - Q(s_{t-1}, a_{t-1}), \tag{4.8}$$

where r_{t-1} is current received reward, $\gamma Q(s_t, \pi(s_t))$ is discounted prediction reward for the next step, and $Q(s_{t-1}, a_{t-1})$ is the current critic value. For APAC we need to calculate the RPE for the last experience, therefore Equation 4.8 is written for the last step. Note that RPE needs to be calculated before training the networks with the new samples.

In a stand-alone actor-critic structure, a_{t-1} would be the same action that the critic has performed in last step. However, in APAC that has a combined structure, the actual action taken may or may not be the same as the actor-critic decided. In this study we therefore study whether using RPE with a^A_{t-1} or a^π_{t-1} would cause any difference or not. This we distinguish here the RPE of the real action taken

$$RPE^A = r_{t-1} + \gamma Q(s_t, \pi(s_t)) - Q(s_{t-1}, a^A_{t-1}) \tag{4.9}$$

from the RPE of the actor prediction

$$RPE^{\pi} = r_{t-1} + \gamma Q(s_t, \pi(s_t)) - Q(s_{t-1}, a^{\pi}_{t-1}). \tag{4.10}$$

The other aspect that we studied in multi-step reaching is the velocity profile of arm movement, which is inapplicable with one-step mapping task. Results of these studies are reported in the next section.

4.4 Results

4.4.1 One-step mapping results

We considered three versions of APAC that represent a) exclusive habits, b) exclusive deliberate planning, and c) arbitration between habit and planning. Exclusive habit is when the arbitrator is set to always pick the action from the habitual system. In this case, the APAC behaves exactly like DDPG. If the arbitrator always selects the action from the inverse model for each step, then the APAC becomes an exclusive deliberate planning controller which we call supervised predictive actor-critic (SPAC) [4]. The third model is when the APAC is able to arbitrate between the actions provided by the inverse model and the actor.

For each condition, we trained 50 independent instances of each model for a total of 1000 episodes. At the end of the 1000th episode, all network parameters were frozen and no more training was applied. Subsequently, each of the independent model instances performed 100 target reaching episodes under the respective training conditions. In the case of the occluded vision, sensory input (i.e. visual target position) was initially presented at time step 0, and subsequently rendered unavailable. Since DDPG has no internal model and requires visual input throughout the task, we only compare APAC with SPAC in the experiments with occluded vision.

All experiments were implemented and tested in Python (3.5) using the TensorFlow (1.3) package [178] on an NVIDIA GeForce GTX 960 graphical processing unit. Statistical analyses were done using Matplotlib 2.1.0.

Figure 4.4 illustrates the percentage of trials that reach the target within 30 action steps during training under different conditions for 1000 episodes each. The pure deliberate planning model SPAC reaches almost near perfect performance very fast

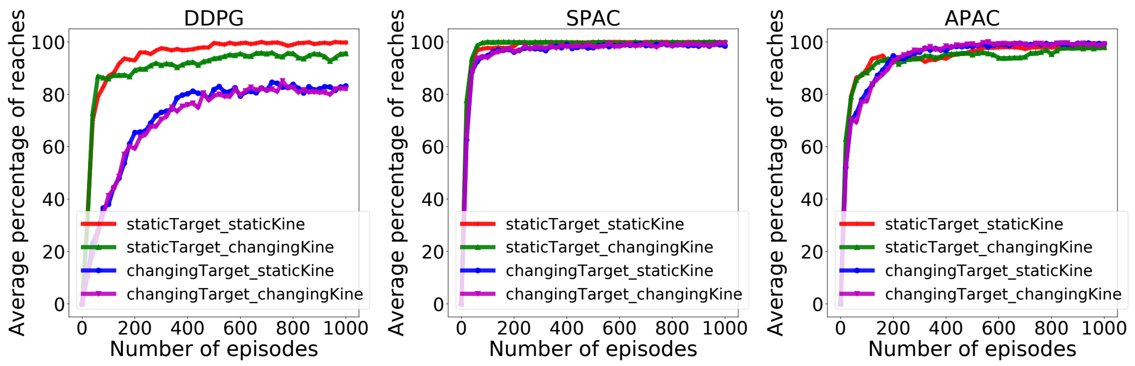


Figure 4.4: Comparison of the proportion of trials in which the target was successfully reached during training between DDPG, SPAC, and APAC. Plots highlight differences in each respective model's performance across the Target/Kinematic experimental combinations. APAC that arbitrates between planning and habits can resist to changes in the target position and kinematic changes, while DDPG is not flexible under either changing condition.

after 100 episodes. Furthermore, SPAC's performance is very robust under different conditions and neither changes in reward function nor in kinematics effects the performance of SPAC very much. DDPG learns to reach almost 100% of the targets only under static target/static kinematics condition. Figure 4.5 shows that performance of DDPG drops slightly under changing kinematics compared to static kinematics; however, its performance drops dramatically (about 20%) under changing target conditions. This is of course expected as habits become invalid solutions under changing environments. Our point here is that APAC can reach almost all of the targets both under static and changing targets as good as SPAC, although it tends to use more habits than planning after a few trials of learning(see Figure 4.8). The speed of learning in APAC is also very high and comparable to SPAC. In this sense, it combines the benefits of DDPG and SPAC.

The above curves give an example of the behaviour of the models during one learning trials. To study how these results generalize we tested the performance of all three models after learning over 50 different learning trials with random initial conditions for the networks. For the static target location we tested on the target location, that was randomly chosen for each learning trial. However, with the changing target location, we decided to cover the possible target locations more systematically and chose a set target point on a regular grid in angle space. Figure 4.6 displays average

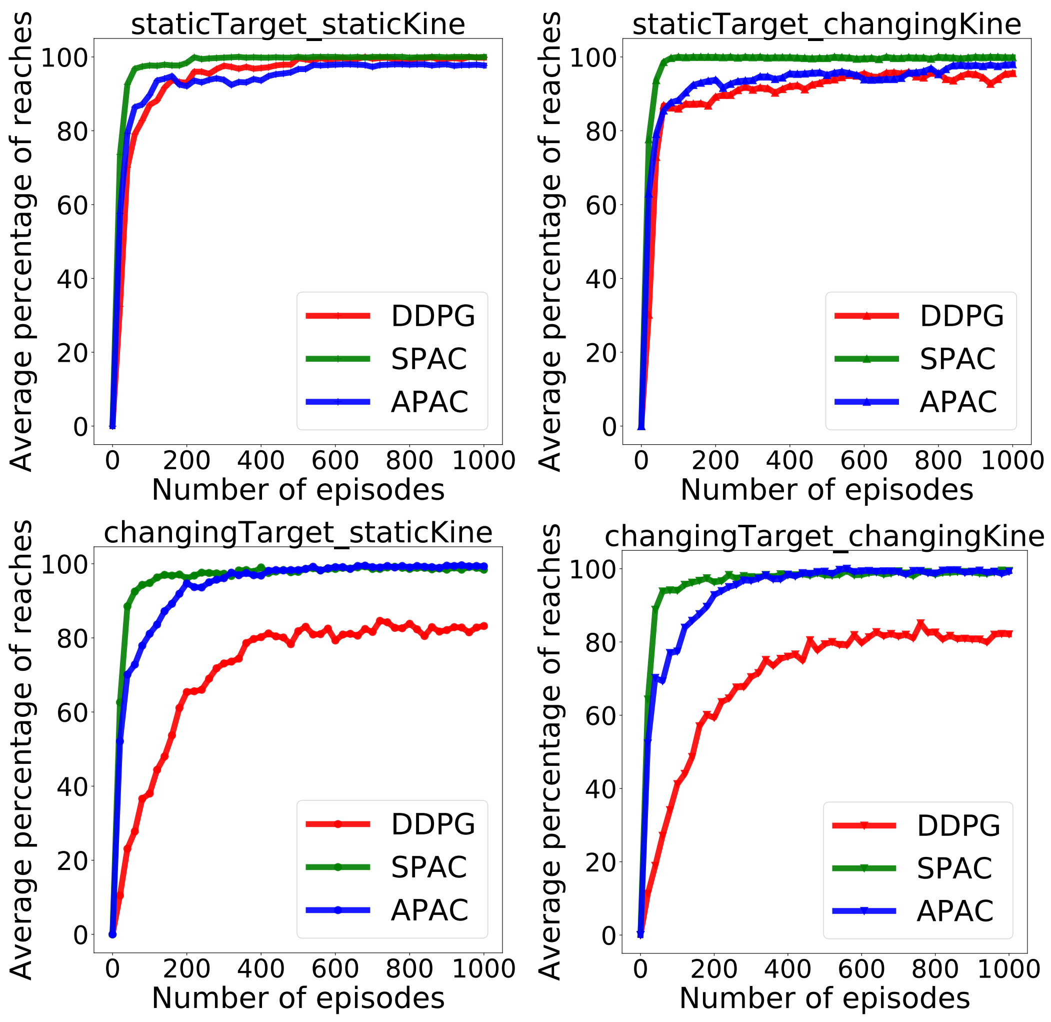


Figure 4.5: Plots highlight differences between models within each experimental combination of target position and kinematics. Performance of DDPG (pure habits) drops when target location is changing, while this has no effect on SPAC (exclusive planning) and APAC (arbitrated).

success rate over the 50 learning trials to reach these targets. All three models under the static target/static kinematics condition reach 100% of the targets. DDPG and APAC have slightly less success under static target/changing kinematics, while SPAC stays flexible under this condition. The major difference between DDPG and APAC become clear under changing target conditions, where DDPG's performance drops dramatically, while APAC obtains very good performance. SPAC is still very flexible to reach targets under changing target conditions. Overall it is remarkable how close APAC stays to the overall performance of SPAC in a situation where deliberative

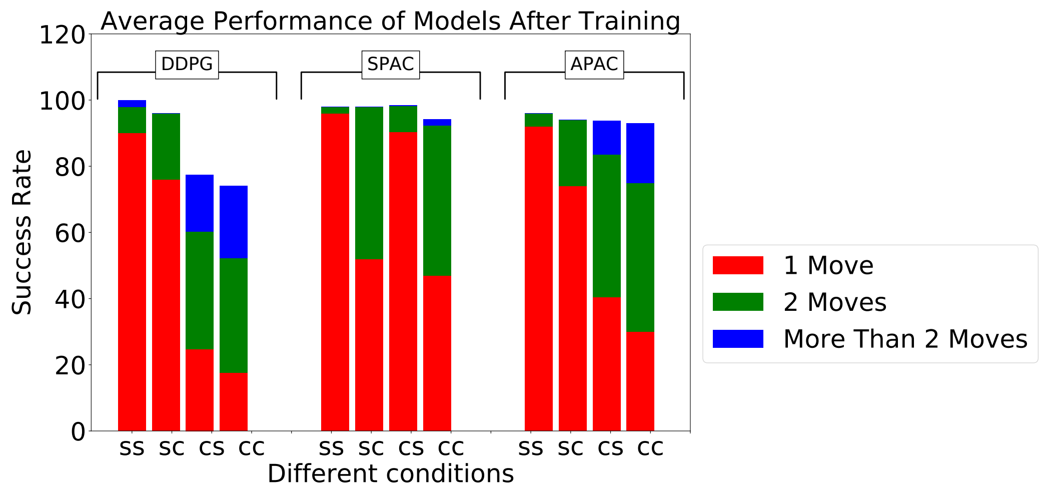


Figure 4.6: Success rate to reach targets during testing by three models under different conditions, where 'ss', 'sc', 'cs', and 'cc' stand for static target/static kinematics, static target/changing kinematics, changing target/static kinematics, and changing target/changing kinematics respectively. The plot also displays the average number of steps to reach 100 targets after training.

planning is the better choice, however it relies more on habitual controller.

The overall success rate does show the entire range of the solutions. We thus included the individual performances in terms of the average number of steps to reach the target. As can be seen, APAC needs to take sometimes more corrective steps to reach the target while an exclusive planning system can optimize the number of steps. This is interesting as this allows for different strategies in solving the task, that of relying somewhat on habitual control when the cost of the movement initiation might be small versus more deliberate planning when the number of action steps might matter. This can explain a form of speed-accuracy trade-off.

Figure 4.7 shows all 50 runs to reach 100 targets of a reaching test under changing target conditions, with (bottom row) and without (top row) changing kinematics, for all three models. The plots illustrate that DDPG has some difficulty reaching the locations at the edges of the possible target area due to non-linearity of the mapping between angles and locations. SPAC can learn the mapping function much better, and the quality of APAC is similar to SPAC. Interestingly, although APAC tends to use more habits than deliberate planning, this model can still reach many more targets than DDPG, almost as good as SPAC.

Figure 4.8 illustrates how APAC gradually shifts from a planning to a habitual

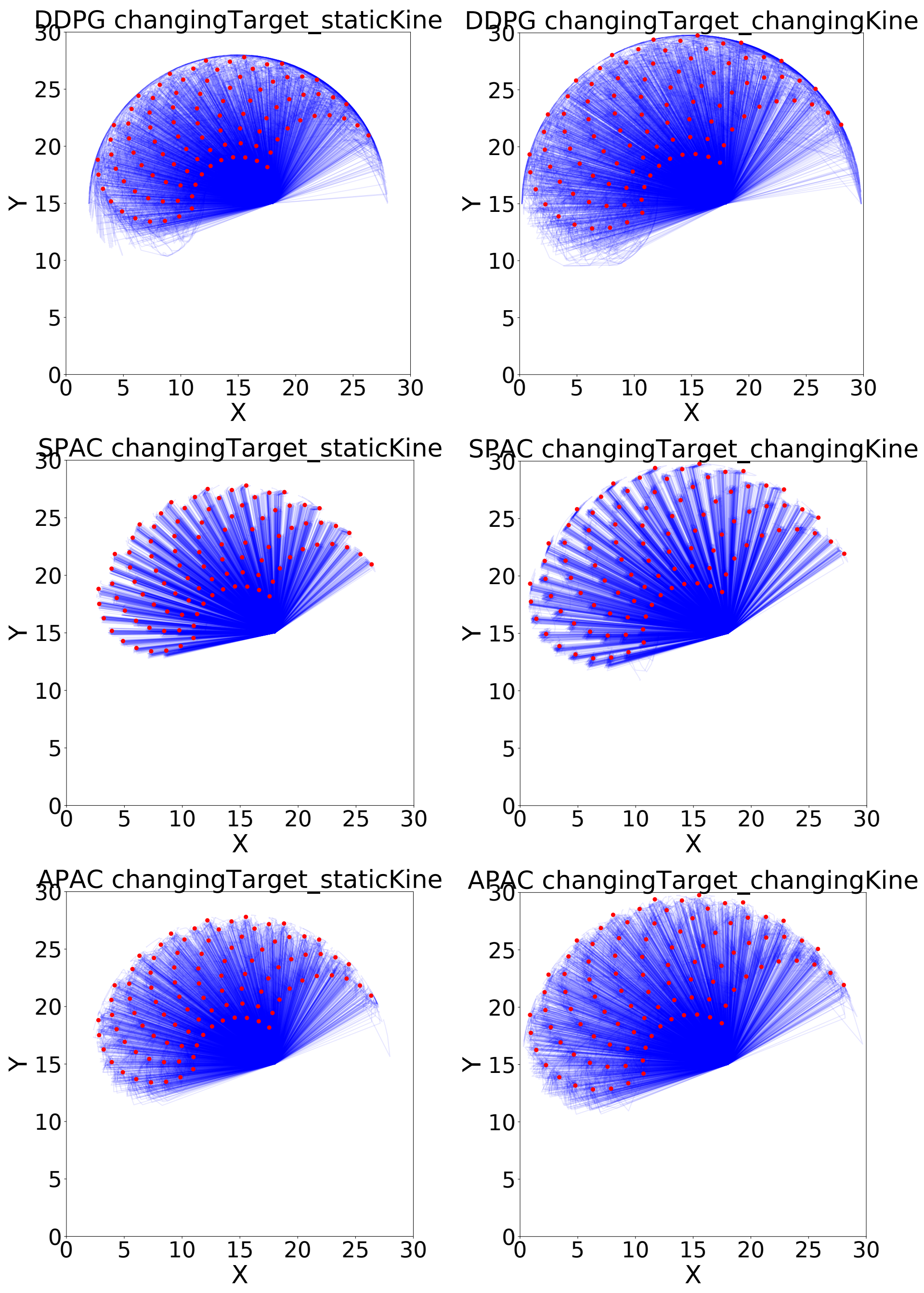


Figure 4.7: Reaching examples of three models under changing target conditions. Blue lines show position of the end-effector at each time step toward the target, while red dots are target locations during the testing phase.

control approach with increasing experience. After around 300 episode, more than 80% of APAC actions were taken from the habitual controller. Of course, SPAC uses planning control throughout the entirety of the task, so no commensurate figure was

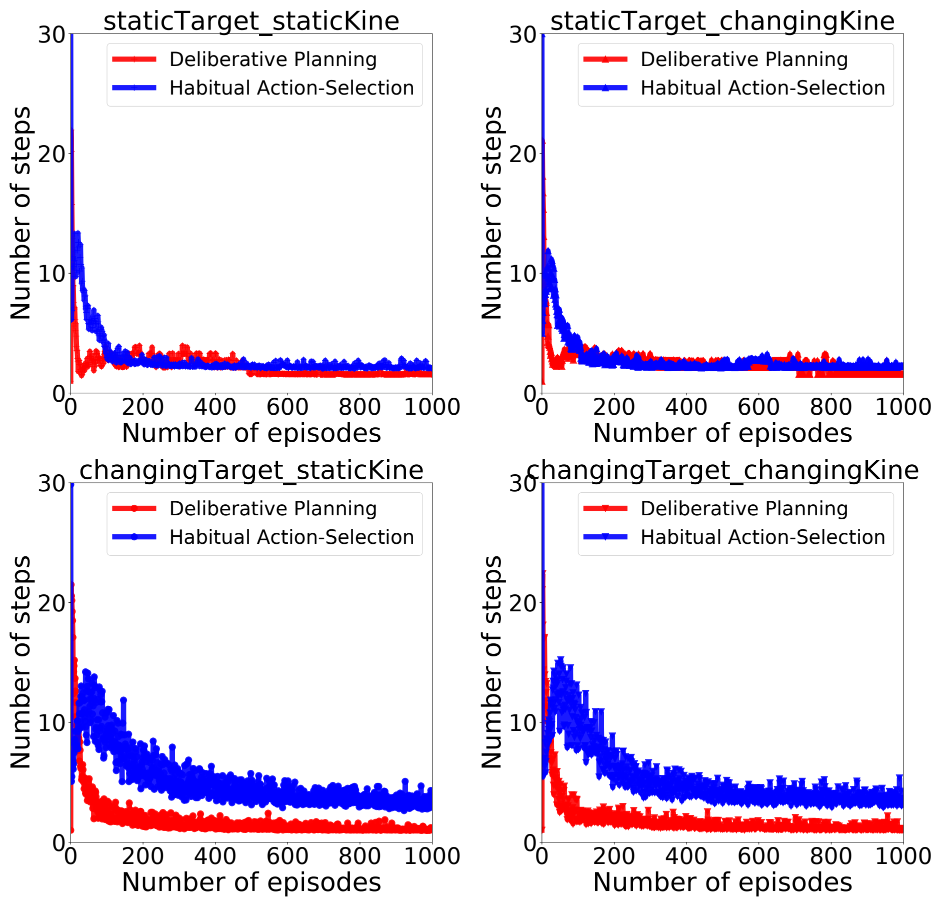


Figure 4.8: Arbitration between habitual and planning by the APAC: For each condition, the blue line indicates the average number of actions which are selected by the habitual controller (i.e. the actor), while the red line demonstrates the average number of actions selected by the planning controller (i.e. inverse model). Results illustrate that planning controller is used early in the training, while later agent tends to use the habits more.

generated for it.

Since a habitual system should be faster than deliberate planning, this figure also illustrates that APAC would be less time-consuming than the SPAC at the same task and under the same condition. To show average time consumption by each model under different conditions, we assumed that each action selected by the deliberative planning takes three times longer than an action selected by the habitual controller. The number here is arbitrary and only chosen to show the general effect. The left column of plots in Figure 4.9 shows the average number of action steps that each of the three models need to complete the task at each episode under different conditions.

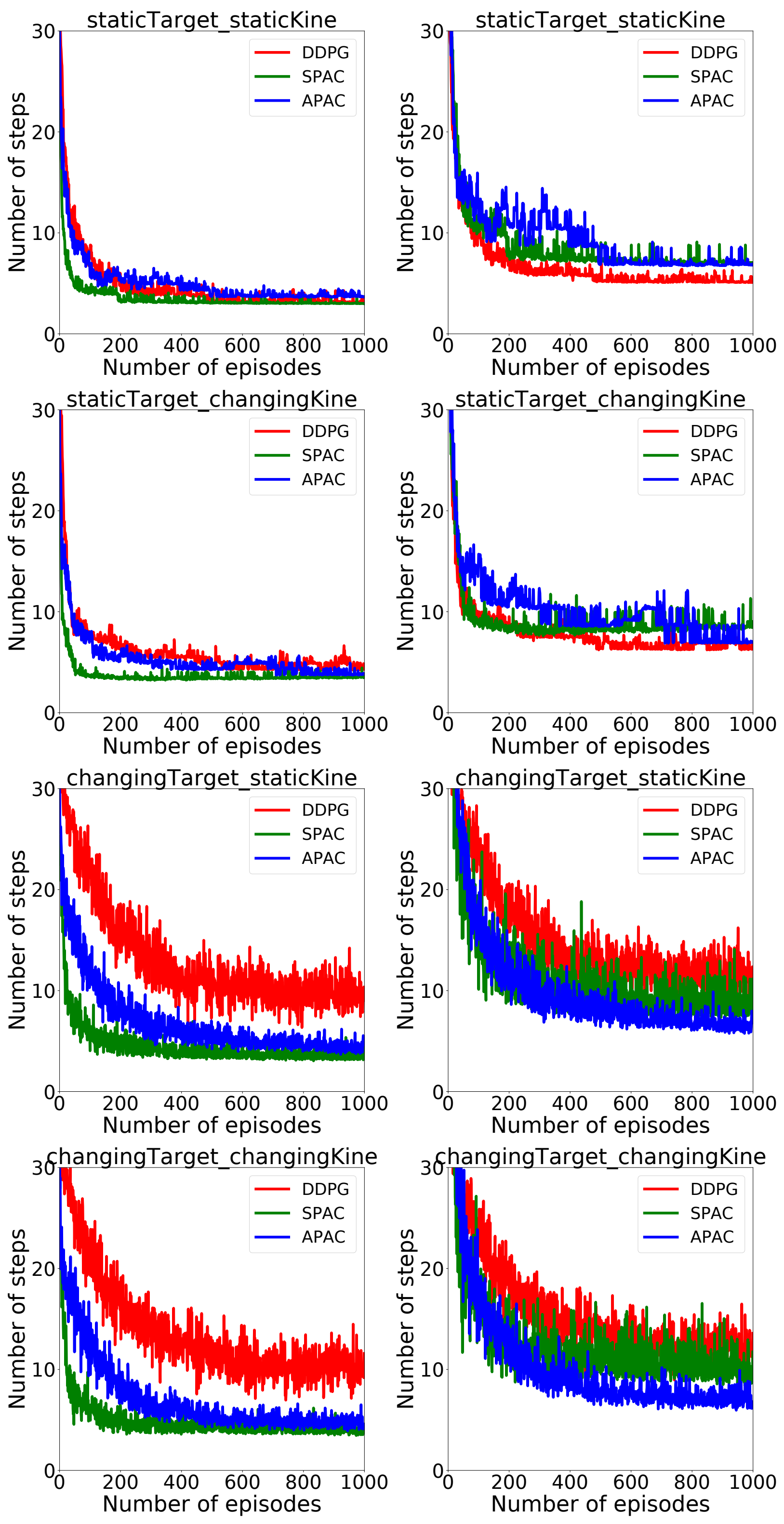


Figure 4.9: **Left column:** Average action steps to complete the reaching task by each model under different conditions. **Right column:** Average time steps to complete the reaching task at each episode under different conditions when a three-times higher cost for a planning control compared to the habitual control is taken into account.

These plots demonstrate that the number of steps to reach the target are almost the same under static target/static kinematics conditions. Under changing target conditions DDPG needs more steps to complete the task than APAC and SPAC. However, when including a higher cost for deliberative planning in the plots shown in the right column of the Figure 4.9, the picture for the average number of time that is needed to complete the task changes. In this case, DDPG needs shorter time under static target conditions. However, under changing target conditions, APAC completes the task of reaching targets faster. DDPG takes longer to finish the task as it needs more corrective actions.

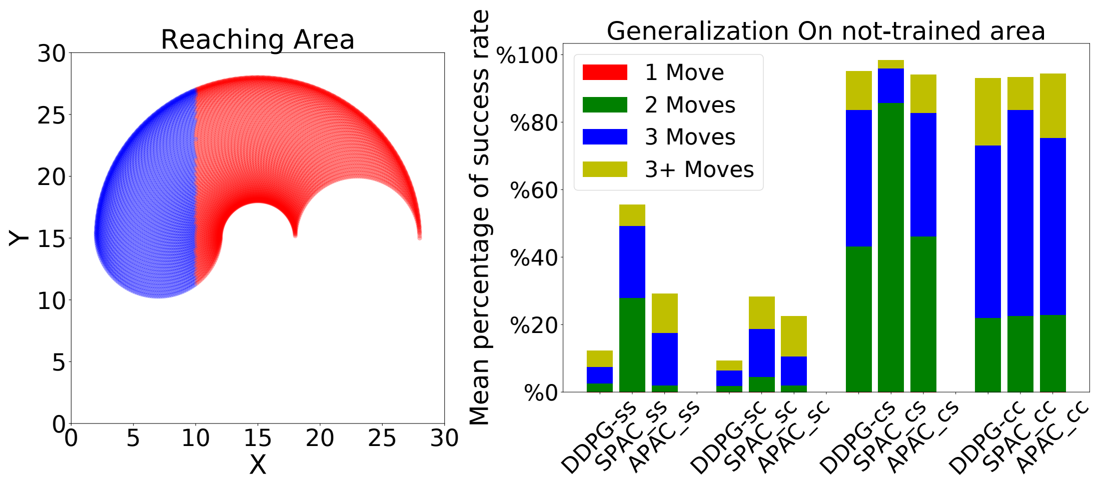


Figure 4.10: **Left:** The reaching area for arm under static kinematics conditions. The red area is used for training while the blue region is for testing. Around edges are more coloured since these points can be reached with many more sets of angles. **Right:** Performance of each model is shown under different conditions to reach about 40 targets located in the blue region. All models obtain a good generalization under changing target conditions.

We also tested the generalization of each model to form of generalization where a whole area of the target zone was not seen during training. This is a form of extrapolation compared to the interpolation trials in the previous generalization experiments. More specifically, we trained each model to reach the target located at a specific region in the environment and we tested each model to reach targets that are located on the unseen area of the environment (see the left most plot in the Figure 4.10). The same distribution of target locations has been used here and only those that are located in the blue area are set as targets for this experiments. Therefore there are about 39 targets under static kinematics conditions and 42 targets under changing kinematics conditions (because of changing kinematics more targets will locate in the testing area). The results show that under static target training, neither model can reach even half of the targets. Their performance is worse under static target and changing kinematics. However, under changing target conditions, all models have obtained a good generalization but they need to take more than one step to reach any target. SPAC has again the best performance among other models under all conditions, while DDPG has the worst performance compared to other two models. These results indicate that learning with a static target hinders generalization as the learner overfits this specific target location.

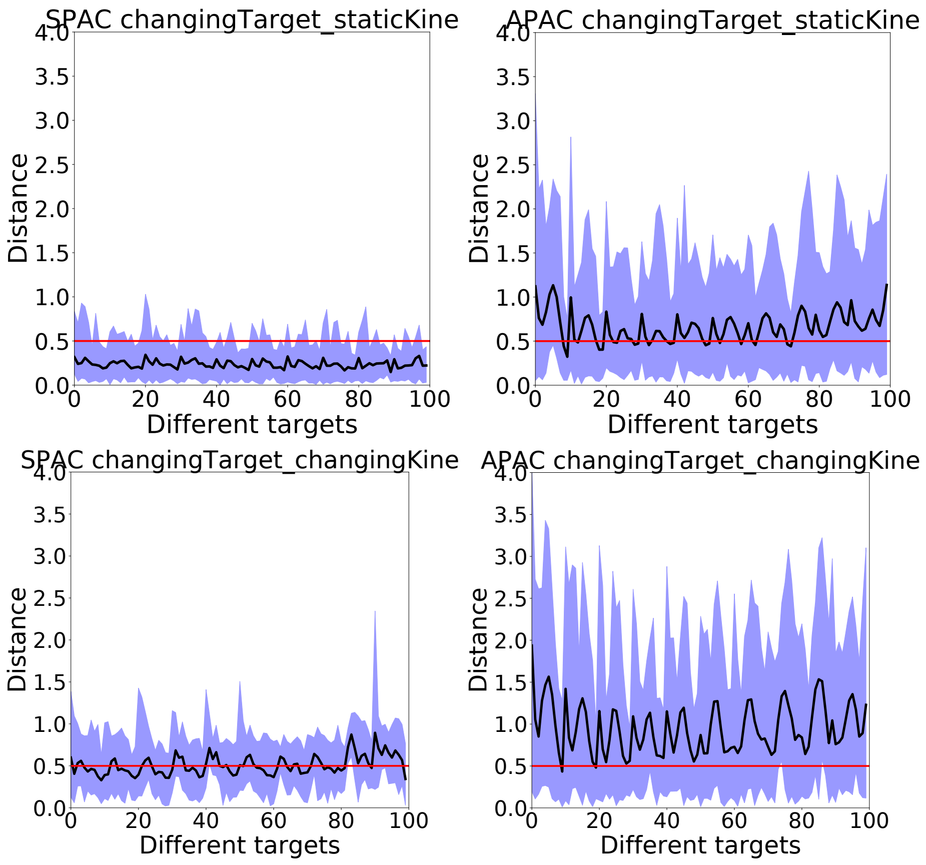


Figure 4.11: Error under occluded vision experiment. The actual distance from the agent to the target location under changing target conditions are shown for SPAC and APAC to reach 100 targets after training over 50 runs. The black line shows the average distance to the target. The area between minimum distance to the maximum distance is coloured in blue. The red line indicates the target zone. In general, SPAC performs better under the dark compared to APAC.

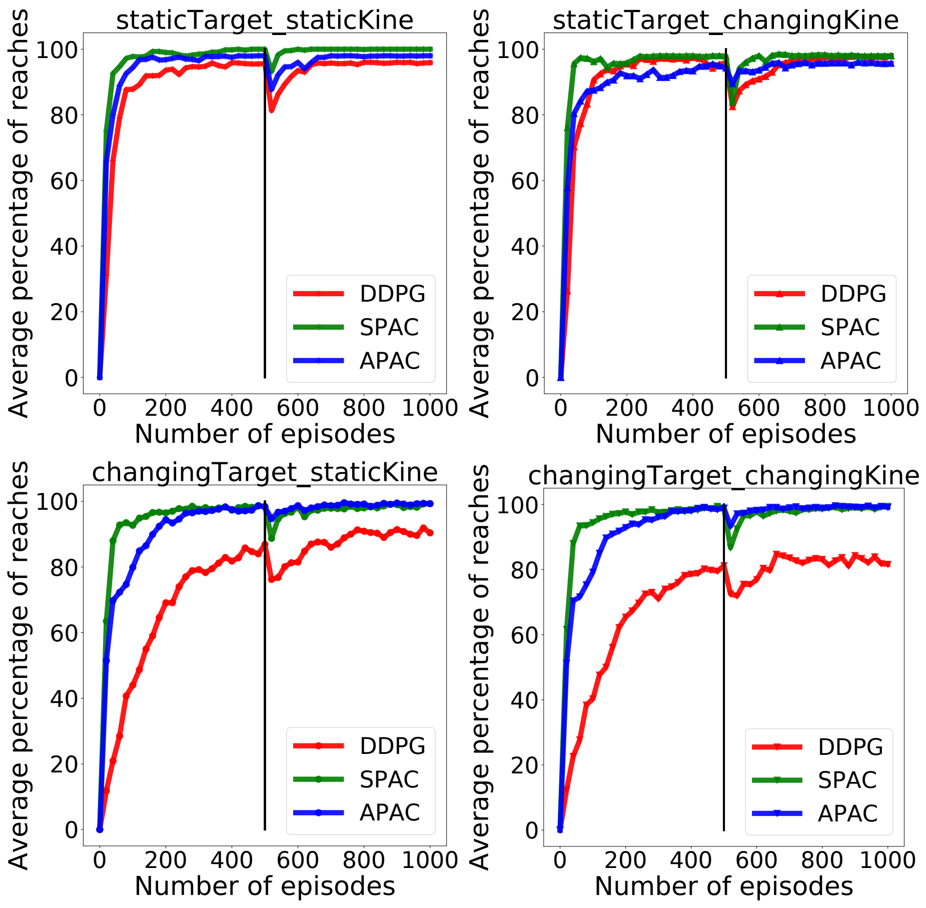


Figure 4.12: These plots illustrate behaviour of the three models under different conditions when a sudden change (increase on both joint by amount of 1 cm) is applied on kinematics at 500th episode. The black line indicates the time that change has been applied.

Finally, we want to show results with occluded vision. Since the habitual controller (DDPG) requires sensory input at all times, only the SPAC and APAC models were compared under this condition. In these experiments, the arm moves toward the target when the target location is only visible at the first step. When the forward model indicates that the agent has reached the target it stops and the actual distance from the agent (here the arm) is measured. Results of these occluded vision test are summarized in Figure 4.11 under changing target conditions since the performance of both models under static target conditions is near perfect.

The target zone is marked by the red line in each plot, while the average distance from the agent to the target location is drawn as a black line. The blue area shows the range of the distances that the arm has experienced during the occluded trials when the forward model thinks that it has reached the target. SPAC shows better performance under occluded vision compared to APAC under all conditions. The average actual distance of the end-effector to the target location under changing target/ static kinematic with SPAC is only about a distance of 0.4, which is less than the target zone radius. However, the APAC model under the same conditions stays about a distance of 1 away from the target. Under changing target/changing kinematics, this average actual distance from end-effector to the target is around 0.7 for SPAC and about 1.5 for APAC. It thus seems that any form of habit should be suppressed by a more advanced arbitrator.

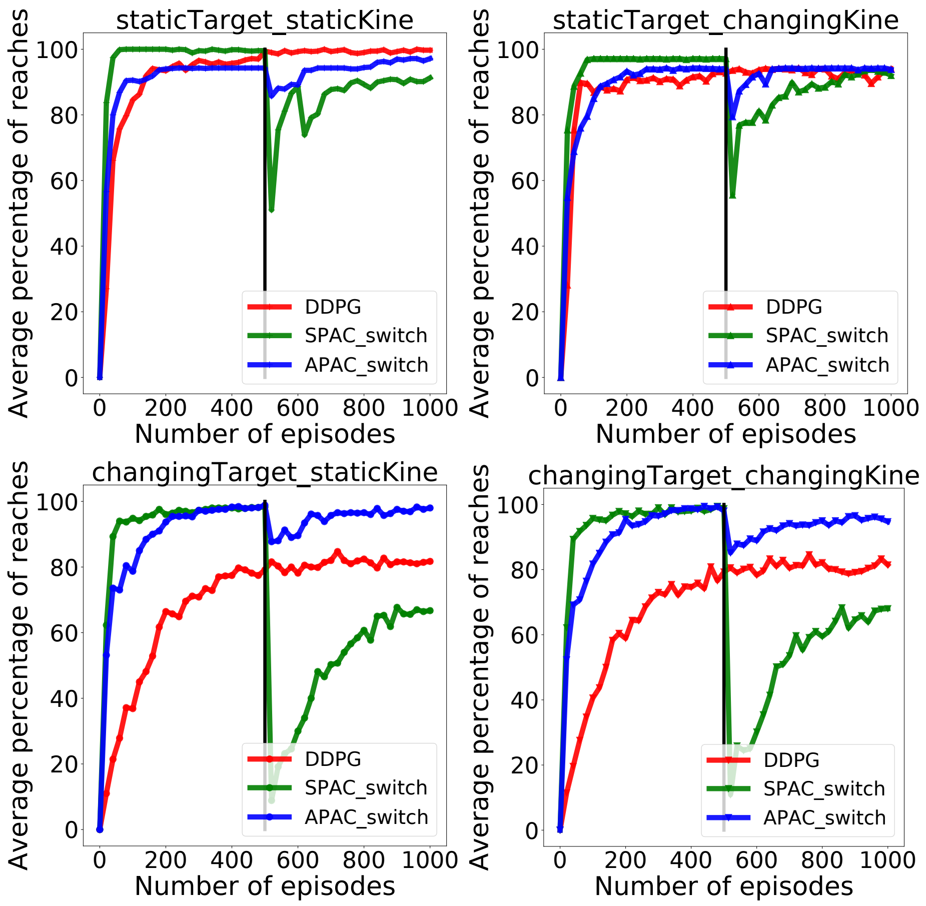


Figure 4.13: Effect of switching from planning to habitual controller is shown under different conditions. All 3 models are force to choose only the habitual controller after 500th episode.

We also examined behaviour of the three models (DDPG, SPAC, and APAC) under a big sudden change. We increase both forearm and arm's length by the amount of 1 cm. As Figure 4.12 displays, DDPG needs more time to get adjusted to the new kinematics and it takes longer for it to learn the new transition function. However, the process of being adjusted to this change is faster in SPAC and APAC.

We also examined the effect of planning on pure habits (see Figure 4.13). Since APAC performs better than DDPG, although it tends to choose more habitual controller than planning, we hypothesized that implying the forward and the inverse model will impact the learning on the habitual controller as well. The reason for this is since the internal models learn faster and provide more accurate samples in the experience replay memory, the habitual controller can benefit from these samples and will be trained better. To examine this hypothesis we design the next experiment.

In this experiment, all three models are trained only as normal for 500 episodes. At 500th episode we forced all models to take the actions only from the habitual controller. DDPG will remain unchanged since it always uses pure habits. SPAC shows a dramatic change in the performance under all conditions. Indeed by switching SPAC to pure habit, the habitual controller starts to learn the task and needs to spend more time to reach the same level of performance by DDPG. However, after 500 episodes of training with APAC, since training the habitual controller is already done with APAC, it takes only a few episodes to reach the DDPG's performance level. Under all conditions, except for the static target/static kinematics, APAC with pure habits outperforms the DDPG. This clearly illustrates that learning planning has a positive impact on habitual controller.

4.4.2 Multi-step reaching results

As it was described above, we examined using two different methods for calculating RPE. Figure 4.14 shows the results of the two RPE calculation method. Interestingly, both RPEs have the same performance under all 4 conditions. This shows that the temporal learning is able to handle the noise on the reward and can learn very fast in presence of the planning controller. Under changing target conditions, the shifting from planning to habits happens faster compared to static target conditions. Note that RPE here has no impact on the training and is only used for the arbitrator to select the action predicted either by the actor or by the inverse model. However, the samples that all networks get trained on are collected based on the real taken action, regardless of being selected by the actor or the inverse model.

Figure 4.15 shows the success rate for all three models under different conditions. All three models perform almost perfect to reach a static target. To reach changing targets, SPAC, which performs pure planning, works better than the other two. Interestingly, the APAC performs better than pure habits (DDPG), although it tends to use more habits than planning (see Figure 4.14). This demonstrates the power of a hybrid system.

Since the first episode of training uses the actions predicted by the actor, the number of actions by the actor is very high in the first episode. However, in the second episode, since the actor has not been trained yet, and the value of RPE is not under the desired threshold, the arbitrator selects the actions predicted by the inverse model.

To show average time consumption by each model under different conditions, we assumed that each action selected by the deliberative planning takes three times longer than an action selected by the habitual controller. The number here is arbitrary and only chosen to show the general effect. Figure 4.16 shows the average total time consumption of the three models to complete the task at each episode under different conditions. In this case, DDPG needs shorter time under static target conditions. In contrast, under changing target conditions, APAC completes the task of reaching targets faster. DDPG takes longer to finish the task as it needs more corrective actions.

Figure 4.17 illustrates trajectories of reaching each target under the changing target/changing kinematics condition, which is the most complex task here. Targets that are located at the edge of the reaching boundary are more difficult to be reached. DDPG has more difficulty to learn the non-linearity for those targets located at the edge, however, SPAC and APAC perform very well to reach those. Since the planning controller provides some good samples for each learner, it helps that the habitual system performs better in APAC than DDPG alone.

We already stressed that the model shows the general behavioural finding in humans that habit form only later in trials while novel actions are dominated by the deliberate planning controller. Another behavioural finding in humans is that the

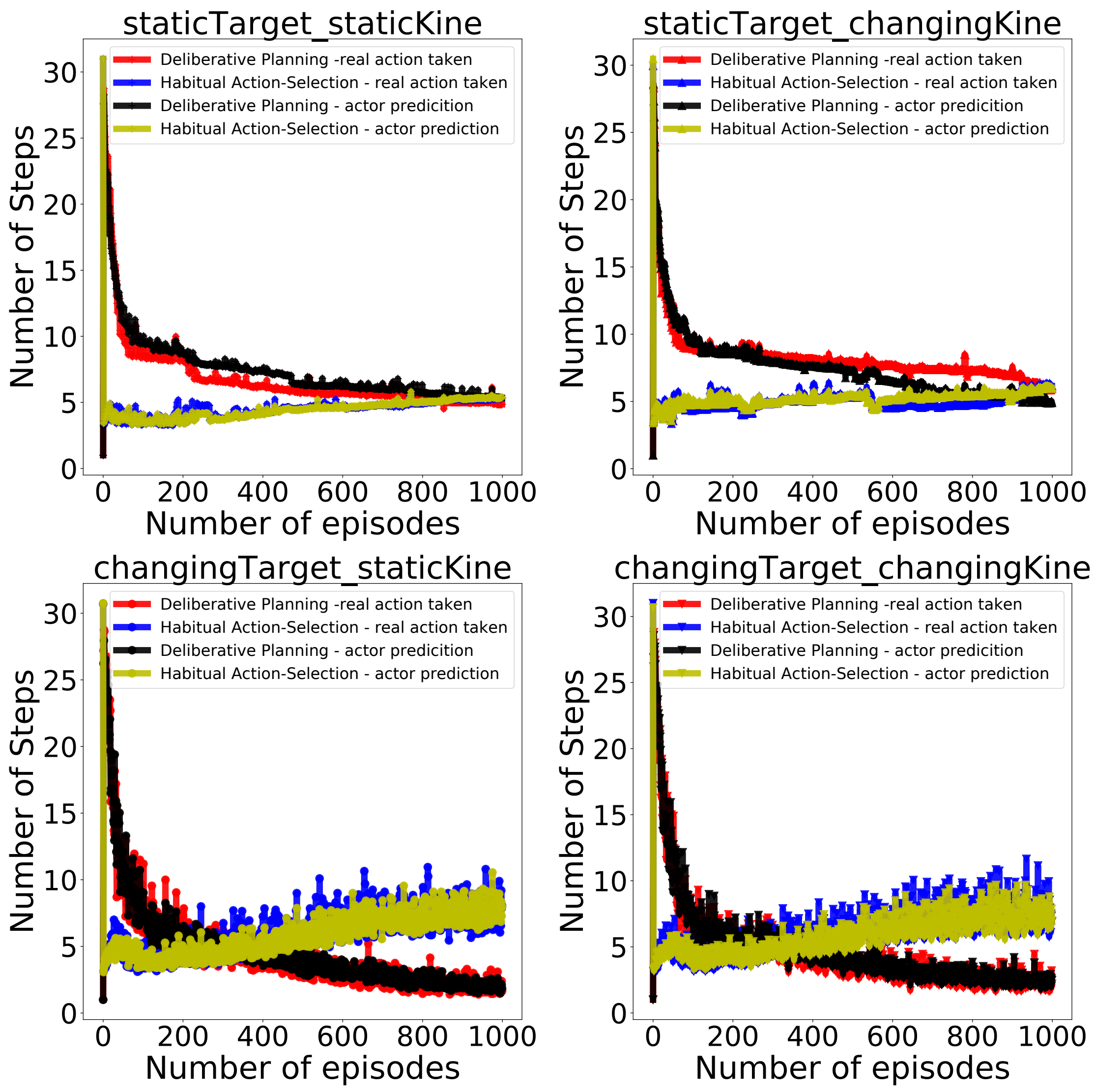


Figure 4.14: Average actions selected by the habitual controller (blue and yellow lines) vs. actions selected by the deliberative controller (red and black lines) at each episode during training that show how action selection shifts from planning to habitual during training. The black and the yellow lines show the behaviour of the system when RPE uses actor prediction, while the red and the blue lines indicate how the APAC behaves when RPE is calculated based on actual taken action.

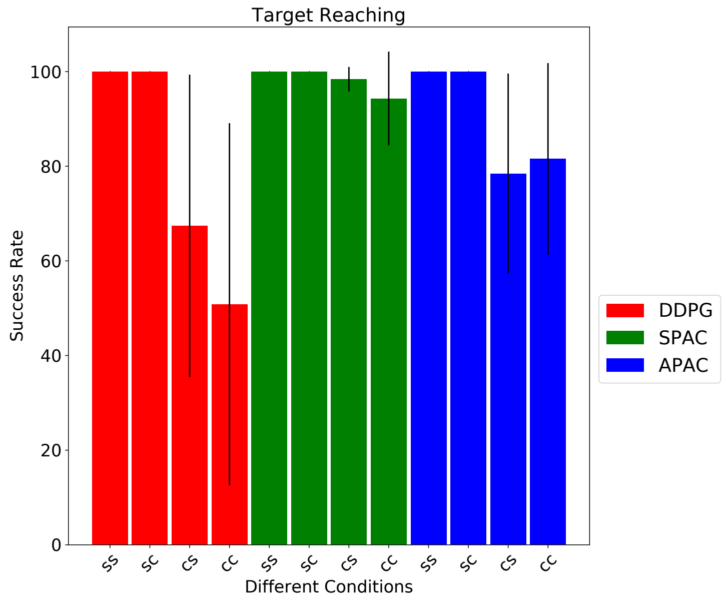


Figure 4.15: Success rate to reach targets after 1000 episodes of training under different conditions. 'ss', 'sc', 'cs', and 'cc' stand for static target/static kinematics, static target/changing kinematics, changing target/static kinematics and changing target/changing kinematics.

velocity of pointing and grasping the target follows a bell-shaped curve [18]. Although the reason for having such bell shape velocity profile remain unanswered, in the proposed model, we obtained this velocity profile in APAC by applying inertia on first two steps of the movement toward the target while the deceleration phase is a natural consequence of minimizing the distance to the target. More specifically, in APAC, the arm tends to take the maximum allowed movement in the first steps toward the target. However, the travelled distance toward the target becomes smaller and smaller when the arm goes closer to the target. Hence, by reducing the desired movement of only the first two steps, which can be attributed to inertia, we show that the velocity profile for reaching forms a bell shape similar to human. To add the inertia in first two steps, we simply divide the selected angles by the arbitrator by 4 in the first step and by 2 in the second step. These smaller angle movements result in overall slower movements in first two steps. The rest of the steps toward the target are able to execute the maximum allowed angles by the arbitrator.

Figure 4.18 represents velocity profile of some of the reaches under various conditions. The figure shows distance travelled by the arm's end effector at a time which

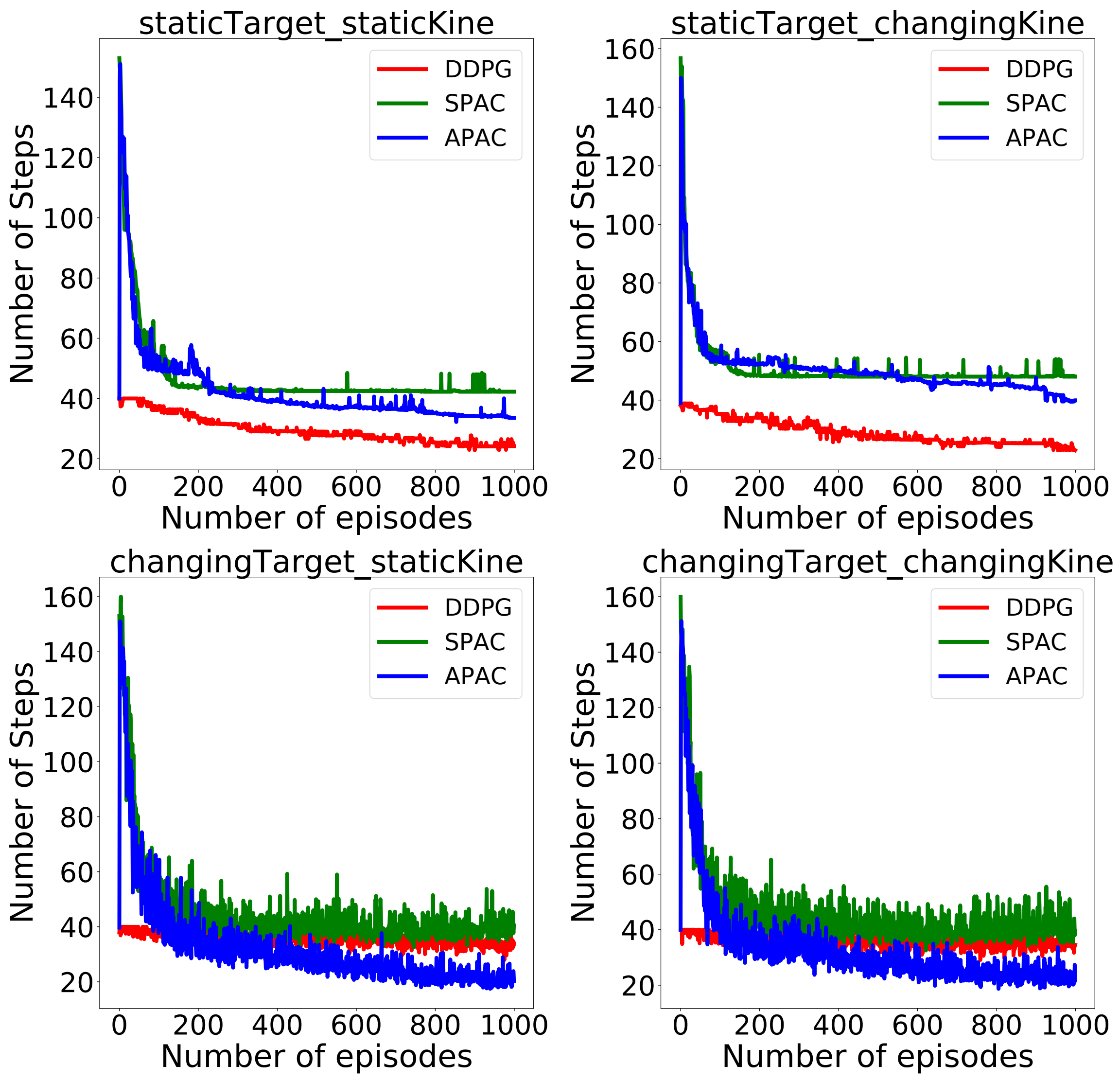


Figure 4.16: Total time consumption (by habits and planning controller) during training using all three models DDPG, SPAC, and APAC under different conditions. Each planning is assumed to take 3 times longer to be calculated than each habitual action.

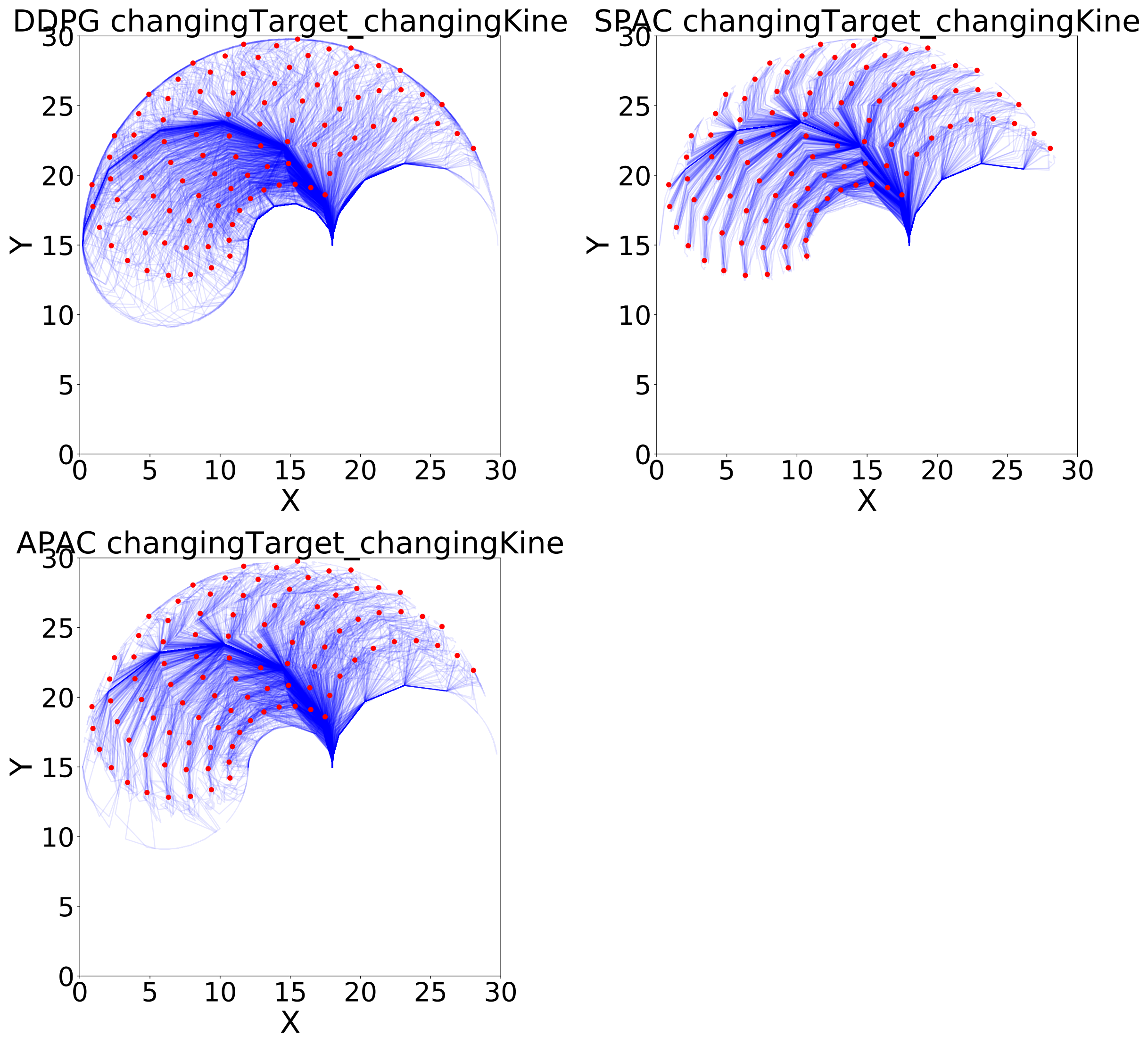


Figure 4.17: Trajectory of reaching targets under changing target/changing kinematics using all three models DDPG, SPAC, and APAC. DDPG cannot reach all targets (red dots) especially those were located at the boundaries of the reaching area.

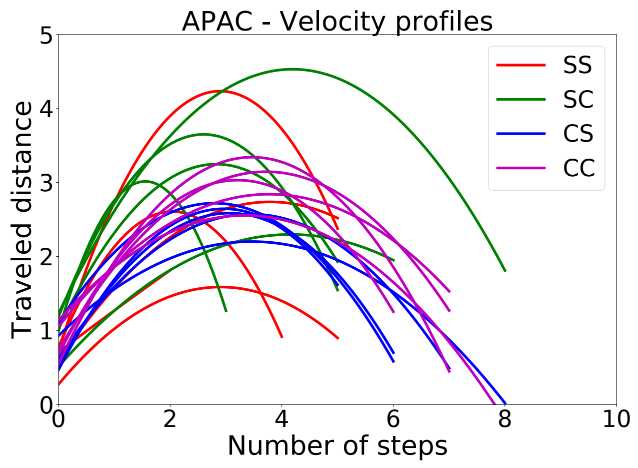


Figure 4.18: Velocity profile of 5 different reaching under static target/static kine (ss), static target/changing kine (sc), changing target/static kine (cs), and changing target/changing kine (ss). The traveled distance of the end-effector toward the target at each step interpreted as the moving velocity at a time. For the first two steps of each movement, inertia is applied to slow down the movement. For more information please see [18].

can be interpreted as the velocity of the movement. The number of steps to reach the target depends on the distance between the target and the arm. The largest travelled distance happens around the middle of the reaches. Curves shown in the figure are the travelled distance at each step for each movement that have been fit to a quadratic polynomial. By adding the inertia at first two steps, the velocity profile forms a bell shape similar to human reaching.

I also examined the impact of the amount of the movement limitation for each model, under different conditions. Figure 4.19 illustrates performance of all three models under different amount of limitation for movement under various conditions. As this figure shows, Small amount of movements makes it very difficult for all models including SPAC and APAC to reach the target for multi-step reaching. Performance of APAC stays very close to SPAC with different limitations applied to the robotic arm. Moreover, the average performance of SPAC and APAC increases with the degree of movement becomes bigger. However, the average performance of DDPG does not increase significantly with bigger movements. Increasing size of the memory replay or longer training may help the models to perform better that should be investigated. Note that the actions always are under the noise applied to the environment by the

Ornstein-Uhlenbeck process. With small movements, this noise provides new samples that make the models incapable to learn the task.

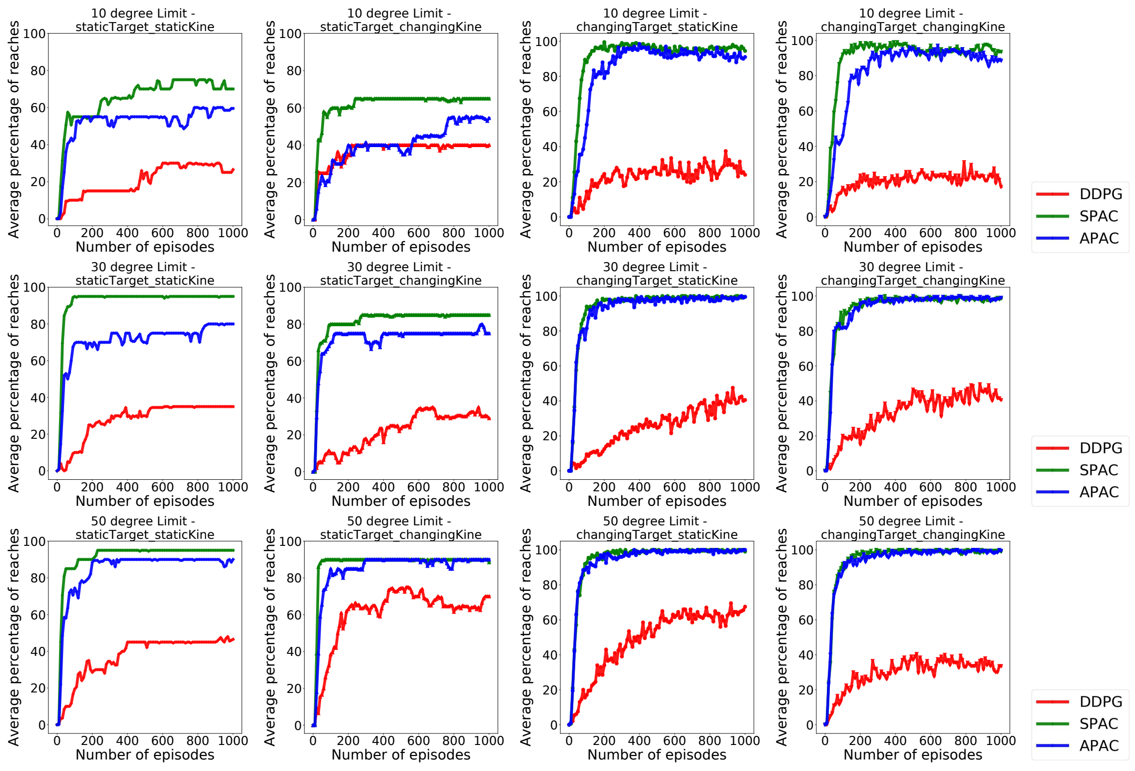


Figure 4.19: Average performance of all three models with 10, 30, and 50 degree limitation under different conditions.

Figure 4.20 also represents the improvement of the performance for each particular model under different conditions. Almost for all models, bigger movements leads to higher performance under any conditions. This should be noted again that the performance of the APAC stays very close to the pure planning (SPAC), however, it tends to use more habitual controller than the planning.

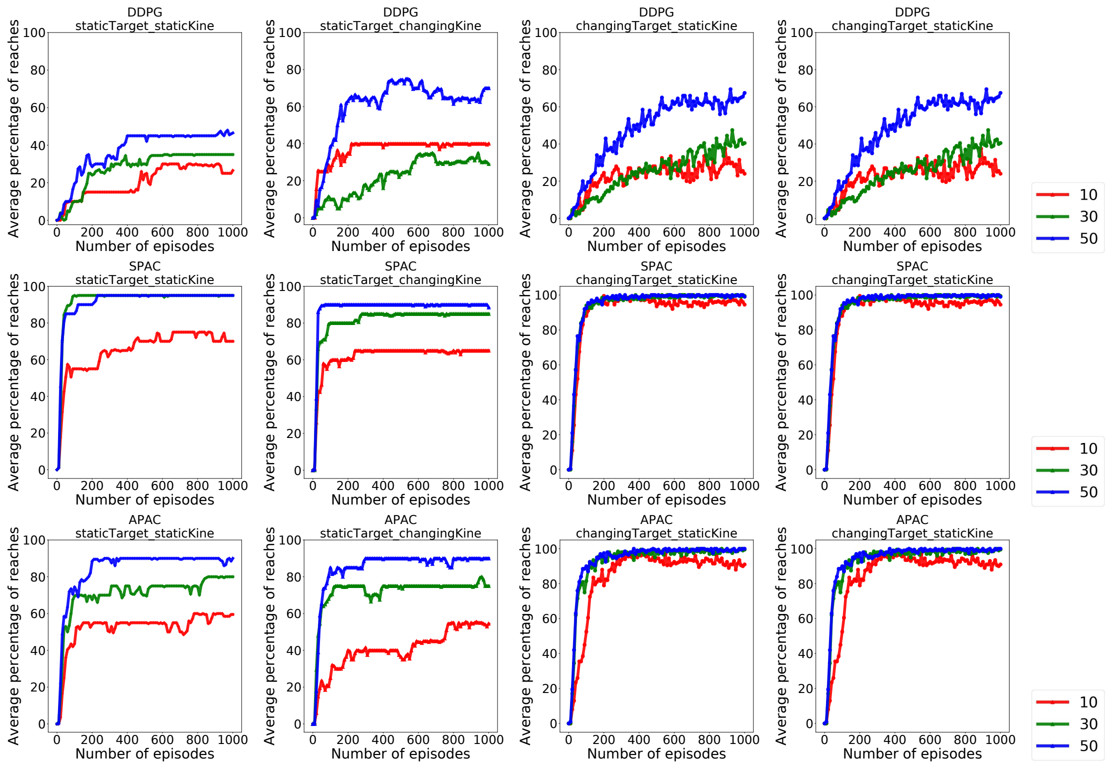


Figure 4.20: Average performance of DDPG, SPAC, and APAC with different degrees of limitation (10, 30, and 50 degrees) under different conditions.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

David Marr suggested 3 levels of understanding [179]. 1) Computational theory, 2) Representation and algorithm, and 3) Implementation. The first step investigates the logic and the goal of the computation. The second step discusses the representation and the algorithm of the model. And the third step, explains how it is built physically. Here, we investigated the target reaching task under computational level. The APAC model presented here, represents the system level of the model and is more on the second level, while the adaptive observer has been implemented on a real robot arm therefore it reaches the third level.

In this dissertation, first, we have shown how an internal model in the form of an observer can be implemented in the neural field framework. Such an internal model is critical to overcome biological constraints such as noisy input, feedback delays, or moving in the dark. The adaptive observer is able to move the arm and reach the target even with occluded vision. It is interesting to note that we do not have a separate internal representation for the sensory and predicted position of the hand. In our case, the map that represents the position of the hand represents the best estimate available, either through vision or by dead reckoning when vision is not available.

In order to understand motor control in humans, it is important to show how well established control architectures can be implemented in a biologically realizable way. We believe that our adaptive observer has some attractive features, namely:

- The end-effector map is both a map of the actual hand position with visual feedback, but includes the mechanisms of the observer in the same neuronal substrate. While this is often somewhat separated in engineering solutions, our system implements both in the same neuronal substrate. This architecture highlights the possible challenge in interpreting neural activity in motor maps as this activity can correlate with current states and predicted states, depending on the specific experiment circumstance.

- Neuronal signals related to reaching movements, including hand and target positions as well as relative positions, have been observed in the posterior prefrontal cortex. It is hence likely that such an internal model could be located there although the predictive nature of the signals has, to our knowledge, not yet been tested. For this, we propose to record during reaching movements with different levels of occluded vision. Our prediction would be that these cells still respond with the predicted target and hand position.

- During the experiments, it became clear that we had to include an adaptive motor component to compensate for delays in the sensory input and motor execution. Such a mechanism is often discussed with reference to the cerebellum in situations of motor adaptation with changed limb dynamics or modified sensory feedback. The model could be applied to such classical experiments. Also, a very attractive feature of the model is that the movement compensation through the internal model is trained by only a few actual movement example from which the system could generalize to other movements. This is important as this mechanism can help to adapt to changing environments and alterations of the motor system itself. Our experiments here have demonstrated the effectiveness in a static environment, but the learning ability of the model sets the stage for further investigations of its behaviour in changing environments.

- Reaching under occluded vision is another feature of this model that is enabled by the presence of the internal models. This feature is called memory-guided movements in human that has been very well studied.

- And last but not least, the neural activity of the hand-target centred map was also used to determine the velocity of the arm movement, by which the adaptive observer forms a bell shape similar to human arm movements.

The adaptive observer, besides many interesting features, cannot explain cooperation or competition between different controller. Therefore, we designed APAC model to study such hybrid system with deliberate planning system and habitual control. The APAC is a system level implementation of the internal model and reinforcement learning structure.

Habitual control will, of course, be very good after long training in static environments. It was hence important to study the model in changing environments. We investigated the behaviour of the proposed model (APAC) under changing target conditions (to manipulate environmental reward function), changing kinematics of the agent (to manipulate the learned transitional model), and with and without vision. We also tested the model under various conditions to see how good they can generalize their learning paradigm.

Not only have human decision-making studies supported the notion that both habitual and planning controls are used during decision-making [71, 77], there is evidence that arbitration may be a dynamic process involving specific brain regions [78]. Our APAC model results suggest that such an arbitration strategy, wherein the planning paradigm is used until the habitual system's predictions become reliable can result in performance that is non-inferior to exclusive planning control in most cases. Thus, the APAC model supports:

- The existence of the two control systems.
- The importance and value of implementing predominantly planning control early in behavioural learning.
- The diminishing importance of planning control with greater experience in a relatively static environment.
- The importance of the arbitrator that arbitrates between the two existing controllers.
- Memory-guided movements under occluded vision.
- And the bell shape velocity profile of movements.

Results show significant improvement in performance when planning is available compared to the pure habitual system under changing the environment including changing in the environmental reward function and kinematics of the agent. In comparison, SPAC and APAC are flexible under these changes. This experiment shows that having an internal model is a key to robustness on the changing environment.

By considering that planning is costly, having another control system that is able to provide cheaper solutions can be useful. In this model, there is no inherent time constraint or computation time difference between the habitual and planning controller. However, if we take this time constraint into account, the APAC is a better model than pure planning (SPAC). Indeed, the overall number of actions taken based on a planned action is less time consuming in the arbitrated model than the number of planned actions taken by the non-arbitrated model when considering some cost of planning.

Since DDPG has no internal model, it cannot use any sort of planning to move under occlusion. However, systems like SPAC and APAC build an internal model of the environment that enables them to anticipate and plan a target even when there is no visual information available. Results show that SPAC can perform better in the dark. This is a good example to show that a more sophisticated arbitrator could take different circumstances into account. For occluded vision, such an arbitrator should suppress habitual actions.

Our application example focused on the implementation of the habitual system as an actor-critic for reinforcement learning and a planning system with a forward and inverse model using supervised learning. There have been previous examples of combining both, some form of supervised learning with RL systems and the use of the internal model. In particular, Dyna-Q [98] and supervised actor-critic [96, 97] are examples of models that bring the capacity of planning into a model-free reinforcement learning space. For example, the supervised actor-critic [96, 97] is able to tune the actor manually and very fast when it is needed. This solution is beneficial when dynamics of the system changes dramatically or a new policy is needed to be learned in a very short time. These authors used a gain scheduler that weights the control signal provided by the actor from the reinforcement learner and the supervised actor. In contrast to supervised actor-critic, our proposed model autonomously learns the internal model and arbitrates between the two controllers automatically and not manually. The intention of our model is to study the interaction of habitual and planning systems in form of an arbitrator and ultimately to understanding human behaviour.

Sutton proposed Dyna-Q that is an integrated model for learning and planning [98]. With respect to our model, Dyna-Q is also a blend of model-free and model-based reinforcement learning algorithms. Dyna-Q can build a transition function and the reward function by hallucinating random samples. Therefore, although it uses a model-free paradigm at the beginning it becomes a model-based solution by learning the model of the world using the hallucination.

In our model, the internal model is used to predict the future state of the agent, unlike Dyna-Q that uses the model to train the critic and anticipate the future reward. Moreover, Dyna-Q starts from the model-free controller and becomes a model-based controller. Hence, while Dyna-Q has focused on the utilization of internal models to learn a reinforcement controller, our study here is concerned with the arbitration of two control systems.

However, since APAC tend to select actions from the planning controller that it learns very fast, it provides more accurate samples in the experience replay memory. Therefore, similar to the Dyna-Q, the habitual system takes advantage of learning from more valuable samples that lead the habitual controller to a better performance compared to the time that it is trained standalone (in pure habitual paradigm).

5.2 Future Works

The proposed adaptive observer is a first attempt to implement a biologically motivated internal model that builds on the previously proposed framework of dynamic neural fields. Such explicit models are only a first step to aid investigations in neuroscience. While the neural field approach is based on neuroscientific evidence [104] to capture common cortical dynamics, more specific investigations of correlating our model with specific brain signatures should follow.

The arbitrated predictive actor-critic is a first attempt to explain possible path for cooperation between the habitual and planning controller. The general view of the proposed model shows that the model is applicable in different applications. In general, it can be implemented in other applications where providing a large enough data set of true labels is difficult or expensive while roughly labelled samples are available or easier to provide. Such system is able to learn the roughly labelled data set and improve its predictions using supervised samples. The output of each controller in APAC for the target reaching are actions. In other applications, other outputs should be learned. Indeed the output can be a word, a sentence, a category, or a class depend on the application and the task.

In APAC model since the task was the target reaching we considered an inverse model as the planner. However, a more general solution can be defined to learn the deliberative planning to make a decision deliberately. For example, for a more complex task, a hierarchical planner might be needed. Heuristic solutions like decision trees and A^* that search through possible actions are also possible. Especially when a system has more than one agent where they need to cooperate to reach a goal, having a cooperative deliberative planner becomes very important.

Moreover, in APAC, the forward model performs as a state prediction error component. We implemented a neural network to learn the forward model, however, using probabilistic models to learn the state prediction are also feasible. The integrator itself is a simple switch in APAC, however, it can be defined by a Kalman-Filter similar to the adaptive observer.

Other implementations of the habitual controller are also possible and should be studied. For example, it could be beneficial to learn a few options to select among them like option-critic architecture [180].

In APAC for target reaching, we assumed that the planning controller only plans one step ahead of its current situation, however, using the forward model, it is possible to plan few steps ahead to have a more accurate movement. The implementation of the APAC is also in a way that both controllers produce their output at the same time. Therefore, the arbitrator is set to choose the action from the habitual controller for the first two steps of each episode. For example, using separate threads to run each controller would be more realistic and the forward and the inverse model can plan a few steps ahead and then pass the output to the arbitrator which will also make the time constraint comparison possible.

The vision for both systems (i.e APAC and adaptive observer) is fixed meaning the view of the visual information is stationary all the time. Even the hand-target centred map in the adaptive observer uses the stationary vision. Using a more sophisticated map that can build such a hand-target map relative to the gaze centered and eye direction in the head would be valuable.

The arbitration mechanism is very simple and basically works as a switch. For example, more biologically plausible implementations like DNFs should be considered. Moreover, the arbitrator needs to be flexible relative to different tasks. For instance, the threshold is defined as a fixed value, however, it might be better if we implement it in a changing, dynamical manner that has to be learned through training. Implementing such threshold in a changing and dynamic manner would be useful to explain how individuals have different behaviours under a specific task and condition.

Both the adaptive observer and the APAC are able to reach the target under the dark since both take advantage of an internal model. Here we only reported such behaviour, however, a detailed study to compare our models with human arm movement under occluded vision would be also interesting.

We also applied fixed limitations on both shoulder and elbow joints. However, APAC may learn to reach the target on a more curvilinear similar to the human if the limitations are not the same on both joints. For example, how the trajectory of arm movement would change if the limitation on the shoulder is smaller or greater than the limitation on the elbow joint.

And last but not least, our proposed models were mainly examined to reach a fixed target. Even for the reaching a moving target, the proposed adaptive observer adjusts its movements at every step based on the current target location and has no anticipation of future location of the moving object. In contrast, Human builds an internal representation of all surrounding moving objects and they can anticipate future states of the objects in their environment which enables them to move safely. For example, human can catch a ball that has been thrown at them. In APAC and adaptive observer, the internal model only learns the kinematics of the agent. Therefore, a more sophisticated model needs to learn the behaviour of moving objects such as speed and trajectory of the movement and predict the future state of the moving objects.

Bibliography

[1] Farzaneh S Fard, Paul Hollensen, Dietmar Heinke, and Thomas P Trappenberg. Modeling human target reaching with an adaptive observer implemented with dynamic neural fields. *Neural Networks*, 72:13–30, 2015.

[2] Farzaneh S Fard and Thomas P Trappenberg. A novel model for arbitration between planning and habitual control systems. *arXiv preprint arXiv:1712.02441*, 2017.

[3] Soeren Strauss and Dietmar Heinke. A robotics-based approach to modeling of choice reaching experiments on visual attention. *Frontiers in psychology*, 3:105, 2012.

[4] Farzaneh S Fard, Abraham Nunes, and Thomas Trappenberg. An actor-critic with an internal model. In *Annual Conference on Cognitive Computational Neuroscience (CCN)*, 2017.

[5] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[6] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[7] David Silver and Demis Hassabis. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 2016.

[8] Stuart Russel and Peter Norvig. Artificial intelligence: A modern approach (3rd edition). *EUA: Prentice Hall*, 2010.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] J Norberto Pires. Robot manipulators and control systems. *Industrial Robots Programming: Building Applications for the Factories of the Future*, pages 35–107, 2007.

[12] Hooman Samani. *Cognitive robotics.* CRC Press, 2015.

[13] William Browne, Kazuhiko Kawamura, Jeffrey Krichmar, William Harwin, and Hiroaki Wagatsuma. Cognitive robotics: new insights into robot and human intelligence by reverse engineering brain functions [from the guest editors]. *IEEE Robotics & Automation Magazine*, 16(3):17–18, 2009.

[14] Maeda Kawato, Y Maeda, Y Uno, and R Suzuki. Trajectory formation of arm movement by cascade neural network model based on minimum torque-change criterion. *Biological cybernetics*, 62(4):275–288, 1990.

[15] Flavia Filimon, Jonathan D Nelson, Ruey-Song Huang, and Martin I Sereno. Multiple parietal reach regions in humans: cortical representations for visual and proprioceptive feedback during on-line reaching. *Journal of Neuroscience*, 29(9):2961–2971, 2009.

[16] A Battaglia-Mayer and R Caminiti. Posterior parietal cortex and arm movement.

[17] Francesco Lacquaniti, John F Soechting, and SA Terzuolo. Path constraints on point-to-point arm movements in three-dimensional space. *Neuroscience*, 17(2):313–324, 1986.

[18] RG Marteniuk, CL MacKenzie, M Jeannerod, S Athenes, and C Dugas. Constraints on human arm movement trajectories. *Canadian Journal of Psychology/Revue canadienne de psychologie*, 41(3):365, 1987.

[19] Matthew Heath, David A Westwood, and Gordon Binsted. The control of memory-guided reaching movements in peripersonal space. *Motor control*, 8(1):76–106, 2004.

[20] David A Westwood, Matthew Heath, and Eric A Roy. No evidence for accurate visuomotor memory: systematic and variable error in memory-guided reaching. *Journal of motor behavior*, 35(2):127–133, 2003.

[21] Denise YP Henriques, Eliana M Klier, Michael A Smith, Deborah Lowy, and J Douglas Crawford. Gaze-centered remapping of remembered visual space in an open-loop pointing task. *Journal of Neuroscience*, 18(4):1583–1594, 1998.

[22] JF Soechting, SI Helms Tillery, and M Flanders. Transformation from head-to shoulder-centered representation of target direction in arm movements. *Journal of Cognitive Neuroscience*, 2(1):32–43, 1990.

[23] Christopher A Buneo, Murray R Jarvis, Aaron P Batista, and Richard A Andersen. Direct visuomotor transformations for reaching. *Nature*, 416(6881):632–636, 2002.

[24] Aaron P Batista, Christopher A Buneo, Lawrence H Snyder, and Richard A Andersen. Reach plans in eye-centered coordinates. *Science*, 285(5425):257–260, 1999.

[25] Gunnar Blohm and J Douglas Crawford. Computations for geometrically accurate visually guided reaching in 3-d space. *Journal of Vision*, 7(5):4–4, 2007.

[26] Flavia Filimon, Jonathan D Nelson, Donald J Hagler, and Martin I Sereno. Human cortical representations for reaching: mirror neurons for execution, observation, and imagery. *Neuroimage*, 37(4):1315–1328, 2007.

[27] J Douglas Crawford, W Pieter Medendorp, and Jonathan J Marotta. Spatial transformations for eye–hand coordination. *Journal of neurophysiology*, 92(1):10–19, 2004.

[28] Jody C Culham, Cristiana Cavina-Pratesi, and Anthony Singhal. The role of parietal cortex in visuomotor control: what have we learned from neuroimaging? *Neuropsychologia*, 44(13):2668–2684, 2006.

[29] Hiroshi Imamizu, Satoru Miyauchi, Tomoe Tamada, Yuka Sasaki, Ryousuke Takino, Benno PuÈtz, Toshinori Yoshioka, and Mitsuo Kawato. Human cerebellar activity reflecting an acquired internal model of a new tool. *Nature*, 403(6766):192–195, 2000.

[30] Ya-weng Tseng, Jörn Diedrichsen, John W Krakauer, Reza Shadmehr, and Amy J Bastian. Sensory prediction errors drive cerebellum-dependent adaptation of reaching. *Journal of Neurophysiology*, 98(1):54–62, 2007.

[31] Matthis Synofzik, Axel Lindner, and Peter Thier. The cerebellum updates predictions about the visual consequences of one's behavior. *Current Biology*, 18(11):814–818, 2008.

[32] Melvyn A Goodale and A David Milner. Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–25, 1992.

[33] Nicholas P Holmes and Charles Spence. The body schema and multisensory representation (s) of peripersonal space. *Cognitive processing*, 5(2):94–105, 2004.

[34] Elisabetta Làdavas, Giuseppe Di Pellegrino, Alessandro Farnè, and Gabriele Zeloni. Neuropsychological evidence of an integrated visuotactile representation of peripersonal space in humans. *Journal of Cognitive Neuroscience*, 10(5):581–589, 1998.

[35] Alessandro Farnè and Elisabetta Làdavas. Dynamic size-change of hand peripersonal space following tool use. *Neuroreport*, 11(8):1645–1649, 2000.

[36] Jody C Culham, Jason Gallivan, Cristiana Cavina-Pratesi, and Derek J Quinlan. fmri investigations of reaching and ego space in human superior parieto-occipital cortex. *Embodiment, ego-space and action*, pages 247–274, 2008.

[37] Jason D Connolly, Richard A Andersen, and Melvyn A Goodale. Fmri evidence for a'parietal reach region'in the human brain. *Experimental Brain Research*, 153(2):140–145, 2003.

[38] Jérôme Prado, Simon Clavagnier, Hélene Otzenberger, Christian Scheiber, Henry Kennedy, and Marie-Thérèse Perenin. Two cortical systems for reaching in central and peripheral vision. *Neuron*, 48(5):849–858, 2005.

[39] Richard A Andersen, Greg K Essick, and Ralph M Siegel. Encoding of spatial location by posterior parietal neurons. *Science*, 230(4724):456–458, 1985.

[40] RA Andersen, GK Essick, and RM Siegel. Neurons of area 7 activated by both visual stimuli and oculomotor behavior. *Experimental Brain Research*, 67(2):316–322, 1987.

[41] Richard A Andersen and Christopher A Buneo. Intentional maps in posterior parietal cortex. *Annual review of neuroscience*, 25(1):189–220, 2002.

[42] P Rondot, J De Recondo, and JL Dumas. Visuomotor ataxia. *Brain: a journal of neurology*, 100(2):355–376, 1977.

[43] NORMAN Geschwind and ANTONIO R Damasio. Apraxia. *Handbook of clinical neurology*, 1(49):423–432, 1985.

[44] Christopher A Buneo and Richard A Andersen. The posterior parietal cortex: sensorimotor interface for the planning and online control of visually guided movements. *Neuropsychologia*, 44(13):2594–2606, 2006.

[45] Bruno B Averbeck, Matthew V Chafee, David A Crowe, and Apostolos P Georgopoulos. Parietal representation of hand velocity in a copy task. *Journal of neurophysiology*, 93(1):508–518, 2005.

[46] Michael Vesia and J Douglas Crawford. Specialization of reach function in human posterior parietal cortex. *Experimental brain research*, 221(1):1–18, 2012.

[47] W Pieter Medendorp, Herbert C Goltz, Tutis Vilis, and J Douglas Crawford. Gaze-centered updating of visual space in human parietal cortex. *The Journal of neuroscience*, 23(15):6209–6214, 2003.

[48] W Pieter Medendorp, Herbert C Goltz, J Douglas Crawford, and Tutis Vilis. Integration of target and effector information in human posterior parietal cortex for the planning of action. *Journal of Neurophysiology*, 93(2):954–962, 2005.

[49] Juan Fernandez-Ruiz, Herbert C Goltz, Joseph FX DeSouza, Tutis Vilis, and J Douglas Crawford. Human parietal 'reach region' primarily encodes intrinsic visual direction, not extrinsic movement direction, in a visual–motor dissociation task. *Cerebral Cortex*, 17(10):2283–2292, 2007.

[50] Paul Cisek. Cortical mechanisms of action selection: the affordance competition hypothesis. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1485):1585–1599, 2007.

[51] Tyson Aflalo, Spencer Kellis, Christian Klaes, Brian Lee, Ying Shi, Kelsie Pejsa, Kathleen Shanfield, Stephanie Hayes-Jackson, Mindy Aisen, Christi Heck, et al. Decoding motor imagery from the posterior parietal cortex of a tetraplegic human. *Science*, 348(6237):906–910, 2015.

[52] Mitsuo Kawato. 20 learning internal models of the motor apparatus. *The acquisition of motor behavior in vertebrates*, page 409, 1996.

[53] Maurice A Smith and Reza Shadmehr. Intact ability to learn internal models of arm dynamics in huntington's disease but not cerebellar degeneration. *Journal of Neurophysiology*, 93(5):2809–2821, 2005.

[54] Michael A Conditt, Francesca Gandolfo, and Ferdinando A Mussa-Ivaldi. The motor system does not learn the dynamics of the arm by rote memorization of past experience. *Journal of Neurophysiology*, 78(1):554–560, 1997.

[55] Reza Shadmehr and Ferdinando A Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *The Journal of Neuroscience*, 14(5):3208–3224, 1994.

[56] RC Miall, DJ Weir, Daniel M Wolpert, and JF Stein. Is the cerebellum a smith predictor? *Journal of motor behavior*, 25(3):203–216, 1993.

[57] RC Miall and GZ Reckess. The cerebellum and the timing of coordinated eye and hand tracking. *Brain and cognition*, 48(1):212–226, 2002.

[58] Reza Shadmehr, Maurice A Smith, and John W Krakauer. Error correction, sensory prediction, and adaptation in motor control. *Annual review of neuroscience*, 33:89–108, 2010.

[59] M Desmurget, CM Epstein, RS Turner, C Prablanc, GE Alexander, and ST Grafton. Role of the posterior parietal cortex in updating reaching movements to a visual target. *Nature neuroscience*, 2(6):563–567, 1999.

[60] Daniel Wolpert. *Daniel Wolpert: The Real Reason for Brains.* TED, 2011.

[61] Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12):1704–1711, 2005.

[62] Yael Niv. Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154, 2009.

[63] R Christopher Miall and Daniel M Wolpert. Forward models for physiological motor control. *Neural networks*, 9(8):1265–1279, 1996.

[64] Daniel M Wolpert, R Chris Miall, and Mitsuo Kawato. Internal models in the cerebellum. *Trends in cognitive sciences*, 2(9):338–347, 1998.

[65] Mitsuo Kawato, Tomoe Kuroda, Hiroshi Imamizu, Eri Nakano, Satoru Miyauchi, and Toshinori Yoshioka. Internal forward models in the cerebellum: fmri study on grip force and load force coupling. *Progress in brain research*, 142:171–188, 2003.

[66] J Randall Flanagan and Alan M Wing. The role of internal models in motion planning and control: evidence from grip force adjustments during movements of hand-held loads. *Journal of Neuroscience*, 17(4):1519–1528, 1997.

[67] JoAnn Kluzik, Jörn Diedrichsen, Reza Shadmehr, and Amy J Bastian. Reach adaptation: what determines whether we learn an internal model of the tool or adapt the model of our arm? *Journal of neurophysiology*, 100(3):1455–1464, 2008.

[68] Marco Iacoboni, Roger P Woods, Marcel Brass, Harold Bekkering, John C Mazziotta, and Giacomo Rizzolatti. Cortical mechanisms of human imitation. *science*, 286(5449):2526–2528, 1999.

[69] Bernard W Balleine and Anthony Dickinson. Goal-directed instrumental action: contingency and incentive learning and their cortical substrates. *Neuropharmacology*, 37(4):407–419, 1998.

[70] Jan Gläscher, Nathaniel Daw, Peter Dayan, and John P O'Doherty. States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–595, 2010.

[71] Nathaniel D Daw, Samuel J Gershman, Ben Seymour, Peter Dayan, and Raymond J Dolan. Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 69(6):1204–1215, 2011.

[72] Earl K Miller and Jonathan D Cohen. An integrative theory of prefrontal cortex function. *Annual review of neuroscience*, 24(1):167–202, 2001.

[73] James C Houk and James L Adams. 13 a model of how the basal ganglia generate and use neural signals that. *Models of information processing in the basal ganglia*, page 249, 1995.

[74] Máté Lengyel and Peter Dayan. Hippocampal contributions to control: The third way. In *NIPS*, volume 20, pages 889–896, 2007.

[75] Russell A Poldrack, J Clark, EJ Pare-Blagoev, D Shohamy, J Creso Moyano, C Myers, and MA Gluck. Interactive memory systems in the human brain. *Nature*, 414(6863):546–550, 2001.

[76] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

[77] John P. O'Doherty, Sang Wan Lee, and Daniel Mcnamee. The structure of reinforcement-learning mechanisms in the human brain. *Current Opinion in Behavioral Sciences*, 1(2014):1–7, 2015.

[78] Sang Wan Lee, Shinsuke Shimojo, and John P ODoherty. Neural computations underlying arbitration between model-based and model-free learning. *Neuron*, 81(3):687–699, 2014.

[79] Richard Stuart Sutton. Temporal credit assignment in reinforcement learning. 1984.

[80] Robert A Rescorla, Allan R Wagner, et al. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical conditioning II: Current research and theory*, 2:64–99, 1972.

[81] Andrew G Barto. 11 adaptive critics and the basal ganglia. *Models of information processing in the basal ganglia*, page 215, 1995.

[82] Wolfram Schultz, Peter Dayan, and P Read Montague. A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599, 1997.

[83] Andrew G Barto, Richard S Sutton, and Christopher JCH Watkins. Sequential decision problems and neural networks. In *Advances in neural information processing systems*, pages 686–693, 1990.

[84] M Waltz and K Fu. A heuristic approach to reinforcement learning control systems. *IEEE Transactions on Automatic Control*, 10(4):390–398, 1965.

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

[86] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[87] Roland Hafner and Martin Riedmiller. Neural reinforcement learning controllers for a real robot application. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2098–2103. IEEE, 2007.

[88] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 387–395, 2014.

[89] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[90] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[91] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.

[92] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR, abs/1509.06461*, 2015.

[93] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

[94] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[95] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, page 0278364913495721, 2013.

[96] Michael T Rosenstein and Andrew G Barto. Supervised learning combined with an actor-critic architecture. *Department of Computer Science, University of Massachusetts, Tech. Rep*, pages 02–41, 2002.

[97] MTRAG Barto. J. 4 supervised actor-critic reinforcement learning. *Handbook of learning and approximate dynamic programming*, 2:359, 2004.

[98] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[99] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[100] Donald O Hebb. The organization ofbehavior new york, 1949.

[101] BWAC Farley and W Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, 1954.

[102] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[103] Michael A Nielsen. Neural networks and deep learning, 2015.

[104] Hugh R Wilson and Jack D Cowan. A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue. *Kybernetik*, 13(2):55–80, 1973.

[105] David J Willshaw and Christoph Von Der Malsburg. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 194(1117):431–445, 1976.

[106] Shun-ichi Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological cybernetics*, 27(2):77–87, 1977.

[107] Thomas P Trappenberg, Michael C Dorris, Douglas P Munoz, and Raymond M Klein. A model of saccade initiation based on the competitive integration of exogenous and endogenous signals in the superior colliculus. *Journal of Cognitive Neuroscience*, 13(2):256–271, 2001.

[108] Wolfram Erlhagen and Estela Bicho. The dynamic neural field approach to cognitive robotics. *Journal of Neural Engineering*, 3(3):R36, 2006.

[109] Dietmar Heinke and Eirini Mavritsaki. *Computational modelling in behavioural neuroscience: closing the gap between neurophysiology and behaviour*, volume 2. Psychology Press, 2009.

[110] Jeffrey S Johnson, John P Spencer, Steven J Luck, and Gregor Schöner. A dynamic neural field model of visual working memory and change detection. *Psychological Science*, 20(5):568–577, 2009.

[111] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE, 2004.

[112] Michael Milford and Gordon Wyeth. Hippocampal models for simultaneous localisation and mapping on an autonomous robot. In *Proceedings of the Australasian Conference on Robotics and Automation, 2003.* Australian Robotics and Automation Association Inc, 2003.

[113] Kevin Gurney, Tony J Prescott, Jeffery R Wickens, and Peter Redgrave. Computational models of the basal ganglia: from robots to membranes. *Trends in neurosciences*, 27(8):453–459, 2004.

[114] Stephan KU Zibner, Christian Faubel, Ioannis Iossifidis, and Gregor Schoner. Dynamic neural fields as building blocks of a cortex-inspired architecture for robotic scene representation. *Autonomous Mental Development, IEEE Transactions on*, 3(1):74–91, 2011.

[115] Sophie Deneve, Jean-René Duhamel, and Alexandre Pouget. Optimal sensorimotor integration in recurrent cortical networks: a neural implementation of kalman filters. *The Journal of neuroscience*, 27(21):5744–5756, 2007.

[116] Gregor Schöner, Michael Dose, and Christoph Engels. Dynamics of behavior: Theory and applications for autonomous robot architectures. *Robotics and autonomous systems*, 16(2):213–245, 1995.

[117] Klaus Kopecz and Gregor Schöner. Saccadic motor planning by integrating visual information and pre-information on neural dynamic fields. *Biological cybernetics*, 73(1):49–60, 1995.

[118] Christoph Engels and Gregor Schöner. Dynamic fields endow behavior-based robots with representations. *Robotics and autonomous systems*, 14(1):55–77, 1995.

[119] Gregor Schöner, Klaus Kopecz, and Wolfram Erlhagen. The dynamic neural field theory of motor programming: Arm and eye movements. *Advances in Psychology*, 119:271–310, 1997.

[120] Randall D Beer. Dynamical approaches to cognitive science. *Trends in cognitive sciences*, 4(3):91–99, 2000.

[121] Michael Milford, Ruth Schulz, David Prasser, Gordon Wyeth, and Janet Wiles. Learning spatial concepts from ratslam representations. *Robotics and Autonomous Systems*, 55(5):403–410, 2007.

[122] Yulia Sandamirskaya and G Schoner. Dynamic field theory of sequential action: A model and its implementation on an embodied agent. In *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pages 133–138. IEEE, 2008.

[123] Yulia Sandamirskaya and G Schoner. Serial order in an acting system: a multidimensional dynamic neural fields implementation. In *Development and Learning (ICDL), 2010 IEEE 9th International Conference on*, pages 251–256. IEEE, 2010.

[124] Thomas Trappenberg. *Fundamentals of computational neuroscience.* Oxford University Press, 2009.

[125] JG Taylor. Neural 'bubble' dynamics in two dimensions: foundations. *Biological Cybernetics*, 80(6):393–409, 1999.

[126] Jason Satel, Farzaneh S Fard, Zhiguo Wang, and Thomas P Trappenberg. Simulating oculomotor inhibition of return with a two-dimensional dynamic neural field model of the superior colliculus. In *ICONIP 2014: 21st International Conference on Neural Information Processing*, pages 27–32, 2014.

[127] John S Brlow. Inertial navigation as a basis for animal navigation. *Journal of Theoretical Biology*, 6(1):76–117, 1964.

[128] Alexei Samsonovich and Bruce L McNaughton. Path integration and cognitive mapping in a continuous attractor neural network model. *The Journal of neuroscience*, 17(15):5900–5920, 1997.

[129] SM Stringer, TP Trappenberg, ET Rolls, and IETd Araujo. Self-organizing continuous attractor networks and path integration: one-dimensional models of head direction cells. *Network: Computation in Neural Systems*, 13(2):217–242, 2002.

[130] SM Stringer, ET Rolls, TP Trappenberg, and IET De Araujo. Selforganizing continuous attractor networks and path integration: two-dimensional models of place cells. *Network: Computation in Neural Systems*, 13(4):429–446, 2002.

[131] Simon M Stringer, Edmund T Rolls, Thomas P Trappenberg, and Ivan E Tavares de Araújo. Self-organizing continuous attractor networks and motor function. *Neural Networks*, 16(2):161–182, 2003.

[132] Bruce L McNaughton, Francesco P Battaglia, Ole Jensen, Edvard I Moser, and May-Britt Moser. Path integration and the neural basis of the 'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663–678, 2006.

[133] Simon M Stringer and Edmund T Rolls. Self-organizing path integration using a linked continuous attractor and competitive network: Path integration of head direction. *Network: Computation in Neural Systems*, 17(4):419–445, 2006.

[134] Daniel Walters, Simon Stringer, and Edmund Rolls. Path integration of head direction: updating a packet of neural activity at the correct speed using axonal conduction delays. *PloS one*, 8(3):e58330, 2013.

[135] Warren A Connors and Thomas Trappenberg. Improved path integration using a modified weight combination method. *Cognitive Computation*, pages 1–12, 2013.

[136] Michael I Jordan. Computational aspects of motor control and motor learning. *Handbook of perception and action*, 2:71–120, 1996.

[137] Michel Desmurget and Scott Grafton. Forward modeling allows feedback control for fast reaching movements. *Trends in cognitive sciences*, 4(11):423–431, 2000.

[138] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current opinion in neurobiology*, 9(6):718–727, 1999.

[139] Daniel M Wolpert, Zoubin Ghahramani, and Michael I Jordan. An internal model for sensorimotor integration. *Science*, 269(5232):1880, 1995.

[140] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[141] Mitsuo Kawato, Kazunori Furukawa, and R Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological cybernetics*, 57(3):169–185, 1987.

[142] Sebastian Raschka. *Python machine learning*. Packt Publishing Ltd, 2015.

[143] Ray J Dolan and Peter Dayan. Goals and habits in the brain. *Neuron*, 80(2):312–325, 2013.

[144] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[145] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[146] Nathaniel D Daw and John P ODoherty. Multiple systems for value learning. *Neuroeconomics: Decision Making, and the Brain,*, 2013.

[147] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[148] Peter Dayan and Kent C Berridge. Model-based and model-free pavlovian reward learning: revaluation, revision, and revelation. *Cognitive, Affective, & Behavioral Neuroscience*, 14(2):473–492, 2014.

[149] Vijay R Konda and John N Tsitsiklis. Onactor-critic algorithms. *SIAM journal on Control and Optimization*, 42(4):1143–1166, 2003.

[150] Yuji Takahashi, Geoffrey Schoenbaum, and Yael Niv. Silencing the critics: understanding the effects of cocaine sensitization on dorsolateral and ventral striatum in the context of an actor/critic model. *Frontiers in neuroscience*, 2:14, 2008.

[151] Roy A Wise. Dopamine, learning and motivation. *Nature reviews neuroscience*, 5(6):483, 2004.

[152] Richard J Beninger. The role of dopamine in locomotor activity and learning. *Brain Research Reviews*, 6(2):173–196, 1983.

[153] Kent C Berridge. The debate over dopamines role in reward: the case for incentive salience. *Psychopharmacology*, 191(3):391–431, 2007.

[154] Kent C Berridge and Terry E Robinson. What is the role of dopamine in reward: hedonic impact, reward learning, or incentive salience? *Brain research reviews*, 28(3):309–369, 1998.

[155] John W Kebabian, Donald B Calne, et al. Multiple receptors for dopamine. *Nature*, 277(5692):93–96, 1979.

[156] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008.

[157] Harm Vanseijen and Rich Sutton. A deeper look at planning as learning from replay. In *International conference on machine learning*, pages 2314–2322, 2015.

[158] Pawe Cichosz. Td (λ) learning without eligibility traces: a theoretical analysis. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(2):239–263, 1999.

[159] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

[160] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930.

[161] Michael SA Graziano, Dylan F Cooke, and Charlotte SR Taylor. Coding the location of the arm by sight. *Science*, 290(5497):1782–1786, 2000.

[162] Robert Desimone and John Duncan. Neural mechanisms of selective visual attention. *Annual review of neuroscience*, 18(1):193–222, 1995.

[163] Soeren Strauss and Dietmar Heinke. A robotics-based approach to modeling of choice reaching experiments on visual attention–appendix.

[164] DJ Weir, JF Stein, and RC Mialt. Cues and control strategies in visually guided tracking. *Journal of motor behavior*, 21(3):185–204, 1989.

[165] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice - a survey. *Automatica*, 25(3):335–348, 1989.

[166] Paul C Bressloff and Stephen Coombes. Neural 'bubble' dynamics revisited. *Cognitive computation*, 5(3):281–294, 2013.

[167] Kechen Zhang. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *The journal of neuroscience*, 16(6):2112–2126, 1996.

[168] Alexis Guanella and Paul FMJ Verschure. A model of grid cells based on a path integration mechanism. In *Artificial Neural Networks–ICANN 2006*, pages 740–749. Springer, 2006.

[169] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[170] DA Wardle. The time delay in human vision. *The Physics Teacher*, 36(7):442–444, 1998.

[171] Kristsana Seepanomwan, Daniele Caligiore, Gianluca Baldassarre, and Angelo Cangelosi. Modelling mental rotation in cognitive robots. *Adaptive Behavior*, 21(4):299–312, 2013.

[172] Jana M Iverson and Mary K Fagan. Infant vocal–motor coordination: precursor to the gesture–speech system? *Child development*, 75(4):1053–1066, 2004.

[173] Claes Von Hofsten. An action perspective on motor development. *Trends in cognitive sciences*, 8(6):266–272, 2004.

[174] Yiannis Demiris and Anthony Dearden. From motor babbling to hierarchical learning by imitation: a robot developmental pathway. 2005.

[175] Jana M Iverson, Amanda J Hall, Lindsay Nickel, and Robert H Wozniak. The relationship between reduplicated babble onset and laterality biases in infant rhythmic arm movements. *Brain and language*, 101(3):198–207, 2007.

[176] Daniele Caligiore, Tomassino Ferrauto, Domenico Parisi, Neri Accornero, Marco Capozza, and Gianluca Baldassarre. Using motor babbling and hebb rules for modeling the development of reaching with obstacles and grasping. In *International Conference on Cognitive Systems*, pages E1–8, 2008.

[177] Jason Satel, Thomas Trappenberg, and Raymond Klein. Motivational modulation of endogenous inputs to the superior colliculus. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 1, pages 262–267. IEEE, 2005.

[178] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[179] David Marr et al. Vision: A computational investigation into the human representation and processing of visual information, 1982.

[180] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

Appendices

Appendix A

Notices of Permission to Use Excerpts from Author's Publications

In this Thesis, large and small excerpts were taken verbatim from two of the author's own published papers [1, 4]. Moreover, for two other papers of the author that have been submitted and are under review, form of the student's contribution to the manuscript is signed and submitted to the graduate studies office.

Both Elsevier and Science direct (publishers of the Journal of Neural Networks) and IEEE (publisher of the proceedings of the International Joint Conference on Neural Networks) state in the documents reproduced on the pages linked below that use of the author's work in their own dissertation is allowed.

Please see the links for permissions:

https://www.elsevier.com/about/our-business/policies/copyright/permissions

https://www.elsevier.com/about/our-business/policies/copyright#Author-rights

https://s100.copyright.com/AppDispatchServlet

https://www.ieee.org/publications_standards/publications/rights/permissions_faq.pdf