Doctoral Thesis Proposal: Robotic transfer of motor skills from low fidelity domains

Juan Camilo Gamboa Higuera
School of Computer Science
McGill University
gamboa@cim.mcgill.ca

November 26, 2015

Abstract

The bootstrap scenario describes the situation where a software agent is placed in a robotic body and, without any prior information, learns how to use its sensors and actuators to perform tasks like servoing and waypoint following, building up new skills for more complex tasks. The ultimate goal of research on such problem is to produce an universal controller, that could be placed on any robotic body and learn how to use it, going from "pixels to policies". As such learning agent relies on gathering extensive experience, it requires large amounts of trials which can carry a high cost and risk; e.g. constant human supervision, degradation of the robot's components, potentially dangerous exploration.

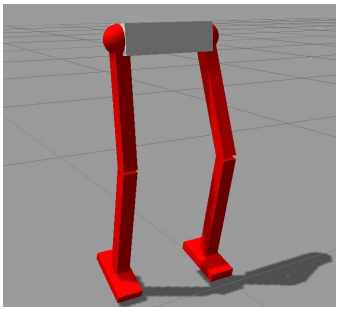
Rather than trying to solve the "robot that learns everything" scenario, we focus on the problem of a robot learning how to move. We assume that the agent can interpret its sensor data as information about its state and it knows its morphology. We further assume that it is given a model of how the world works, in the form of physically based simulators. Using this information, the robot must obtain control policies specifying the control commands that bring about a desired outcome in the environment.

The use of physically based simulators would allow the agent to learn a motor task without the cost and risk associated with real world trials. However, physically based simulators are low fidelity models of reality, so the agent must adapt its knowledge when applying it in the real world. The learning agent has to determine which models are good enough for learning a motor task, learn to perform the task using such models, then *transfer* the learned controller to the physical system. We provide a description of the methods we have used for learning swimming controllers from simulation and transferring such controllers to a real robot.

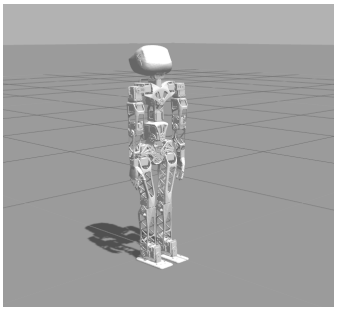
As an extension, we propose a system in which data from learning motor control tasks could be transferred between robots with different morphologies, when the descriptions of the morphologies are available to the learning agent. We envision a robotic learning system that is able to transfer motor skills between a source and a target domain addressing the following open questions: Can a robot learn a skill from low fidelity models, and directly use it in the target domain? How do we use new data to correct inaccurate models? Can we determine when using a particular model has a negative impact on performance? How should a robot pick which model to use for learning a task?

1 Introduction

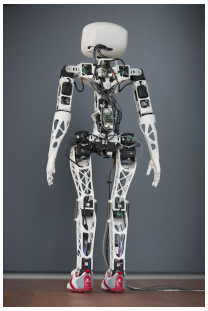
Successful robotic systems will need to adapt to the unforeseen. It is impossible to program a robot for every situation it will face in the future: the real world is dynamic, not fully observable, task requirements may change over time, and robot sensors and actuators degrade with repeated use. Similar to how humans are able to adapt to novel situations using a rather unreliable set of sensors and actuators, robots will have to "learn" from every situation they are presented with, either from sensing information, or from interacting with humans. In either case, learning requires collecting data from interactions to build models or mechanisms to predict the outcomes of future actions. The quality of these predictions generally depends on the amount of data available. But collecting data with a physical robot can be costly, or potentially



(a) Simplified biped simulation



(b) Humanoid simulation



(c) 3D printed humanoid

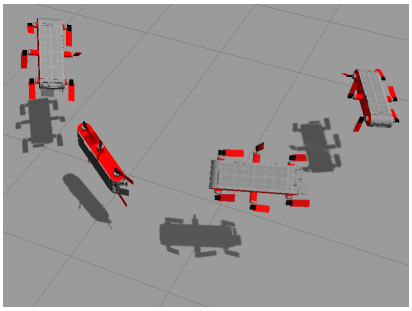
Figure 1: Is it possible to download a "walking" program to each of the three situations, making adjustments to the program while avoiding learning from scratch? How can we apply knowledge obtained from the biped simulation on the physical robot? (The 3D printed humanoid is the Poppy platform (Lapeyre et al., 2014))

dangerous. Thus robot learning needs to be **data efficient** in order to be practical. An effective robotic system will have to address these two conflicting requirements: collecting sufficient data and using data efficiently.

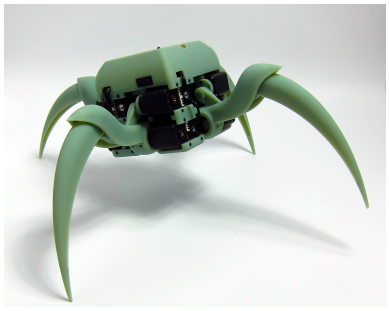
One way to be data efficient, is to share knowledge between robots, or across tasks. As an example consider the robotic systems shown on Figure 1. If a biped robot first learns how to balance in standing position, it should be able to use this knowledge to learn how to walk. And if we transform this biped into a humanoid robot, it should be able to take advantage of the similarities between these two morphologies to avoid learning from scratch. A similar point can be made about simulated or analytical models: if such models make reasonable approximations, can a robot learn to perform a task using such models, and adapt its knowledge to quickly execute the task on a physical realization of the robot? If the robot had access to a model that perfectly replicated the relevant bits of reality necessary to successfully execute the task, then the answer would be yes: a solution can be found using optimal control methods (Bertsekas, 1995; Jacobson and Mayne, 1970). However, such models are rare, with the most accurate models requiring expensive computations that make them impractical for mobile robotic applications in unstructured environments. In the case where no model is available, the Reinforcement Learning (RL) paradigm (Sutton and Barto, 1998) provides a framework for building models, or not using one at all, (Sutton et al., 1999a; Peters et al., 2010; Levine and Koltun, 2013), but such methods require a large amount of data for effective learning.

Model based reinforcement learning (MBRL) and transfer learning (TL) are techniques that attempt to tackle the data efficiency problem. In MBRL (Sutton, 1990; Atkeson and Santamaría, 1997), the robot builds a model from its interactions with the environment. Such model is used to simulate future interactions and to select appropriate actions for each situation accordingly; reducing the number of interactions with the real world as the model becomes more accurate. Recent work has shown the applicability of MBRL techniques to complex robotic systems (Deisenroth and Rasmussen, 2011; Meger et al., 2015; Moldovan et al., 2015), but the computation time remains too high for being applicable to physical mobile robots. Another body of recent work has shown how to combine MBRL and TL to reduce the number of interactions on a physical system (Cutler et al., 2014; Cutler and How, 2015; Ha and Yamane, 2015), but it has only been demonstrated for relatively simple robotic systems; e.g. balancing an inverted pendulum.

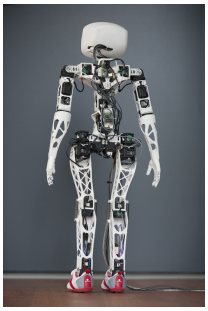
TL is the formalization of the problem of reusing data across tasks, situations, or robot morphologies, in order to speed up learning on a target domain (Taylor and Stone, 2009). With such goal, it is hard to separate the problem of transfer learning from that of learning in general. In our proposed research, the distinction is clearer: we attempt to transfer knowledge obtained from **low fidelity** models to a target physical robotic system. By low fidelity models we refer to any black box that receives as input any robot state and action, producing a prediction of the **change** in the state. Our goal is to produce a robotic system that learns in simulation (using low fidelity models) and only needs to make small adjustments when applying its knowledge to the real world. Similar to recent work in control theory (Censi, 2012), this can be seen



(a) Swimming Gaits on the AQUA underwater swimming platform



(b) Biomimetic Locomotion



(c) Humanoid Locomotion

Figure 2: Our research focuses on learning **motor skills** which can be used as building blocks for more complex behaviours. Our main target platform is the AQUA robot (Sattar et al., 2008), but we also envision deploying our algorithms for low cost 3D printable platforms, similar to the Aracna (Lohmann et al., 2012) or Poppy (Lapeyre et al., 2014) platforms.

as projecting a robotic system to an approximating or simplifying description of the system, learning a controller with such description, then using the inverse projection to obtain a controller for the original system. This inverse projection is not necessarily a bijection[1], thus additional adjustments need to be done before obtaining an appropriate controller. Note that we do not restrict ourselves to learned models: we want to use existing models as a way of reducing the real world data requirements. We base ourselves on the observation that existing rigid body simulators, while grounded on unrealistic assumptions, they make reasonable (and plausible) predictions of the behaviour of physical bodies under the effect of gravity, frictional contacts, and joint constraints (Baraff, 1997), or bodies immersed in water (Si et al., 2014).

Since we are providing low fidelity models of motion dynamics for a robotic agent to test, our initial focus will be on learning motor behaviours and skills, as opposed to the "robot that learns everything" scenario (Kuipers et al., 2006; Censi, 2012; Levine et al., 2015a). These behaviours and skills can be defined as tasks where the goal is to decide upon the appropriate motor control commands to bring a robot to a desired state, or to cause a desired outcome in the environment. Example tasks, illustrated in Figure 2, include performing swimming manoeuvres with a six legged underwater robot, or walking with a biped robot. In such cases, the motor control commands would be the torques applied to each joint. The target of learning is to find a mapping from states, or observations, to control commands which produce a desired behaviour on the robot, i.e. **control policy**. Our initial objective is to transfer control policies learned on low fidelity models, or simulators, to a physical robot. Given the generality of the optimal control, RL, and TL frameworks, we use a similar framework, based on Markov Decision Processes (MDP), to formulate our problem.

Although a multitude of methods have been proposed for transferring control policies, successful transfer of motor skills between robots remains an open challenge – even between different realizations of the same robot. As mentioned before, our proposed approach consists of using a simulator, where general physical principles are encoded, to minimize the amount of data that is required by the robot in the real world. This introduces two questions that we aim to address with our work:

1. **Which methods can we use to adjust previously learned control policies to a new target robotic system?**
2. **Given a target system and an ensemble of simulators, how should we select the appropriate simulator, or an appropriate combination of such simulators, to learn a control policy for the real robot?**

[1]The low fidelity model might be missing degrees of freedom, or perceived state dimensions. When using the inverse projection, there will be state or control dimensions that need to be "filled in".

Answering these questions would not only allow us to synthesize controllers for mobile robotic systems using data efficiently, it would also minimize the cost in terms of human supervision and robot component degradation. We envision a learning system which consists of simulators running **in the cloud** (Censi, 2014), that would operate as follows (illustrated in Figure 3): 1) When a robot is required to perform a new task, it can send the task specification to the server, along with some of its recorded experience data. 2) The experience data is used to select an appropriate model, or combination of models, that are similar to the robot for the specified task. This data can also be used to update the available models if needed. 3) The task specification and selected models are used to select one or more of the previously trained policies. 4) Given the candidate policies, the selected models, and the robot's experience data, a new policy is synthesized. 5) The robot evaluates the policy obtained from the server, from where it has some options: it can continue to adjust that policy by itself, or it can ask the server for a new adjusted policy. If we are able to determine which models are better suited for a particular robotic platform, we can also use them for planning (LaValle and Kuffner, 2001), with the learned control policies used as temporally extended motor commands, or motor primitive for more complex tasks (Schaal, 2006), or options (Sutton et al., 1999b) In the following sections, we provide some background and related work, describe the methods we have used so far, and propose a series of questions to be addressed by our research. Finally, we propose a research plan to address these questions.

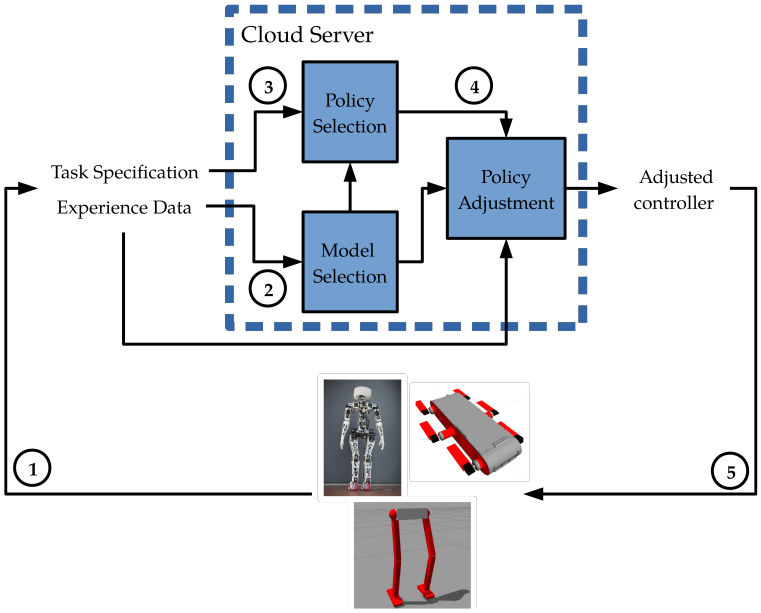


Figure 3: Our vision of a cloud computing server for transferring motor policies, which allows robot clients to download controllers which (in the best case) work in one shot or (in the worst case) require updating the low fidelity models, or creating a new ones.

2 Related work

The Robot Learning Problem

In the fields of artificial intelligence and robotics, the problem of **learning** refers to the design of software for autonomous agents in which the final design cannot be completely specified by the designers. Rather, a part of it can only be finalized by interactions of the agents with the environment. As illustrated on Figure 4, we refer to a **robot** to the conjunction of a set of sensors and a set of actuators that allow for interaction with the physical world. An **agent** is the software that decides how to use the robots actuators, based on information obtained from the robots sensors. In the following pages we use robot and agent interchangeably, except in

occasions where we want to differentiate between the hardware and the software.

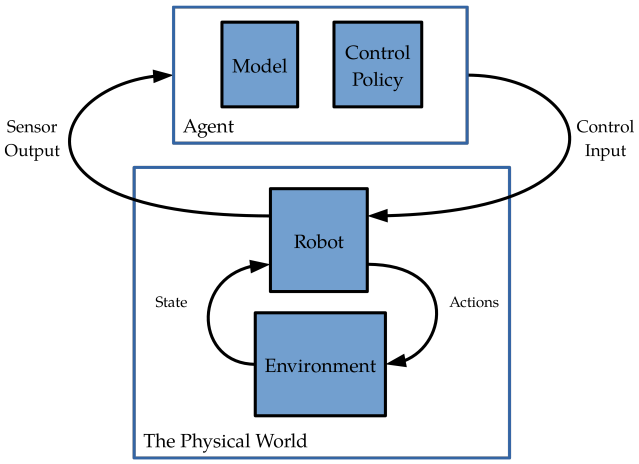


Figure 4: A robot agent interacts with an environment and uses it sensors to perceive the state of the environment. It can use these observations to update any internal model it may have of the world, and to keep track of progress in a task. It may receive a reward/cost signal from the environment, or compute it internally, which it uses to evaluate its performance.

The **bootstrap scenario** (Kuipers et al., 2006; Censi, 2012) provides a general description of the robot learning problem. In the bootstrap scenario, a robot starts with no prior information about the world, its morphology, or an interpretation of its sensors. The robot is given input from an uninterpreted stream of sensor data, an output stream for its actuators (also uninterpreted), and the notion of causality; i.e. the robot knows that the world can be described as a dynamical system, but it does not have an exact description of such system. The goal of bootstrapping is to find a procedure for a robot that learns a sequence of transformations that go from the uninterpreted sensor streams to control policies; i.e. an agent that can learn to use any set of sensors and actuators.

Some theoretical results have been shown for low dimensional state and action spaces, with most of the computational effort spent in interpreting the sensor data (see the recent work by Censi (2012)). An experimental end-to-end robot learning system has been demonstrated in recent work (Levine et al., 2015a,b), where an agent could learn to perform simple manipulation tasks with a 7-DOF[2] articulated arm, from visual feedback only. While these results are impressive, there is still a long way to go to apply such methods on more complex morphologies, and more complex tasks. Since these approaches involve an agent that needs to "learn everything", the amount of data (interactions with the environment) needed to produce useful behaviours is generally very large, except for very constrained tasks.

Using the bootstrap scenario as inspiration, we concentrate instead on the following problem: if an agent knows the morphology and an interpretation of the sensing data from every robotic body it will be deployed in, determine a procedure to *transfer* controllers from one robot body to another. This problem subsumes that of learning from a low fidelity models, or set of such models, as in both cases the robot interacts with different worlds on which the robot has knowledge about its sensor and actuators, and we wish to transfer control policies between both (see Figure 5). To reiterate, we are interested in learning from simulators as a way to reduce the amount of data needed from the physical robotic systems. To make the problem more tractable, we put aside the problem of learning and transferring perception, and focus on transfer learning of motor control.

Motor Control Learning in Robotics

The goal of motor control learning in robotics is to find control commands that produce a desired change in the environment. The theory of Markov Decision Processes (MDP) provides an elegant way to formalize this

[2]Degrees of freedom.

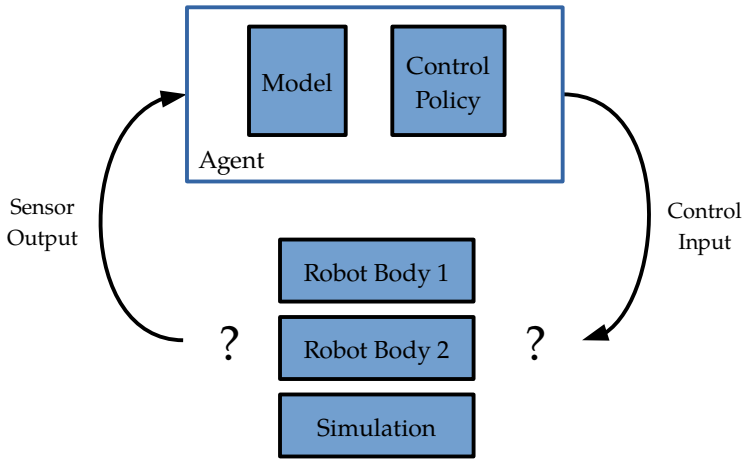


Figure 5: If we find a mechanism for transferring control policies between different robot bodies, the same mechanism should work for transferring between a between a simulator and the robot bodies.

problem, as has been done in optimal control theory (Todorov and Jordan, 2002; Tassa, 2011; Haith and Krakauer, 2013) and reinforcement learning (Schaal and Atkeson, 2010b; Peters and Schaal, 2008). Under such formalism, an agent must optimize a control policy from cost functions that penalize undesired states (alternatively, reward functions that attain a high value at desired states).

We consider discrete time problems, as our controllers and models are implemented on a digital computer. At each time step t, the agent observes the state of the environment $\mathbf{x}_t \in \mathcal{X}$, and chooses a control command $u_t \in \mathcal{U}$ according to a control policy $\pi(\mathbf{u}_t \mid \mathbf{x}_t)$, producing a state transition drawn from the state transition dynamics $\mathcal{T}$ (usually expressed as a distribution with density $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{u}_t)$). The desired behaviour in a particular task is specified by the instantaneous cost function $c(\mathbf{x}_t, \mathbf{u}_t)$. The goal is to obtain an *optimal policy*, which minimizes the expected cumulative cost

$$V_\pi(\mathbf{x}_t) = \mathbb{E}_\tau \left\{ \sum_{k=t}^{H} c(\mathbf{x}_k, \mathbf{u}_k) \right\} \tag{1}$$

where $\mathbf{x}_t$ is the initial state, H is the robot's planning horizon, and τ is the sequence of states $\{\mathbf{x}_0, ..., \mathbf{x}_H\}$ obtained by following policy π. The function $V_\pi(\mathbf{x}_t)$ is also called the *value* of a policy, or of the starting state $\mathbf{x}_t$ under the policy π. Here we focus on finite horizon tasks, but extensions to other formulations is also possible (Sutton et al., 1999a; Baxter and Bartlett, 2001; Erez et al., 2012).

In continuous state spaces, approximate solutions to this problem can be found either by methods of trajectory optimization (Abbeel et al., 2007; Pan and Theodorou, 2014) or policy search methods (Deisenroth and Rasmussen, 2011; Levine and Koltun, 2013; Peters et al., 2005). We are more interested in the latter, as we aim to obtain *controllers* that generate the control commands as a function of state, rather than sequences of control commands. In general, given that the dynamics of a robotic system can be hard to model, two main classes of approaches have been suggested in the literature. The first one is to *learn a model of the dynamics* $\mathcal{T}$ using function approximation methods and then compute a policy based on the learned model; this is the MBRL approach. In MBRL, the value of a policy is calculated via "rollouts"[3] on the learnt model. The second one aims to find improvements to the policy directly from interactions with the environment, called model free methods. In this sort of methods, a policy is assumed to be parametrized by a vector θ; e.g. a neural network (Levine and Koltun, 2013) or a weighted sum of radial basis functions. Although successful, model free methods require a considerably larger experience dataset than model based methods.

Since our goal is to reduce the number of real world trials extensively, we focus on MBRL methods. We have successful one such method, PILCO (Probabilistic Inference and Learning in Control) Deisenroth and Rasmussen (2011), to obtain swimming controllers for a variety of manoeuvres on the AQUA class of underwater legged robots (Meger et al., 2015). While this method allowed the robot to learn swimming controllers

[3]Using the model as a simulator and the known immediate cost to compute $V_\pi(\mathbf{x}_0)$

in a small number of trials (5 to 10), the cost in terms of robot degradation and human supervision is still very high. This is because building and evaluating models is computationally expensive: for every 15 seconds of real robot experience it takes between 5 and 10 minutes of controller optimization. We managed to make a better use of the deployment time by learning multiple tasks in parallel, using the idle time between learning iterations to interleave the multiple learning tasks. Still, this meant that any single task would only be learnt successfully after around 5 hours of deployment time.

We see two solutions to this problem: we can build an automated system for setting up and performing experiments with a robot, or we try to do most of the learning in simulation. Since the first option can be prohibitively expensive, we aim for the second one. This is in part inspired by the learning systems that have been demonstrated in the computer animation community, that produce feasible kinematics for a variety of simulated creatures.

Artificial Life and Learning from Simulators

The problem of motor control learning has also been studied in the physics based animation under the umbrella of Artificial Life. One of the first demonstrations of emergent motor controllers was demonstrated by Sims (1994), where a set of generative rules for morphologies and controllers, in conjunction with rigid body simulation, were used to obtain swimming, walking and hopping creatures. Similar work by Terzopoulos et al. (1994) described a procedure for an artificial ecosystem of various species of fish, which proved realistic enough to simulate the actual behaviour of fish in the ocean.

More recent work has shown how the gaits of swimming creatures observed in nature can emerge from black box optimization using a simulation of the creatures bodies as articulated rigid bodies immersed in a fluid (Tan et al., 2011). However such impressive results came at a great computational expense: simulations of 1 to 10 seconds took from several hours to two days. Similar work has been proposed for optimizing central pattern generators for swimming simulation of humans Si et al. (2014), again at a very high computational cost. The accuracy of these types of simulations is dependent on the computational resources available. Since researches do not have access to infinite computations their models are only approximate. A question that arises often in the computer animation community is: How aggressive can approximations be in order to speed up computations, while still producing plausible physical behaviour?.

Some recently proposed methods have shown that it is not necessary to simulate the fluid around rigid bodies to obtain plausible motions of submerged bodies (Weissmann and Pinkall, 2012; Xie and Miyata, 2014). In fact, that has been the approach used in naval architecture, aeronautics and traditional control: produce models that are simple to compute and are valid in restricted regimes. The approach we have taken to simulate the AQUA robot underwater is based on these ideas, as explained in section 3.1. By tuning a small number of parameters, one can replicate in simulation the trajectories followed by physical objects in the world. Thus physically based animation techniques may allow us to find the kinematic trajectories that demonstrate successful executions of a given task, and use these trajectories to train the desired control policies.

Using kinematic trajectories generated from physically based simulation is a promising idea, since recent work on motor control learning has shown that it is possible to use trajectory information to guide the optimization of a general class of parametrized control policies; a method called Guided Policy Search (Levine and Koltun, 2013). Thus, we could first have an agent learn to perform a task in a general physical simulator, and then use simulated trajectories to obtain a policy on the physical robot. As we describe in section 3.3, this can be done by supervised learning. Unlike Guide Policy Search, we could avoid generating the trajectories on the physical robot, reducing the number of real world trials needed (hopefully to just one).

Learning robotic control from simulators is certainly not a new idea. But it remains an unsolved problem, as evidenced by the large body of literature on the subject in recent years (Abbeel et al., 2006; Kober and Peters, 2012; Lee et al., 2013; Cutler et al., 2014; Cutler and How, 2015). As mentioned by Sims (1994) and confirmed by Todorov et al. (2012), if the simulation allows cheating, any optimization algorithm will exploit it, producing physically unrealistic trajectories. In our research, we attempt to use the insights from *transfer learning* techniques from a souce simulator to a target robot, in combination with a "inverse transfer" in which we correct for modelling inaccuracies, pruning out unrealistic behaviours produced in the source

domain.

Transfer Learning

In our recent work (Meger et al., 2015), we experimented with learning swimming behaviours for a six legged robot (see Figure 2a) using a simplistic simulated model (Georgiades, 2005), and attempted to use the learned controllers and predictions from the simulator to speed up learning in the real world. While we were able to learn control policies successfully both in simulation and on the real robot, combining data from both domains was detrimental in terms of data efficiency. Our approach was naïve since we combined data from two different "worlds" and assumed that they were produced by the same environmental variables. Our approach suffered from **model bias**, leading to suboptimal controllers in the initial real world trials,; thus requiring large amounts of data to compensate for our incorrect prior. These results were expected, but slightly puzzling, as the behaviours learned in simulation qualitatively resembled reality. Our simulator did not capture hydrodynamics with high fidelity. But according to expert users, it did replicate the behaviours found in the real world when trying to control it via teleoperation.

As mentioned in the previous sections, if we assume that the kinematics of the robotic system are described with some degree of accuracy, we could in theory attempt to transfer controllers from a simulator to a real robot. One underlying assumption that we make is that we have a description of the morphology of the robot, and that this morphology does not change between the simulation and the real robot body. So it makes sense to think of transferring trajectories from one domain to the other. What happens if our assumptions break? These questions have been studied in the area of Transfer Learning for Reinforcement Learning tasks.

TL techniques aim to use knowledge or experience data from one task domain to facilitate learning with limited data on a different task, feature space, or distribution. The idea of TL has been studied for supervised and unsupervised learning (Pan and Yang, 2010), and for RL tasks (Taylor and Stone, 2009). The idea is that previous computations should be re-used whenever possible in order to speed up optimization on a target task. This is related to the concept of reward shaping where the difficulty of a task is changed from easier to harder tasks (Ng, 2003), or the concept of analogical reasoning in general artificial intelligence. Transfer learning methods have also been proposed for cases where the state and action spaces change across domains (Taylor and Stone, 2007); e.g. between the benchmark inverted pendulum, cart-pole, mountain car tasks and simulated quadrotor tasks (Ammar et al., 2012, 2015b).

Although, there are many different transfer learning approaches, the pipeline of transfer learning is usually the following:

1. The agent learns to perform a task in a source domain, described by $(\mathcal{X}^{(S)}, \mathcal{U}^{(S)}, \mathcal{T}^{(S)}, c^{(S)})$
2. It obtains mappings from elements of $(\mathcal{X}^{(S)}, \mathcal{U}^{(S)}, \mathcal{T}^{(S)}, c^{(S)})$ to the target domain $(\mathcal{X}^{(T)}, \mathcal{U}^{(T)}, \mathcal{T}^{(T)}, c^{(T)})$
3. It uses such mappings to transfer knowledge in the form of
   - experience sample instances $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1}, c(\mathbf{x}_t))$
   - Approximations to the state value function $V_\pi(\mathbf{x}_t)$, or state action value $Q_\pi(\mathbf{x}_t, \mathbf{u}_t)$
   - state transition dynamics $\mathcal{T}$

   from the source to the target domain.
4. The agent continues learning in the target domain.

When transferring knowledge results in an improvement in the target we say that there is **positive transfer**. When the opposite occurs, we call that situation **negative transfer**. In general, positive and negative transfer is measured in terms of sample complexity: the number of trials, or samples, needed to achieve a threshold performance. One of the drawbacks of the existing TL techniques is that there are no guarantees on when negative transfer will happen; thus every TL method has the potential of resulting in negative transfer (Taylor and Stone, 2009).

More related to our work, Cutler et al. (2014) propose to use a chain of domains, ordered from lowest fidelity (a simulator that does not model any dynamics) to highest fidelity (running experiments on the

actual robot). In this case, the authors address the problem of inaccurate modelling by allowing the learning agent to update the transition dynamics of a source domain, $\mathcal{T}^{(S)}$ with samples of a target domain, $(\mathbf{x}_t^{(T)}, \mathbf{u}_t^{(T)}, \mathbf{x}_{t+1}^{(T)}, c(\mathbf{x}_t^{(T)}))$. While promising, this method was only shown to work in discrete state and action spaces. Unfortunately such methods are not directly applicable to our target domain; i.e. mobile robotic platforms with a high number of dimensions (12 DOFs in the AQUA robots). In more recent work (Cutler and How, 2015) the same authors attempt to use PILCO to transfer policies and experience data from a simulator to a physical realization of the robot (an inverted pendulum) by first learning a dynamics model $\hat{\mathcal{T}}^{(S)}$ in the simulated domain, and using it as a prior for the target domain. PILCO learns dynamics models by fitting a Gaussian Process regression model to the experience data, so using the $\hat{\mathcal{T}}^{(S)}$ as a Bayesian prior for $\hat{\mathcal{T}}^{(T)}$ is relatively straightforward.

The question we may ask is, why fit a model to an already available model (i.e., the true dynamics the simulator $\mathcal{T}^{(S)}$)? The authors claim that this is done in order to correctly integrate the uncertainty from the simulated model into the predictions made by the model learned from real world data. As more data from the real world becomes available the prior becomes less relevant. A drawback of this approach is that as the robotic system becomes more complex, with more local minima when following the policy gradient, more data will be needed to form an informative prior model. In such case, the learned prior model $\hat{\mathcal{T}}$ could become more expensive to evaluate than any simple simulator. As with previously described transfer learning methods, this method provides no guarantees on the performance in the target domain.

3 Proposal Topic I: A Robotic System that Learns from Simulators

In this section we describe our approach for learning motor controllers, and for simulating articulated robots underwater. Even though we describe an approach only for the underwater domain, it is general enough that it should work with other types of robots.

3.1 Fast and Plausible Physically Based Models

We express the dynamics of a robotic system as the differential equation

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \psi) \tag{2}$$

where ψ is a vector of parameters for the dynamics model. In simulation, we consider the state to contain the robot's position, orientation and linear and angular velocities, although this information might not be available to the real robot. We are currently performing our simulations using the Open Dynamics Engine (ODE), with a few modifications. In ODE, an articulated robot is represented in maximal coordinates; each link in the robot's body is modeled as a rigid body with 6 DOFs, and the body is held together through internal forces that account for joint constraints. For each rigid body, we can express the dynamics of its motions as

$$\begin{bmatrix} \mathbf{M} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{l}} \end{bmatrix} = \mathbf{F}_{ext} + \mathbf{F}_{int} \tag{3}$$

where $\mathbf{v}$ and $\mathbf{w}$ are the linear and angular velocities of the rigid body, $\mathbf{p}$ and $\mathbf{l}$ are the linear and angular momentum, $\mathbf{M}$ is the mass matrix, $\mathbf{J}$ is the inertia tensor, $\mathbf{F}_{ext}$ represents the external forces such as gravity, or buoyancy and drag in the underwater domain, and $\mathbf{F}_{int}$ represents internal forces such as the ones enforced by joints constraints and contacts. The forces required to enforce a joint constraint between two bodies i and j from the following velocity constraint

$$\mathbf{J_i} \begin{bmatrix} \mathbf{v}_i \\ \omega_i \end{bmatrix} - \mathbf{J_j} \begin{bmatrix} \mathbf{v}_j \\ \omega_j \end{bmatrix} = 0 \tag{4}$$

where J_i is a Jacobian matrix with one row for each constrained degree of freedom; e.g. for a hinge joint (one DOF) $\mathbf{J}_i$ has 5 rows. The constraint aims to keep the relative velocities of the two bodies close to 0. For walking robots, we also need to model contact constraints between the robot's body and the ground. In ODE this is done using the friction pyramid model (Baraff, 1994). At every point of contact between two bodies, or a body with the ground, a contact force must be computed, which we can split into normal and tangential components $\mathbf{F}_c = F_\perp \mathbf{n} + F_{\parallel,1}\mathbf{t}_1 + F_{\parallel,2}\mathbf{t}_2$. The contact force must satisfy the following linear complementarity conditions

$$v_\perp \geq 0, \qquad F_\perp \geq 0, \qquad F_\perp v_\perp = 0 \tag{5}$$

where the first constraint states that the only allowed velocities are the ones where there is no interpenetration, the second constraint states that the contact force must be repulsive along the normal direction of the contact, and the third constraint states that the contact force should only be applied if there is actual contact (i.e. when $\mathbf{v}_\perp = 0$). For the perpendicular forces we have

$$|v_{\parallel,i}| < 0, \qquad F_{\parallel,i} = \mu F_\perp |v_{\parallel,i}| > 0, \qquad F_{\parallel,i} = -\mu F_\perp |v_{\parallel,i}| = 0, \qquad |F_{\parallel,i}| \leq \mu F_\perp \tag{6}$$

The first two conditions model when there is slip in the contact: the friction force is proportional to the normal force and acts in the direction opposite to the slip motion. The third condition states what happens when there is no slip: the friction force must lie within the friction cone (otherwise, there would be slip). Using the dynamics in Equation 3, along with the constraints in Equations 4, 5 and 6, we can solve for the joint and contact forces using Dantzig's algorithm (Baraff, 1994). Using a numerical integration scheme, we can use Equation 3 to simulate articulated robots in vacuum. We note that this is not the only method for solving the problem of contacts with multiple articulated bodies, and that this remains an active area of research in robotics (Drumwright et al., 2012; Todorov et al., 2012; Drumwright, 2015).

To simulate underwater robots like the AQUA platform we also need to include hydrodynamic effects. So we modify Equation 3 as

$$\left(\begin{bmatrix} \mathbf{M} & 0 \\ 0 & \mathbf{I} \end{bmatrix} + \mathbf{K}_F \right) \begin{bmatrix} \mathbf{v} \\ \omega \end{bmatrix} = \mathbf{F}_g + \mathbf{F}_b + \mathbf{D} \begin{bmatrix} \mathbf{v} \\ \omega \end{bmatrix} + \mathbf{F}_{int} \tag{7}$$

Here $\mathbf{K}_F$ and $\mathbf{D}$ are the added mass and drag tensors, which depend only on the geometry of the rigid body, and F_g, F_b are the forces due to gravity an buoyancy acting on the center of mass of the rigid body. For the gait learning work using the AQUA robot (Meger et al., 2015), we used the approach of modelling the robot's body and its paddles as boxes to obtain $\mathbf{K}_F$ and $\mathbf{D}$, which resulted in a model with 15 parameters per rigid body to account for hydrodynamic effects (Georgiades, 2005). We are currently integrating more recent work on underwater rigid body simulation (Weissmann and Pinkall, 2012) to reduce the number of parameters to 3 (the mass of the robot, the density of the fluid, and a scaling factor that accounts for some unmodelled quantities), but requiring a CAD model of the robot to precompute quantities related to the hydrodynamic effects. For the majority of robotic systems in existence there are CAD models available, so obtaining one would not be a problem.

We have successfully integrated our physics based underwater simulation to the Gazebo robot simulator (Koenig and Howard, 2004), achieving simulations that run 10 times faster than real time.The proposed model is fast as it does not require modelling the fluid surrounding the robot. This benefit comes at a price: since we are not modelling turbulence in the fluid caused by the robot's motion, a swimming robot like AQUA, or an underwater vehicle using rotating propellers, will not generate thrust. In our prior work, we used a simple model that computes thrust as a function of periodic motion of a rigid body. An alternative that we propose is to use the data from real world trials to account for the lack a of thrust forces:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \psi) = f_{sim}(\mathbf{x}, \mathbf{u}, \psi_{sim}) + f_{corr}(\mathbf{x}, \mathbf{u}, \psi_{corr}) \tag{8}$$

In this case we need to careful to not incur in "double dipping", data used to obtain the parameters correction model ψ_{corr} cannot be used to estimate the parameters of ψ_{sim}., unless we jointly estimate both vectors. We note that the simulated model is linear in the parameter vector ψ, except for the friction coefficient μ.

Coming back to the bootstrap scenario, this might seem like giving a robotic agent a considerable amount of prior information. But we note that what the agent has at its disposition is a world in which in sensor information is interpreted, and the ability to reset to an initial position. Besides, the point that we want to make is that giving an approximate model is better than giving it nothing, in the context of motor control learning. Now the problem of using these physically based models is that, as has been pointed out in the literature Sims (1994); Todorov et al. (2012), if the physics engine allows any shortcut, it will be exploited by the optimization algorithm. Likewise, if the optimal behaviour in the real world does not produce an optimal behaviour in the simulator, learning in the prior model will result negative transfer.

3.2 Improving Inaccurate Models

The usual purpose of simulation in robotics is to test the validity of perception and control algorithms. In this proposed research, we take the view that a robot can transfer knowledge from simulation to a real robot, so as to offload learning to a server and minimize the amount of real world trials in learning a task. Even though authors in the field have expressed that the only useful simulator is one that is accurate (Schaal and Atkeson, 2010a; Kober and Peters, 2012) or trained from the robot's experience, we hypothesize that using trajectory information (ignoring the control commands) will allow us to transfer policies from simulation.

However, this relies on the assumption that the trajectories produced by the simulator are feasible in the real world. When this does not occur, we need to update the simulator parameters to correct for modeled dynamics, and fit a correction term to account for unmodelled dynamics; this is the reason why we express the dynamical models as Equation 8. To do this we suggest the following procedure. First we use a numerical integration scheme to transform Equation 8 into discrete time dynamics. For example

$$\mathbf{x}_{t+1} = F(\mathbf{x}_t, \mathbf{u}_t, \psi) = x_t + hf(\mathbf{x}_t, \mathbf{u}_t, \psi_{sim}) \tag{9}$$

where $h > 0$ is the simulation time step. Every time we run a policy π_k in the real world, we obtain a dataset $\mathcal{D}^{(T,\pi_k)}$ consisting of state-action-state triples $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$ for $t \in [0, ..., H-1]$. If the policy follows

a trajectory close to the one predicted in simulation we are done, otherwise we need to update the simulator. We follow the approach of Nguyen-Tuong and Peters (2010) to estimate F with Gaussian Process regression, using the simulated dynamics as a mean prior. After this is done, a new control policy π_{k+1} is obtained by using the new simulated model and we repeat the process.

The previous procedure is just the textbook System Identification procedure Ljung (1998). Recent work by Ross and Bagnell (2012) proposed a method for selecting the appropriate dataset to learn the parameters of a model. It consists of selecting a combination of state-action samples $(\mathbf{x}, \mathbf{u})$ from $\mathcal{D}^{(t,\pi_k)}$ and some exploration strategy ν (e.g. expert demonstrations), evaluating the transitions using the real world or a generative model to obtain state-action-state triples, and using the resulting dataset to fit the model. We aim to investigate this procedure further in our research, extending it to multi-step predictions.

Ideally, we should fit the parameters of f_{sim} and f_{corr} minimizing the long term multi-step prediction error, as described in the literature for estimating helicopter dynamics (Abbeel et al., 2005; Punjani and Abbeel, 2015). The reason for using a multi-step objective is that the real world dynamics, particularly in the underwater domain, depend on unobservable quantities like the state of the fluid surrounding the environment.

Our procedure is done under the assumption that we have access to the internals of a simulator. In the case we don't, we propose two options. The first one, similar to the work by Abbeel et al. (2006), is to just fit the bias term f_{corr}. This amounts to using the simulator as a mean prior for estimating the model (Ha and Yamane, 2015; Cutler and How, 2015). As mentioned before, this approach would still make bad predictions in regions of the state space not visited by the real robot. The second option is to fit a function that maps predictions from the simulator f_{sim} to predictions from the real world. More specifically, we would like to obtain a function G such that

$$\mathbf{x}_{t+1}^{(T)} = G(F(\mathbf{x}_t, \mathbf{u}_t, \psi)) = G(x_t + hf(\mathbf{x}_t, \mathbf{u}_t, \psi_{sim})) \tag{10}$$

Note that the second option is a generalization of the first, but may include other effects like affine transformation of the state or the state derivatives, along with other types of non linear transformations. Such approach also remains to be investigated further. We also recently learned about the work of Konidaris and Doshi-Velez (2014), in which the authors describe a similar class of problems under the paradigm of hidden parameter Markov Decision Processes (HIP-MDP). In HIP-MDP, a set of MDP tasks is related by a set of latent parameters. In our case, such latent parameters could correspond to ψ_{sim}, and idea that we are considering for our future directions.

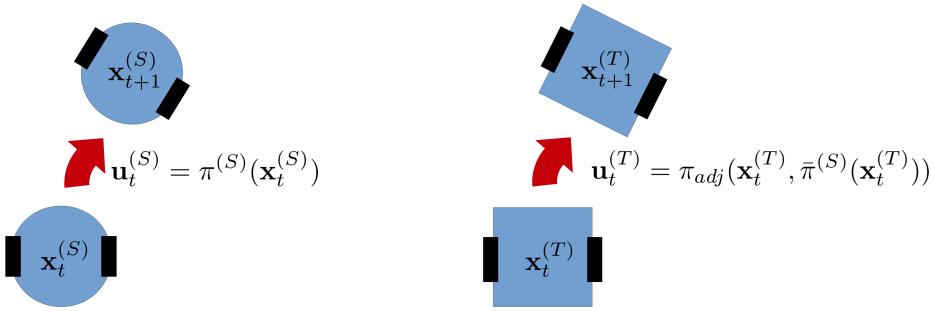
3.3 Policy Transfer

In this section we describe our current approach for transferring policies between a source simulator and a target domain, along with some preliminary results on the benchmark cart-pole task. The methods for learning control policies that we have implemented so far include Probabilistic Inference and Learning in Control (PILCO) (Deisenroth and Rasmussen, 2011) and Probabilistic Differential Dynamic Programming (PDDP) (Pan and Theodorou, 2014). Our goal is to determine if it is possible to get good performance **in the first trial** of a task on a real robot, by transferring policies learned from simulators. Our approach here is to compute a *policy adjustment* function, that maps policies from the target to the source domain, illustrated in Figure 6.

Using the simulated environment described in Section 3, we learn a control policy for a particular locomotion task, producing a policy $\pi^{(S)}$. We assume that the kinematic trajectories of the simulator are feasible in the real world, ignoring controls and forces. Formally, we assume that if the policy produces a trajectory, or set of trajectories, of the form $\tau^{(S)} = \{\mathbf{x}_0^{(S)}, ..., \mathbf{x}_H^{(S)}\}$, then for every $t \in [0, ..., H-1]$ there exists a control command $\mathbf{u}_t^{(T)}$ and pair of states $\mathbf{x}_t^{(T)}, \mathbf{x}_{(t+1)}$ in the target environment such that

$$\mathbf{x}_t^{(S)} \approx \mathbf{x}_t^{(T)}, \mathbf{x}_{t+1}^{(S)} \approx \mathbf{x}_{t+1}^{(T)} \implies p(\mathbf{x}_{t+1}^{(T)} \mid \mathbf{x}_t^{(T)}, \mathbf{u}_t^{(T)}) \gg 0, \text{ according to } \mathcal{T}^{(T)} \tag{11}$$

We implement the notion of state similarity of the first condition via a simple euclidean distance metric. However, there are several situation in which this metric would be too restrictive. For example, in a mobile



(a) Behaviour in the source domain

(b) Desired behavior in the target domain

Figure 6: The goal of the policy adjustment π_{adj} is to replicate trajectories from the source domain in the target domain. We do this by generating a dataset $(\mathbf{x}_t^{(S)}, \mathbf{u}_t^{(S)}, \mathbf{u}_*^{(T)})$, where $\mathbf{u}_*^{(T)} = g^{(T)}(\mathbf{x}_t^{(S)}, \mathbf{x}_{t+1}^{(S)})$, and training π_{adj} by supervised learning; e.g. GP regression or locally weighted linear regression

robotic system, the change in state produced by a particular control action may be invariant to translations and rotations. A more appropriate metric would measure state similarity by looking at the relative state transitions. One has to be careful with this approach because some relative state transitions that look similar might be realized by very different control commands. One possible approach would be to compute an approximate bisimulation metric (Ferns et al., 2011), or to obtain a metric via unsupervised learning (Ammar et al., 2014b).

We also assume that it is possible to estimate the inverse dynamics of the robot system as a function $g^{(T)} : \mathcal{X}^{(T)} \times \mathcal{X}^{(T)} \rightarrow \mathcal{U}$; i.e. a function that takes as input two states and outputs the action that produces the transition between them. Since we will only query information about states that are "close" in time and in the state space, we may safely assume that $g^{(T)}$ exists. Given these assumptions, we propose the following scheme for transferring a policy. For any given motor control task that we want to learn:

1. Execute a learning algorithm in the source domain $(\mathcal{X}^{(S)}, \mathcal{U}^{(S)}, \mathcal{T}^{(S)}, c^{(S)})$, obtaining:
   (a) An experience dataset $\mathcal{D}^{(S)}$, consisting of state-action-state triples: $(\mathbf{x}_t^{(S)}, \mathbf{u}_t^{(S)}, \mathbf{x}_{t+1}^{(S)}))$
   (b) A current best policy $\pi^{(S)}$
2. Set $\bar{\pi}_0^{(S)} = \pi^S$. Optional: use a learned[4] or user provided source to target mapping.
3. Execute $\bar{\pi}_k^{(S)}$ in the target domain $(\mathcal{X}^{(T)}, \mathcal{U}^{(T)}, \mathcal{T}^{(T)}, c^{(T)})$, obtaining:
   (a) An experience dataset $\mathcal{D}^{(T)}$, consisting of state-action-state triples: $(\mathbf{x}_t^{(T)}, \mathbf{u}_t^{(T)}, \mathbf{x}_{t+1}^{(T)}))$
4. Using $D^{(T)}$, estimate (or update) the inverse dynamics of the target domain $g^{(T)} : \mathcal{X}^{(T)} \times \mathcal{X}^{(T)} \rightarrow \mathcal{U}$
5. Use $D^{(S)}$, $D^{(T)}$ and $g^{(T)}$ to estimate a policy adjustment function $\pi_{adj} : \mathcal{X} \times \mathcal{U} \rightarrow \mathcal{U}$ (illustrated in Figure 6)
6. Set $\bar{\pi}_{k+1}^{(S)}(\mathbf{x}) = \pi_{adj}(\mathbf{x}, \bar{\pi}_0^{(S)})$

In Step 5 of the previous procedure, we find a, possibly nonlinear, transformation from source to target policies. We do this by generating a dataset with samples of the form $(\mathbf{x}_t^{(S)}, \mathbf{u}_t^{(S)}, \mathbf{u}_*^{(T)})$, where $\mathbf{u}_*^{(T)} = g^{(T)}(\mathbf{x}_t^{(S)}, \mathbf{x}_{t+1}^{(S)})$. The first two items represent the action picked by the source policy at some target state. The third item is the action *that replicates the behaviour* of the source policy in the target domain. We use this dataset to train π_{adj} by supervised learning.

[4]We discuss how such a mapping could be learned in Section 4

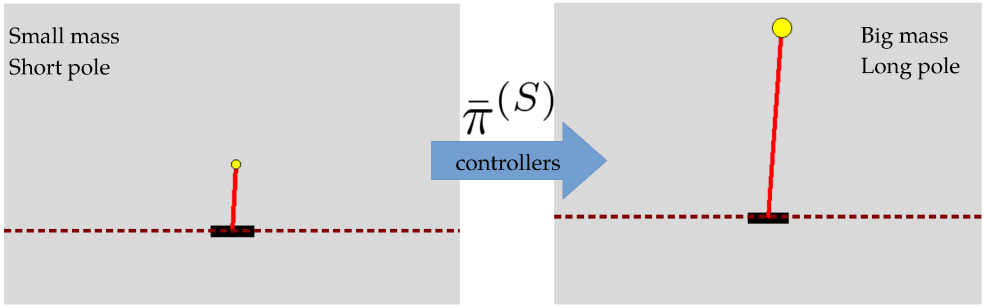


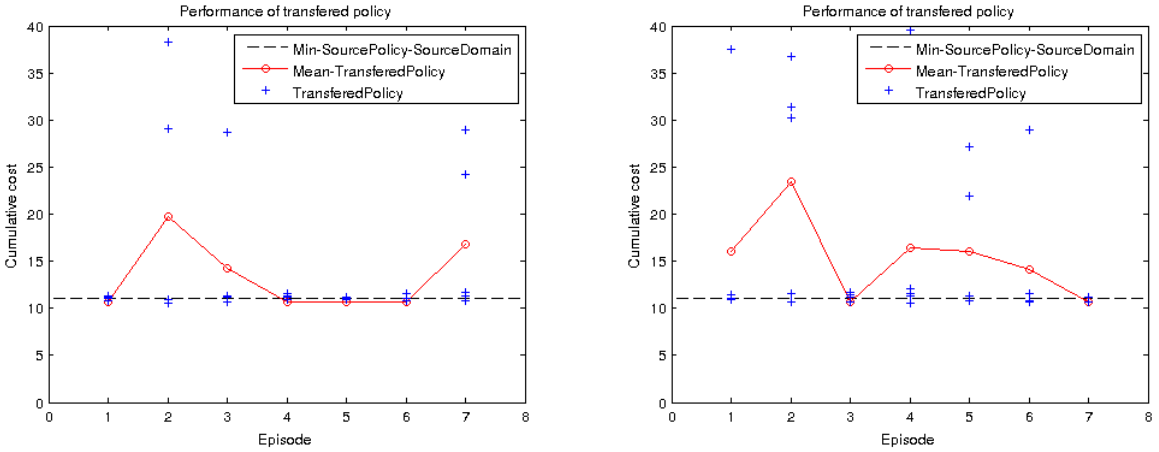
Figure 7: The cart-pole scenario: We tested our policy transfer scheme by varying the cart mass, pendulum mass, and friction coefficient at the target domain. The original parameters we used where $g = 9.82\text{m/s}^2, l = 0.5\text{m}, m = 0.5\text{kg}, M = 0.5\text{kg}, b = 0.1N/m/s$, where g is the acceleration due to gravity, l is the length of the pole, m the mass of the pendulum, M the mass of the cart, and b the coefficient of friction. The control input is a horizontal force applied to the cart (in black), and the goal is to balance the pole (in red) in an upright position.

We experimented with this method in the cart-pole domain, where the goal is to balance an inverted pendulum on a mobile cart on rail. We used PILCO to learn tasks from scratch, as illustrated in Figure 7. We are currently in the process of integrating PILCO with the simulator learning setup of Section 3.2. Learning the task with PILCO required 10 iterations in near 40 minutes of computation on a 2.53 GHz Intel Core i3 laptop with 4GB of RAM. Figure 8 shows the number of iterations required on average, when varying one of the cartpole simulation parameters. Our transfer method took about 5 minutes to transfer the same task, running unoptimized code. Given the relative simplicity of this method, we were surprised that it worked at all in the successful cases. We were less surprised by the failure modes, i.e. policies are very sensitive to a change in the pole length since the optimal trajectory deviates in the target domain deviates considerably with respect to the original one. Our current goals for this algorithm are to test on the data we collected with the AQUA robot (Meger et al., 2015), and close the loop by updating the simulator with the methods described in Section 3.2.

One major drawback with this naïve approach is that it is highly dependent on the quality of the initial policy. An alternative approach is to modify the policy parameters directly, which we aim to test in our future research. But our idea behind estimating the π_{adj} transformation is that it can be used for *different tasks*, since π_{adj} is a function of the state and the desired control command. The selection of which data points to use to train π_{adj} requires more attention. Selecting starting states from the source domain $\mathbf{x}_t^{(S)}$ is desirable as it defines the trajectory that we want to follow, but it has the problem that we may not have enough data in $D^{(T)}$ to compute the inverse dynamics $g^{(T)}$ for such states. On the other hand, selecting starting states from the target domain $\mathbf{x}_t^{(T)}$ is desirable because this corresponds to regions of the state space where we can compute $g^{(T)}$. However these states might have not been visited when training the source policy, and would not help bringing the robot closer to the optimal behaviour.

A possible way of improving our method is change the learning method used in simulation, for a trajectory optimization method and use multiple simulated trajectories to train a policy for the real robot. The stages of such learning system would roughly be: 1) Use trajectory optimization algorithm (e.g. DDP) in simulation to generate multiple desired trajectories, 2) perform supervised learning to train π_{adj} for transferring the learned policy to the target domain, 3) train a parametrized policy in the target domain from mapping the control command of the simulated trajectories through π_{adj}. We could use the simulated trajectories in conjunction with the target experience data to train the policy directly, but learning π_{adj} has the potential of working for multiple tasks.

Unlike the prior work on using a simulator in RL (Kober and Peters, 2012; Cutler and How, 2015), we are not continuing to optimize the policy in the target domain, using the source policy to seed the optimization. Instead, we use our assumption to synthesize a control policy through the policy adjustment function π_{adj}, which should work for multiple tasks. Our approach has strong similarities with the Dataset Aggregation



(a) Transfer to cartpole with a 200% friction increase

(b) Transfer to a cartpole with 1000% friction increase

Figure 8: Sample results when varying the friction coefficient across transferred domains. The dotted line represents the cumulative cost of the source policy $\pi^{(S)}$ in the source domain. The solid line is the average performance of the transfered policy across 5 trials with random initial conditions; i.e. drawn from a Gaussian distribution with zero mean and variance of 0.1 for each state dimension. The markers show the cost of individual performances at each trial. Notice how we got the transfered policy to work in the first trial, after one run of the original policy, for both cases, with occasional failures with high cumulative cost. We hypothesize the failures, which make up a minority of the trials, were due to a source policy that was not very robust.

DAGGER algorithm (Ross et al., 2010), except that we have no expert that could demonstrate the trajectories, other than the policy trained on a *different domain* (a simulator). A key question that remains to be investigated is that of deciding if a model is good enough to try the policy transfer approach, or if we need to update the model either by fitting the parameters of the existing simulator model, or constructing a new one.

3.4 Evaluating and Selecting Source Models

Knowing when a model is good enough for transfer is a key issue we need to address if we aim to transfer data to and from a simulator. Applying a policy learned from an inaccurate simulator directly on a target robotic platform will lead to suboptimal performance (Atkeson and Santamaría, 1997); i.e. to negative transfer. Here we take a look at two questions: how to decide when a model needs improvement, and how to select a model, or combination of models, if multiple are available. This means that we want the agent to be *self aware*, since it has to make decision on when to try a policy in the real world and when to improve it in the real robot; which brings the problem to the domain of **active learning**.

Recent work on Knows What it Knows (KWIK) learning provides a theory for how to approach the question of self awareness (Li et al., 2011). In the KWIK framework, a learning algorithm can decide to make a prediction about an outcome in the environment, with error smaller than ϵ with probability at least $1-\delta$, or report that it can't make a prediction, denoted with the symbol $\perp$. When applying this framework in a robot learning setting, the idea is to encourage gathering more experience in situations where the learning agent would report $\perp$. By using an exploration strategy that encourages this, it can be demonstrated that certain class of models can be learned with a number of samples that is polynomial in $1/\epsilon$, $1/\delta$ and a measure of the size of the hypothesis space.

In our case it is not clear we can obtain such bounds, but the ϵ, δ parameters and the $\perp$ symbol can be integrated into a scheme for deciding to update the model vs transferring the policy. The agent would

decide to update the model by comparing the original trajectories from the simulator and the executed trajectories in the real robot. For every triplet $(\mathbf{x}_t^{(T)}, \mathbf{u}_t^{(T)}, \mathbf{x}_{t+1}^{(T)}))$ obtained from the real robot, we can evaluate the prediction that the simulated model would make. From this set of predictions, we can compute the probability of making an error smaller than ϵ. If it is at least $1-\delta$, we decide to use the dataset to train the policy adjustment π_{adj}, otherwise we use the dataset to update the model and continue learning in the simulator. A similar approach was proposed by Cutler et al. (2014) but for discrete state and action spaces.

Measuring the probability of making a prediction error can be done by using Gaussian Process regression to fit our simulated model from Equation 8; i.e. we assume our model $f(\mathbf{x}, \mathbf{x}, \psi)$ will be drawn from a Gaussian Process

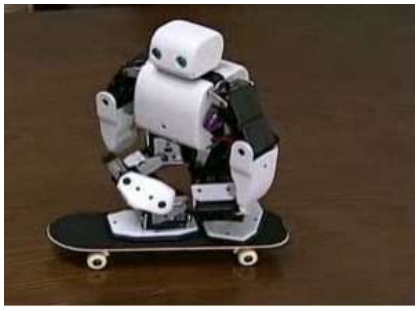
$$f(\mathbf{x}, \mathbf{x}, \psi) \sim \mathcal{GP}\left(f_{sim}, k\left(\begin{bmatrix}\mathbf{x}_A \\ \mathbf{u}_A\end{bmatrix}, \begin{bmatrix}\mathbf{x}_B \\ \mathbf{u}_B\end{bmatrix}\right)\right) \tag{12}$$

where k is a *kernel* function that models how the output of the dynamics co-vary for any given pair of input vectors $\left[\mathbf{x}_A, \mathbf{u}_A\right]^T$, $\left[\mathbf{x}_B, \mathbf{u}_B\right]^T$, and we assume that our simulator is a prior mean function; see (Rasmussen, 2004) for a more complete treatment. Every prediction made by the fitted model with GP regression has is modelled as a Gaussian distribution, so it is straightforward to determine the probability of making an error smaller than ϵ by integrating the density function of the predictions from the GP model. Using GP regression for for fitting the dynamics has also the benefit of interpretability. If we assume that the underlying dynamics are deterministic (true for the simulator), then the stochasticity of the model represents model uncertainty due to unmodelled dynamics.

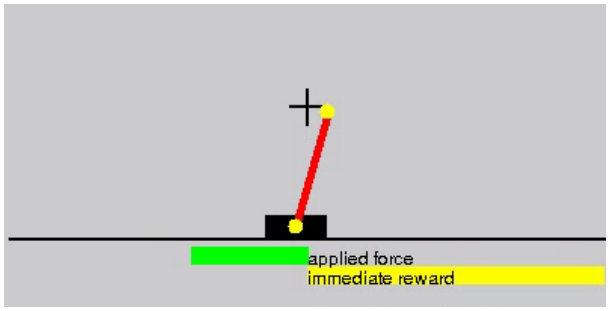
Once we have established a method for training a simulated model, we can use this method to produce an ensemble of models using different parameters or training data; e.g. learning swimming gaits in fluids with different characteristics. A robot that has access to such ensemble, now faces the problem of model: some models will be better suited than others for any particular combination of task and environment. The problem of model selection will be more clear when dealing with different robot morphologies, where a robot must select models from robots that are the most similar to itself. A clean formalization of this problem may be given by following the Bayesian model selection approach. Similar to the *responsibility signal* technique of Doya et al. (2002), if we get a dataset $\mathcal{D}^{(T)}$ from trajectories executed in the real robot, we can compute the likelihood of such dataset being predicted by any model $\hat{f}_i, i \in [1...n]$ of the ensemble. We can measure which model is the most likely by computing

$$\mathcal{P}\left(\hat{f}_i \mid \mathcal{D}^{(T)}\right) = \frac{p\left(\mathcal{D}^{(T)} \mid \hat{f}_i\right) p\left(\hat{f}_i\right)}{\sum_j p\left(\mathcal{D}^{(T)} \mid \hat{f}_j\right) p\left(\hat{f}_j\right)} \tag{13}$$

where $p\left(\hat{f}_i\right)$ is a prior that encodes some knowledge on which model should be used for the particular target. If we assume that the data samples are uncorrelated and assume a Gaussian Process distribution (Equation 12) for our models, then computing the term $p\left(\mathcal{D}^{(T)} \mid \hat{f}_i\right)$ is straight forward. In the case of trajectory following, the assumption of uncorrelated samples does not hold, since they were drawn from a distribution induced by the applied policy, and because they are temporally correlated. An appropriate way of computing this term should account for such correlations, and is a subject of ongoing research.



(a) A robot on a skateboard



(b) Another robot on a skateboard: the cartpole scenario

Figure 9: A humanoid robot on a skateboard can be seen as a considerably more complicated task than balancing a cartpole. Still, given that the analogy is valid, is there any knowledge that could be transfered from one domain to the other? The robot on the left is the PLEN robot `http://www.plen.jp/index_en.html`. The image on the right is a snapshot of the cartpole balancing task using PILCO (Deisenroth and Rasmussen, 2011)

4 Proposal Topic II: A Robotic System the Learns from Multiple Robot Morphologies

A natural extension to the framework for learning from simulated environments is to try transferring control policies between related tasks across different morphologies. For instance, policies and experience obtained from learning how to balance a cart-pole could be very useful in learning how to balance a humanoid robot on a skateboard (Figure 9), if the learning agent can determine how to map the parts of theformer to the parts of the latter. This idea might sound questionable, because the increase in complexity from a cartpole balancing task to humanoid balancing task. However, the advent of 3D printing provides some justification for our ideas.

With 3D printing we have the possibility of manufacturing robot bodies at low cost with fast iterations. Intuitively, any knowledge gathered about the dynamics of the world will be useful across iterations of the robot's morphology; i.e. we may re-use models in order to find policies for a new robotic system. If the changes in the robot's morphology are incremental, then transferring policies across different versions of the same robot should speed up the goal of finding controllers for particular behaviours. If we assume that the policy transfer mechanisms between iterations is composable, then it becomes conceivable that we can transfer policies from a very simple robot to a highly complex one, by chaining the transfer mechanisms of robots with intermediate complexity. However, this is highly speculative as we would need to demonstrate that the assumption of composable policy transfer mechanisms is valid.

We don't think this problem will be solved anytime soon, so we propose two research goals that could help us going in that direction. One is to develop techniques that allow us to compute sensible similarity metrics between robotic systems for particular tasks. Provided that the robotic systems do have some underlying similarity which can be treated, at least locally, as a metric, we can use this to select the most appropiate source model to learn from. The other goal is finding mappings between the descriptions of the robotic systems. This, combined with simulated environments as the ones described in Section 3, will assist in realizing our vision of the cloud robotic learning system depicted in Figure 3. As we haven't done much work on these topics, this section is slightly speculative. But these are the research ideas that motivated the idea of learning from simulators in the first place.

To our knowledge, there is no prior work on transfer learning across different physical robot morphologies. The closest work we have found is the recent work by Ammar et al. (2015a, 2014a), which is based on learning cross-domain mappings in a with a similar scheme to the one we propose in Section 3.3, but using intermediate projection spaces between domains, where the similarity between states can be computed as some affine metric. But no such approach has been demonstrated for transferring motor tasks between

different physical robotic systems. We suspect that this is because the computational requirements of such methods are very demanding.

4.1 Finding similarities and mappings between robotics systems

One of the first descriptions of the problem of finding mappings across different robotic learning tasks was discussed by Taylor and Stone (2007), where mappings were provided by a human designer. In that work, while no task was tried on a physical robot, did demonstrate that if a mapping between states and actions is provided, value functions learned in the source domain could greatly improve learning in a target domain. In subsequent work, the same authors demonstrated the use of such mapping to transfer policies across domains. Providing the mappings required an inspection by the designer to determine which state dimensions and which actions were "similar" across domains, successfully speeding up the learning process in the target domain.

However, we believe such approach might be very hard to apply in continuous state and action spaces; we might be able to give a qualitative description of similarity by comparing state instances, but providing a mapping a priori could be more challenging. Taylor and Stone (2009) suggest that learning the task mapping from observed data in the source and target domains may be more appropriate, listing some of the methods that existed at the moment of their publication. Most of the methods listed required some input from the designer in the form of allowed mapping rules (i.e. grouping state variables that correspond to distances, positions, angles, etc).

One of the proposed approaches obtained the state and action mapping through supervised learning, in a similar manner to the approach we described in Section 3.3, and used it to transfer instances from the source experience dataset to the target domain. The transferred experience was used to guide exploration during learning in the target task. However, it only allowed for mapping state variables, whereas a more general approach would need to map regions of state space. In addition, their proposed approach did an exhaustive search for the appropriate inter task mappings, a method for which the search space grows exponentially with the number of dimensions, making their method less appealing for robotics applications.

Very recently, a variety of approximate methods for finding task mappings through projection and dimensionality reduction have been proposed to learn mapping between source and target domains from data. One example is the idea of manifold alignment by Wang and Mahadevan (2009). In manifold alignment, the idea is to use dimensionality reduction techniques to transform two disparate datasets into manifolds with the same number of dimensions, then finding a mapping between the two manifolds to combine, or directly finding a common lower dimensional manifold for the two datasets. Bocsi et al. (2013) propose to use Principal Component Analysis (PCA) from the source and target domain datasets, heuristically specifying a common number of principal components to use, and fitting an affine transformation between the two low dimensional manifolds. Then, the data from the source domain is used to fit a kinematic model for the target domain – their application was learning kinematic models with data from multiple articulated robotic manipulators.

The work by Ammar et al. (2012) proposes projecting the datasets to *higher* dimensional spaces with the same number of dimensions, then learning a mapping between the two using supervised learning. Their approach is then used to transfer data samples from the source domain to the target domain, and continue learning in the target domain. Their work has recently been extended (Ammar et al., 2015b) to instead use the manifold alignment approach of Wang and Mahadevan (2009) to transfer policies between simulated domains. The idea of manifold alignment presents a powerful way of transferring policies across related domains, however the question of whether they work for physical robots is still waiting to be answered[5].

We are interested in trying the approach of manifold alignment for learning cross domain mappings between different robot morphologies, which relies on the assumption that the exists a common latent space shared by the source simulator and the target domain. A couple questions that we aim to investigate are: When does this assumption break? Is it always beneficial to use this technique? How can we apply this to the case where we want to combine data from a large number of simulated models or robot morphologies?

[5]In this situation it is important to remind ourselves of the words of a renowned physicist: "In theory, theory and practice are the same. In practice, ..."

As stated in Section 2 the question of whether a particular mapping will produce negative transfer remains an open problem, and it is one that we wish to investigate.

4.2 A cloud architecture for learning controllers across robotic morphologies

The last step in realizing our vision depicted in Figure 3 would be putting all of the pieces from the previous sections together. This possibly requires a great software engineering effort, and we are not claiming that we could rely on the capacity of a single graduate student and his supervisor to carry it out. But, as we note in the next section, we do have some of the pieces together in an initial version of the learning infrastructure used for our previous experiments (Meger et al., 2015) and have developed a server based platform for planning robotic waypoint navigation tasks (Gamboa-Higuera et al., 2012). In this very short section, we are just going to state that the final goal of our research is to demonstrate one such system. This is highly speculative, as its realization depends on the success of the components described in the previous sections.

The idea of a cloud based system[6] for offloading the computations of robotic systems has mostly been explored for visual tasks like navigation and grasping, where object detection and tracking, detecting affordances, and simultaneous localization and mapping are needed (Waibel et al., 2011; Kehoe et al., 2013). In such cases the robots offload the task of interpreting sensor data to the cloud. We aim to extend the work on this area by demonstrating a learning system where any robot could upload its morphology, some experience data, and a specification of a task, to obtain a policy that, in the best case, works in a single shot or, in the worst case, enables the collection of new data to improve the existing models.

We would like to try an experiment similar to the work by Song et al. (2015), where a simulated model is used to design a 3D printed robot. Instead of designing the robot, we would take a design from the internet, train a policy in simulation, 3D print the model and test the policy on the 3D printed robot. We believe this is achievable in a shorter term than the one required for a cloud based system of Figure 3.

5 Proposed research plan

In this section we list the progress we have made to this date, and the proposed milestones to evaluate progress in our research plan

5.1 Progress to date

Most of the work we have done so far has been listed in Section 3. Here we present a list that summarizes our current progress.

Models for Physically based Simulation of Robotic Systems: We have implemented a simulation module that accounts for hydrodynamics, which can be used to model bodies immersed in a fluid; e.g. underwater robots, or fast moving objects in air (Section 3.1). We have described a group of methods for updating simulator parameters, and fitting a correction term that accounts for unmodelled dynamics. We currently have an implementation of the semi-parametric method described in 3.2. We have integrated our software with the Gazebo Robot simulator (Koenig and Howard, 2004), and provided a drop-in replacement interface to switch between a real and simulated AQUA robot using the Robot Operating System middleware Quigley et al..

Policy Transfer Methods: We have also integrated existing policy search (Deisenroth and Rasmussen, 2011) and optimal control (Pan and Theodorou, 2014) methods into the ROS framework, in order to use them with the AQUA robot. We have successfully used this software to run our first experiments on learning motor behaviours from scratch with the AQUA robot, and our first experiments with naïve bidirectional transfer of experience and policies between the simulator and the real robot, work that was presented at ICRA 2015 (Meger et al., 2015). We have started our first approach to policy transfer, which we called *policy adjustment*. We have tested this approach on a benchmark RL task with promising results, and have the infrastructure ready for testing in on the AQUA robot.

[6]For a detailed survey of the development of the cloud robotics idea see Kehoe et al. (2015); Cameron and Hurd (1984)

Model Evaluation and Selection Methods: We have written software for the model evaluation scheme described in Section 3.4, which aims to decide when to improve an existing model, or when to optimize the policy adjustment. We are currently running experiments with this method. The model selection method described in the same section is still at the preliminary design stage.

Infrastructure for Robotic experiments: In the past year, we have been developing a offsite learning system that allows us to run multiple robot learning tasks in parallel in a remote server, making the best use possible of the idle time between learning iterations. We have integrated the PILCO and PDDP algorithms into a Python package with a common infrastructure for learning models, defining cost functions and optimizing policies. We aim to extend this software package with additional learning and optimal control methods. This should allow us to test our transfer algorithm with a heterogeneous class of optimizers, policy classes and cost functions. This work has been integrated with the ROS framework.

5.2 Proposed Milestones

To guide and evaluate progress towards our research goals, we propose the following list of milestones. This is a tentative list, that will evolve according to any new development we make. These first items correspond to our short term goals, for which we can estimate dates with more certainty.

- **Policy Transfer Experiments on a Real Robot:** Our most immediate milestone is to test the approach we described in Section 3.3 for a physical robot, which is the goal of this thesis. We are planning to run such experiment by the end of the Summer of 2015, with preliminary tests using data from our prior experiments happening during the course of the Summer term.
- **Autonomous Model Evaluation Experiments on a Real Robot:** The approach for model evaluation described in Section 3.4 will be tested first in simulation, then on the real robot. We aim to test this approaches before the end of the Fall 2015 term.
- **Additional Policy Transfer Methods:** The approach we described in Section 3.3 might not work in practice. We have discussed how to use other policy transfer schemes based on matching trajectory data for training a policy directly in the target domain, without the requirement of learning a policy in the simulated domain. This is research that will be carried out during the course of the year 2015, as we gain more insights from the original approach. Candidate methods for policy transfer are likely to be based on prior work that relies on trajectory optimization techniques (Atkeson et al.; Levine and Koltun, 2013; Pan and Theodorou, 2014).
- **Model Selection:** While we will keep investigating methods for model selection, we currently do not have multiple models to select from. As our software becomes more stable, we aim to test our algorithms with a wider class of mobile robotic system for which we have access. Our current plan is to obtain multiple models for one particular robot morphology, the AQUA robot, under a variety of settings that drastically change the dynamics of the robot: changes in water density (fresh vs salt water), changes in robot ballasting, and a possible new version of the AQUA robot. Data for these models will be collected during the 2015-2016 academic year. Experiments with model selection are likely to happen offline, during the same timeframe. Ideally we would like to do at least one field experiment where we can validate the model selection procedure during the first half of the year 2016.

The following list of milestones constitute our medium term research plan, to be carried out before the end of the 2016-2017 academic year.

- **Additional Model Selection methods:** One very interesting approach for dealing with model uncertainty is to treat the model parameters as latent variables Konidaris and Doshi-Velez (2014). We would like to explore such methods in a setting with multiple simulated models; i.e. we would like to combine the idea of latent simulator model parameters, with latent simulator model classes.
- **Obtaining mapping and distance metrics between different domains:** As recent research has shown (Taylor and Stone, 2007, 2009; Barrett et al., 2010; Ammar et al., 2012, 2014a, 2015a,b),

finding the appropriate mappings between the source and target domains state and action spaces is a critical piece for any knowledge transfer scheme. Our goal is to apply similar techniques in our research thread on learning from multiple simulated models.

- **A cloud based server for heterogeneous robotic systems:** We plan to extend or, more likely, replace our current offsite so that we can a) collect experience data from multiple robots, b) query an extensive set of simulated models, c) learn policies in such models, and d) query the system for an appropriate prior policy given a task specification and robot morphology. We already started work in this direction, as described in the **Infrastructure for Robotic experiments** item.

The following items are our "long shot goals", for which the progress on the prior tasks is critical, and could possibly require starting another PhD.

- **Online learning on a mobile robot:** Given a starting policy, or set of policies from a simulator, we would like to continue to improve the policy online in a mobile robot with intermittent access to the internet. This requirement will make a large class of existing policy search methods impractical. We would like to see this happening in the near future, but we do not have a clear idea on how to proceed. Hopefully, our own learning during the next year will allow us to transfer some of our old ideas to a real time setting.
- **Test our transfer ideas with a 3D printed robot:** We would like to include the robot morphology into the policy optimization loop, such that we can adapt policies from one morphology to the next one. Recent approaches have shown that it is possible to obtain better performance with appropriate transfer, than with random initialization or with *tabula rasa* learning. We would like to test 3D printing different robot bodies, and use the same library of policies and models to try to perform various motor tasks on them. Candidates include the Aracna (Lohmann et al., 2012) or Poppy (Lapeyre et al., 2014) platforms, but we would likely start with simpler platforms.

6 Conclusion

We have presented a summary of our ideas on a contribution to the Robot learning problem, exemplified by the bootstrap scenario (Kuipers et al., 2006; Censi, 2012). We take the view that physically based simulation, while inaccurate, still provides enough information to a robot about certain general rules; e.g. gravitation, contacts and friction forces and hydrodynamics. These are rules that would be very hard to generalize in the bootstrap scenario, where a robot must learn how to interpret its sensor and actuator data in order to produce repeatable behaviour. As knowledge acquired from a simulator may not be directly applicable on a real robot, we propose a series of techniques for bidirectional transfer between a simulated model and a target environment.

We propose to extend these ideas to transferring information between different robot morphologies via our learning from simulation framework. We presented some of our preliminary ideas on the subject, along with a proposed research plan to help us reach our goal. We envision a learning system that can run in the "cloud" providing a motor control learning service for robots deployed across different domains. Such service would allow for offloading the learning and exploration work to a simulated environment, drastically reducing the amount of data that a robot would need to learn to perform a task.

References

Pieter Abbeel, Varun Ganapathi, and Andrew Y Ng. Learning vehicular dynamics, with application to modeling helicopters. In *Advances in Neural Information Processing Systems*, pages 1–8, 2005.

Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2006.

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1, 2007.

Haitham B Ammar, Karl Tuyls, Matthew E Taylor, Kurt Driessens, and Gerhard Weiss. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 383–390. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

Haitham B Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1206–1214, 2014a.

Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014b.

Haitham Bou Ammar, Eric Eaton, José Marcio Luna, and Paul Ruvolo. Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. 2015a.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. 2015b.

Christopher G. Atkeson and Juan Carlos Santamaría. A comparison of direct and model-based reinforcement learning. In *Proceedings of the 1997 IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico, USA, April 20-25, 1997*, pages 3557–3564, 1997.

Christopher G Atkeson et al. Using local trajectory optimizers to speed up global optimization in dynamic programming.

David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 23–34. ACM, 1994.

David Baraff. An introduction to physically based modeling: rigid body simulation i—unconstrained rigid body dynamics. 1997.

Samuel Barrett, Matthew E. Taylor, and Peter Stone. Transfer learning for reinforcement learning on a physical robot. In *Int. Conf. on Autonomous Agents and Multiagent Systems - Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010.

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

Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

Botond Bocsi, Lehel Csató, and Jochen Peters. Alignment-based transfer learning for robot models. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE, 2013.

James Cameron and Gale Anne Hurd. The terminator, 1984.

Andrea Censi. Bootstrapping Vehicles: A formal approach to unsupervised sensorimotor learning based on invariance. Technical report, California Institute of Technology, 2012. URL `http://purl.org/censi/2012/phd`.

Andrea Censi. Design Patterns for Learners Parallelization, or: the joy of waking up in the morning and finding the experiments all done. Technical report, Laboratory for Information and Decision Systems/MIT, October 2014. URL `http://purl.org/censi/2015/despl`.

Mark Cutler and Jonathan P. How. Efficient reinforcement learning for robots using informative simulated priors. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015. IEEE.

Mark Cutler, Thomas J. Walsh, and Jonathan P. How. Reinforcement learning with multi-fidelity simulators. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014.

Marc P. Deisenroth and Carl E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2011.

Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

Evan Drumwright. Rapidly computable viscous friction and no-slip rigid contact models. *arXiv preprint arXiv:1504.00719*, 2015.

Evan Drumwright, Dylan Shell, et al. Extensive analysis of linear complementarity problem (lcp) solver performance on randomly generated rigid body contact problems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5034–5039. IEEE, 2012.

Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. *Robotics: Science and Systems VII*, page 73, 2012.

Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011.

Juan Camilo Gamboa-Higuera, Anqi Xu, Florian Shkurti, and Gregory Dudek. Socially-driven collective path planning for robot missions. In *in Proceedings of the 9th Canadian Conference on Computer and Robot Vision (CRV '12)*, Toronto, Canada, May 2012.

C. Georgiades. Simulation and control of an underwater hexapod robot. Master's thesis, McGill University, 2005.

Sehoon Ha and Katsu Yamane. Reducing Hardware Experiments for Model Learning and Policy Optimization. *IEEE International Conference on Robotics and Automation*, 2015.

Adrian M Haith and John W Krakauer. Theoretical models of motor control and motor learning. *Routledge Handbook of Motor Control and Motor Learning*, page 7, 2013.

D.H. Jacobson and D.Q. Mayne. *Differential dynamic programming*. Modern analytic and computational methods in science and mathematics. American Elsevier Pub. Co., 1970.

Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-based robot grasping with the google object recognition engine. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4263–4270. IEEE, 2013.

Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *Automation Science and Engineering, IEEE Transactions on*, 12(2):398–409, 2015.

Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

N. Koenig and A Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, 2004.

George Konidaris and Finale Doshi-Velez. Hidden parameter markov decision processes: An emerging paradigm for modeling families of related tasks. 2014.

Benjamin J Kuipers, Patrick Beeson, Joseph Modayil, and Jefferson Provost. Bootstrap learning of foundational representations. *Connection Science*, 18(2):145–158, 2006.

Matthieu Lapeyre, Steve N'Guyen, Alexandre Le Falher, and Pierre-Yves Oudeyer. Rapid morphological exploration with the poppy humanoid platform. In *IEEE-RAS International Conference on Humanoid Robots*, page 8, 2014.

Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

Suchan Lee, Jason Yosinski, Kyrre Glette, Hod Lipson, and Jeff Clune. *Evolving gaits for physical robots with the hyperneat generative encoding: The benefits of simulation*. Springer, 2013.

Sergey Levine and Vladlen Koltun. Guided policy search. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1–9, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015a.

Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. *arXiv preprint arXiv:1501.05611*, 2015b.

Lihong Li, Michael L Littman, Thomas J Walsh, and Alexander L Strehl. Knows what it knows: a framework for self-aware learning. *Machine learning*, 82(3):399–443, 2011.

Lennart Ljung. *System identification*. Springer, 1998.

Sara Lohmann, Jason Yosinski, Eric Gold, Jeff Clune, Jeremy Blum, and Hod Lipson. Aracna: An open-source quadruped platform for evolutionary robotics. In *Artificial Life*, volume 13, pages 387–392, 2012.

David Meger, Juan Camilo Gamboa Higuera, Anqi Xu, Philippe Giguère, , and Gregory Dudek. Learning legged swimming gaits from experience. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

Teodor Mihai Moldovan, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. Optimism-driven exploration for nonlinear systems. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

Andrew Y Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

Duy Nguyen-Tuong and Jan Peters. Using model knowledge for learning inverse dynamics. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2677–2682. IEEE, 2010.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowledge and Data Engineering*, 2010.

Yunpeng Pan and Evangelos Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*, 2014.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. 2008.

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Natural actor-critic. *Proceedings of the European Conference on Machine Learning (ECML)*, 2005.

Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI), Physically Grounded AI Track*, 2010.

Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May 2015. IEEE.

Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. Ros: an open-source robot operating system.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*, 2012.

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. *arXiv preprint arXiv:1011.0686*, 2010.

Junaed Sattar, Gregory Dudek, Olivia Chiu, Ioannis Rekleitis, Philippe Giguère, Alec Mills, Nicolas Plamondon, Chris Prahacs, Yogesh Girdhar, Meyer Nahon, and John-Paul Lobos. Enabling autonomous capabilities in underwater robotics. In *Proc. of the IEEE/RSJ Int. Conf. on Int. Robots and Systems (IROS)*, 2008.

S. Schaal and C.G. Atkeson. Learning control in robotics. *Robotics Automation Magazine, IEEE*, 2010a.

Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive Motion of Animals and Machines*. Springer, 2006.

Stefan Schaal and Christopher G Atkeson. Learning control in robotics. *Robotics & Automation Magazine, IEEE*, 17(2):20–29, 2010b.

Weiguang Si, Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Realistic biomechanical simulation and control of human swimming. *ACM Transactions on Graphics (TOG)*, 2014.

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM, 1994.

Seungmoon Song, Joohyung Kim, and Katsu Yamane. *Development of a bipedal robot that walks like an animation character*. 2015.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. 1990.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. Citeseer, 1999a.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999b.

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. Articulated swimming creatures. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11. ACM, 2011.

Yuval Tassa. *Theory and Implementation of Biomimetic Motor Controllers*. PhD thesis, 2011.

Matthew E Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, pages 879–886. ACM, 2007.

Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 2009.

Demetri Terzopoulos, Xiaoyuan Tu, and Radek Grzeszczuk. Artificial fishes: Autonomous locomotion, perception, behavior, and learning in a simulated physical world. *Artificial Life*, 1(4), 1994.

Emanuel Todorov and Michael I Jordan. Optimal feedback control as a theory of motor coordination. *Nature neuroscience*, 5(11):1226–1235, 2002.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. Roboearth. *Robotics Automation Magazine, IEEE*, 18(2):69–82, 2011.

Chang Wang and Sridhar Mahadevan. Manifold alignment without correspondence. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, San Francisco, CA, USA, 2009.

Steffen Weissmann and Ulrich Pinkall. Underwater rigid body dynamics. *ACM Trans. Graph.*, 2012.

Haoran Xie and Kazunori Miyata. Real-time simulation of lightweight rigid bodies. *Vis. Comput.*, 2014.