Modeling for robotics and pneumatic actuation using PDEs

Svetoslav Kolev

A report for the Final Examination
submitted in fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Emo Todorov, Chair

Dieter Fox

Eric Rombokas

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Modeling for robotics and pneumatic actuation using PDEs

Svetoslav Kolev

Chair of the Supervisory Committee:
Affiliate Professor Emo Todorov
Computer Science & Engineering

Successful robotic control depends on truthful and robust dynamic models. While system identification is generally employed to compute such models, there are challenges to applying system ID. In the context of dynamic object manipulation the difficulty arises from the non-linear nature of the contact phenomena. On the other hand, with pneumatically actuated robots, the challenge is devising a suitable model class that captures the intrinsic dynamics of the system.

The primary goal of this thesis is to push the state of the art in those areas. The thesis makes the following contributions towards this goal:

- A novel approach to the system identification problem that solves both the problems of estimation and system ID jointly. We show that these two problems are difficult to solve separately in the presence of discontinuous phenomena such as contacts. The problem is posed as a joint optimization across both trajectory and model parameters and solved via Newton's method.

- A novel way of modeling pneumatically actuated systems based on Computational Fluid Dynamics and implemented as a Partial Differential Equation solver that is augmented to handle full robotic systems.

- Validation of that model and exploration of the advantages compared to previously employed models via the use of Reinforcement Learning in a simulated environment.

- System Identification of PDE-based pneumatic model and demonstration of its advantages over the existing methods for modeling pneumatic systems.
- Successful transfer of agents learned through Reinforcement Learning to real hardware.

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

1.1 Motivation

Accurate physics models are essential for any robotic control task. There are many aspects of modeling a robot's environment ranging from visual understanding of the objects around a robot and their relationships, to robot's own body and most importantly the relationship between a robot's body and its surroundings. In this thesis we focus on the latter two aspects. There are also different ways of modeling a system. On one side there are approaches starting from first principles in physics and building up towards a full robotic system. On the other side are the approaches that bypass formal physics and instead focus on direct input (sensor data) to output (usually control signal) link, and are usually have to be task-specific. Neither approach on its own is sufficient in enabling autonomous robots and a combination of the two is the path towards autonomy. In this thesis, we focus on the former (physics based) approach.

One important aspect of robot control is manipulation of objects. In particular the contact between an end-effector and object's surface is of great interest as that is the only way in which a robot effects the environment, regardless of its manipulator shape. The problem is also very dynamic, as gravity usually drives the system towards failure (a robot dropping its manipulation target), making manipulation one of the most challenging robotic tasks. Reliable modeling is necessary for achieving dexterous environment manipulation.

Another interesting aspect is actuation. While electrically driven motors are most common, pneumatically actuated robots offer advantages in terms speed and strength. However they come with much more complex dynamics, which complicates their control. Building better understanding of pneumatic systems will enable their wider spread use in robotics.

1.2 Thesis: contribution and organization

This thesis investigates the challenges of modeling robotic systems in two aspects: Modeling contact interaction between end-effectors and objects as well as modeling pneumatic actuators. The specific contributions are:

1. Novel solution to the system identification problem in the presence of contacts. We tie the problem to state estimation and show that system ID and state estimation are inextricably linked in certain cases. We use our Phantom Haptic Device based robotic platform with 3D printed silicone covered fingers as well as IMU and motion tracking instrumented 3D printed object. Details are discussed in chapter 2.

2. Novel Partial Differential Equation based model for pneumatically actuated robotic systems. The framework builds upon existing Computational Fluid Dynamics ideas and expands them towards modeling of a full robotic environment. Details are found in chapters 4 and 5.

3. Successful Sim-2-Real of RL-trained agent to a real system with third order dynamics. Details can be found in chapter 9.

Chapter 3 lays out some background theory that we build upon. In section 3.2 we explore the previous work in modeling pneumatic actuators and in section 3.1 we explain the Reinforcement Learning framework that we employ afterwards.

Chapter 4 explains the foundations of Computational Fluid Dynamics that we build upon. In chapter 5 we expand the basic CFD model to allow it to model actual robotic systems, and this is where most of our contribution lies.

Next, chapter 6 shows some preliminary work exploring the applicability of a PDE-based model to real systems, while 7 compares the performance of our model against previous work in a simulated environment.

Next, chapter 8 shows system identification of the so constructed PDE model.

Chapter 9 shows successful Sim2Real transfer of agents learned in simulated environment via Reinforcement Learning towards real hardware. We demonstrate that in basic building block tasks such as pressure tracking our method outperforms existing approaches. We also successfully show Sim2Real transfer on a third order system.

Finally, chapter 10 is the conclusion of this thesis and offers a discussion on the ideas presented earlier and offers possible direction for further study.

Chapter 2
SYSTEM ID WITH STATE ESTIMATION

Successful model based control relies heavily on proper system identification and accurate state estimation. We present a framework for solving these problems in the context of robotic control applications. We are particularly interested in robotic manipulation tasks, which are especially hard due to the non-linear nature of contact phenomena.

We developed a solution that solves both the problems of estimation and system identification jointly. We show that these two problems are difficult to solve separately in the presence of discontinuous phenomena such as contacts. The problem is posed as a joint optimization across both trajectory and model parameters and solved via Newton's method. We present several challenges we encountered while modeling contacts and performing state estimation and propose solutions within the MuJoCo physics engine.

We present experimental results performed on our manipulation system consisting of 3-DOF Phantom Haptic Devices, turned into finger manipulators. Cross-validation between different datasets, as well as leave-one-out cross-validation show that our method is robust and is able to accurately explain sensory data.

2.1 Introduction

Accurate models and state estimation are crucial components of model-based robot controllers. Such controllers usually utilize a planner with which the control algorithm imagines a future and then optimizes the plan to best satisfy user specifications. There are two critical requirements for this strategy to work: 1) We need to have an accurate estimate of our current position, otherwise the resulting plan would not make any sense and 2) we need to have an accurate physical model so that we are confident that the plan is feasible and will indeed be realized by our robot.

Model predictive control (MPC) is a particular implementation of that idea which is able to cope with slight estimation and modeling inaccuracies. The idea is that deviations from the plan accumulate slowly and smoothly; if we replan fast enough then the controller will still achieve the goal. MPC has been applied to contact rich tasks such as humanoid robot walking [6], where despite it being in simulation an MPC approach is still needed because the simulation model is different than the one used for planning. Model based control has even been applied to dexterous hand manipulation [15] where the planner has to reason about multiple acyclic contact events.

Still, applying model based controllers to real-world contact rich problems has been an elusive task. Even if we had perfect contact information, many tasks would still be difficult. For example, in manipulation tasks friction is of great importance, where not only the contact state matters but the friction force and normal forces are what allows one to hold an object against gravity, or change its configuration. This problem manifests itself both in the planner, which has to plan those contact forces, as well as in the estimator that needs to reason about them. Due to their nature, contacts introduce very sharp nonlinearities in the dynamical model, which makes the control and estimation tasks much harder. Even slight error in state information will cause incorrect contact state which will result in significant misprediction when used by a controller.

Robots are usually equipped with plenty of sensors, including force, tactile and motor torque sensors in an attempt to cope with those challenges. Similarly a human hand is covered with tactile sensors and our muscles are equipped with tendons which have force sensors embedded in them. Our brains are able to fuse this information into an intuitive state estimate. We are also able to predict the effects of our muscle actions on the objects we grasp. Similarly we aim to enable robots to do the same in what we call physically consistent state estimation. Physical consistency means our estimator is able to explain all the observed sensory data with its dynamical model of the world.

The standard procedure for system identification of a robot is to collect data while it is doing a task and then optimize over certain model parameters so as to maximize the predictive accuracy of the model. For example, if we have access to an inverse dynamic simulator, given a state trajectory it can compute the motor torques that must have produced that behavior, which we can compare with the collected data and determine success. Or with a forward simulator we can judge by its ability to accurately simulate certain time interval of the future. That strategy works well for smooth systems, but it fails as soon as we encounter contacts as shown in Figure 2.1. For example, trying to estimate the coefficient of friction between an end effector and a surface would be impossible if we had a distance of even $1mm$ between them in our state estimate (the typical error of motion capture system like Vicon and a robot like a Phantom). This is the case because estimating friction requires knowledge of the normal force, which in physics simulators is greatly dependent on the interpenetration (or distance in the case of remote contacts) between the two objects. Therefore it is essential to consider both problems of estimation and system identification together.

The main contribution of this paper is a joint system identification and estimation framework that optimizes over both a trajectory and dynamical model with the goal of achieving high consistency. To the best of our knowledge this is the first paper to present such a combination while using a full-featured physics simulator, MuJoCo [35]. Furthermore we propose solutions to associated challenges such as tangential contact deformations and remote contact sensing.

This paper is organized as follows. In section 2.2 we discuss previous approaches to system

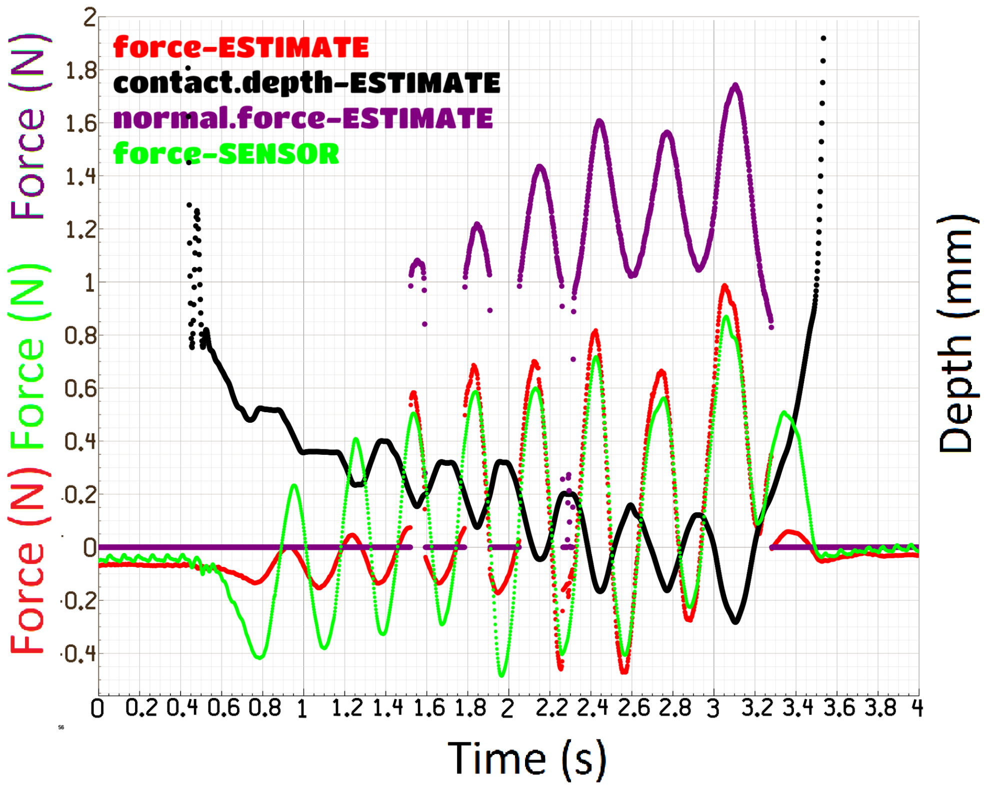


Figure 2.1: Example of system identification failure trying to estimate the contact parameters between a robotic finger and a 3D printed object. The finger is sliding on the surface applying various amounts of pressure. The black graph shows the distance between the end-effector and the stationary object, estimated purely from motion capture and joint encoder inputs. When the finger is too far from the object (more than $0.2mm$ away) the physics simulator predicts zero normal force (purple graph), making the estimated sensor reading (red graph) very different than the true sensor reading (green graph). The fluctuations of the red graph when not in contact are caused by the simulated sensor measuring the inertial forces of the moving end-effector.

identification and estimation. In section 2.3 we outline the mathematical formulation of the optimization algorithm. After that in section 2.4 we detail some of the challenges in solving the posed optimization problem while in section 2.5 we detail some of the contact modelling challenges we encountered and their solutions. In section 2.6 we describe the hardware setup and data collection. Section 2.7 presents the results. We conclude in section 2.8 and discuss possible venues for future work.

2.2 Related Work

In our previous work [16] we developed an extension to Extended Kalman Filter (EKF) that had two important differences: Instead of keeping the state of the system for the last step only, we kept a fixed length interval and instead of a single Gauss Newton step we allowed for a varying number, with the usual lag/accuracy tradeoff. Looking at multiple steps in the past provides the estimator with richer context which can be used to arrive at a more physically consistent estimate. In this work we extend the algorithm to handle system identification at the same time as estimation.

Zhang et al. also considered the problem of object tracking and identification of contact parameters together by combining a visual datastream and tactile feedback [43]. However, they used a 2D physics engine and also their estimation is done via Particle Filtering which proved too slow for realtime, while our gradient based approach has been successfully applied to realtime applications [16]. Wan et al. attempt to solve the problems of state estimation, parameter estimation (system identification) and dual estimation (combination of the two) using an Unscented Kalman Filter (UKF) [39]. However, their results are not on robotic systems, but on artificial data, and it is not clear whether robotic system parameters can be estimated in an UKF manner, without considering the whole trajectory.

The SLAM (Simultaneous Localization and Mapping) problem is very similar to ours[19][23]. Mapping corresponds to system identification (learning about the environment) and localization to state estimation. Instead of mapping the tangible aspects of the environment we are modeling dynamical properties of the environment and also the state of a robot consist of many DOFs, whereas in SLAM the state is usually 2D or 3D. SLAM is solved via landmark tracking techniques [19] as well as by dense vision approaches [23].

Vision based systems for state estimation have been very popular recently. Tanner et al. developed DART that uses a depth camera to track articulated bodies [28] and extended it by incorporating tactile information to enable tracking of objects even when they are partially occluded [29]. They use dense camera information which enables gradient approaches and consider contacts and interpenetrations carefully but do not use a full dynamical model and only consider a few predefined states. Particle filtering is another technique applied to estimation in contact rich tasks by Koval et al. [14], as well as by Chalon et al. [3] who also combined vision and tactile sensing. Koval et al. optimized the number of particles required by placing them on the contact manifold but still the number of particles required grows exponentially as the number of contact points increases. While vision methods provide good estimates based only on internal sensors they cannot provide the required accuracy that is needed by a physics simulator for planning, because for stiff contacts even small inaccuracies have great effect on the dynamics. We view this approach as complimentary to ours. For example, a vision based system can replace our motion capture system, as it is expected that robots should be able to operate in arbitrary environments that are not equipped with motion capture systems.

With regards to contact modeling, Gilardi et al. provide an overview of the various approaches and attributes to consider when implementing contact models [7]. Real contacts are rather complex and no existing physics simulator captures all aspects. We mention how we solve some of the arising problems in section 2.5. Drumwright et al. survey the various implementations of multibody systems with contact [5]. Verscheure et al. identify contact dynamics with a stiff robot and using extremely precise measurement tools and do not use a general purpose physics model [37]. It is not clear how easily a general contact model can be incorporated into a physics engine, which is why we think it is important to perform the system identification within the framework of a physics engine.

The physics simulator that we use, MuJoCo [35], is a general purpose physics engine and allows simulation of robotic systems with multiple joints as well as handles contacts between bodies, equality constraints and tendons. MuJoCo relies on a new formulation of the physics of contact described in [35]. The computation of contact forces in forward dynamics reduces to a convex quadratic program, while in inverse dynamics the contact forces are computed analytically. This model also allows soft contacts, and has a rich parametrization making it suitable for system identification. Using a physics simulator such as MuJoCo bridges the gap towards control as MuJoCo has been successfully applied in trajectory optimization for control [15] [6].

2.3 Problem Formulation

Currently, the mode of operation of our framework is as follows. The robot moves around, either via teleoperation, or following a predefined trajectory, possibly interacting with the manipulated object. We record sensory data including motion capture data, an IMU, a 6-DOF force sensor at the end-effector, as well as joint angles and torques. We also construct a model in MuJoCo that mirrors the realworld scene. That model includes parameters such as masses, inertias, friction coefficients, contact softness, etc. We then ask the question: What is the robot/object trajectory and the set of model parameters that best explains the sensory data? We pose that question as an optimization problem as follows.

We model the trajectory as a sequence of states:

$$\mathbf{Q} = \{q_1, q_2, \ldots, q_n\}$$

where n is the number of timesteps we consider. Each state is a vector $q_i = (\theta_1, \ldots, \theta_{k'}, x, y, z, q_w, q_x, q_y, q_z)$, with $\theta_1, \ldots, \theta_j$ representing joint angles and $x, y, z, q_w, q_x, q_y, q_z$ being the cartesian position and quaternion orientation of the manipulated object. We denote by j the number of joint angles and by $k' = j + 7$ the number of state variables. We do not consider velocities or accelerations separately but compute them via finite differencing from position data:

$$v_i = \frac{q_i - q_{i-1}}{h}$$

$$a_i = \frac{v_{i+1} - v_i}{h} = \frac{q_{i+1} + q_{i-1} - 2q_i}{h^2}$$

When computing the difference of quaternions we convert it to a 3D rotational velocity. Therefore the size of the velocity vector is k, same as the degrees of freedom of our system and one less than the number of state variables. The number of optimization variables is therefore nk.

The inputs to the optimization problem are the sensor readings and control signals for all timesteps:

$$\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$$

and

$$\mathbf{U} = \{u_1, u_2, \ldots, u_n\}$$

Each sensor vector consists of l elements $s_i = (s_i^1, s_i^2, \ldots, s_i^l)$, which includes joint angle sensors, motion capture position and orientation, the IMU data as well as the force sensor readings.

At the core of the optimization lies the physics simulator MuJoCo. It is used to predict accelerations or torques, in forward dynamics and inverse dynamics mode respectively. In forward dynamics mode the simulator computes system accelerations for a given timestep, given position, velocity and control signal for each DOF. In inverse dynamics mode, given position, velocity and acceleration at a given timestep it computes the generalized forces for each DOF, including the unactuated ones, required to produce the given motion. Since we compare the predicted forces to the control signal it is convenient that they have the same dimension so we define the control signal for the unactuated DOFs as 0.

The basic dynamics formulas are

$$(\hat{a}_i, \hat{s}_i) = \text{fwd}(q_i, v_i, u_i)$$

and

$$(\hat{\tau}_i, \hat{s}_i) = \text{inv}(q_i, v_i, a_i)$$

In both cases $\hat{s}_i$ is the predicted sensor output given the state and control signal, produced using MuJoCo via a generative sensor model.

Since we are doing system identification, we also optimize over a vector of c system parameters: $\mathbf{P} = (p_1, p_2, \ldots, p_c)$, and generally we also have lower and upper bounds for these parameters: $p_i \in [l_i, u_i]$. The system parameters that we consider are detailed in section 2.3.2.

With this setup we can formulate the following optimization problems:

- Problem formulation using inverse dynamics

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{i=1\ldots n} \|\hat{a}_i - a_i\|^{*_3} + \sum_{i=1\ldots n} \|\hat{s}_i - s_i\|^{*_2}$$

- Problem formulation using forward dynamics

$$\min_{\mathbf{P},\mathbf{Q}} \sum_{i=1\ldots n} \|\hat{\tau}_i - u_i\|^{*_1} + \sum_{i=1\ldots n} \|\hat{s}_i - s_i\|^{*_2}$$

This optimization is done offline with the idea that the estimator part can be run at realtime once a suitable model is identified, as done in [16].

2.3.1 Cost terms

The difference between the two formulations is the cost term related to accelerations, in the case of forward dynamics, which is substituted to a cost term related to torques in the case of inverse dynamics. The norms used are denoted by $*_i$, meaning they are not just quadratic norms but can be arbitrary convex function. We call each element of the vector $\hat{s}_i - s_i$ a residual and generally we compute the norm of a residual vector r with the formula $\|r\| = \sum_i w_i f_i(r^i)$, where w_i is a weight and f_i can be any smooth convex function. For example we used the standard $f(r) = r^2$ or $f(r) = e^{r^2} - 1$, which behaves similarly to the quadratic close to 0 but acts more like a barrier function further away. The details of specifying norms and residuals are outlined in [6]. Table 2.1 shows the set of cost terms that we used to generate our results.

Table 2.1: Cost terms for forward dynamics. Acceleration terms are the difference between acceleration predicted from the physics engine and acceleration computed via finite differencing from the estimated trajectory. Sensor terms are deviations from sensor readings.

Term	*Units*	*Cost Term*
Force sensor	N	r^2
Torque sensor	Nm	$10^2 r^2$
Joint acceleration	$\frac{rad}{s^2}$	$10^{-5} r^2$
Joint position sensor	rad	$20 * 0.02^2 e^{\frac{1}{2}\frac{r^2}{0.02^2}}$
Object acceleration	$\frac{m}{s^2}$	$10^{-2} r^2$
Object rotational acceleration	$\frac{rad}{s^2}$	$10^{-4} r^2$
Object position Vicon sensor	m	$10 * 0.001^2 (e^{\frac{r^2}{0.001^2}} - 1)$
Object orientation Vicon sensor	rad	$10 * 0.005^2 (e^{\frac{r^2}{0.005^2}} - 1)$
Gyro sensor	$\frac{rad}{s}$	$10^{-1} r^2$

2.3.2 Physics parameters

The physics parameters that we optimize for fall in 3 categories. This first category is kinematic parameters. For example, in our datasets we have the object hanging on a string. This configuration introduces an inequality constraint (limiting the length of the string) in our system and MuJoCo models that with the same machinery as contacts, since an inequality constraint is very similar to frictionless contact. Therefore the string forces are greatly dependent on the position of its anchor point as well as the object. Similar to state information we can use a motion capture system to estimate the anchor position but we will have small inaccuracies again. Therefore we include the anchor position in our set of model parameters to optimize over. When alternating trajectory optimization and model optimization, we noticed very slow convergence in the case of hanging object. The reason is that given an inaccurate anchor position, the best trajectory is one that is shifted by the same amount as the error of the anchor, and given offset trajectory, the best anchor position is similarly shifted. Our joint optimization approach avoids this issue by taking optimization step with both sets of variables together.

The second category are intrinsic dynamic parameters of our robots. Those are the parameters that govern its interaction-free motion and include the positions of center of mass of all links, their masses and inertia matrices, as well as joint friction and damping. That is done in a separate process because those parameters are best inferred with a contact free behaviour.

The most important category is the parameters related to contacts. Many of the contacts encountered in manipulation tasks are with soft rubber-covered fingers. There are hard contacts as well, such as an object sitting on a table. In order to accommodate these different cases MuJoCo supports contact softness. When objects collide they experience each other's inertia. One way MuJoCo simulates softness is by scaling the apparent inertia depending on distance or penetration between the objects. In this paper the apparent mass scale factor increases quadratically with penetration. Another set of parameters is the more familiar spring parameters, namely damping and stiffness. Overall these parameters result in a position dependent mass on a spring damper system for contacts. The details of the MuJoCo contact model are explained in [35]. Of course the other important parameter is the friction which determines the shape of the friction cone. These are all parameters that we are interested in optimizing, with the option of them being different between different pairs of objects.

2.4 Optimization Algorithm

We solve the optimization problem with Newton's method. It is a second order numerical optimization method, meaning it requires both gradient and a Hessian with which we iterate the solution until convergence. The method is summarized as follows: $x^i \leftarrow x^{i-1} - \alpha\mathcal{H}^{-1}g$. The derivatives are computed via finite differencing, as our physics simulator is not yet capable of producing derivatives of the dynamics. Since our optimization variables are non-homogeneous (model parameters and state variables), the computations follow different paths.

Since a point in our trajectory only affects the dynamics of the neighboring timesteps, computing the gradient is not very computationally expensive. It only takes $3kn+n$ dynamics evaluations to evaluate the cost and the gradient for a given trajectory. Again, since the effect of changing a trajectory point is only local, the trajectory Hessian is band diagonal and is also relatively cheap to compute via finite differencing. It will take additional $\frac{9k(k+1)}{2}n$ dynamics evaluations to compute it exactly. However, we do not actually use the real Hessian as it is not positive definite, which would make the use of Newton method infeasible. Instead we use an approximation of the Hessian: $\mathcal{H} \approx J^T J$, where $J_{ij} = \frac{\partial r_i}{\partial x_j}$, with r_i being residual number i and x_j being the j^{th} optimization variable. Computing J also allows us to compute the gradient by $g = Jr$. Computing J costs us $3nk$ dynamics evaluations and computing r is done by evaluating the dynamics for each timestep, hence the total number of $3nk + n$ evaluations for both the gradient and the approximated Hessian.

Unlike a state variable, a change in a model parameter, in general, will change the dynamics for all timesteps. Therefore evaluating the parameters gradient and approximated Hessian costs us cn, where c is the number of parameters. Computing the part of the approximated Hessian between the trajectory variable and the model parameters comes at no additional cost, given we have the residuals from the trajectory perturbations and model perturbations.

A typical trajectory of 20 seconds at 200Hz with 14 state variable yields an optimization problem with 56000 variables. Computing $\mathcal{H}^{-1}g$ directly is impossible. Therefore we use Cholesky decomposition and backsubstitution. Fortunately the approximated Hessian is rather sparse with a band diagonal structure and there are fast algorithms for computing Cholesky decomposition and performing the backsubstitution. When we add c parameters to the optimization, the Hessian gets expanded with c dense rows and columns. Fast algorithms exist even in those cases and, overall, the computation of $\mathcal{H}^{-1}g$ takes less than 10% of the total execution time of our algorithm.

Our framework is flexible and it allows us to optimize only model parameters, only trajectory, or both at the same time. There are cases when trajectory optimization, which is computationally heavier, is not necessary (e.g. system identification of smooth dynamics), as well as cases when system identification does not make sense (e.g. running the system in realtime). Similarly to Wu et al. [42] we tried an EM approach to solve the combined problem, but noticed very slow progress in certain cases and moved to a full joint optimization.

When optimizing over just model parameters we found the Matlab Optimization Toolbox to be sufficient. However it did not scale well to hundreds of thousands of variable in trajectory optimization. Therefore we used minFunc[27] which provides a wide selection of algorithms and options but we found that Basic Newton method with Wolfe Line Search performs best for our problems. Still, even minFunc had trouble with computing $\mathcal{H}^{-1}g$, because, due to its huge size, $\mathcal{H}$, is numerically close to singular and the Cholesky decomposition algorithm fails, in which case minFunc was defaulting to the extremely slow eignevalue decomposition. We modified that part of the algorithm by adding some small constant to the diagonal of the Hessian in a Levenberg - Marquardt fashion so that it is clearly positive definite: $\mathcal{H} \leftarrow \mathcal{H} + \mu I \quad s.t. \quad \mathcal{H} \succ 0$.

2.5 Modelling

When defining the optimization problem we emphasize consistency over accuracy because accuracy is not well defined in the presence of modeling errors, and no physics engine can simulate the world with absolute accuracy. For example, a common way of representing robots is with joints and links, represented as rigid bodies. However, when forces are applied on such a structure long robotic links bend under the load. Then they no longer satisfy the rigid body hypothesis and in turn the joint angles lose their meaning as predictors of end-effector position via forward kinematics. Such non-rigidities are usually small but also much greater than the noise floor of joint-angle sensors. Another example is soft-body contacts. Like human fingers, robotics fingers are usually covered with soft rubbery material. Short of using Finite Element Methods there is no clear definition of an end-effector position. In

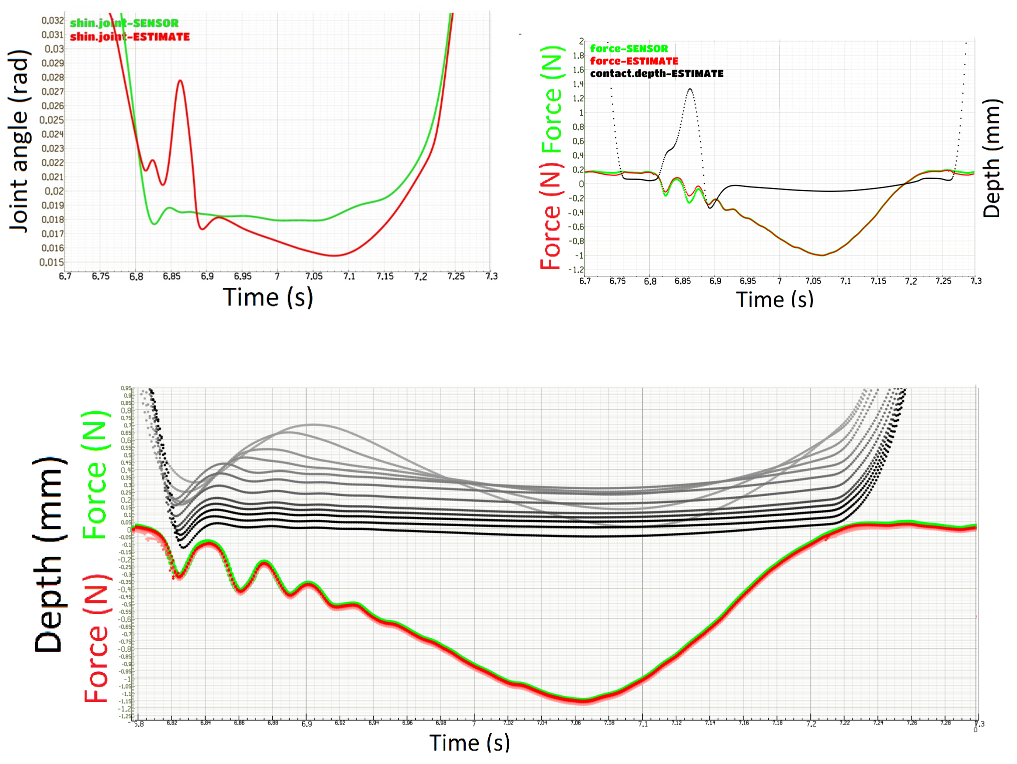


Figure 2.2: The top figures show the result of running our estimation algorithm on a sample run consisting of the robot tapping an object. The remote contact distance is set at $0.1mm$. Clearly seen are artifacts in the estimates of the joint angle (top-left figure) and the resulting penetration error (black graph on the top-right figure). This is a local minima of the optimization. Still, the sensor output was predicted well (red graph) matching the raw sensor output (green graph). The bottom figure shows the result of successive optimizations done with different contact sensing distances, varying from $2mm$ (gray graph) to $0.1mm$ **(black graph)**. Each optimization was seeded with the result of the previous, avoiding the local minima as a result.

this section we detail specific modelling choices that proved helpful in solving the system identification and estimation problems.

2.5.1 Remote Contacts

One feature that MuJoCo supports is contact forces between object that are not interpenetrating. As we mentioned in section 2.3 the apparent mass that the colliding bodies feel during contact depends on their interpenetration but we can extend that outside the boundaries of the objects so that forces are felt at distance greater than 0. That remote distance is another parameter that can be varied. This can be best thought of as an invisible soft pillow that cushions the impact between two bodies. This feature allows the trajectory optimization algorithm to see gradients and orient itself better in the almost discrete search space. By setting this parameter to be relatively large in the beginning we can guide the optimization process in its initial stages and then progressively decrease to 0 it to make realistic estimates. An example of our use of this idea is shown in figure 2.2. Also, that idea can help with control for hand manipulation [20].

2.5.2 Springy end effector

While the contact softening mechanisms in MuJoCo allows us to cope with rigid body violations in the normal direction of the contact, they still cannot explain tangential deformations. For example, when holding a heavy object the friction force would deform a human's skin in tangential direction, similar to what happens with the soft silicon rubber of the end effector. In order to cope with that we introduced extra joints for the end effector, therefore making the end effector non-rigidly attached to kinematic structure of the robot. The joints have spring-dampers attached to them that always aim to return them to nominal position. The idea is that surface deformations will be explained by load and displacement in those virtual springs. Naturally the spring-damper coefficients are another set of parameters we optimize over. Another benefit is that these springs can also explain bending and deformations of the

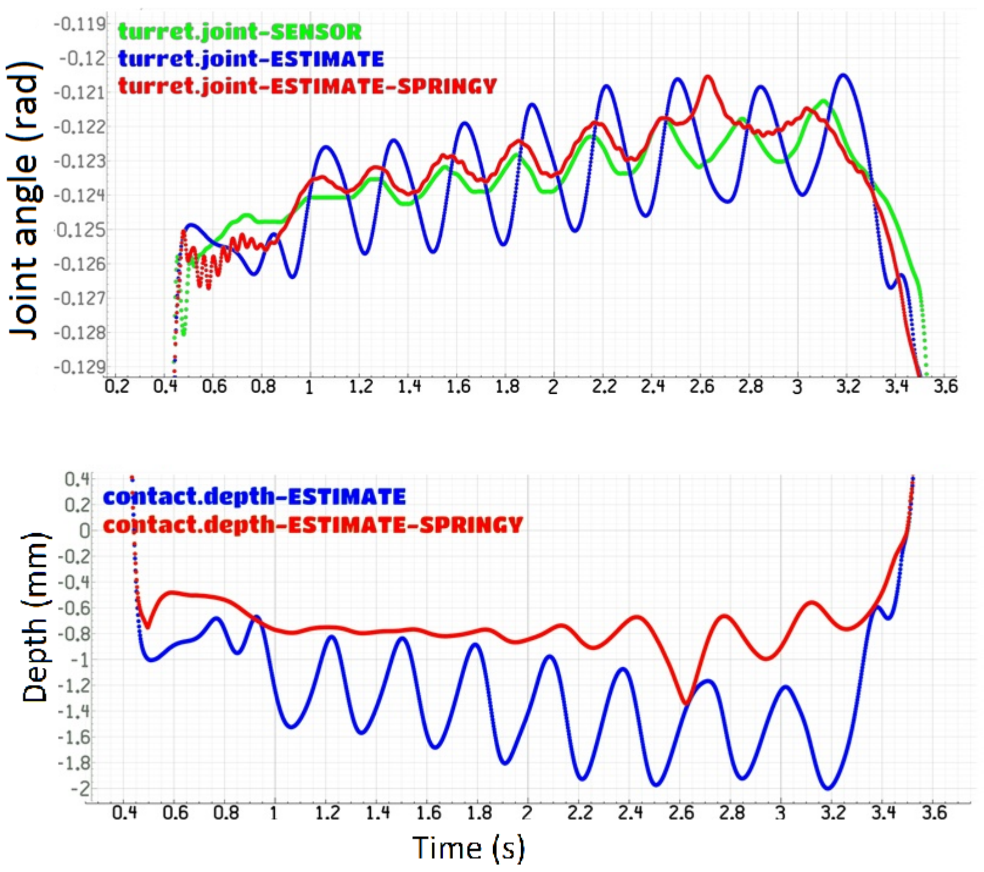


Figure 2.3: Example of the effect of modelling the end effector as attached with a spring to the rest of the robot. In this experiment the end effector applies force on the object with a circular motion, causing lateral deformations of its silicon cover. **Top:** an estimate of a joint angle. Two things are to be noted here: 1) The joint estimate when not using a springy end effector (blue graph) has much higher amplitude compared to the sensory reading (green graph). This is result of the compliant robot links. 2) It also is phase-shifted, as a result of the silicon cover. The estimate with springy end effector (red graph) tracks the sensory readings much better. **Bottom:** Estimated contact penetration between the finger and the object. The estimate with the springy end effector (red graph) is much flatter than the one with rigid end effector (blue graph). As the finger was pressed against the object throughout the experiment the contact depth should not exhibit $1mm$ fluctuations.

kinematic structure. Figure 2.3 shows how the springy end-effector helps explain the sensory data better.

2.6 Experimental Platform

2.6.1 Hardware Overview

We explore these algorithms with our Phantom Manipulation Platform. It consists of several Phantom Haptic Devices. We use each of them as a robotic finger. Each haptic device is a 3-DOF cable driven system shown in figure 2.4. Joints are equipped with optical encoders with resolution of about 5K steps per radian. The actuation is done by Maxon motors (Maxon RE 25 #118743) that despite the low reduction ratio are able to achieve 8.5N instantaneous force and 0.6N continuous force at nominal position. Even though they were designed as haptic devices the API allows for direct torque control which is what allows us to use them as general manipulators. The control loop is running at 2KHz.

For testing and developing our algorithms we use a single robot that performs various motions against an object. The robot was equipped with a silicon covered fingertip to enable friction and reliable grasping of objects. The softness of the rubber was one of the challenges that motivated us to carefully model contacts. There is a 6D force/torque sensor (ATI Nano 17) attached between the end effector and the robot. For our manipulation object, we used a 3D-printed cylinder with known sizes.

We also rely on Vicon motion capture system, which gives us position data at 240Hz. While being quite precise ($0.1mm$ error), the overall accuracy is significantly worse ($< 1mm$) due, in part, to imperfect object and manipulator models. The manipulated object is also equipped with an IMU (gyroscope and accelerometer) producing inertial data at 1.8KHz.

2.6.2 Data collection

When collecting data for system identification we want to have no unmodelled external perturbances. With our Phantom manipulation platform this is easy to achieve as we have

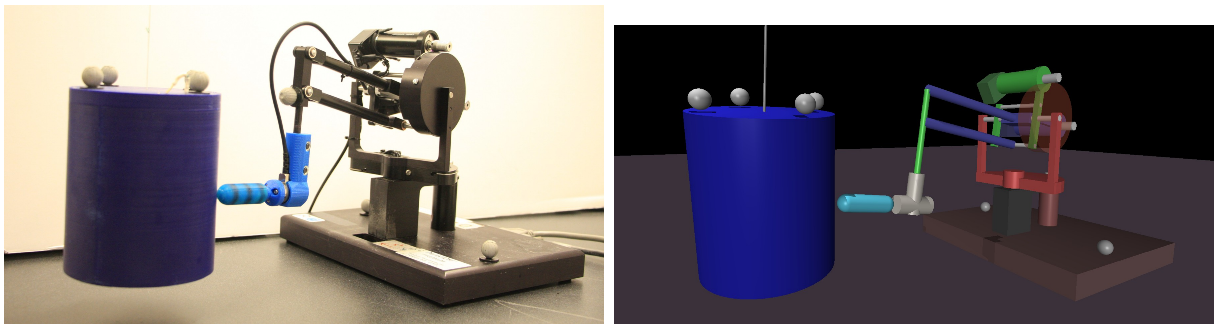


Figure 2.4: **Left:** A single phantom robot manipulating a 3D-printed cylinder. **Right:** Model of the scene in MuJoCo. The white spheres are Vicon markers used for motion tracking.

several identical robots and teleoperation is a viable option since they are back-drivable. The experimental data that involves interaction with the manipulated object was collected via teleoperation with force feedback. For identifying the intrinsic robot model, we used a PID controller following a predefined trajectory.

Each sensor had different update rates therefore input data is collected at different frequencies. It poses a great software engineering challenge to handle non-homogeneous input frequencies. Therefore we resample all inputs at varying frequency. We assign finer resolution to interesting parts of the trajectory (contacts between robot and object) and lower resolution when the robot is just moving in the air. The frequency thus varied between 200Hz and 1KHz. Inaccuracies produced by this operation are easily tolerated since our state variables are optimization variables as well and any problems introduced by this operation will be cleaned up in the optimization phase.

2.7 Results

Figure 2.6 is an excerpt of our results showing the loadcell readings during a typical tap of the end effector on the hanging object. We note the oscillatory nature of the contact, as well

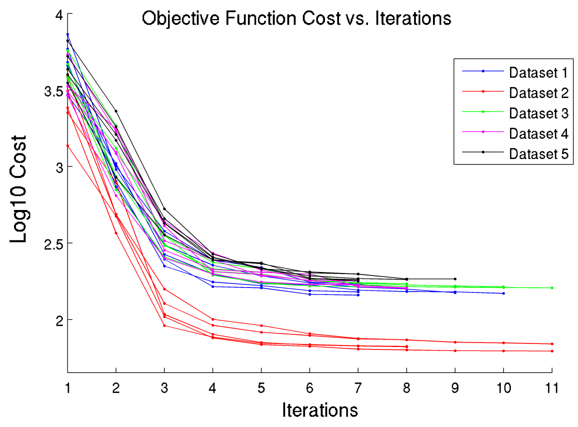


Figure 2.5: Shown are the optimization traces for 5 datasets, each dataset colored differently. Here we run only the estimation part of our framework and for each dataset we perform 5 runs each using different model parameters found as described in section 2.7.2.

the intermittent period within the contact duration where the force readings are flat. What happens is that the finger bounces off the object and contact reoccurs several times until a firm contact is established. This is due to the compliant structure of the robot as well as to the weight difference between the robot links and the object which is about 3 times heavier.

Figure 2.5 shows the objective function value as a function of optimization iteration. The overall shape is consistent across different datasets. The number of iterations required to reach close to the minimum is about 6.

2.7.1 Leave-one-sensor-out prediction

Here we show that our framework is robust to missing sensory readings. We run the estimation two times: 1) not penalizing for deviations from the force output of the ATI Nano17 and 2) not penalizing for deviations from the torque output. This simulates missing sensor readings. Figure 2.6 shows that our predictions of the missing sensor readings match the sensor output well.

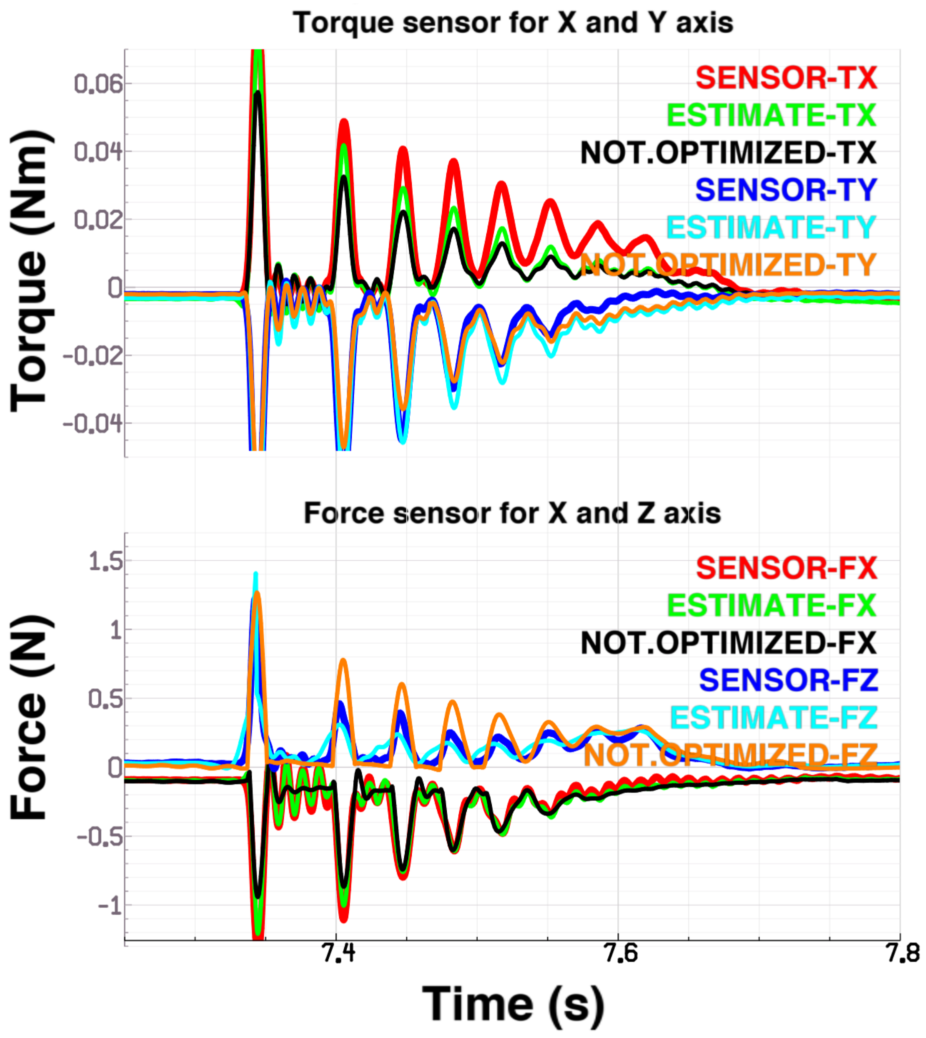


Figure 2.6: ATI Nano17 sensor readings during typical interaction between the end effector and a hanging object. Shown is the difference between the sensory readings (red and blue graphs), the estimates predicted by our physics simulator at the end of a full trajectory estimation, considering both sensors in the cost function (green and cyan graphs) and the estimates generated when not penalizing deviations of the corresponding sensor channel (black and orange graphs). While considering all sensor inputs in the optimization gives us more accurate estimates, omitting some sensors does not result in overfitting and our framework correctly predicts the missing sensor values. Only 4 out of the 6 sensor channels are shown since the other two closely match the shown channels.

Table 2.2: Model parameters values from different datasets

	Parameter			
	Coefficient of friction	*Contact stiffness*	*Spring stiffness*	*Spring damping*
Dataset #		$\frac{N}{m}$	$\frac{Nm}{rad}$	$\frac{mNm}{\frac{rad}{s}}$
Nominal	1.000	100	50.0	10.0
$ID\#1$	0.247	221	48.4	19.9
$ID\#2$	0.288	94.8	38.5	3.2
$ID\#3$	0.268	196	41.9	16.6
$ID\#4$	0.213	292	46.9	12.4
$ID\#5$	0.250	338	41.4	20.2

2.7.2 Cross-validation between datasets

Here we show that our framework is robust to overfitting. We perform joint system identification (sysID) and estimation on each of 5 datasets independently. The model parameters found in each of the 5 runs are shown in table 2.2. Then we perform only the estimation part of the algorithm on all 5 datasets using the model parameters from: 1) hand tuned model parameters and 2) the five sets of parameters found found previously when running sysID (labeled $ID\#1 \ldots ID\#5$) on each of the datasets. The minimized objective function values are summarized in table 2.3. Within each dataset the cost when using the parameters estimated from the same dataset does not differ significantly from the cost when using parameters identified using the other datasets. In all cases, the resulting cost is significantly lower than when using the hand-tuned model parameters. The model parameters found using different datasets are fairly close with few exceptions. The contact stiffness and the damping of the end-effector spring parameters vary between the different datasets. This is discussed in the next section.

Table 2.3: Cross-validation between different datasets

	Model parameters from					
Dataset #	*Manual*	*ID#1*	*ID#2*	*ID#3*	*ID#4*	*ID#5*
1	246.7	**112.8**	122.8	103.2	114.4	106.3
2	124.6	60.96	**50.69**	58.45	56.10	56.97
3	296.8	126.0	154.8	**133.4**	126.7	151.9
4	282.7	143.2	144.4	126.9	**128.2**	137.3
5	342.8	118.5	157.9	111.0	171.0	**147.1**

2.7.3 Robustness to parameter initialization

We run our algorithm on the same dataset, starting with 33 different initial parameter values. Table 2.4 shows the resulting distributions of final cost and the estimated parameters. Again we note that the contact stiffness and the damping of the end-effector spring parameters have bigger variance. This indicates that the datasets do not contain enough variability of dynamical motion which will be addressed in future work.

2.8 Conclusion and Future Work

We present a framework for system identification and dynamically consistent state estimation. We show results for basic robot-object interactions. By enabling consistent state estimation we hope to improve model based controllers in the real world. We show that using a hand-tuned model parameters produces worse estimates than when using system identification together with the estimation.

There are two venues for future work. First, we will improve the optimization tools by

Table 2.4: Final values distribution

Parameter	Units	Mean	Variance
Final Cost		126.7	18.0
Friction coefficient		0.2770	0.0079
Contact softness	$\frac{N}{m}$	272.3	102.9
Spring stiffness	$\frac{Nm}{rad}$	41.3	0.93
Spring damping	$\frac{mNm}{\frac{rad}{s}}$	15.86	1.51

enabling constrained optimization in the parameter space. A problem we encountered is the optimizer asking our physics simulator to reason about non-physical values such as negative mass or coefficient of friction. Another improvement would be to optimize over different behaviors jointly as well as to increase the complexity of the scene by adding more robotic fingers.

While we performed various validations and show that our optimization is robust to change in input data, it remains to prove its effectiveness to control applications. The most important venue for future work is to implement realtime version of the algorithm and apply the realtime estimation in a model based controller. Closing the loop with a controller would be the ultimate validation tool as we can quickly determine what is the quality of the results needed to achieve certain tasks. Another advantage is that we will be able to check which modelling choices are helpful in estimation and not a hindrance in control. In general, a more complicated model would give us better explanatory power in the estimation phase but would make controller or planner's work harder. Keeping a sane middle level is very important and closing the loop is what would allow us to do so. We see our framework as a tool to discover new ways to model robots and their interaction with the environment in applications of state estimation and control.

2.9 Applications

Training agents with Reinforcement Learning has been challenging due to the requirement for large amounts of training data. Model-based Reinforcement Learning is an alternative to this approach, with the potential problem of mismatch between the real world and the simulated system.

Our framework has recently been used for successful simulation-to-reality transfer of Reinforcement Learning agent in (Lowrey et. al [17]). We used our system identification procedure to identify the models of 3 Phantom robots, as well as the dynamic parameters of the same cylinder. Then, using *Natural Policy Gradient* (explained in more detail in section 3.1.2) we trained an agent to control the 3 robots to push the object around so it follows a desired trajectory. The experimental setup is shown in figure 2.7.

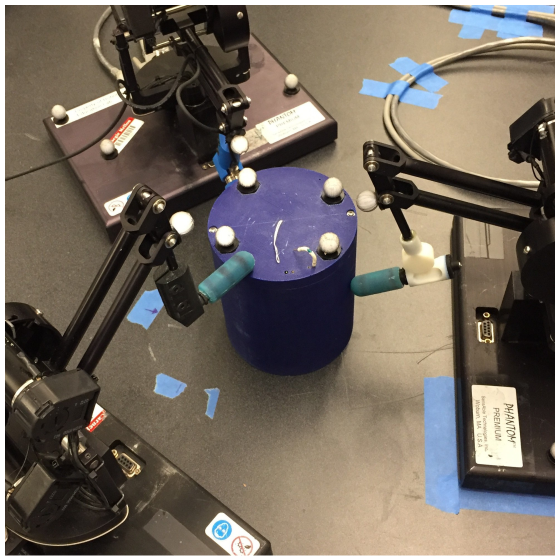


Figure 2.7: Phantom robots manipulating an object.

One of the contributions of this work is the training with an ensemble of models makes the learned policies more robust to modeling errors, thus compensating for difficulties in system identification. The mass of the cylinder is varied and then a policy is trained that works simultaneously on all the different masses is executed on the real system. Complete results and description of the methods can be found in [17].

Chapter 3

BACKGROUND

3.1 Reinforcement Learning

In our model comparison in chapter 7 we use Reinforcement Learning (RL) as a tool to analyze the differences between the two models. RL is a suitable environment in which to do such comparisons, because the product of Reinforcement Learning algorithm is a feedback policy that solves a given task. Due to exploration such policies are generally robust to small amounts of noise and model differences, hence, when executed on different models will highlight their differences quite effectively.

In this section we explain the basics of Reinforcement Learning. Section 3.1.1 lays the foundations and defines important concepts. Section 3.1.1 introduces policy gradient methods for solving RL as well as the basic algorithm *REINFORCE* [41]. Finally section 3.1.2 explains briefly an improvement to *REINFORCE*, called *Natural Policy Gradient*, introduced in [12] and its application to continuous control.

3.1.1 MDP Formulation

A Markov Decision Process (MDP) is defined as the tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \rho_0, \gamma, T\}$. $\mathcal{S} \subseteq \mathbf{R^n}$ is a set of states, $\mathcal{A} \subseteq \mathbf{R^m}$ is a set of actions that generally can be state dependent but in the particular continuous control setting that we have does not depend on the state. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R}$ is the reward distribution while $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition distribution. Often $\mathcal{R}$ and sometimes $\mathcal{T}$ are deterministic functions, like in our simulated environments, but for the purpose algorithm exposition we treat them as stochastic. Largely, algorithms are agnostic to whether $\mathcal{R}$ and $\mathcal{T}$ are deterministic or not. ρ_0 is a distribution over initial states. The basic premise of an MDP is that we have an agent starting according to the initial distribution, and in each turn the agent select an action from $\mathcal{A}$ and then transitions according to $\mathcal{T}$ and receives reward according to $\mathcal{R}$. Usually this behavior (an episode) continues for a number of steps T or until stopping criteria are met (at the terminal states) and the goal is to maximize the discounted total collected reward. Equation 3.2 explains the use of γ for discounting. There are many variations on the exact formulation, which are found in [32].

In order to simplify exposition we drop the hard requirement that episodes have length T by transforming the $\mathcal{M}$. We extend the state space by a time index t and denote all states at $t = T + 1$ as terminal states. Therefore we can treat each episode as potentially infinite and T as a random variable.

We seek to find a stochastic policy $\pi : \mathcal{S} \to \mathcal{A}$, a probability distribution over action space for each state, that governs the behavior of our agent. We define rollout as a sequence of state, action and reward triples

$$((s_1, a_1, r_1), (s_2, a_2, r_2) \dots (s_T, a_T, r_T)) \tag{3.1}$$

collected according to the MDP $\mathcal{M}$ and the policy π as follows: $s_1 \sim \rho_0$, $a_i \sim \pi(s_i)$, $r_i \sim \mathcal{R}(s_i, a_i)$ and finally $s_{i+1} \sim \mathcal{T}(s_i, a_i)$. Return is defined as a function of a rollout

$$G_t = \sum_{t=1}^{T} \gamma^{t-1} r_t \tag{3.2}$$

which can easily be computed from a rollout. The discount factor $\gamma \in [0, 1]$ prioritizes immediate rewards and it is useful to speed up learning, since we can never be sure what effect our immediate action would have on the far future. It is discussed in more detail in [30] and [32].

For a deterministic system it does not make sense to have a stochastic policy, but this formulation helps certain algorithms with exploration and eventually given enough training the distribution would collapse to a point given the chance [12]. We denote π_θ a policy parametrized by and differentiable w.r.t θ.

For a given MDP $\mathcal{M}$ we introduce the following useful functions. The value function is defined as

$$V^\pi(s) \doteq \mathbb{E}_\pi \left[G_t | s_t = s \right] \;=\; \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^t r_{k+t+1} \middle| s_t = s \right] \tag{3.3}$$

is the expected return starting from a state s and following the policy π. The Q function is very closely related and is defined as

$$Q^{\pi}(s,a) \doteq \mathbb{E}_{\pi}\left[G_t | s_t = s, a_t = a\right] \;=\; \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^t r_{k+t+1} \middle| s_t = s, a_t = a\right] \tag{3.4}$$

The difference between the value function and the Q function is that the Q function assumes the first action is chosen already. And finally we have the advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{3.5}$$

indicates how much better is choosing action a than the default policy behavior. It is of prime importance in learning policies.

Policy Gradient Methods

We define our objective function as a measure of performance as

$$J(\theta) \doteq \mathbb{E}_{s_1 \sim \rho_0}\left[V^{\pi_\theta}(s_1)\right] \tag{3.6}$$

The *policy gradient theorem* [32] states that

$$\nabla J(\theta) \propto \int_s \mu^{\pi_\theta}(s) \int_a Q^{\pi_\theta}(s,a) \nabla \pi_\theta(a,s) \tag{3.7}$$

Here the gradient is w.r.t θ and $\mu^{\pi_\theta}(s)$ is a probability distribution, the normalized expected amount of time we spend in state s following the policy π_θ. Now we can use data obtained from a rollout to approximate the policy gradient. We drop the subscript θ where appropriate. First we note that

$$\int_s \mu^{\pi}(s) \int_a Q^{\pi}(s,a) \nabla \pi(a,s) = \mathbb{E}_{\pi}\left[\int_a Q^{\pi}(s_t,a) \nabla \pi(a,s_t)\right] \tag{3.8}$$

because the probability of being in state s is approximated without bias by sampling from the policy in the form of rollouts [32]. Continuing with the substitution with real data

$$\begin{aligned} \nabla J(\theta) = g \propto \mathbb{E}_{\pi}&\left[\int_a Q^{\pi}(s_t,a) \nabla \pi(a,s_t)\right] \\ &= \mathbb{E}_{\pi}\left[\int_a \pi(a,s_t) Q^{\pi}(s_t,a) \frac{\nabla \pi(a,s_t)}{\pi(a,s_t)}\right] \\ &= \mathbb{E}_{\pi}\left[Q^{\pi}(s_t,a_t) \nabla \ln \pi(a_t,s_t)\right] \\ &= \mathbb{E}_{\pi}\left[G_t \nabla \ln \pi(a_t,s_t)\right] \end{aligned} \tag{3.9}$$

The last step follows because the return is an unbiased estimator of the Q function [32]. So finally we have

$$\widehat{\nabla J(\theta)} = \widehat{g} = \sum_{i=1}^{N} G_i \nabla \ln \pi_\theta(a_i, s_i) \tag{3.10}$$

where the triplets (G_i, s_i, a_i) are obtained from potentially numerous different rollouts. Now we have sample based estimate of the policy gradient $\widehat{g}$ and can implement the simplest iterative gradient ascent algorithm called *REINFORCE* [41]:

$$\theta^{k+1} = \theta^k + \alpha \widehat{g} \tag{3.11}$$

The return in equation 3.10 can be combined with any function that does not depend on the particular action taken. A popular choice is to subtract an estimate of the value function $G_t - \widehat{V}(s)$, which is an approximator of the advantage function. This idea can be further extended to *Generalized Advantage Estimation* strategy [30], which is what we use. For approximating the value function $\widehat{V}(s)$ we use a simple neural network.

3.1.2 Natural Policy Gradient

One problem with the standard *REINFORCE* algorithm is the difficulty of choosing the learning rate α. Moreover the algorithm is not invariant to the scale of the parameters, which generally necessitates low learning rate and that slows down optimization greatly. Specifically the units of ∇J are $\frac{1}{return}$, so the learning rate α has units of $\frac{parameters}{return}$ needs to map between parameter space and inverse return space, which is not an intuitive task, since the parameters, in general, have vastly different ranges.

One solution to this problem is to find a better search direction. The *Natural gradient*[1] in the form of $F_\theta^{-1} g$ is a better direction when the vector of parameters θ parameterize a probability distribution. Here F_θ is the Fisher Information Matrix. Kakade et. al [12] extends this idea to *Natural Policy Gradient* and provides sample based estimator for F_θ:

$$\widehat{F_\theta} = \frac{1}{N} \sum_{i=0}^{N} \left[\nabla \ln \pi_\theta(a_i, s_i)\right] \left[\nabla \ln \pi_\theta(a_i, s_i)\right]^T \tag{3.12}$$

Another way to arrive at same search direction is to form a local trust-region optimization problem around the current estimate θ_k, similarly to [31] and [25]. We linearize the reward around the current estimate and use the Fisher Information metric as the distance metric.

$$\begin{aligned} \min_{\theta} \quad & \widehat{g}^T(\theta - \theta_k) \\ \text{subject to} \quad & (\theta - \theta_k)^T \ \widehat{F_{\theta_k}} \ (\theta - \theta_k) < \delta \end{aligned} \tag{3.13}$$

This constraint based on the Fisher Information Metric ensures that the new parameter vector θ_{k+1} corresponds to a probability distribution such that average Kullback–Leibler divergence between the old and new distribution satisfies $D_{\mathrm{KL}}(\pi_{\theta_k} \| \pi_{\theta_{k+1}}) < \delta$. This has a nice intuitive interpretation: Modify the parameter vector so as to increase the total reward, while staying close to current policy, and exactly how close is determined by δ. The solution to this optimization problem [25] is given by

$$\theta_{k+1} = \theta_k + \sqrt{\frac{\delta}{\widehat{g}^T \ \widehat{F_{\theta_k}}^{-1} \widehat{g}}} \widehat{F_{\theta_k}}^{-1} \widehat{g} \tag{3.14}$$

This reduces the task of choosing a stepsize to selecting δ, which has a much easier interpretation than α in section 3.1.1.

Algorithm 1: Natural Policy Gradient

```
1  Initialize the policy parameters $\theta_0$
2  Initialize the value function approximation $\widehat{V}_0$
3  for $k \leftarrow 0$ to $K$ do
4      Sample episodes $(e_1, \ldots, e_N)$ by rolling out the current policy $\pi_\theta$
5      for each timestep t in each trajectory do
6          Compute return $G_t$ according to 3.2
7          Compute state value estimates $v_t = \widehat{V}_k(s_t)$
8          Compute generalized advantages $A_t$ according to [30]
9          Compute $\nabla_\theta \ln \pi_{\theta_k}(a_t, s_t)$
10     end
11     Compute the policy gradient $\widehat{g}$ according to 3.10
12     Solve $\widehat{F_{\theta_k}}^{-1} \widehat{g}$ using conjugate gradient and perform the policy update 3.14
13     Update the value function $\widehat{V}_k \rightarrow \widehat{V_{k+1}}$ using the return-state pairs $(s_i, G_i)$
14 end
```

Our implementation is based on Rajeswaran et. al [25] and a pseudocode is provided in algorithm 1.

3.2 Existing Models for Pneumatic Actuators

Modeling pneumatically actuated robotic system involves several components. The path of the air starts at our pressure source, usually a compressor. Then it passes through several on/off valves, condensers, purifiers, etc. and a thick pneumatic line that splits into many small individual lines. Each small line in turn, passes through (usually proportional) control valve, after which follows another line that leads to one end of a pneumatic actuator. In a kinematic system, usually each DOF's actuator is connected to two pneumatic lines due to inability to create negative (below atmospheric) pressure. The proportional valves also offer exhaust port. Figure 3.1 shows a diagram of pneumatic system with pneumatic actuators.

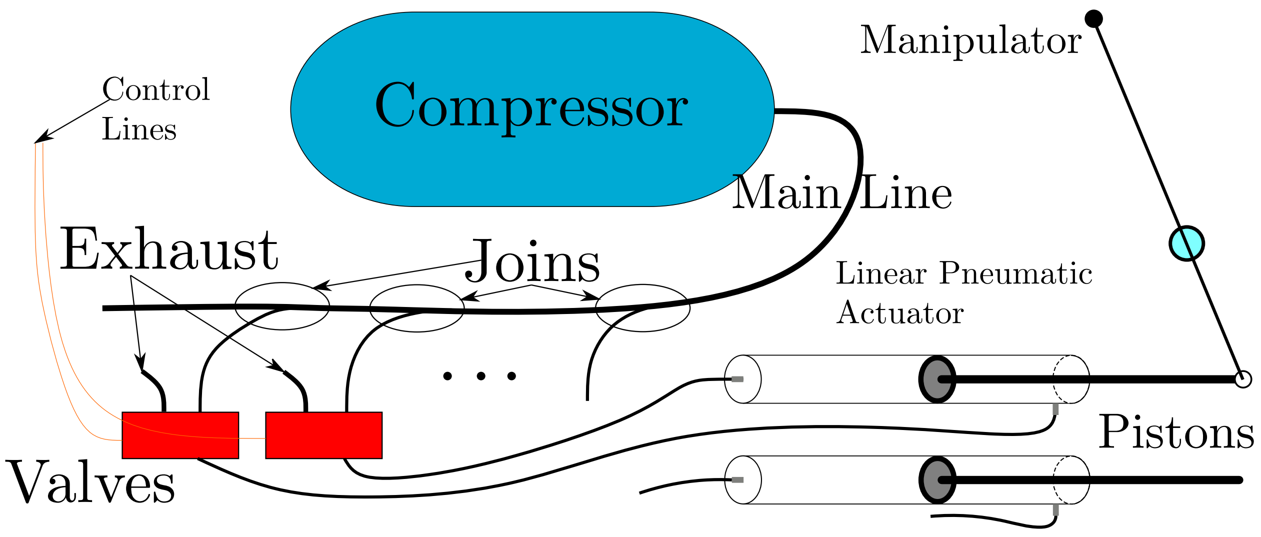


Figure 3.1: Pneumatically actuated robotic system

Previous research has focused primarily on modeling the flow through the final valve before the actuator [2][34][33]. The flow is traditionally assumed to satisfy the Orifice Plate (fig. 3.2) flow model that assumes a region of high pressure, a region of low pressure and a variable-area orifice in between to estimate the mass flow through the valve. Additionally there is a fudge factor included to account for imperfect setup (shape of orifice, etc.). Often that model is deemed insufficient and various data-fitted approximate equations have been developed in [2] and [33]. The details of this model and how it relates to our work is discussed in more detail in section 3.2.1.

Other research based on the orifice plate model include [9] and [10] which focus on estimating cylinder pressure without the use of sensors as a cost saving mechanism. While having a good estimator is very important, nowadays the cost of pressure sensors is insignificant compared to the rest of a pneumatic system. Moreover, having a good model allows for better observer as well as a better controller.

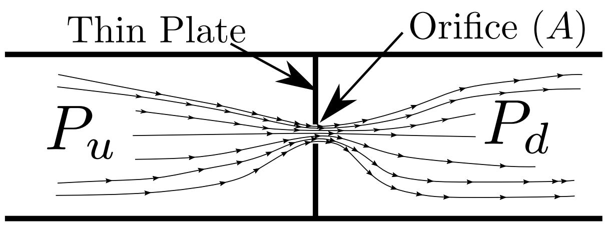


Figure 3.2: Thin Plate Model

When a command is sent to a valve there is a delay in the valve response. This is largely ignored in existing literature. Few exceptions include [2], which acknowledge it but decide not to model the valve motion. Only Hurmuzlu et al. in [26] model the spool motion but they do not do data fitting on the valve model at all and only assume circular orifice modulated by PID-controlled valve and calculate the flow through the valve using the Orifice plate model.

The delay due to the finite signal propagation speed in a pneumatic lines is largely ignored with the exception of [26], which is discussed in section 3.2.2.

3.2.1 Orifice Plate Model

The orifice plate model is the most commonly used for representing the flow through a pneumatic valve. It assumes there are two regions called upstream (higher pressure p_u) and downstream (lower pressure p_d) and a thin plate with an orifice between them (fig. 3.2).

Then depending on p_u, p_d and the orifice area the flow is assumed to be in one of two conditions: choked and un-choked. Choked flow means the flow is fully saturated and only depends on the upstream pressure. The conditions for choked flow are that

$$\frac{p_d}{p_u} < (\frac{2}{\gamma+1})^{\frac{\gamma}{\gamma-1}} \tag{3.15}$$

The un-choked flow is given by

$$\dot{m} = \frac{C_e A_\nu p_u}{\sqrt{RT}} \sqrt{\frac{2\gamma}{\gamma-1}\left(\left(\frac{p_d}{p_u}\right)^{\frac{2}{\gamma}} - \left(\frac{p_d}{p_u}\right)^{\frac{\gamma+1}{\gamma}}\right)}, \tag{3.16}$$

where R is the specific gas constant, γ is the heat capacity ratio of the gas and C_e is a discharge coefficient that depends on factors such as the shape of the orifice, etc. We refer to this as the fudge factor and is usually empirically estimated. The choked flow is given by

$$\dot{m} = \frac{C_e A_\nu p_u}{\sqrt{RT}} \sqrt{\gamma\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma+1}{\gamma-1}}}. \tag{3.17}$$

Some authors [2] choose to model the valve as a converging nozzle with variable minimal area, but the resulting formulas are the same. There are two problems with these approaches. First, the assumptions for a converging nozzle or orifice plate do not exactly reflect the reality of a valve internals. Usually the air travels through a maze of ducts and chambers and then escapes through a variable area orifice. However, the extra obstacles (fig. 3.3) along the way complicate the flow and do not resemble the theoretical basis of the model. This problem might be overcome with empirical fitting of the fudge factor as well as empirical relationship between valve command and effective orifice area, such as done by Todorov et al. [34]. Figure 3.3 shows a typical proportional valve.

The bigger problem perhaps is the requirement of knowing p_u and p_d. Most research assumes that p_u (or p_d in the exhaust case) is constant and equal to the supply pressure (or the atmospheric in the exhaust case). However, as mentioned, there is a long pneumatic line between the compressor and the valve and the pressure at the input of the valve drops whenever it is opened and there is a pressure gradient formed throughout the whole pneumatic

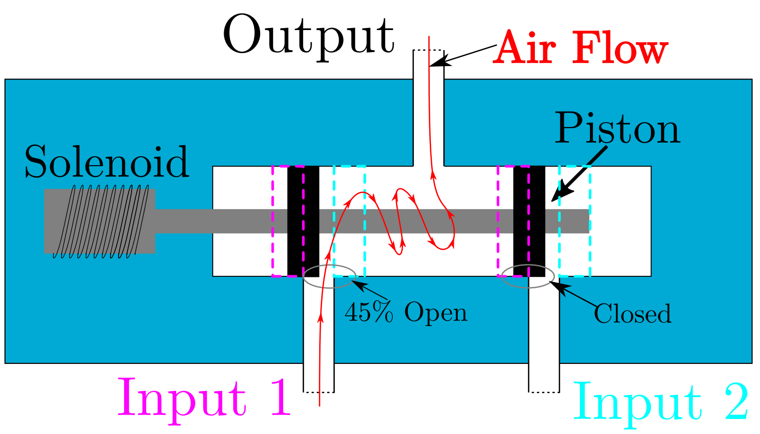


Figure 3.3: Proportional Valve in the $2/3$ configuration. It has 2 inputs and the piston controls which input and to what extent is connected to the output. Magenta shows the piston fully open towards Input 1, while cyan corresponds to the piston fully open to Input 2. The piston is usually actuated via a solenoid (see chapter 5.4).

line between the actuator and the compressor. This can be alleviated by using sensors at all the ports of each valve. However, that doesn't solve the problem that we cannot model the evolution of the upstream and downstream pressures and such model will have poor predictive capabilities, which is essential for agile control.

3.2.2 PDE-based model

Hurmuzlu et al. [26] present a PDE model of the air flow between the valve and the actuator. It is not the full set of equations that govern compressible flow. It is missing terms related to temperature and energy. They solve the simplified equation analytically, and derive delay and attenuation of the flow entering the actuator relative to the flow at the valve. This largely solves the propagation delay problem. Our work is originally inspired by this work and we attempt to model the full featured set of equations governing 1-dimendional compressible flow.

We extend the PDE idea to include the whole pneumatic line, including long actuators, as opposed to just modeling the line between a valve and its associated chamber. We also extend to handle multiple actuators connected to the same source input line.

While [26] again use the orifice plate model to model the valve flow, we chose to use simpler method for now, that naturally fits in the PDE framework. The specific choice of valve model is orthogonal to our work and can be swapped with any other model, which we intend to investigate in the future. Currently we aim to show the need for a model that can model flow through pneumatic lines more accurately.

Chapter 4
FLUID DYNAMICS BACKGROUND

This chapter gives the basic background knowledge in Computational Fluid Dynamics. In section 4.1 and 4.3 we present the main equations that govern air flow through a pneumatic line. In section 4.4 we present methods for solving those PDEs. The specific ways CFD is applied towards pneumatically actuated system is detailed in chapter 5.

4.1 Fluid Dynamics

We begin with a short overview of fluid modeling. More details can be found in Statistical Mechanics and Thermodynamics books. An excellent reference on the topic is [13]. This serves mainly to familiarize the reader with the topic.

A fluid is a collection of molecules and atoms each having position, mass, velocity as well as internal energy state. The internal energy only applies to molecules and represents the energy of the molecule rotation around its axes and other internal degrees of freedom (DOFs). Normally those particles have random velocities bouncing against each other in a random walk. While that is physically accurate such models are in general useless due to the impossibility of observing or simulating such large number of entities. Therefore in fluid analysis we have converged to space-averaged quantities, that describe the essential behavior of those particles in the limit at any given point. We have density (mass per unit volume), usually denoted with ρ and units of $\frac{kg}{m^3}$. With velocity a simple averaging would lead to information loss, therefore we split the velocity into a squared magnitude average (random motion) and regular averaging (directional motion). The simple average corresponds to fluid velocity at a given point in space and is represented with u and units of $\frac{m}{s}$, and can be represented by 1, 2 or 3 scalars depending on the dimension of the simulation. The averaged magnitudes give rise to another intuitive quantity: pressure (denoted by P and units of $Pa = \frac{kg}{s^2 m}$). The random motion as well the internal particle energy are together called internal energy (denoted with I and units of $J = \frac{\mathrm{kg\,m}}{s^2}$) together give rise to the sensation of temperature (denoted by T and units of K).

The 4 variables (ρ, u, P and T) are not independent. As it turns out the internal energy state and the random motion are tightly coupled and under ideal gas law we have $P = \rho RT$, where R is the specific gas constant. There are two more variables that are often used: *momentum* and *energy*. Momentum, M, (per unit volume) is specified in units of $\frac{\mathrm{kg\,m}}{\mathrm{s\,m^3}}$ and is computed by $M = \rho u$. Energy, E, (per unit volume) is specified in $\frac{J}{m^3} = \frac{kg}{\mathrm{s^2\,m}}$ and is computed as the sum of kinetic energy ($\rho\frac{u^2}{2}$) and internal energy $I = \frac{P}{\kappa-1}$, where κ is the heat capacity ratio of the gas. Interestingly, energy per unit volume has the same units as pressure. Collectively density, momentum and energy are called conservative variables since their integral over a closed system does not change in the absence of external forces.

The concepts of ideal gas law, specific gas constant and heat capacity ratio are terms from basic fluid mechanics and are explained in introductory books such as [22] and [40]. From now on we assume we have ideal gas law conditions, which stops being useful only at extremely high or low pressures and temperatures. For everyday phenomena, such as pneumatic actuators, we can assume it holds. We also assume the gas in question is standard air and κ and R refer to the specific constants for air. We also assume those constants are constant with respect to the state of the gas, which is a good approximation while the actual dependency can easily be incorporated.

The set of state variables used to represent the state in a simulation is not particularly important since is it straightforward to convert between them. The one place where it matters is for finite volume approximations where the non-linearity of the transformations between them comes into play and might violate the conservative laws. We chose to use density, velocity and pressure (ρ, u, P) as they are more intuitive and immediately useful, as well as making the derivation of the PDE easier to understand.

4.2 *Conversion formulas*

Table 4.1 shows the conversion formulas between all the dependent units and more importantly between the conservative units and the standard intuitive units.

Name	Symbol	Units	Relation with others
Density	ρ	$\frac{kg}{m^3}$	$\rho = \frac{P}{RT}$
Velocity	u	$\frac{m}{s}$	$u = \frac{M}{\rho}$
Pressure	P	$\frac{N}{m^2} = \frac{kg}{s^2 m}$	$P = (E - \frac{1}{2}\rho u^2)(\kappa - 1)$
Temperature	T	K	$T = \frac{P}{\rho R}$
Momentum per volume	M	$\frac{kg}{m^2 s}$	$M = \rho u$
Energy per volume	E	$\frac{kg}{s^2 m}$	$E = \frac{\rho u^2}{2} + \frac{P}{\kappa - 1}$

Table 4.1: Units conversion table

4.3 *Euler Equations*

What follows is a short sketch derivation of the basic equations of compressible gas flow. This particular approach is useful in understanding some of the solution methods in the next section, which would be non-intuitive with just a PDE as a starting point. A full detailed derivation can be found in [24].

We seek to find a relationship between the spatial and temporal derivatives of ρ, u, P. We only focus on the 1-dimensional case since, so momentum and velocity are scalars. To help derive that it is useful to think of a small interval δx (cell) and try to understand what happens to it in time δt, in particular to the quantities of mass, momentum and energy. We assume that right is the positive spatial direction. In the following equations A denotes the cross sectional area of the 1-dimensional pneumatic line we are simulating, which can vary with space but not with time so $\frac{\partial A}{\partial t} = 0$. The usual 1-dimensional equations one can

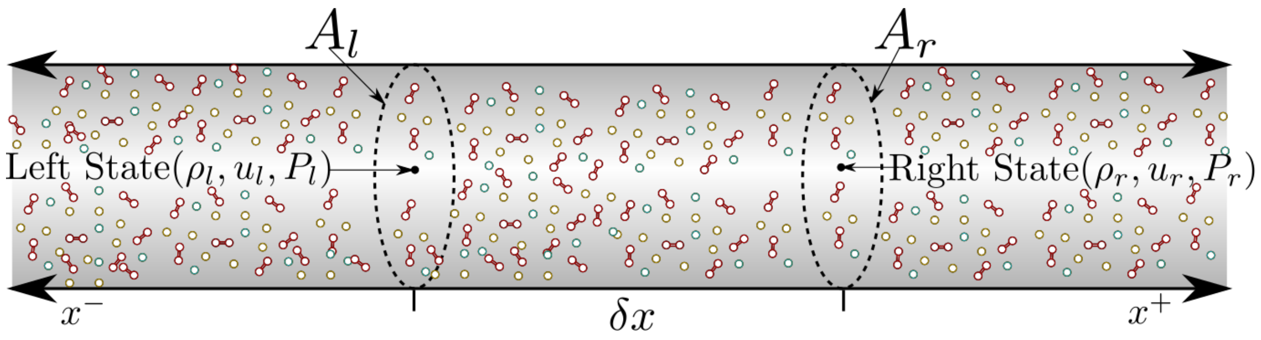

A_l
A_r
Left State(ρ_l, u_l, P_l)
Right State(ρ_r, u_r, P_r)
x^-
δx
x^+

Figure 4.1: 1-Dimensional fluid system

find in the literature do not have the area term, as with varying area the flow cannot be fully categorized as 1D due to transverse effects, it still is a very good approximation and is particularly useful for our purposes. Figure 4.1 shows a diagram with the infinitesimal cell δx and its walls.

The amount of mass present in a cell is $m = \rho A \delta x$. In time δt it changes because some of it left and some other mass came into the cell. Both can happen through either left or right boundary depending on the velocity of the fluid at those boundaries. Overall we have

$$rhoA\delta x|_{t+\delta t} = \rho A \delta x|_t + \rho_l A_l u_l \delta t - \rho_r A_r u_r \delta t \tag{4.1}$$

where l and r subscripts denote the left and right boundary respectively. In the limit of $\delta t \to 0$ and $\delta x \to 0$ we have

$$A\frac{\partial}{\partial t}\rho + \frac{\partial}{\partial x}(\rho A u) = 0 \tag{4.2}$$

The amount of momentum in a cell is

$$mu = (\rho A \delta x)u \tag{4.3}$$

The change in momentum is due to three factors. First is the amount of matter with given mass and velocity that enters or leaves the cell. At the left boundary, for example, matter with mass $\rho_l A_l(u_l \delta t)$ and velocity u_l enters the cell, for a combined effect of $\rho_l A_l u_l^2 \delta t$ change in momentum. The second effect is the force of the neighbor cell acting for a time of δt. At the left boundary that is $P_l A_l \delta t$. The third contribution is the wall reaction force due to change in area given by $P(A_r - A_l)$. The total formula becomes

$$(\rho A\delta x)u|_{t+\delta t} = (\rho A\delta x)u|_t + \rho_l A_l u_l^2 \delta t - \rho_r A_r u_r^2 \delta t + P_l A_l \delta t - P_r A_r \delta t + P(A_r - A_l)\delta t \quad (4.4)$$

which in the limit of $\delta t \to 0$ and $\delta x \to 0$ becomes

$$A\frac{\partial}{\partial t}(\rho u) = -\frac{\partial}{\partial x}(\rho A u^2) - \frac{\partial}{\partial x}(PA) + P\frac{\partial A}{\partial x} \quad (4.5)$$

Similar analysis can be applied for energy as well. Energy in a cell is

$$A\delta x(\frac{\rho u^2}{2} + \frac{P}{\kappa - 1}) = A\delta x E \quad (4.6)$$

First source of change is the transport of matter with energy from its neighbors. On the left side we have $\delta t u_l A_l E_l$. Second contribution is the work done on/by our neighbors. On the left side we have force $A_l P_l$ acting for a distance of $u_l \delta t$ for a contribution of $A_l P_l u_l \delta t$. Overall we have

$$A\delta x E|_{t+\delta t} = A\delta x E|_t + \delta t u_l A_l E_l - \delta t u_r A_r E_r + A_l P_l u_l \delta t - A_r P_r u_r \delta t \quad (4.7)$$

which in the limit of $\delta t \to 0$ and $\delta x \to 0$ becomes

$$A\frac{\partial}{\partial t}(\frac{\rho u^2}{2} + \frac{P}{\kappa - 1}) = -\frac{\partial}{\partial x}(Au(E + P)) = -\frac{\partial}{\partial x}(Au(\frac{\rho u^2}{2} + P\frac{\kappa}{\kappa - 1})) \quad (4.8)$$

If the area is constant we can drop it and simplify the equations. If we extend to 3 dimensions, we would need to account for contributions from all 6 neighbors as well as add equations for the other two components of velocity. All can be still written succinctly in a vector form. The analysis is still the same with basic application of Newtonian physics. From those equations various simplifications for certain applications, such as treating ρ as constant for liquids, since they are, in general, not compressible, or discarding ρ for sound simulation since it can be assumed it is proportional to P. In our case, however we need all 3 variables.

The difficulty in fluid dynamics simulation is that in 3 dimensions it is extremely computationally expensive, as often we need to capture tiny phenomena caused by the chaotic behavior of the system, which requires small δx. Fortunately in our case we can model the system as 1-dimensional, albeit imperfectly.

4.4 Solution Methods

One of the most important considerations in solving computational fluid dynamics problems are the choices regarding discretization. With time, we assume we know the state of the system at time t and wish to obtain the solution at time $t+\delta t$. Things are more interesting with the spatial discretization. Generally we split our region of interest into smaller cells. There are two main ways to think about this discretization. Finite Difference (FD) approaches are concerned with the value of the center of the cell, while Finite Volume (FV) methods keep values that are the cell-averaged value for a particular variable. The difference is usually subtle, but generally FV methods allow better reasoning about conservation laws and are sometimes more intuitive.

For ease of notation, once discretized we are going to refer to time and space with integers. For example ρ_j^i will denote the value of ρ at time index i at cell j. We are going to start with the premise that we have the state of the system at time index n at each cell and wish to obtain the solution at the next time step $n+1$, separated by δt. Also, where relevant we are going to omit the area (A) as a variable to simplify exposition.

Section 4.5 explains the basic method for solving any PDE. Section 4.6 explains more advanced methods, which we ended up using at the end. Finally section 4.7 explains how we choose δt.

4.5 Direct Methods

The most straightforward way of solving the Euler equations is to integrate them directly. First we need to obtain the temporal and spatial derivatives of the quantities in the Euler equations. Within the equations we have only temporal derivatives on one side and only spatial derivatives on the other.

Let's first consider the temporal derivative of a quantity Q. We are only concerned with a single cell so we omit the subscript in this paragraph. An obvious strategy is to define

$$\frac{\partial Q}{\partial t} = \frac{Q^{n+1} - Q^n}{\delta t} \tag{4.9}$$

This is a standard approach with first order time derivative approximation. It can be extended to higher orders, such as Runge-Kutta, which is beyond our scope. For a concrete example let's take $\frac{\partial}{\partial t}(\rho u)$. There are two ways to discretize it. One is directly set $Q = \rho u$, where we obtain

$$\frac{\partial}{\partial t}(\rho u) = \frac{(\rho u)^{n+1} - (\rho u)^n}{\delta t} = \frac{\rho^{n+1}u^{n+1} - \rho^n u^n}{\delta t} \tag{4.10}$$

The other approach is to first expand the derivative:

$$\begin{aligned}\frac{\partial}{\partial t}(\rho u) =& u\frac{\partial \rho}{\partial t} + \rho\frac{\partial u}{\partial t} = u^{\mathbf{m}}\frac{\rho^{n+1} - \rho^n}{\delta t} + \rho^{\mathbf{m}}\frac{u^{n+1} - u^n}{\delta t} \\ =& \frac{u^m\rho^{n+1} - u^m\rho^n + \rho^m u^{n+1} - \rho^m u^n}{\delta t}\end{aligned} \tag{4.11}$$

Here m can stand for either n or $n+1$, which is another choice that we have to make, which we'll discuss in a bit.

Let's turn onto the spatial derivative of a quantity Q. Again, if Q is a composite quantity it is possible to first expand it so each derivative term appears individually like

$$\frac{\partial}{\partial x}(\rho u^2) = u^2\frac{\partial \rho}{\partial x} + 2\rho u\frac{\partial u}{\partial x} \tag{4.12}$$

Let $(\frac{\partial Q}{\partial x})_i^n$ denote the spatial derivative of Q at index i at time n. Next consideration is the order of the approximation. Forward

$$(\frac{\partial Q}{\partial x})_i^n = \frac{Q_{i+1}^n - Q_i^n}{\delta x} \tag{4.13}$$

and backward

$$(\frac{\partial Q}{\partial x})_i^n = \frac{Q_i^n - Q_{i-1}^n}{\delta x} \tag{4.14}$$

finite differences are examples of first order derivative approximation. One could combine them and achieve central difference

$$(\frac{\partial Q}{\partial x})_i = \frac{Q^n_{i+1} - Q^n_{i-1}}{2\delta x} \tag{4.15}$$

which is a second order approximation and very widely used. Higher order approximations can be derived as well. Spatial discretization is the tool that transforms our PDEs into a system of ordinary differential equations.

Let us now investigate what are the implications of all the choices we can make. The various combinations give rise to different algorithms which are outlined in detail in [38] and [21].

We can use the discretization strategies to turn our PDE into a regular system of equations, that we can use to solve for the next timestep. Our unknowns are $\rho_i^{n+1}, u_i^{n+1}, P_i^{n+1}$ for all i. If we expand the time derivatives from the left-hand side of the Euler equations, use present time $(m = n)$ in the time derivative approximation and evaluate the spatial derivatives at present time, we obtain very simple system of linear equations. Moreover the system is separable, so for each cell we can solve simple linear system with 3 equation. If we use the conservative quantities directly and do not expand the time derivative get even simpler equations that are all disjoint, except we need to derive the nominal variables from the conservative quantities detailed in chapter 4.2. If we use use the next timestep $(m = n + 1)$ in the time derivative approximation, then we obtain, again for each timestep, a system of 3 non-linear equations, which can be solved easily with an iterative method.

However easy, these methods require very small timestep and are impractical [38]. Moreover, if we use more than first-order spatial derivative approximations, the simulation becomes unstable very quickly [8] [38].

One change we can make is to use the next timestep for computing the spatial derivatives. This is exactly the same idea as an implicit Euler integration since we already converted the PDE to an ODE. Now we have our unknowns $(\rho_i^{n+1}, u_i^{n+1}, P_i^{n+1})$ in both sides of the equations and moreover, adjacent cells can no longer be solved independently. So we have to solve a giant nonlinear system. Fortunately the system has fairly easy form that allows for numerous iterative methods to be applied [21]. For example we can use Newton's method which is an iterative procedure that we can seed either with the state from the current timestep, or with the solution from the explicit integrator. For a nonlinear system

$$f_i(x_1, \ldots, x_m) = 0 \tag{4.16}$$

for $i \in [1 \ldots m]$, the basic formula is

$$x^{k+1} = x^k - J^{-1} f(x^k) \tag{4.17}$$

where J is the Jacobian of the system with

$$J_{ij} = \frac{\partial f_j}{\partial x_i} \tag{4.18}$$

The Jacobian of our system is band-diagonal, which allows for efficient solving of the update equation with either LU or Cholesky decomposition, as well as with second layer of iterative solvers such as Gauss-Seidel or Jacobi methods. The non-exact iterative solvers are preferable since we do not require the exact solution to the inner problem, as the outer step is itself approximate, and with an iterative solver we can make use of very few inner iterations, which is faster than full LU decomposition.

These are called implicit methods and are very stable and allow for large timestep. However, they are not very accurate and also require complex iterative solver.

There exists a solution to the accuracy problem with semi-implicit methods. Instead of computing the spatial derivatives at the current timestep or the next we can compute them as a weighted sum of both timesteps. This adds little complexity to the implicit method since the values at the current timestep are known. More specifically instead of explicit central differencing

$$(\frac{\partial Q}{\partial x})_i = \frac{Q^n_{i+1} - Q^n_{i-1}}{2\delta x} \tag{4.19}$$

or implicit

$$(\frac{\partial Q}{\partial x})_i = \frac{Q^{n+1}_{i+1} - Q^{n+1}_{i-1}}{2\delta x} \tag{4.20}$$

we can have a combination of both in the form of

$$(\frac{\partial Q}{\partial x})_i = \frac{Q_{i+1}^{n+1} - Q_{i-1}^{n+1}}{2\delta x}\alpha + (1-\alpha)\frac{Q_{i+1}^{n} - Q_{i-1}^{n}}{2\delta x} \tag{4.21}$$

where α can be any value between 0 and 1, with those extremes corresponding to explicit and implicit differencing respectively. Integration with $\alpha = \frac{1}{2}$ results in both a stable method and second order spatial accuracy, albeit still at the cost of complex iterative solver [38]. We did not pursue this line of methods further and instead focused on Godunov's method.

4.6 Flux Methods

4.6.1 Riemann Problem and Sod Shock Tube

An important problem in PDEs is solving the initial value problem composed of two piecewise constant initial regions. In gas dynamics it is called Sod shock tube with the conditions usually being constant temperature, zero velocity and different pressure in both sides. At the beginning of the simulation there is a shockwave (travelling at the speed of sound) plus two slower waves. There exist exact solutions for those simple kind of problems and simulating their propagation is a good tool to test fluid dynamics solvers. Figure 4.2 shows a sample solution.

The Sod shock tube problem is particularly important for us since it mimics the situation where the control valve in our pneumatic system opens suddenly and the compressed air rushes through the pneumatic line towards the cylindrical actuator. Therefore we extensively used it to evaluate the algorithms as well as a debugging tool.

4.6.2 Basic Flux Method

Let's go back to our derivation of the Euler equations and remember that at each cell the change of mass, momentum and energy is due to interaction with its neighbors. Using the Finite Volume discretization approach we know the total amount of each of the conserved quantities in each cell. Therefore if we calculate the interaction of each cell with its two

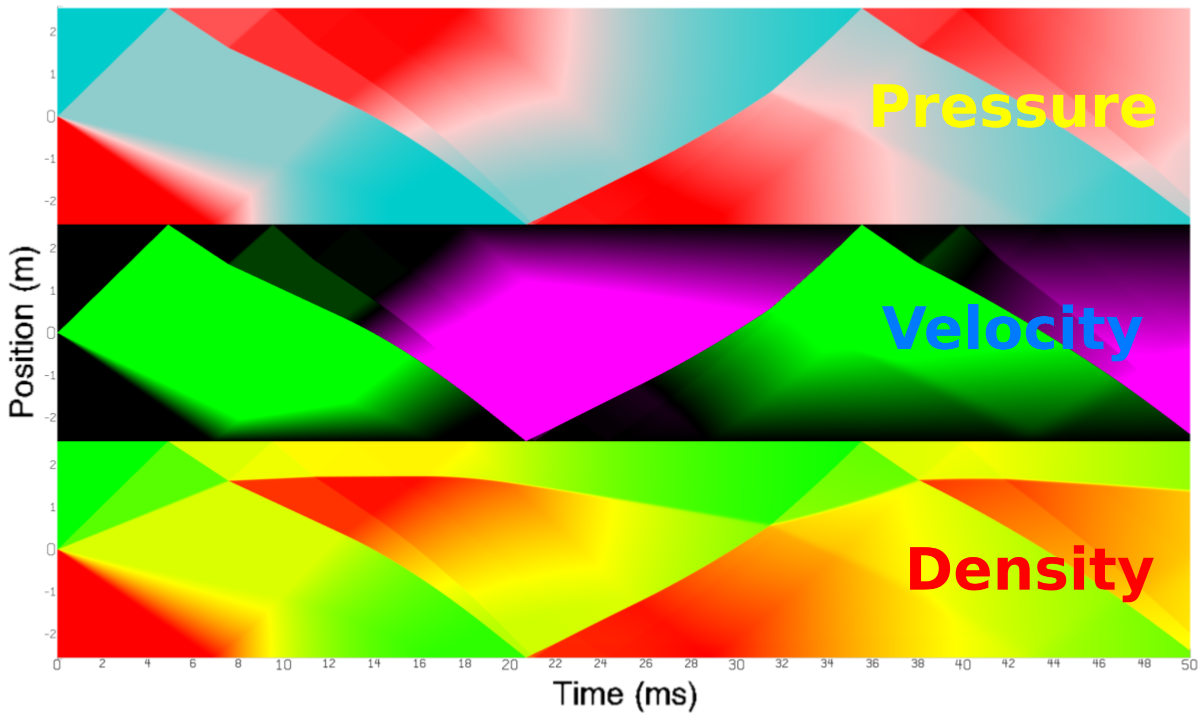


Figure 4.2: The three channels of the Sod shock tube solution. Simulation is $50ms$ of a tube of total length of $5m$. The two ends of the tube are sealed (just like a pneumatic cylinder), therefore the waves reflect and complex interplay between them happens.

neighbors at their boundary explicitly and add to the current amount we can update the system easily. As discussed in section 4.3, the two components of the interaction are the transport of quantities due to velocity and the effects of the force due to pressure. Collectively they are called flux and the amount of flux that the left cell loses is exactly the amount of flux that its right neighbor gains. Therefore if we compute the flux for each intercell boundary, we can use it to update both cells with just switching the sign. Flux has three components that are expressed in units of mass, momentum and energy per second.

The flux through a point in space is with given state (ρ, u, P) is:

$$\begin{aligned} &(A\rho u, A\rho u^2 + AP, u(0.5 * A\rho u^2 + AP\frac{\kappa}{\kappa-1}) + APu) \\ =&A(\rho u, \rho u^2 + P, u(\rho\frac{u^2}{2} + P\frac{\kappa}{\kappa-1}) + Pu) \end{aligned} \tag{4.22}$$

The flux between cells i and $i+1$ is then determined by the state at their intercell boundary. We denote that state by $\rho_{i+\frac{1}{2}}$, $u_{i+\frac{1}{2}}$, and $P_{i+\frac{1}{2}}$. For a quantity Q the most obvious solution would be to set it as the average of its neighbors $Q_{i+\frac{1}{2}} = \frac{Q_i+Q_{i+1}}{2}$. Using this approximation we obtain exactly the same numerical solution as solving directly with explicit integration and central differencing as outlined in section 4.5, which is very unstable.

4.6.3 Godunov's Method via Riemann solution

Instead of simply averaging both neighbors to find the value at the intercell boundary, one can consider the mini-Riemann problem formed at the boundary of the two cells. Akin to the Sod shock tube, we have two regions with different state. The solution to this problem will be locally valid (assuming the initial values are correct: see section 4.6.4) and using the part of the solution at $x = 0$ (figure 4.3) will provide much better approximation for the intercell boundary state. This is the basis of Godunov's method [36].

It is interesting that we only need the solution at $x = 0$, but that is the exact boundary between the cells and we only care at the state exactly at the boundary, as that would determine the flux between the cells. The solution to the Riemann problem is constant along a line of constant x/t, therefore we look for the solution at the $x/t = 0$ line (fig. 4.3). Of course this method is still an approximation, because the state is not guaranteed to remain constant in the duration of δt, but in practice gives much better results, nicely combining the power of numerical simulation with the analytical solution to a simpler problem.

There is in general no analytical solution to the Riemann problem for the Euler equations. However, once the pressure in the "star" region (see figure 4.3) is found, the rest can be found analytically. To find the pressure in the "star" region one has to solve a non-linear equation with Newton's method [36]. There are also various approximate Riemann solvers [36] that can be used directly, or as a seed to the exact method. The exact procedure can be found in [36]. Even though it is an iterative method, each iteration only involves a handful of scalar computations and we discovered that 2 iterations is enough to reach stable solution.

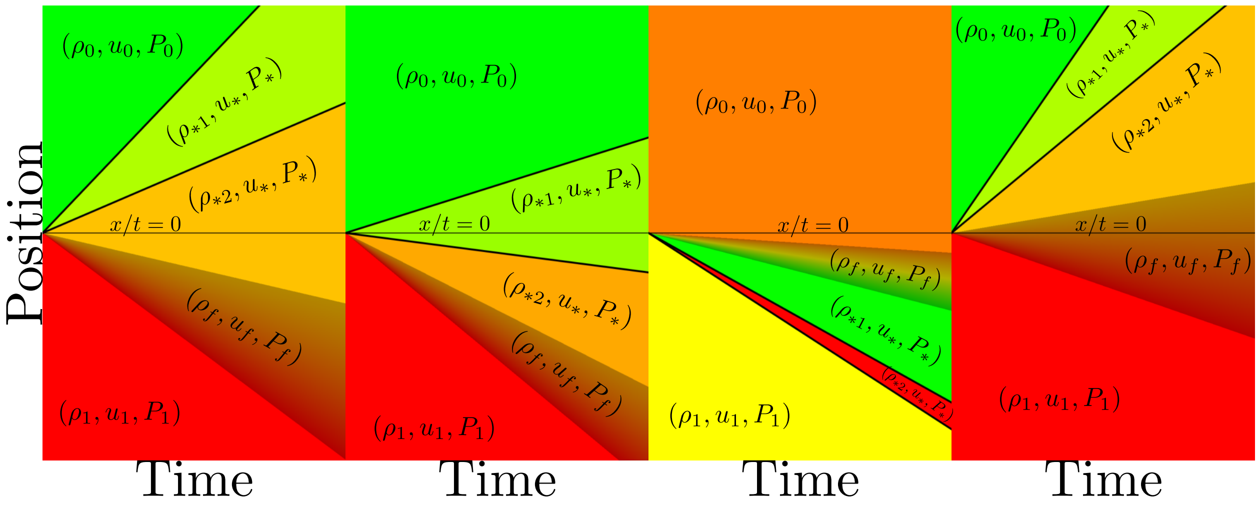


Figure 4.3: Example solution to few Riemann problems. The "star" region has constant u^* and P^*, while the density is split along the middle wave. The "fan" region (grayed area) has continuously varying state that can be easily found once the "star" region is solved. A complex solver is needed as the ${}^x\!/_t = 0$ line can be in any of the regions between the 2 waves (marked with solid lines) and the "fan" region. Shown are 4 cases, while in practive there can be 10 different relative positions of the ${}^x\!/_t = 0$ line, the two sharp waves and the "fan" region.

4.6.4 MUSCL Reconstruction

Godunov's method is first order accurate [36]. One way to increase the resolution of the method is to better approximate the initial values for the mini-Riemann problem. The basic version just assumes that the values are constant in each cell. Instead, we can allow for a linear variation like in fig. 4.4a. Then at each cell boundary ($i + {}^1\!/_2$) we use the rightmost value from the left cell ($u^L_{i+1/2}$) and the leftmost value from the right cell ($u^R_{i+1/2}$) as our inputs to the Riemann problem (fig. 4.4b).

One thing to note is that the reconstruction needs to happen in the set of conservative variables (ρ, ρu, E). We are doing linear extrapolation and while we are keeping the integral of the extrapolated variable the same, the non-linear relationship between (rho, u, P) and

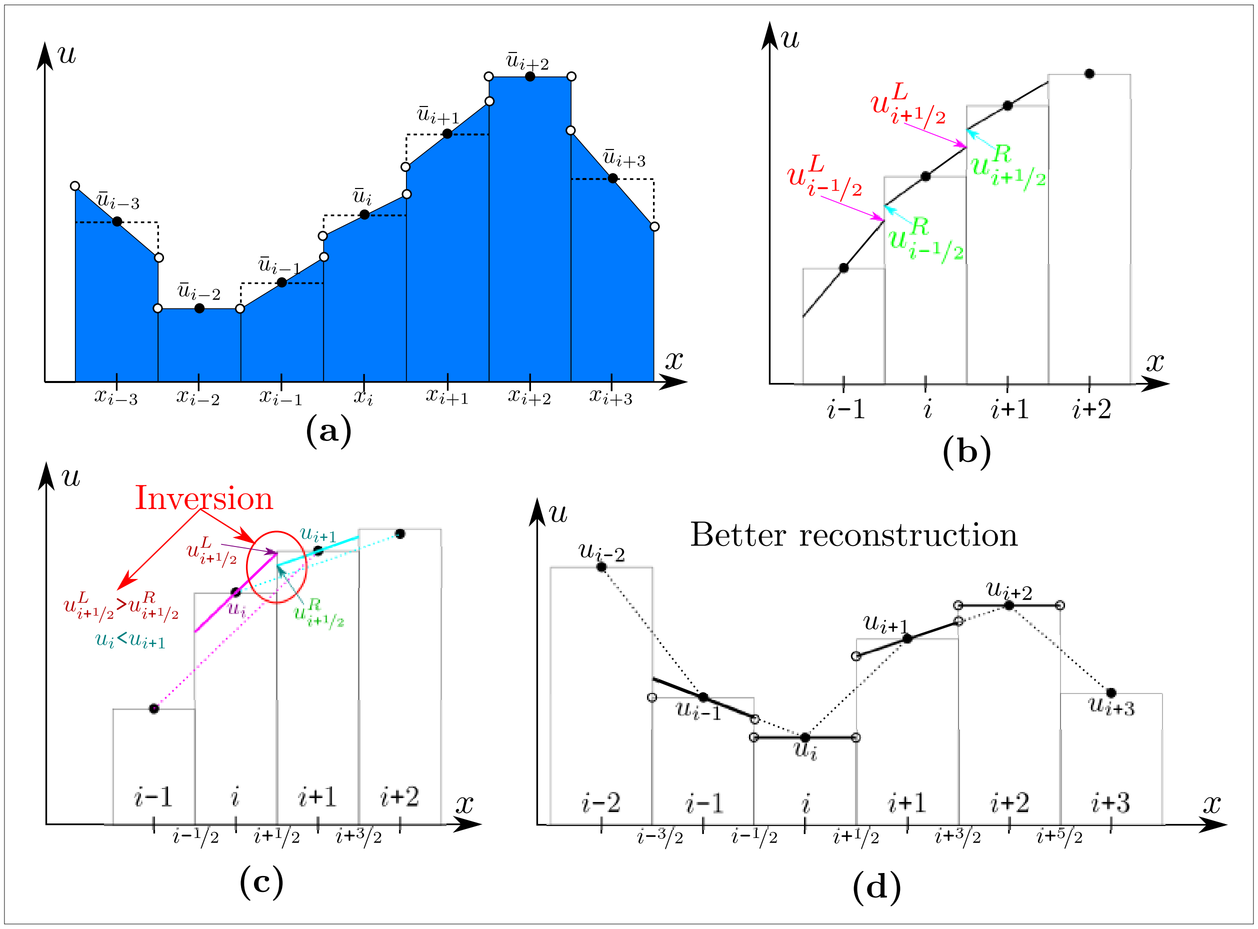


Figure 4.4: *MUSCL* Reconstruction

(ρ, ρu, E) means we are going to change the amount of the conserved quantities, which is physical violation.

One can simply approximate the slope in each cell using its two neighbors in a scheme akin to central differencing (fig. 4.4c). However that can lead to situations where the reconstructed values in the left cell is higher than the reconstructed value in the right cell, while the left cell value itself is smaller than the right one. This can lead to undesirable oscillations in the solution. A smarter way is to limit the slope of cell i to the smaller of the slopes to its two neighbors, as well as set it to zero (constant value in the cell) when the two neighbors are both above or below the current cell (fig. 4.4d). That ensures that the reconstruction does not introduce new extrema and therefore a stable solution.

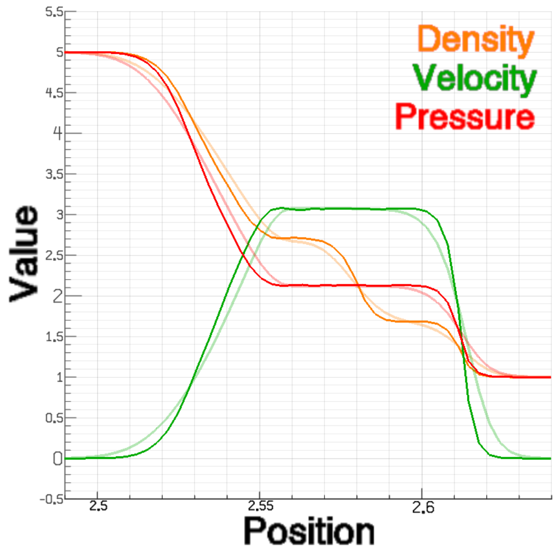


Figure 4.5: Comparison between *MUSCL* reconstructed simluation (solid color) and basic Godunov's method (light color).

That basic idea is at the center of *MUSCL*, which stands for *Monotonic Upstream-centered Scheme for Conservation Laws*. There is a lot of research into smarter ways to limit the slope using so called slope limiters [36], as well as schemes using higher order reconstructions, such as quadratic variation inside each cell. However, we did not investigate as we it turns out we do not need extreme spatial resolution. Using first order methods smears the sharp frontier of waves (fig. 4.5). Similar effects are also caused by the viscosity of air travelling through a small tube (see chapter 5.1) so although we had a basic version of *MUSCL* we did not use it for most of subsequent experiments.

4.7 Choosing δt

As with any physics simulation δt is one of the most important parameters. The Courant–Friedrichs–Lewy condition [4] normally governs the choice of δt:

$$C = \frac{u_{max}\delta t}{\delta x} \leq C_{max} \tag{4.23}$$

where C is called the CFL number and u_{max} is the maximum velocity of the conserved quantities we are simulating. Note that this is not the the physical fluid velocity, but the shock wave propagation speed (the radial lines in fig. 4.3). C_{max} is method-dependent number that is less than 1 for explicit methods such as some flux methods, including Godunov's method. For implicit methods it can be higher than 1, which allows for higher δt. Therefore we have the following requirement for δt:

$$\delta t \leq \frac{C_{max}\delta x}{u_{max}} \tag{4.24}$$

For values of $C_{max} = 1$, $u_{max} = 500{}^{m}/{}_{s}$ (the typical speed of the shock wave after opening the valve is higher than speed of sound due to the pressurized air) and $\delta x = 1cm$ we need to have $\delta t \leq 20\mu s$. In practice, however, we found that under those conditions we need $\delta t \leq 17\mu s$, otherwise the simulation becomes unstable.

The intuitive explanation is that at $C_{max} = 1$ in time δt, the shock will travel exactly one cell distance of δx. Higher δt would mean that cells that are not neighbors have exchanged information within a timestep. Most methods only assume neighboring cell interactions, therefore higher δt would violate that assumption. One possible reason for needing even smaller δt than what equation 4.24 would suggest is that within a cell we have influence from both sides and when the informations from both neighbors traveling in opposite directions meet, it creates non-linear interaction. Since the CFL condition gives us just a guideline, in practice we choose whatever δt gives us stable solutions for each particular problem.

Chapter 5

PDE BASED PNEUMATIC SYSTEM

We now present our solution to the components mentioned in chapter 3.2 of the pneumatic system, namely the pneumatic lines, the proportional valves and the actuator itself. Although the solution is via unified Partial Differential Equation (PDE) model, each component influences the air flow differently, so we present the each of them separately. This chapter is our work that builds on the basic CFD solution that we outlined in chapter 4.

Here we introduce the elements that deviate from the idealized model into a more realistic pneumatically actuated system. In section 5.1 we explore how pipe length and area affect the flow strength. Section 5.2 details how temperature considerations affect the flow. Section 5.3 explains how we can fit a valve model into our PDE as well as model cylinders and variable pipe cross sectional areas. Section 5.4 details the motion of the valve piston. In section 5.5 we move away from single line and onto more realistic pneumatic systems with interesting topologies. In section 5.6 we explain how the pneumatic system we built interacts with a general purpose physics simulator and allows for pneumatic control of arbitrary robotic systems. Finally, section 5.7 deals with implementation details.

5.1 Viscosity

When a fluid flows near a surface its velocity at the surface is zero and increases further away from the surface [11]. This is due to the viscosity of the fluid and the relative friction between the layers of fluid. There is friction between the layers, because they actually move at different velocities, with the velocity at the edge being 0 due to the no-slip condition [11]. This creates the boundary layer effect [11], or no-slip condition. Lower viscosity leads to sharper boundary layer and also retards the flow to a lesser degree. Although air is not

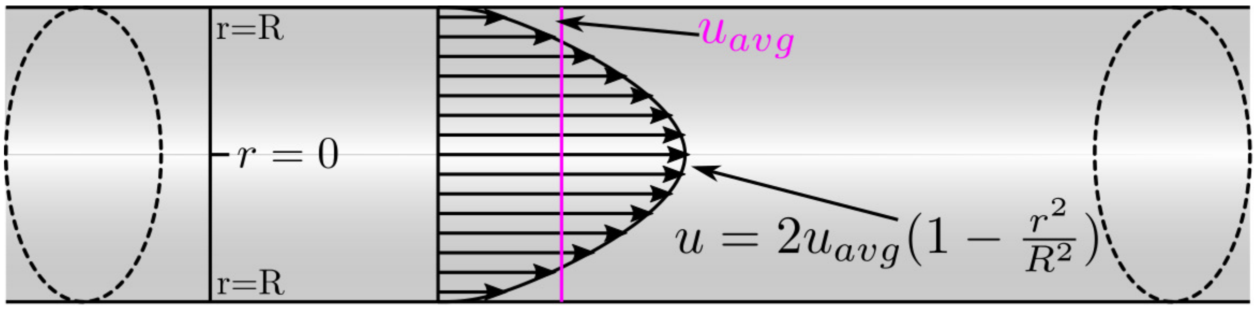


Figure 5.1: Hagen–Poiseuille velocity profile

particularly viscous the small diameter of the pneumatic lines makes the effect noticeable.

According to the Hagen–Poiseuille law [11], the velocity profile of a fluid flowing inside a tube is not uniform but follow a parabolic profile (fig. 5.1). If the average velocity in a tube of radius R is u_{avg}, then the velocity at a distance r from the center is given by

$$u = 2u_{avg}(1 - \frac{r^2}{R^2}) \tag{5.1}$$

The shear stress (force per unit area) between two layers of fluid moving at different velocities is given by $\tau = \mu \frac{\partial u}{\partial r}$, where μ is the coefficient of dynamic viscosity which is around $2 \cdot 10^{-5} \frac{N \cdot s}{m^2}$ for air. Using this one can show that the total restraining force at a cell with width δx is

$$F_r = 2u_{avg}\mu(2\pi)\delta x \tag{5.2}$$

As can be seen, higher viscosity leads to more restraining force, as do higher velocity and longer segment. It is interesting that the diameter of the tube does not affect the restraining force, which means that the effect will be much bigger in a narrower tube, as the same force will be acting on a smaller mass.

The only modification to the Euler equations then is to add a term $-F_r\delta t$ to the momentum equation. Intuitively, one might also want to incorporate the friction loss in the energy term. However, that would be wrong since the energy is not lost, just converted to heat and remains in the system. That would get taken care of automatically. For a given set of mass,

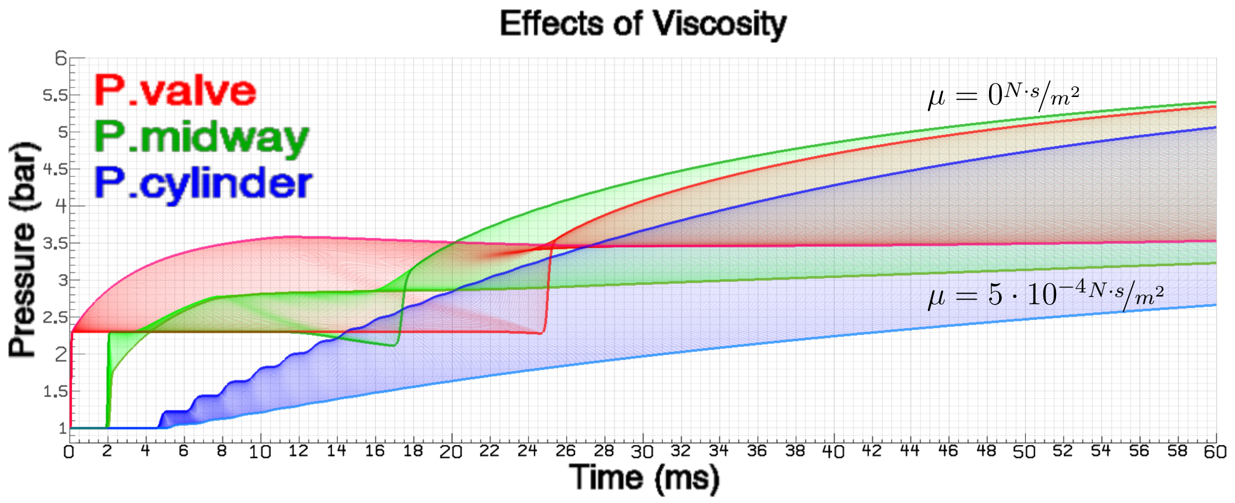


Figure 5.2: Pressure inside a pneumatic line and actuator at different viscosities. Red graphs represent a pressure tap immediately after the valve, green graphs correspond to pressure tap midway between the valve and the cylinder, while blue graphs are the cylinder pressure. The solid lines represent the values for $\mu = 0$ and $\mu = 0.0005$, while the shaded region is obtained by smoothly varying the viscosity between the two bounds.

momentum and energy, reducing the momentum leads to lower velocity, which lowers the kinetic energy, and with constant total energy, the temperature would rise.

Figure 5.2 shows the effect of adding viscosity to the simulation. Clearly viscosity retards the wave propagation, delays it and attenuates it. Overall there is less mass flow through the pneumatic system and it would take longer to fill up an actuator.

5.2 Temperature

Most previous pneumatic models assume constant temperature. However, that is incorrect since the rapid movement of the air from the compressed side to the cylinder leads to cooling at the cylinder entrance, as well as heating up at the far end of the cylinder. That can be even felt by touching it. In the situation of filling up an actuator with compressed air using the idealized Euler equations we will get huge temperature variations across the pneumatic line, even after allowing for everything to settle. However that is not realistic, and temperatures do return to room values due to contact with the pipe.

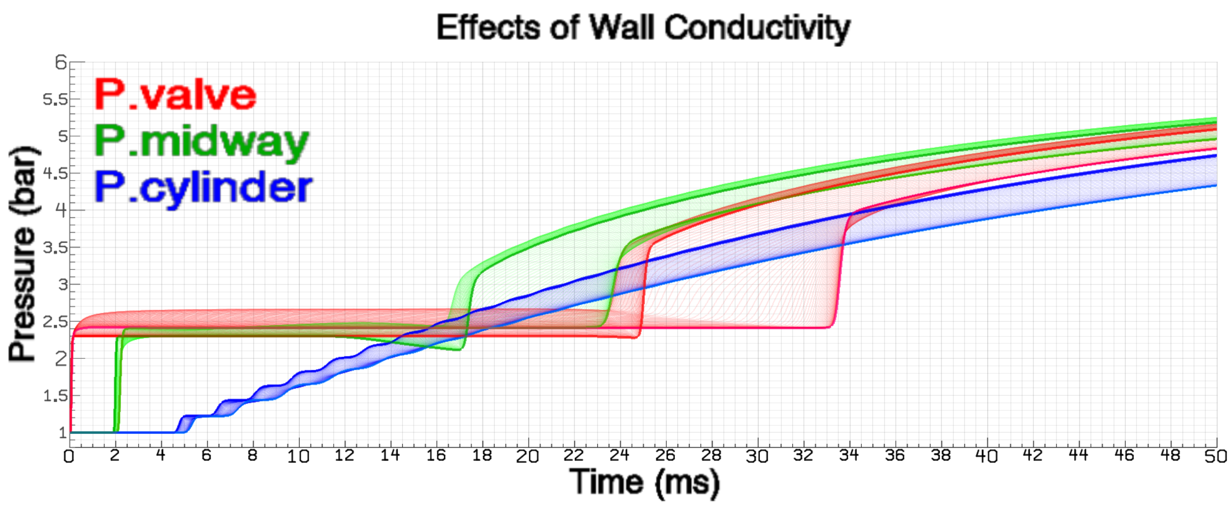


Figure 5.3: Pressure inside a pneumatic line and actuator at different pneumatic line conductivities. Red graphs represent a pressure tap immediately after the valve, green graphs correspond to pressure tap midway between the valve and the cylinder, while blue graphs are the cylinder pressure. The solid lines represent the values for $\kappa = 0$ and $\kappa = 0.0005$, while the shaded region is obtained by smoothly varying the conductivity between isentropic ($\kappa = 0$) and isothermal ($\kappa = \infty$).

We model that heat conduction simply as a product of contact area, temperature differential (between working fluid and room) and heat conductivity coefficient (κ). More concretely we add to the energy equation at each cell

$$(T_{room} - T)\kappa(\delta x\sqrt{A})\delta t \tag{5.3}$$

The reason the cross-sectional area (A) is under the square root is because the outer surface area of the cell is proportional to $\sqrt{A}$ and δx. The exact formula is not important since the conversion constants will be absorbed by κ, which is to be determined empirically, since this is an only an approximation to the real phenomenon. Example effect can be seen in fig. 5.3. While the air flow is attenuated, it is to much smaller degree, compared to viscosity.

Previous work in modeling pneumatic systems (chapter 3.2) have assumed either isentropic, or isothermal processes. Isentropic means, no energy leaves or enters the system (or in our case at least, no thermal energy, since the actuator is doing work). Isothermal assumes that the temperature of the working fluid is always constant. Both assumptions are wrong and the truth is somewhere in the middle. When $\kappa = 0$ thermal energy cannot escape the system so we have isentropic process, while $\kappa = \infty$ the fluid equalizes infinitely fast to the room temperature, giving us isothermal process. Varying κ, allows us to smoothly vary between the two regimes.

5.3 Variable Area and Valves

The whole pneumatic system consists of different lines and cylinders with different internal width or cross-sectional area. Main feed line from the compressor is usually thicker, while the transport line from the control valve to the cylinder is usually narrow. In our framework we model pneumatically actuated cylinders as just wider pipes. That is not strictly necessary, but simplifies the computational model somewhat. We handle this variation by specifying cross-sectional area for each cell of the simulation. At the intercell boundary we need to know the area though which the flux goes through. We use the smaller of the two neighbors as the effective area, unless otherwise specified (fig. 5.4).

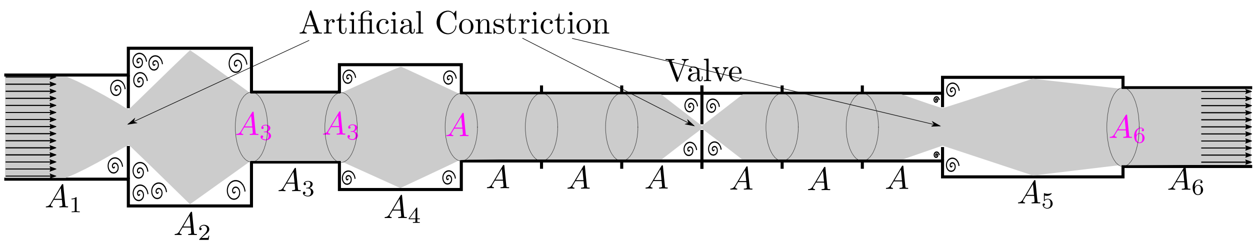


Figure 5.4: Modeling Variable cross-sectional area

This is just an approximation as the Riemann problem at the boundary between two different areas does not model effectively the reality. For example, such sudden obstructions usually lead to some amount of velocity loss due to turbulent flow. Figure 5.4 shows a sketch of the flow through variable area pipe. The gray region in the figure are where the majority of the flow is happening and the vortices in the corners are where some eddy flows happen. We have not investigated such additions to the model yet, but could be introduced as increased viscosity as those dead flows just convert kinetic energy to thermal energy. It is unlikely that such detail is needed or can be modelled correctly.

5.4 *Valve model*

One reason to manually select the effective area between neighboring cells is to model the action of a valve. Since we are working with a proportional valve by scaling the effective area down from the maximal opening we can approximate the desired behavior (fig. 5.4).

When the air flows from the inputs to the output of the valve, it has to navigate through complicated contraptions (fig. 3.3). Therefore, it loses part of its kinetic energy, which we model as added viscosity just for the valve-adjacent cell. Results are shown in 6.3.

In principle, the valve transmission mechanism can be modelled as a black box and using a function approximator compute the flow given the state at the ports and the state of the valve piston. That approach is compatible with all other methods of modeling pneumatic systems. However, we did not investigate such approach yet.

One interesting property of our proportional valves is the lag between setting the desired position and the valve actually going there. We found out that the behavior of the valve can be modeled as if controlled by a PID controller

$$\ddot{x} = \max(u_{max}, k_p(x - x_{target}) + k_d\dot{x} + k_i \int (x - x_{target}) \tag{5.4}$$

where x is the actual position and x_{target} the desired position set by the voltage reference given to the valve. Note that we use voltage as a proxy to valve position as the actual position is meaningless. What we need to know is that 10V corresponds to fully open towards input 1, 0V corresponds to fully open towards input 2 and 5V is closed towards both. Figure 5.5 shows the motion of the valve starting from all integer voltage levels and going to 0V as well

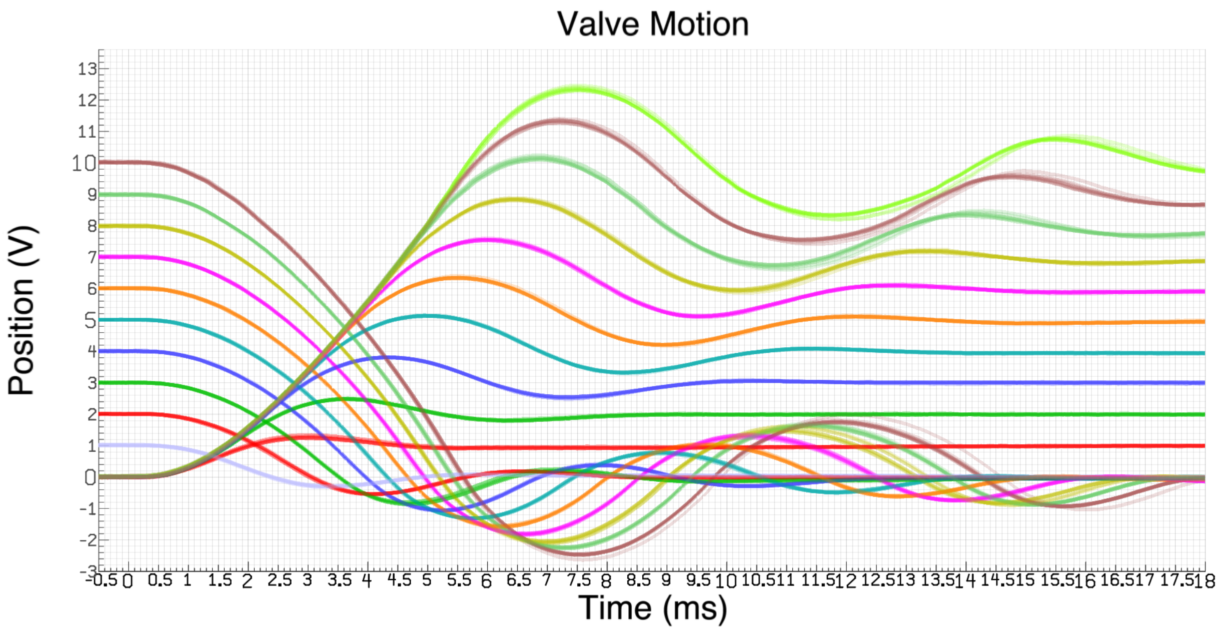


Figure 5.5: Valve motion: Each line is actually superposition of 10 different runs. They overlap almost perfectly, suggesting very realiable valve.

as starting at 0V and going to all levels. Each experiment is repeated 10 times and the valve performs remarkably consistent.

We show our results at modeling it in section 6.2.

5.5 Joints and Boundaries

So far we have only looked at infinite 1-dimensional pneumatic lines, without considering finite networks of pipes. Any realistic pneumatic system will consist of multiple valves and chambers, in fact each pneumatic cylindrical actuator consist of two chambers each connected to a separate valve, which itself has two more connections, one towards the pressurized air source and one towards the room (fig. 3.1).

First we need to model the boundaries of our system. Those include the piston wall, as well as the source and sink for the air (compressor and room respectively). The compressor and room are modeled as just cells at the end of a line segment with constant state: compressor pressure and ambient pressure respectively at room temperature and zero velocity. More interestingly boundaries of cells at end of a line that is blocked are modeled as not transmitting any actual flux. We still need to include the reaction force of the wall equal to $-PA$ for a contribution of $-PA\delta t$ to the momentum, which, at rest, balances the same magnitude of force from the other direction.

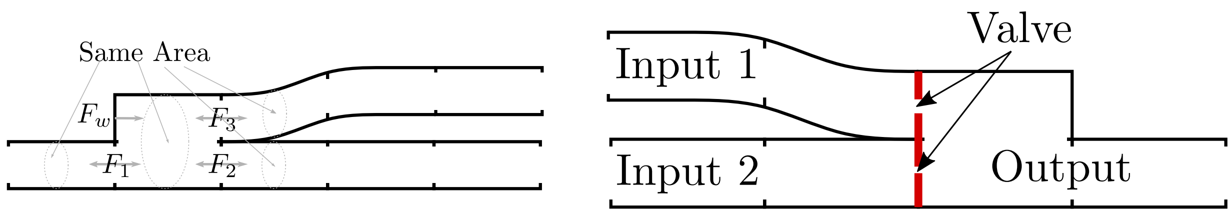


Figure 5.6: Pneumatic pipes linking. The **left** figure shows how we model general pipe segments joining, while the **right** shows how we can apply this idea to model a valve with 2 inputs and 1 output. Using the variable area mechanism we ensure that only a single of the inputs has positive effective intercell area to the output.

We model links between segments by attaching the first or last cell of a line segment to a cell in another segment. We then add one more neighbor to the cell at the point of attachment, with the flux computations following the same scheme. For each additional neighbor we add, we need to provide a fictitious wall at the opposite end of attachment to provide the reaction force (see fig. 5.6), otherwise momentum would start accumulating. That addition is quite necessary and is artefact of the fact that we model all of our flow as 1-dimensional with clear left and right directions. In reality a flow that splits in two lines cannot be truly 1-dimensional and will have component of velocity that is perpendicular to the nominal direction. However, we only consider 1-dimensional velocity as well, therefore we need such additions to approximate reality.

5.6 Physics Integration

Lastly we need to connect our pneumatic system to an actual dynamical system so we can do interesting things with it. So far we have described how we model the air propagation inside pipes and cylinders. The interaction with a physics engine has 2 parts. First, the pressure inside a cylinder causes force at the piston. That force is used as an input to a physics engine, which after computing the forward dynamics knows the velocity of the piston, which then gets input back to the pneumatic simulation. Figure 5.7 explains this visually.

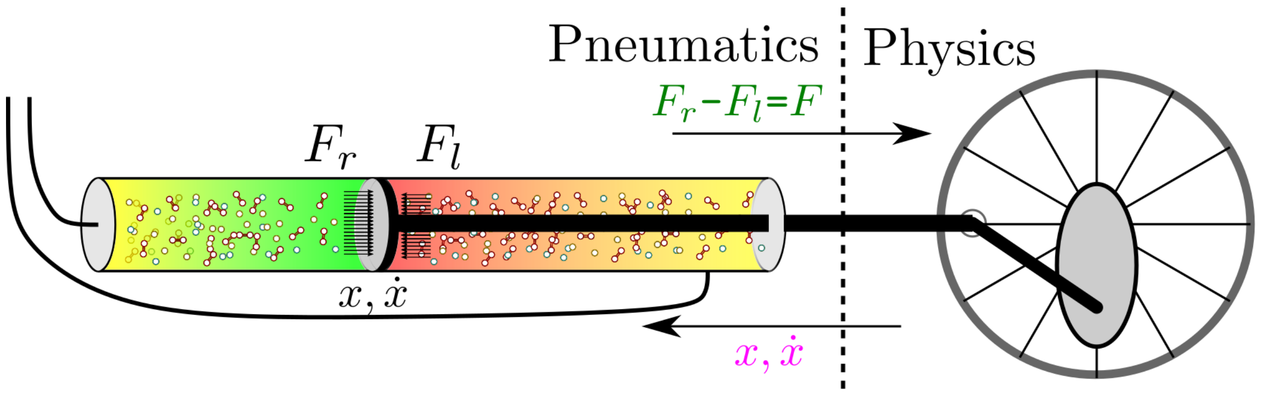


Figure 5.7: Integrating pneumatic model with a physics simulator.

After obtaining velocity from the physics engine we just set that piston to have constant velocity until the next update step. When the piston moves, the volume inside the cylinder changes with all the corresponding changes in internal energy. When the volume is shrinking the piston force is doing work to the fluid, while when the volume is expanding the fluid is losing energy (doing work) to the piston and the rest of the dynamical system. Moreover, since each cell in our simulation has fixed δx, as the piston moves the number of active cells inside the cylinder changes and we track such changes dynamically.

It is not important that this loop runs at pneumatic simulation speeds. We just use the timestep that is relevant for the physics simulation, and then the pneumatic system performs as many smaller steps at its own δt to match the larger timestep.

5.7 Implementation Details

To specify a pneumatic system one needs to list all the line segments, which includes their individual cells. For each cell we need to know the cross-sectional area (A), the length (δx) as well as the effective transmission areas to left and right neighbors. Moreover we need to specify between which cells is there a valve. Next we need to specify how are the line segments connected. Each end of a segment can either be closed (as in cylinder wall for example), constant state (compressor or room) or it can join another line segment. Finally we need to specify some of the line segments as cylinders.

That information then is used to distribute the computations on a GPU or a CPU. We currently have a CUDA implementation which due to the highly parallel nature of the task is significantly faster than the CPU implementation. The basic steps in the computation can be described as follows:

1. **Piston Motion — for each chamber**
    1.1. Change cell connectivity and sizes if needed.
    1.2. Update state of boundary cell due to work done by or to the piston.
2. **Collect data — for each cell**
    2.1. Gather neighbor information in case we need to do *MUSCL* reconstruction.
    2.2. Collect the state (rho, u and P) as well as A and δx.
3. **Valve motion — for each valve**
    3.1. Apply valve motion as per section 5.4.
    3.2. Save the resulting effective area for step 5..
4. **Reconstruct the state — for each cell**
    4.1. If first order reconstruction set left and right to be the same as average.

4.2. Otherwise compute the **MUSCL** using data from 2..

4.3. Save the left and right state for this cell.

5. **Flux computation — for each itercell boundary**

5.1. Compute the flux using Godunov's method or an approximation thereof.

5.2. Compute the wall reaction force, in case of different left and right cell areas.

5.3. Save the flux to the left and to the right at this boundary.

6. **Integration — for each cell**

6.1. Collect the fluxes with their associated direction and add them.

6.2. Apply the effects of heat loss $(T_{room} - T)\kappa(\delta x\sqrt{A})\delta t$.

6.3. Apply the effects of viscosity $F_r = 2u_{avg}\mu(2\pi)\delta x$.

6.4. Multiply by δt and normalize by volume $(V = A\delta x)$.

6.5. Convert cell's state to conservative variables and add computed flux.

6.6. Convert back to nominal $(rho,\, u,\, P)$ and write back.

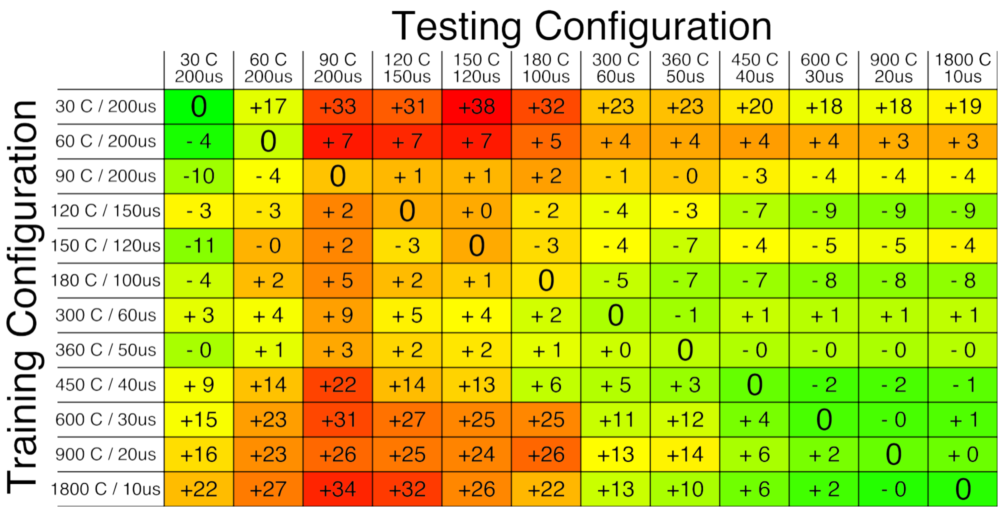
The reason for splitting in 6 distinct steps is that each of them is independent and can be parallelized. Doing all 6 steps for each cell in a sequence would result in having to compute each flux twice (once for each cell at left and right side of the intercell boundary). The most computationally heavy step is 5., where the Riemann problem is solved. However, overall, this step is just about 30% of the total computation time on the GPU since memory transfers are relatively expensive. Overall on a NVIDIA GeForce 1080 GTX one step takes approximately 5-10μs. That number is largely independent from the number of cells, because usually a GPU has much higher capacity for parallel work. That capacity is actually fully utilized when doing reinforcement learning, for example, with multiple simultaneous rollouts.

Chapter 6
BASIC VALIDATION

6.1 Cell size considerations

So far we have been talking about δx, but have not mentioned what values would be appropriate to use. It is of huge significance as using higher δx generally allows for proportionally higher δt as well as using less cells to simulate a given system. Therefore the overall result is that for a given simulation the computational time is inversely proportional to the square of the cell sizes, or crudely speaking $\mathcal{O}(^1/_{\delta x^2})$.

At the moment we have not done extensive exploration of the subject other than to notice that agents trained at a given system perform just as well when evaluated at identical system with just δx and δt changed. Figure 6.1 shows the results of evaluating 12 agents trained on systems ranging from 30 cells with a $\delta t = 200\mu s$ to the identical system with 1800 cells (scaling δx from $1cm$ to $60cm$) and $\delta t = 10\mu s$. We used our 1-dimendional tracking environment (section 7.1) for this testing and the overall training process is describes in section 7.


Testing Configuration

Training Configuration

	30 C 200us	60 C 200us	90 C 200us	120 C 150us	150 C 120us	180 C 100us	300 C 60us	360 C 50us	450 C 40us	600 C 30us	900 C 20us	1800 C 10us
30 C / 200us	0	+17	+33	+31	+38	+32	+23	+23	+20	+18	+18	+19
60 C / 200us	- 4	0	+ 7	+ 7	+ 7	+ 5	+ 4	+ 4	+ 4	+ 4	+ 3	+ 3
90 C / 200us	-10	- 4	0	+ 1	+ 1	+ 2	- 1	- 0	- 3	- 4	- 4	- 4
120 C / 150us	- 3	- 3	+ 2	0	+ 0	- 2	- 4	- 3	- 7	- 9	- 9	- 9
150 C / 120us	-11	- 0	+ 2	- 3	0	- 3	- 4	- 7	- 4	- 5	- 5	- 4
180 C / 100us	- 4	+ 2	+ 5	+ 2	+ 1	0	- 5	- 7	- 7	- 8	- 8	- 8
300 C / 60us	+ 3	+ 4	+ 9	+ 5	+ 4	+ 2	0	- 1	+ 1	+ 1	+ 1	+ 1
360 C / 50us	- 0	+ 1	+ 3	+ 2	+ 2	+ 1	+ 0	0	- 0	- 0	- 0	- 0
450 C / 40us	+ 9	+14	+22	+14	+13	+ 6	+ 5	+ 3	0	- 2	- 2	- 1
600 C / 30us	+15	+23	+31	+27	+25	+25	+11	+12	+ 4	0	- 0	+ 1
900 C / 20us	+16	+23	+26	+25	+24	+26	+13	+14	+ 6	+ 2	0	+ 0
1800 C / 10us	+22	+27	+34	+32	+26	+22	+13	+10	+ 6	+ 2	- 0	0

Figure 6.1: Evaluation results. Shown are the percentage differences of tracking error compared to the evaluation on the same exact cellsize and timestep system, which is the diagonal. The maximum increase in tracking error is about 30% which is much less than when the agents were trained with the traditional pneumatic models (see section 7.5).

6.2 *Valve Motion*

We outlined the model of the valve motion in section 5.4. Here we show the results of fitting the unknown PID parameters as well as the rest of the valve piston parameters. We commanded the valve to transition between several voltage levels and measured the actual valve position. We then solved a nonlinear Gauss Newton regression for the unknown parameters.

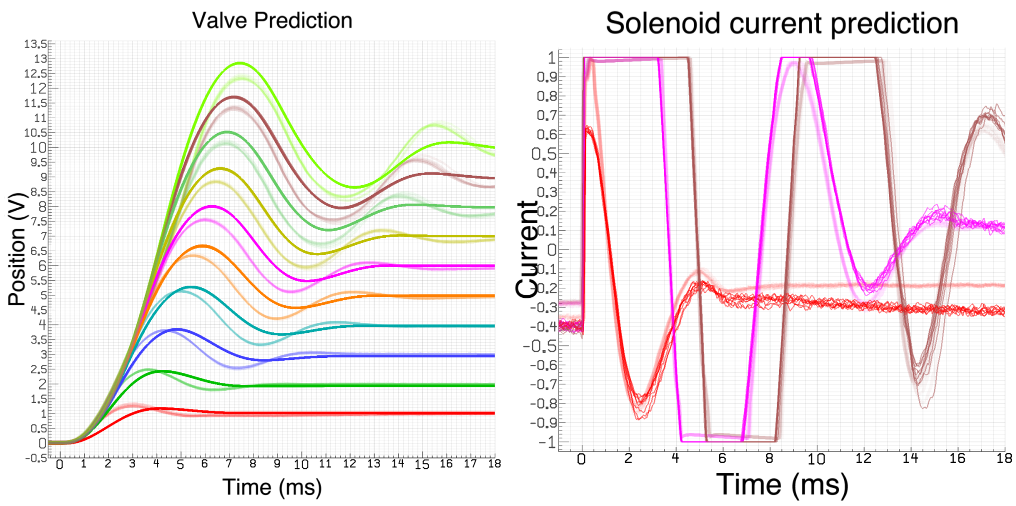


Figure 6.2: Valve motion prediction. **LEFT:** Predicting piston position for 10 different transitions. **RIGHT:** Predicting solenoid current for 3 different transitions.

Figure 6.2 shows the result of our identification procedure. Solid lines represent our predictions, while the light lines are the actual transitions. This is shown for few transition from $0V$ position. While the prediction is not perfect, it captures the initial slope and overshoot well. Predicting the solenoid current is actually almost perfect. The alternative of using just a step function at $t = 0$ is clearly much inferior.

Even though the valve is extremely consistent, more precise prediction in only possible with non-parametric approaches. The task is essentially to reverse engineer the software on the valve micro-controller. It was not deemed necessary to build a better model at this point.

6.3 *Valve Flow*

While we have not performed full system identification against a physical system, initial comparisons look very promising. First, fig. 6.3 shows how the computed flow of our PDE-based model compares with the flow computed using the Thin-plate model (section. 3.2.1). The comparison is done for a fixed valve opening, and while we have not accounted for the fudge factor in the previous model, we should still expect them to be close to each other, within a scaling factor. However what we note is that as the inflow pressure gets closer to the cylinder pressure the two models diverge more and more. This in itself does not mean one is more accurate than the other but it shows where their differences occur.

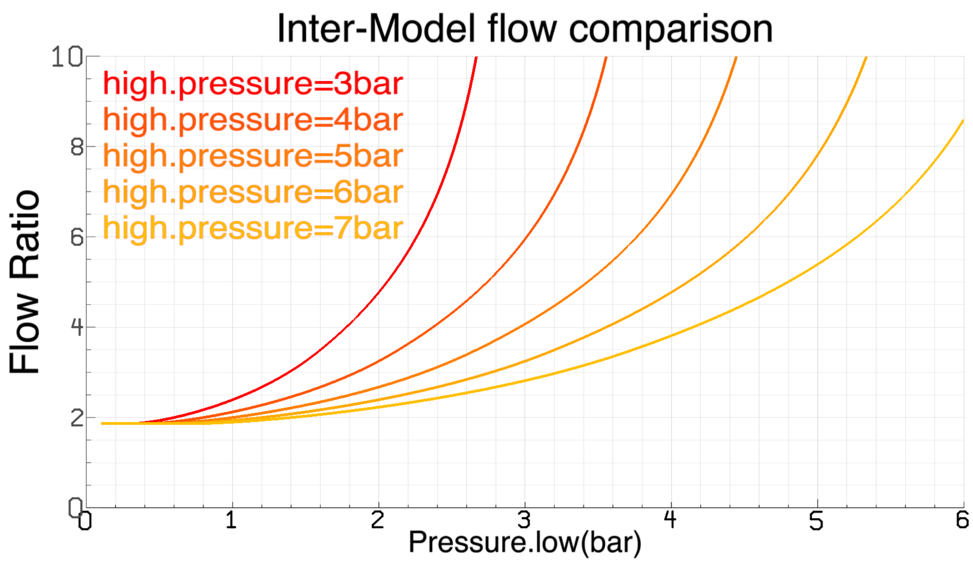
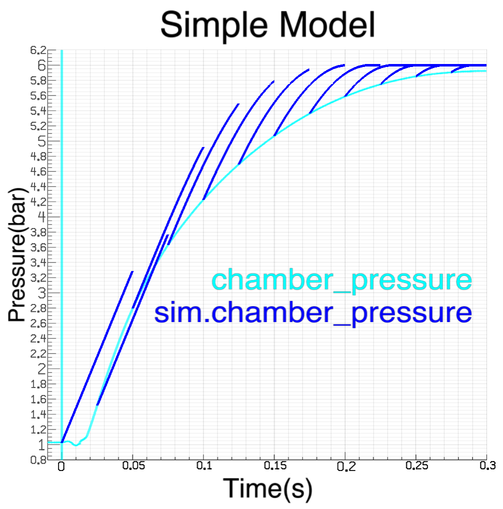
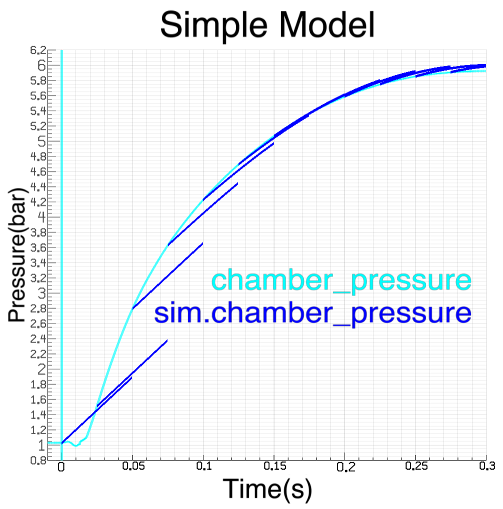


Figure 6.3: Ratio between flow computed using Thin-Plate model and by solving Riemann problem and.

Next, we do a simple system identification of the simple model for that specific valve opening (fully open) and try to match the initial portion (immediately after valve opening, fig. 6.4a) of the cylinder pressure as well as for the latter stages (fig. 6.4b). We notice that when fitted for one it grossly mis-estimates the other portion.



(a) Matching the initial portion of the flow



(b) Matching the latter stages

Figure 6.4: Simple model matching the pressure evolution of flow filling in a cylinder after fully opening the valve.

Finally figure 6.5 shows the performance of our PDE-based model. This is open loop simulation of valve opening. No systematic system identification is performed, just hand-tuned parameters. The results look very promising, even though pressure tracking at various taps are not perfect. More systematic system ID will follow in future work.

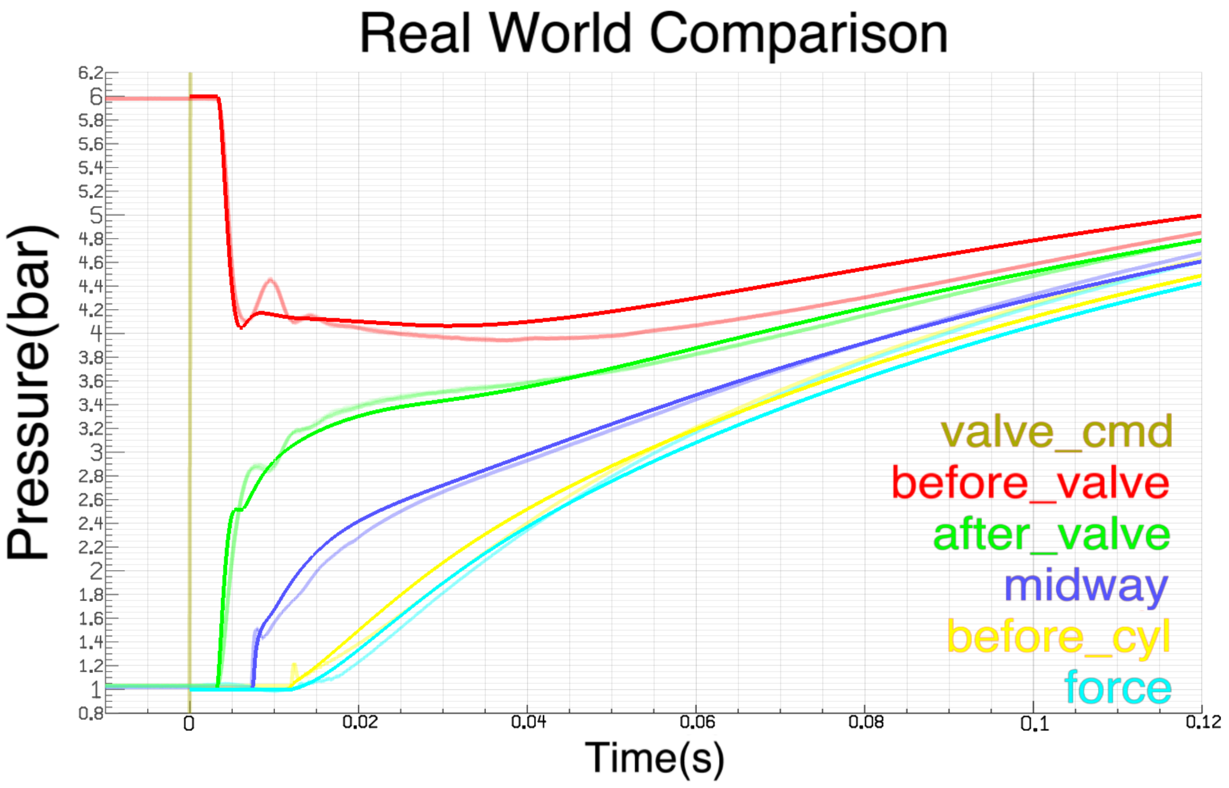


Figure 6.5: Light colors correspond to real world data, while solid colors are our simulation. The valve opens at time 0 and the pressures taps displayed are as follows: upstream of valve just before entrance, downstream of valve just after the exit, midway to cylinder, just before cylinder entrance pressure at end of cylinder, computed via force sensor at piston end.

Chapter 7

VALIDATION WITH REINFORCEMENT LEARNING

In this chapter we evaluate the fitness of simplistic pneumatic models to serve as training environments for RL agents controlling a pneumatically actuated physical systems. In section 7.1 we introduce the RL tasks that we use as well as the specific parameters of the learning algorithm explained in section 3.1.2. Section 7.2 introduces a novel feature augmentation strategy that greatly speeds learning. Section 7.3 shows the usefulness of the extra state variables afforded by the PDE-based model. Section 7.4 sets up the exact comparison mechanism.

7.1 Testbed environments

We use the *Natural Policy Gradient* algorithm for Reinforcement Learning outlined in section 3.1. For our policy class we chose a 2 hidden layer neural network (NN) with **tanh** activations. The NN takes as input a vector of observations $O(s_t)$ and produces a vector of control signals u_t. To make it stochastic we further have a parameter σ_i for each component of $u_{t,i}$ that determines its standard deviation. Then we sample $a_{t,i}$ from $\mathcal{N}(u_{t,i}, \sigma_i^2)$.

For the value function approximator we use the same observation vector augmented with time, since the value function is time dependent. Similarly we use 2 hidden layer neural network with **RELU** activations and optimized with ADAM. Table 7.1 lists the algorithm hyperparameters that we used for most of the experiments.

We selected few dynamical systems on which to demonstrate our method. They start from a basic 1-dimensional position tracking. Next we have 2-dimensional two-link arm. Then we have a cart-pole system to demonstrate more fine grained dynamic and dexterous tasks. Finally we have an in-plane walking *dog*, to demonstrate the effect on a more realistic

Discount factor	γ	0.995	explained in equation 3.2
GAE exponent	λ	0.97	explained in [30]
KL trust radius	δ	0.1	explained in equation 3.13
CG iterations		10	to solve for NPG, equation 3.14
Policy architecture		[32,32]	
Number of episodes		64	explained in section 3.1.1
VF architecture		[128,128]	
VF learning rate		10^{-3}	for value function update
VF iterations		500	number of iterations per NPG step
VF batch size		256	

Table 7.1: Parameters values for NPG

system.

For the 1-dimensional system we model it as a cart on a rail. The cart has mass and is connected to the piston of a single cylinder (fig. 7.1A) . Its goal is to follow a desired trajectory on the rail with 4 different options. The simplest task is to move to a desired position from a random location. The next three tasks are to follow a trajectory with 3 different profiles: superposition of smooth sine waves, square-like wave and saw-tooth-like wave.

For our tracking tasks, the reward we use is $-\alpha|e| - \beta log(|e|) + \gamma$, where $e = pos - target$ is the tracking error. The extra *log* term encourages more close following of the target, which we found empirically. Unlike many control tasks we do not need to penalize the control signal as it does not directly influence force.

The observation vector includes the current position, the target, as well as their difference: the error signal. Moreover we include the velocity, the actual position and velocity of the valve piston, as well as the pressure in the two chambers of the cylinder.

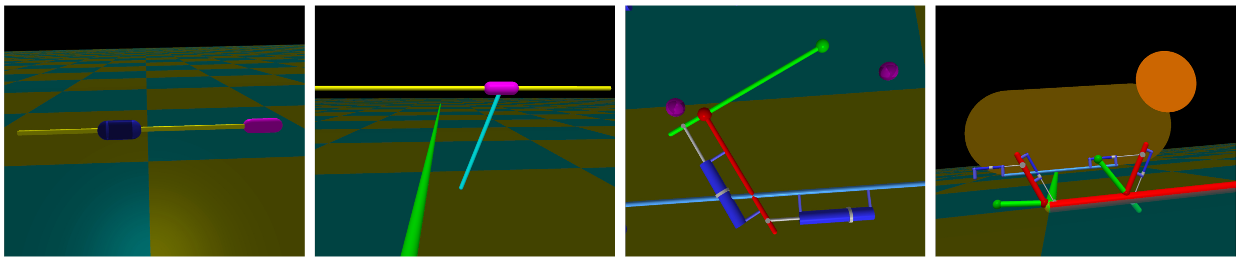


Figure 7.1: The different test system that we use throughout this paper. From left to right: **(A)** 1-D Tracking, **(B)** Cartpole Swingup, **(C)** 2-link arm tracking, **(D)** Dog Walking

To promote better trajectory tracking, we add additionally information about the future targets. Specifically we add the differences of the current position and few points in the future within the next 0.5 seconds. While that may be unrealistic in some situations, it greatly enhances trajectory tracking if that is the goal. Moreover, in any scenario, where we form any plans for the future instead of just reacting to current observations we would need to be able to take the future requirements and incorporate them into the present action. And, finally, this is done to compare different pneumatic models, so putting such requirements into the policy is justified.

The 2D two-link arm is shown in fig. 7.1B. Similarly we have the same tasks as the 1D system, with the only difference being the crank-lever attachment as opposed to direct drive. The reward is similar to 1D case, but we use the Cartesian distance to the target. The feature vector is analogous to the 1D system, but we include information for each joint and each end-effector DOF.

Next is the cart-pole system (fig. 7.1C). It is just like the basic 1D system but with a rotating rod with mass at the end attached to the cart. Here we have 2 different tasks: swing-up and tracking. Swing-up is the usual cart-pole swing-up task, starting from a random orientation. The goal in cart-pole tracking is to track a desired cart position, while holding the rod vertical. This demonstrates fine dexterous control. The observation vector includes like usual the position and velocity of the two degrees of freedom as well as the pneumatic features. For the tracking task we add the tracking features we mentioned previously.

The reward in the swing-up case is of the form $\alpha|1-x|^2+\beta|\pi-\theta|-\gamma log(|e|)-\delta$, where x is the normalized position on the rail with $x=0$ being the middle. The α term encourages staying in the center of the rail. Additionally we terminate the episode when the cart hits the ends of the rail.

The tracking reward is just the 1D tracking reward, with the addition of episode termnation when the pole falls more than $\frac{1}{4}rad$.

Lastly we have a *dog* system which is just two 2-link arms attached at a torso to simulate 2-legged moving creature (fig. 7.1D). The goal is to simply to move forward, and correspondingly the reward is the velocity. The features are basically all the joint angles and velocities as well as pressures in each of the 8 chambers.

7.2 Positional encoding

Reinforcement Learning relies on a lot of components working together to achieve its purpose. On one hand we have the specific algorithm, the choice of baseline function approximator as well as the choice of policy class. On the other end, we have the model of the environment and how realistic is it, as well as the sensor outputs. What binds those together is the sensory output which feeds both into the policy function as well as the baseline estimation function. With the advent of neural networks, the sensory output is usually directly fed into a neural network, relying on the idea that neural networks are universal function approximators. Often overlooked are ways to augment the sensory data in a way that helps the learning process.

We discovered that in certain scenarios a simple feature augmentation scheme based on trigonometric functions drastically speeds up the learning process as well as help achieve higher reward. Often when we have a tracking problem, the error between current position and desired position is used as one features. Instead of the single scalar f feature we put a sequence of features $\sin(f \cdot b^n)$ and $\cos(f \cdot b^n)$, for each $n \in [0, N]$, where b is a chosen base constant that we choose to range between 2 and 7, and N varies such that $b^N \sim R$, where

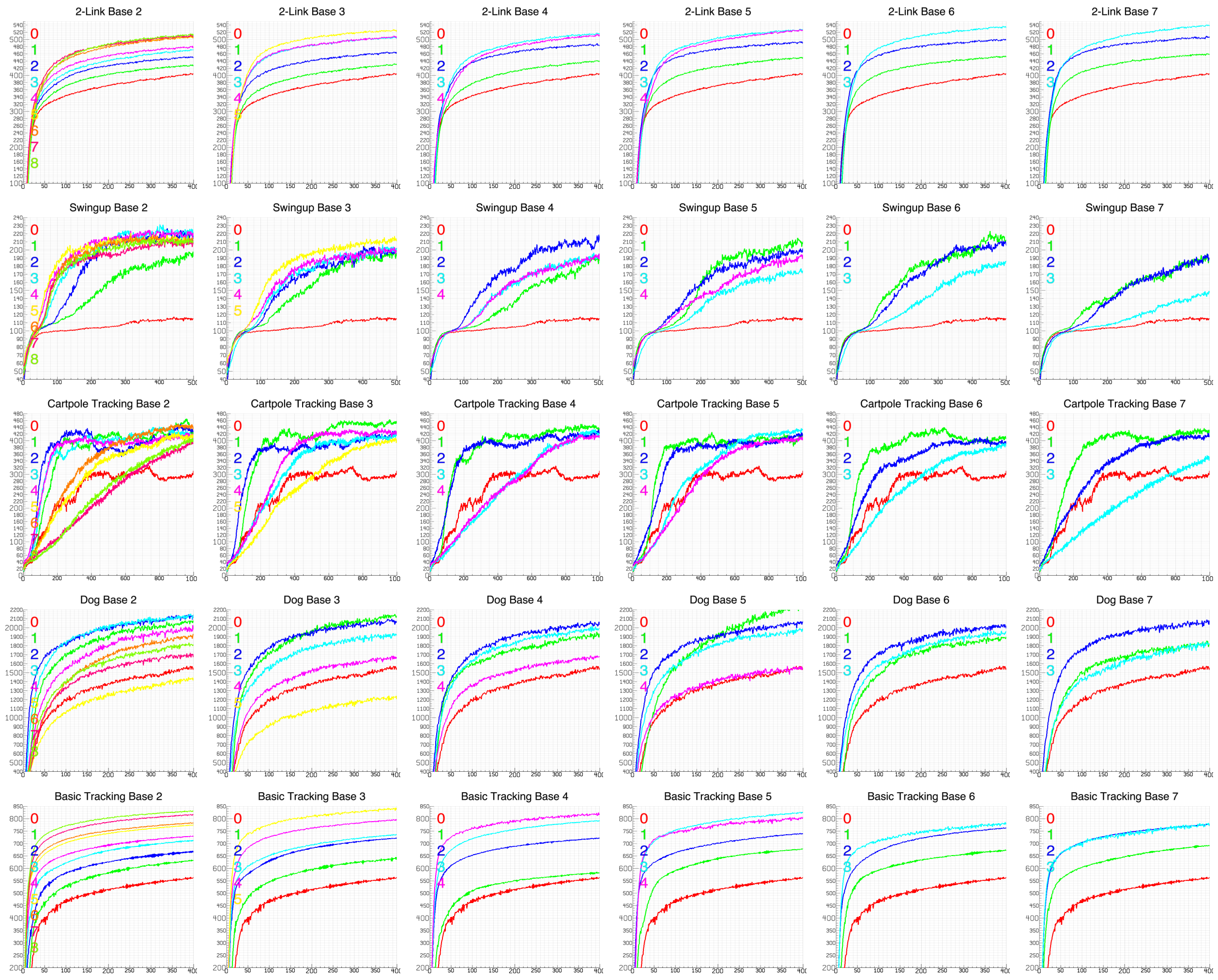


Figure 7.2: Effect of positional encoding on training performance for few of our environments as well as for different power bases b. Each row is a new environment, while each column is a different base. The different lines in each picture correspond to different number of extra features added with red corresponding to not using positional encoding, i.e. $N = 0$.

R is the desired zoom factor. Essentially what this scheme does is presents the feature f at different zoom levels (with circular overflow) to the RL algorithm, to allow it to operate at different scales. The factor R sets the zoom level. We discovered that RL-based tracking algorithms are very bad at operating both at higher level when error signals are high, as well as small scale when errors are tiny, due to the inherent lack of structure in the neural networks.

This scheme is directly borrowed from the computer vision community, for example in [18], this technique is called positional encoding and allows them to achieve much improved visual acuity. Similar technique is employed in RL as well in [25], however, the authors there use RBF kernels, which are just random collection of sinusoidal functions with different frequencies and phases. Essentially they will achieve the same effect given enough functions, albeit in a less systematic way.

This scheme helps not just with tracking tasks but with other tasks as well as shown in fig. 7.2. Performance is greatly enhanced for almost all combinations provided $N > 1$. Not a significant difference between different power bases was observed.

7.3 Comparison with and without extra state observations

One advantage of using complicated model is that is allows us to have a more detailed picture of the internal state of the system we are simulating. In the context of feedback control and Reinforcement Learning that is an opportunity to use the extra available state as part of the feedback loop or observation vector in the case of Reinforcement Learning.

Specifically we have access to the state of the air throughout the pneumatic line. We have not only the pressure but also the density and the velocity of the compressed air. We add those three state variables from few evenly spaced locations along our pneumatic lines to the observation vector when learning our agent with RL. This allows for better control decisions as well as more accurate value function which allows for faster learning.

Figure 7.3 shows sample learning curves for two different configurations where using the extra features makes the training faster as well as reach higher maximum return.

7.4 Comparison setup

When computing a control law one usually utilizes a model of the actual system with the hope that it will behave sufficiently close when executed on the real physical system. Especially

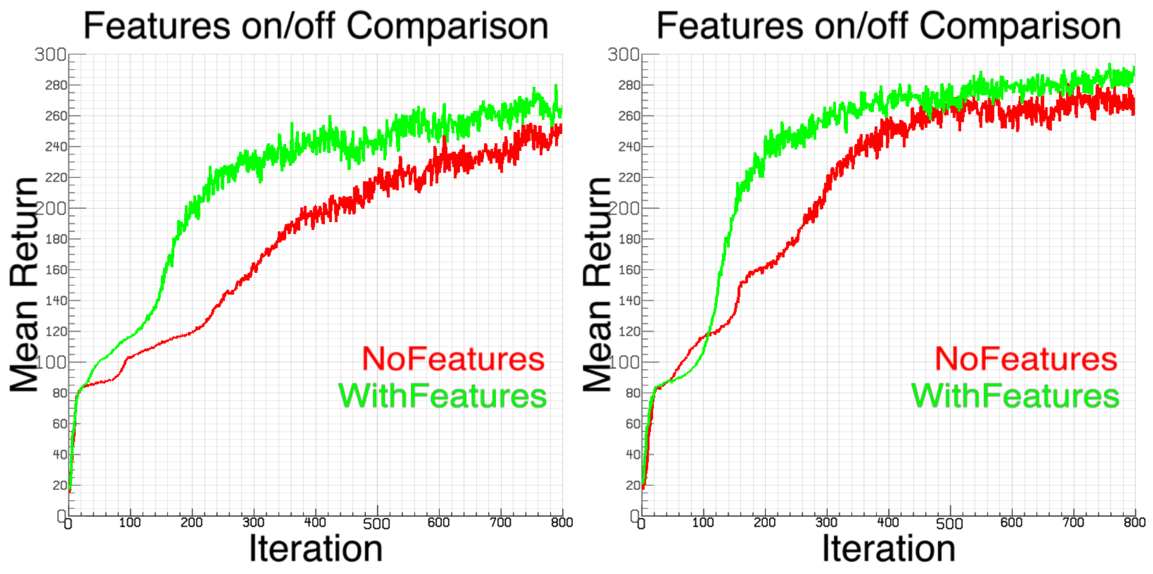


Figure 7.3: Training curves for two configuration with and without the extra features afforded by our model.

useful is a having a feedback law. Reinforcement Learning usually produces policies which are in essence feedback policies, since the current state influences the control output.

In order to validate the need for our approach to pneumatic modeling, we evaluate the performance of an agent trained on a dynamical system equipped with a simplistic model on the same dynamical system with just the pneumatic part interchanged with our sophisticated model. This is preliminary step before using this approach on the actual physical systems available in our lab.

The *simple* model in particular that we use is based on [34]. It was explained in section 3.2.1 and the specific way in which we parameterize it in section 6.3.

In essence, we are treating our PDE-based model as the *physical system.* Therefore before using any other model to learn a policy we need to do system identification to match it to the real system. Since the only unknowns in that model is the valve flow, that is the part we identify. Other parts such as the source pressure, cylinder size, etc. we keep the same between the two systems. To do system ID we *collect* data from the PDE-based model of a

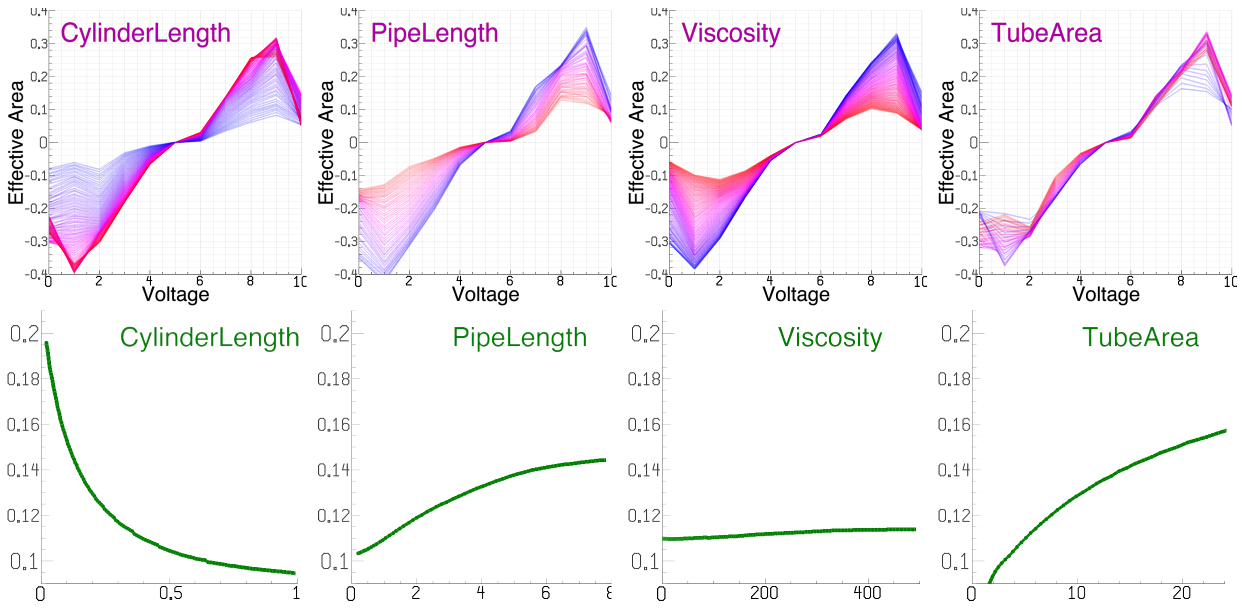
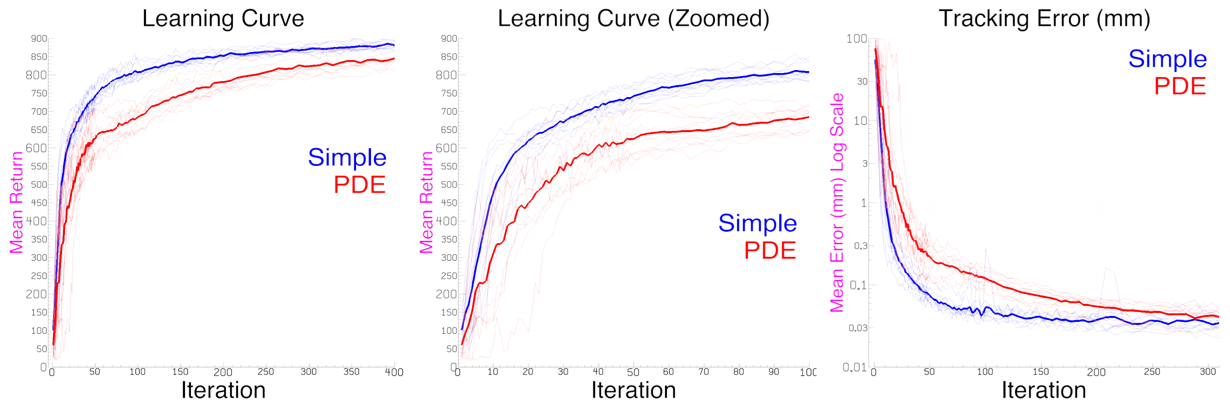


Figure 7.4: System match sample: **Top**: The horizontal axis is the commanded valve area expressed in voltage, while the vertical axis is the scaling applied to the tube cross-sectional area. The different shades (from **red** to **blue**) correspond to different setting of one parameter with **red** corresponding to numerically higher values. **Bottom**: The fitting error. Vertical axis is pressure RMSE in bars, while the horizontal is the value of the varied parameter.

valve filling up a locked cylinder with air and measuring the pressure just before the entrance (we do not have observation of the pressure inside the cylinder). We do this for various initial conditions and estimate the parameters of the *simple* model.

Figure 7.4 shows how varying certain parameters of the PDE-based system affect the effective valve area. The length of the cylinder, the length of the connecting pipe as well as the viscosity of the fluid have great effect, while the cross sectional area of the pipe does not. The reason is that the effective area is scaled by the pipe area anyways, so it it should not be a factor. Also it can be seen that for short cylinders it is more difficult for the two systems to agree, judging by the higher final error.



(a) (b) (c)

Figure 7.5: Comparison of the learning curves for the PDE-based agent and the simple pneumatic model based agent. Shown are 10 learning curves with the corresponding tracking errors in 7.5c, and also the average of the 10. The simple agent consistently both learns faster and achieves higher score and is able to more closely track the desired trajectory. Figure 7.5c shows the tracking error in log scale for each iteration of the training.

7.5 *Basic tracking*

We start with a simple 1-dimensional system with 1 cylinder and 2 valves with typical system parameters: (). The goal is to track a moving target with a mass attached to the piston of the cylinder. The target moves along a superposition of random sine waves. We vary several parameters of this setup to explore the difference in the models under different conditions. This is explored in detail in section 7.6. For now we keep the system as specified.

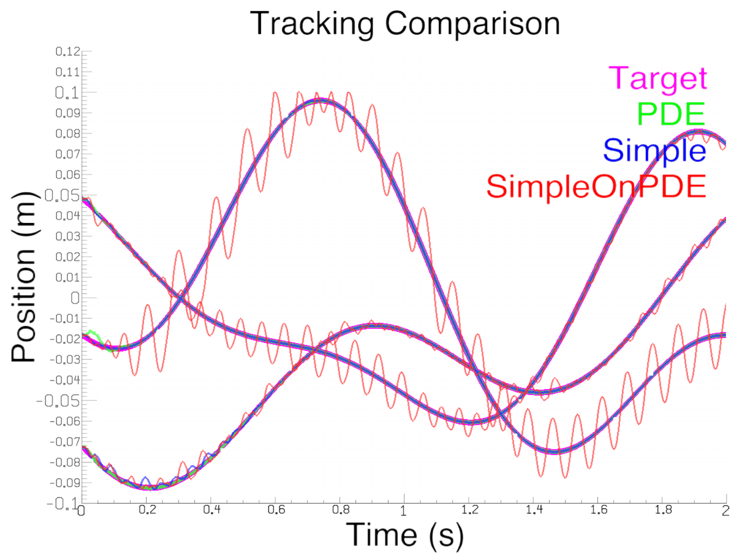
We train agents based on both models (the PDE-based model as well as the simple model) for 400 iterations. Figure 7.5 shows typical learning curves for the problem. It is a very easy problem, so learning happens fast in the beginning and the vast majority of the iterations are spent fine-tuning the behavior. Both agents achieve similar levels of performance, though the PDE-based model is slightly behind. This is to be expected because it is a more complex dynamical system and it takes more time to learn the extra parameters due to the extra observation features.

Also note how the difference is much bigger when expressed in average tracking error, as opposed to in RL score. The RL score is calculated as the mean return, which, for each timestep, includes the immediate reward plus future discounted rewards. This is the score that the RL algorithm tries to optimized.

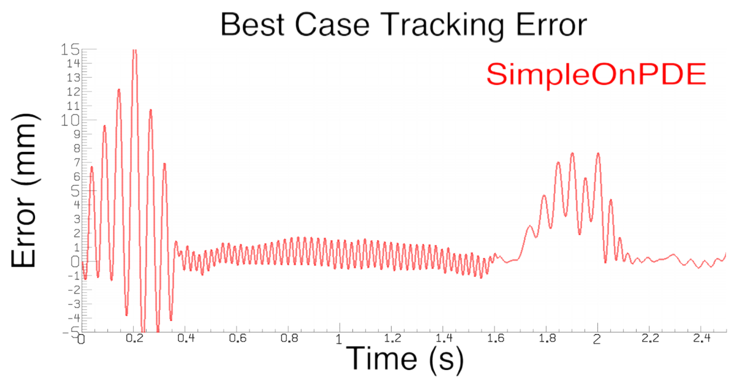
Once we have an agent trained with the simple model we modify the PDE-based model to give observations that match the observations from the simple model. This means excluding any extra pressure measurements besides the cylinder pressure. Now we can run the agent trained with the simple model on the PDE-based model and observe the difference in behavior. Figure 7.6 shows the typical tracking behavior. While it grossly tracks the target, huge oscillations are clearly visible on some of the trained agents. Numerically, the PDE-trained agent has about $0.036912mm$ tracking error, the agent trained on the simple model has an error of $13.557506mm$. Moreover, figures 7.6c and 7.6d show comparisons of the control signals that were used. The control signal from the simple agent is clearly oscillating as well.

So far we have tested the agent obtained after 400 iterations on the simple model. What if we test the intermediate agents that are not fully trained? It turns out most of the time they are much better. Figure 7.7 shows the evaluation score and average error for the agent at all points of its training curve. Clearly it can be seen that very early in the training process the agent achieves its best performance on the PDE-based model. Figure 7.6 shows the results from evaluating the best policy out of all iterations. While it looks much better (with average tracking error of $0.549736mm$), small oscillations still remain in the position (fig. 7.6b), as well as huge oscillations in the control signal (fig. 7.6e).

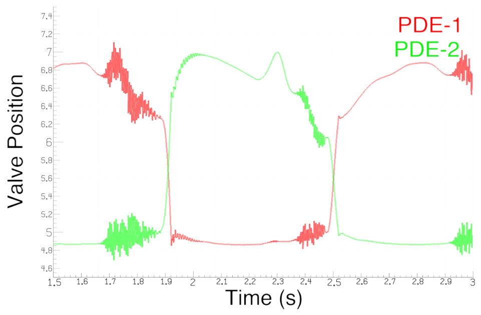
We have not investigated in detail why early stopping performs better, it is likely due to overfitting for the simple model. The learning process start by capturing the essential aspects of the task (such as positive correlation between valve command, pressure, velocity and ultimately position), which are similar between the two models and proceeds to fine tune to the specifics of the simple model. While the performance is much closer to the



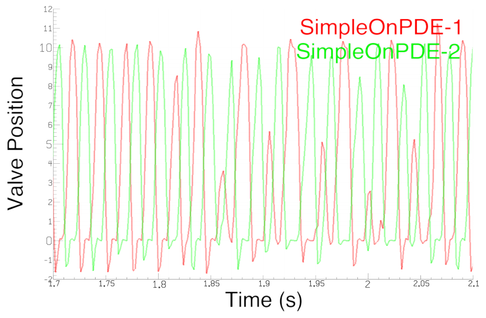
(a) Tracking performance after 400 iterations. The two graphs PDE and Simple are the tracking performance of the two agents when tested on the same model they were trained on. Virtually perfect tracking. SimpleOnPDE shows the performance of the agent trained on the simple model and evaluated on the PDE-based model.



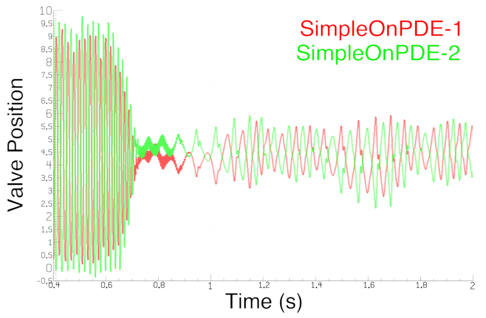
(b) The tracking error with the best iteration of the simple agent. If shown overlaid on the target the performance is visibly very similar, therefore only the residual is shown.



(c) The valve position during evaluation of the PDE-based agent



(d) The valve position during evaluation of the simple agent, tested on the PDE model



(e) Best case iteration of the simple agent. Still there are wild oscillations.

Figure 7.6: Performance comparison between agents trained on a PDE-based model and on a simple model. The graphs on the left show the tracking performance, while the right graphs show the control signal as it is desirable to have smooth control signal as well.

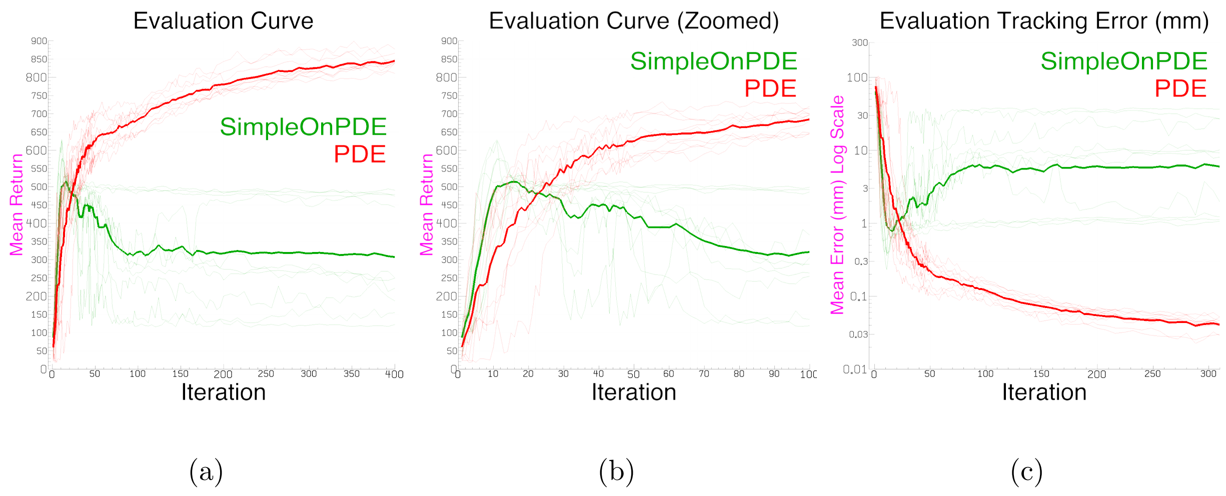


Figure 7.7: Comparison of the evaluation results at intermediate stages for the PDE-based agent and the simple pneumatic model based agent, both tested on a PDE-based model. Shown are 10 learning curves with the corresponding tracking errors in (7.7c), and also the average of the 10. The simple agent starts learning better but quickly reaches a plateau and eventually is not able to track as well. (7.7c) shows the tracking error in log scale for each iteration of the training.

PDE-based model when choosing the best training iteration, the validity of such approach is questionable. A practitioner cannot know when to stop training, and will usually train until flattening of the training curve (fig. 7.7) which results in poorly performing agent. The training process is also very non-deterministic, as seen from the huge variance at the end of the training process (fig. 7.7). We continue to report the best-case for some of the following experiments, where it makes sense, as the error is more stable.

7.6 Ablation study

Here we show how changing a parameter of the system affects the relative performance of the two agents. The parameters that we are varying are:

1. Cylinder length (between $4cm$ and $1m$)

2. Cylinder cross sectional area (between $30mm^2$ and $500mm^2$)

3. Length of pipe from valve to cylinder (between $20cm$ and $10m$)

4. The ratio between maximum force of the piston to the gravitational weight of the object being moved. (between 1 and 50)

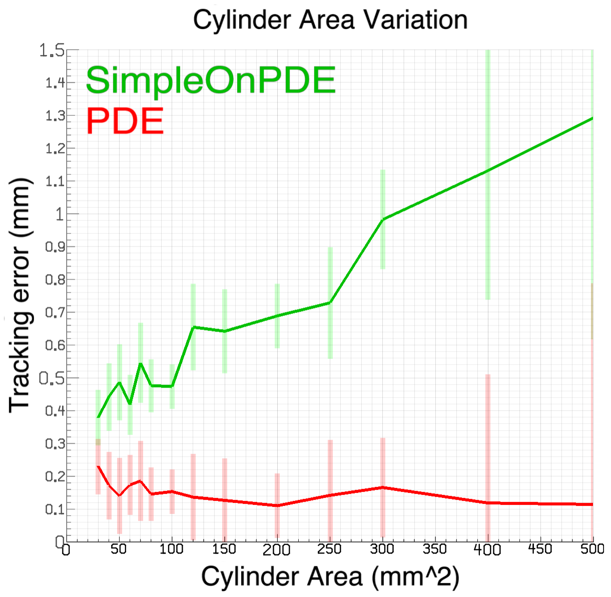
In order to account for the randomness we train 10 different agents on the simple model for each variation of the parameters and report the result from the iteration that achieves the best performance when evaluated on the PDE-based model. This is grossly overstating the performance of the simple model but we use this strategy to highlight the scenarios in which our PDE-based model is more relevant.

Figure 7.8 shows the average results against a single agent trained on the PDE-based model. As the cross-sectional area increases we see the simple model is deviating from the PDE-based model (fig. 7.8a). This is likely because the cylinder chamber contains more volume and it is slower to react to changes in demand. Therefore a more accurate model is needed to react quickly and extract the maximum performance.

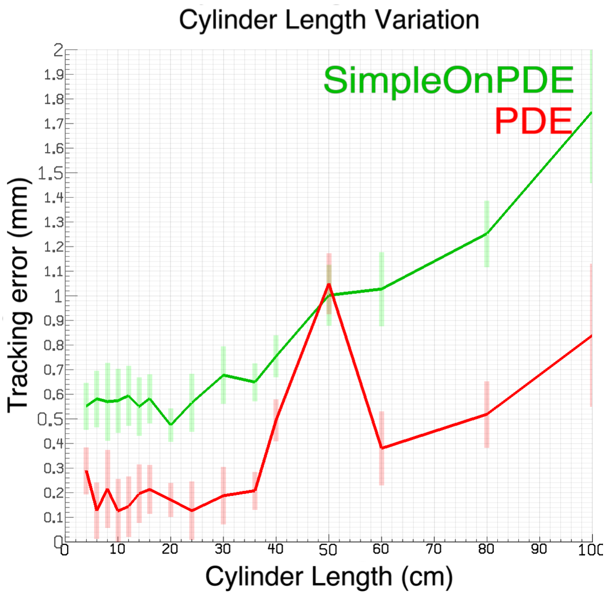
The same is observed when the actuator is overpowered relative to the task requirements (fig. 7.8c). This is typical for pneumatic systems as they have very high volume to force ratio while often connected to small moving parts. This additional agility requirement is not easily met with a simplistic model as again there is a need to react quickly.

Varying the actuator length does not change the relative difference of the two models.

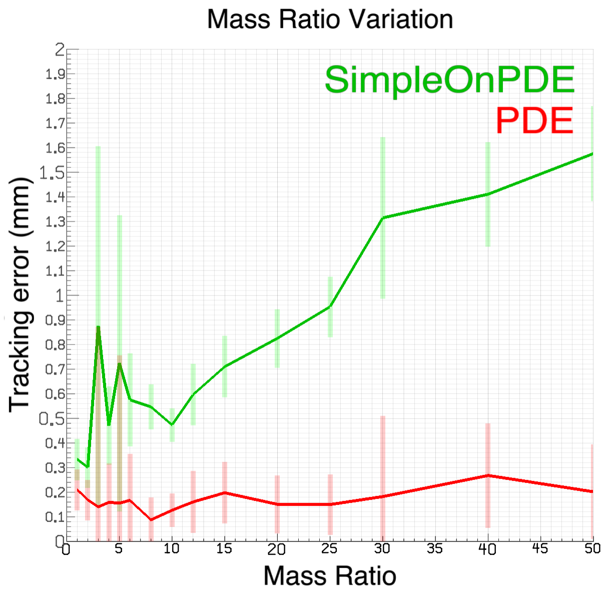
Finally, the biggest difference is observed when changing the length of the pipe from the valve to the actuator (fig. 7.8d). This is likely due to hidden delay of air travel time that is not modelled by the simple model.



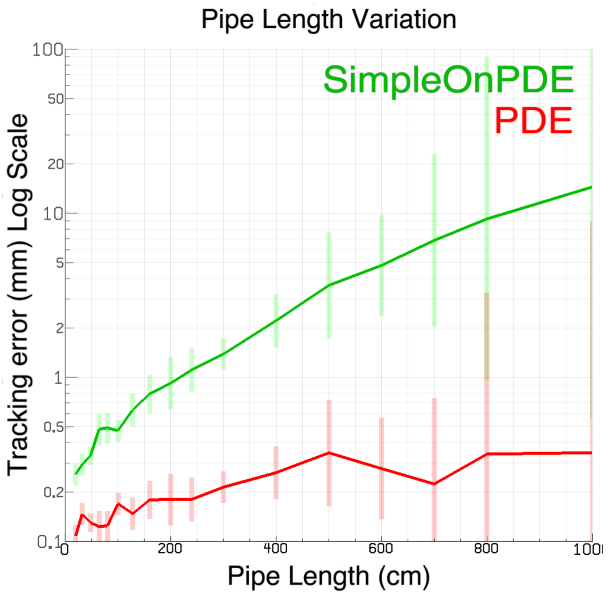
(a) Variation of cylinder cross-sectional area



(b) Variation of cylinder length

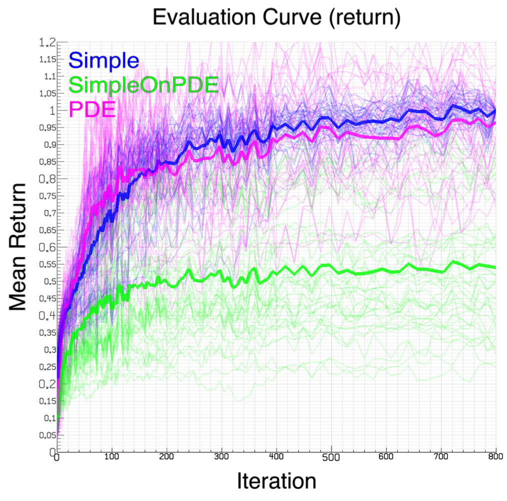


(c) Variation of ratio between max-force and mass of object

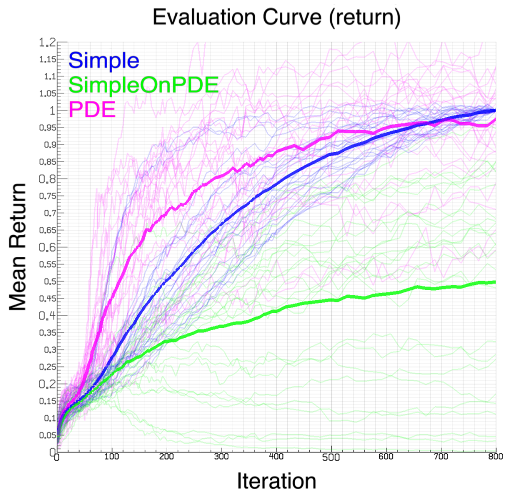


(d) Variation of pipe length between valve and cylinder

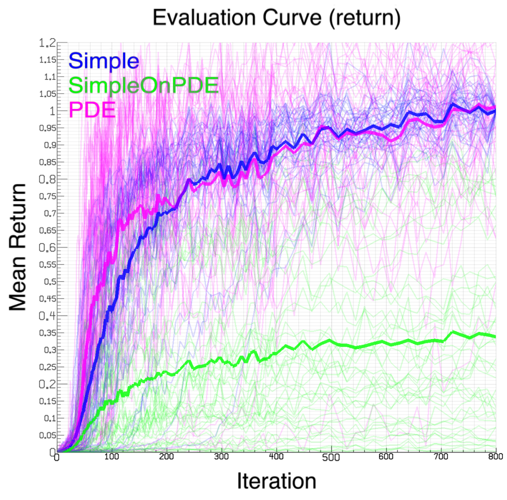
Figure 7.8: Evaluation tracking error showing the relative performance of the two agents when varying parameters of the system. Shown are also the distributions, as vertical bars, as each point is produced by averaging 10 different training sessions.



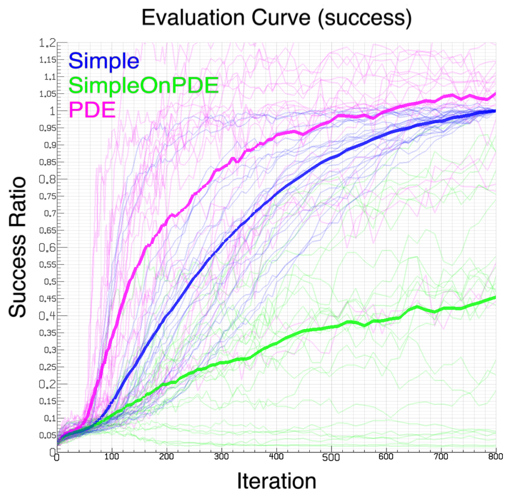
(a) Swingup mean return



(c) Upright tracking mean return



(b) Swingup success



(d) Upright tracking success

Figure 7.9: Evaluation graphs showing the performance of agents trained on a given pneumatic model and evaluated on possibly different model. The left graphs show results for the swingup task while the right graphs show upright tracking task. The top graphs show the mean return (the RL objective function), while the bottom show the percentage of total time the agent was in a successful position.

7.7 The rest of the environments

7.7.1 Cartpole

Here we show a comparison of the models on various tasks involving a cart-pole system. Cart-pole systems requires agile and responsive policy so therefore is well suited to show the differences between the pneumatic models.

We model cart-pole system by having a single pneumatic cylinder connected directly to a moving cart, onto which a rod is attached. The cart and the rod have different masses. We investigate the performance of our pneumatic models on two tasks:

1. Standard swingup task starting from a random initial state.
2. Following a desired horizontal trajectory while keeping the rod up upright. The initial state has the rod upright, and the task is deemed as failure if the rod falls or the cart deviates more than few percent of the target.

We also choose 100 random configurations of model parameters. These include:

1. Track Length ($60cm$ to $4m$)
2. Actuator cross-sectional area ($50mm^2$ to $500mm^2$)
3. Pneumatic line cross-sectional area ($5mm^2$ to $50mm^2$)
4. Length of rod ($10cm$ to $1m$)
5. Ratio of rod to cart mass (Constant total mass of $2kg$) (0.2 to 5)

Out of all configurations we only report the results for those that were successful, as some configurations yielded impossible tasks.

For each configuration we train 1 agent on our PDE-based model, and 8 agents (with different random seed) on an equivalent simple model. We then evaluate the performance of those 8 models on the PDE-based model.

Results are shown in fig. 7.9. For the tracking task success means not allowing the rod to fall beyond 15 degrees from vertical as well as staying within 2% of track length away from the target. For the swingup task success means staying within 3 degrees of vertical position with very low velocity. The graphs represent the evaluation of the agent at each iteration of training. The lightly colored graphs are all the different system configurations while the solid lines are simply the mean of the light lines. The blue graphs show the results of the agent trained on the simple model and evaluated on the same. The magenta graphs correspond to the PDE-based model, while the green graphs represent the agents trained on the simple model and evaluated on the PDE-based model. The dark lines show the mean evaluation results after each iteration of the RL algorithm. The lighter lines show individual results for each configuration. It can be seen that for some configurations there is less advantage to using complex PDE-based models, while for others it using the simplified model is not applicable at all.

7.7.2 2-link arm

For the 2-link arm environment we vary the system parameters as follows:

1. The two masses at the knee and end-effector ($200g$ to $2kg$)
2. Length of the two links ($15cm$ to $40cm$)
3. Actuator cross-sectional area ($100mm^2$ to $700mm^2$)
4. Pneumatic line cross-sectional area ($5mm^2$ to $50mm^2$)

For all configuration there was some behavior that was more or less tracking the target, with different levels of success as the ranges are quite large and often yielded underpowered systems.

Results are shown in fig. 7.10, and the interpretation is same, with the difference is that we count as success being within 1% of the target since it's an easier task than tracking and balancing a rod. We also normalize the success ratio as different task configuration have different difficulties and it is impossible to compare the raw success ratio. Overall the average success ratio across all configurations, at the end of the learning curve was around 40%.

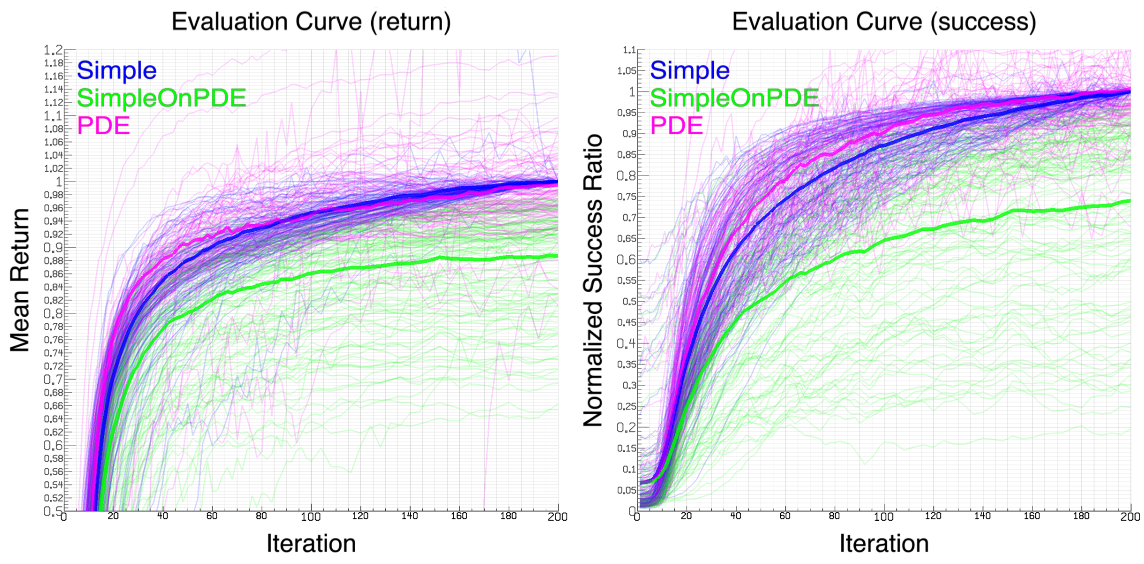


Figure 7.10: Evaluation graphs showing the performance of agents trained on a given pneumatic model and evaluated on possibly different model. The **LEFT** graph show the mean return (the RL objective function), while the **RIGHT** show the percentage of total time the agent was in a successful position.

7.7.3 Dog

For the Dog-like creature system we only trained one configuration as it is much slower. Similar effects were observed with the agents trained on the simplistic model underperforming when tested on the PDE-based model. Results are shown in figure 7.11.

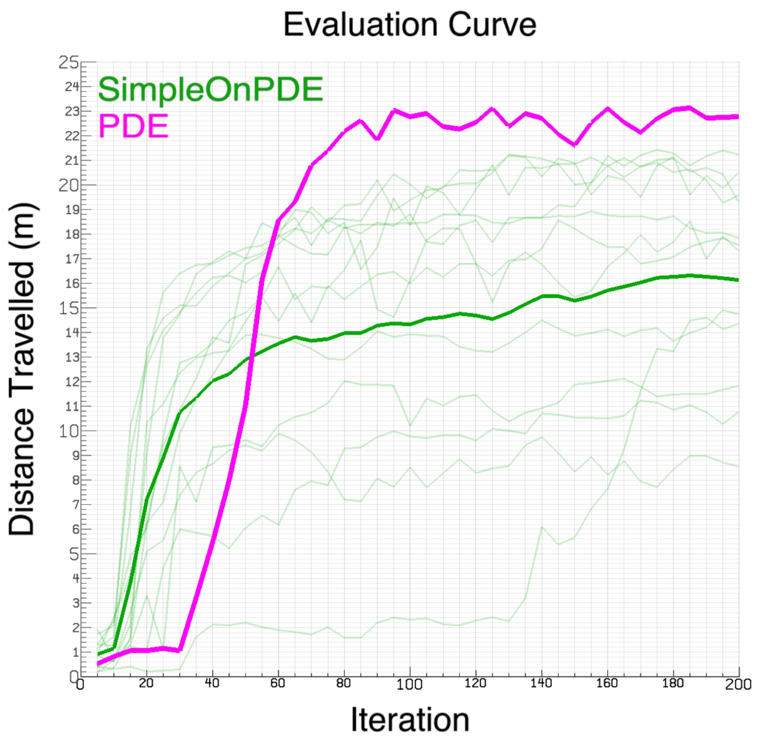


Figure 7.11: Results after evaluating intermediate agents on the PDE-based model. We trained 12 different agents on the simplistic system with different random seeds and the bold line is the average.

Chapter 8

SYSTEM IDENTIFICATION

8.1 System ID method

In this chapter we perform System Identification of the basic pneumatic system. We investigate to what extent is our PDE-based model able to predict pressures at various locations inside a pneumatic line given a valve control signal as well as how does it compare with the Thin-Plate model. The test setup is shown in fig. 8.1.

8.1.1 Pneumatic system setup

To generate data for system identification we need to excite the system. At first we excite our system in the simplest possible way: Changing valve position from fully open towards the room pressure to fully open towards the pressurized air source (the compressor). Next we generate some more interesting behaviour by using a basic PID controller to achieve a desired pressure trajectory at the end of the pneumatic tube. This random behavior is enough to excite the system to go explore various pressures, pressure velocities, as well as

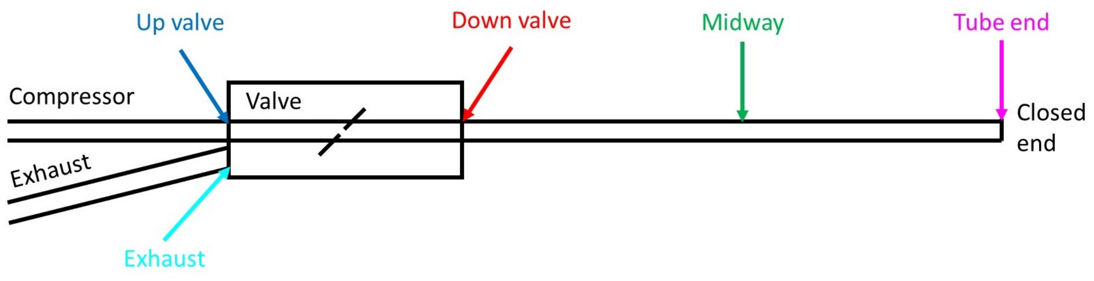


Figure 8.1: Basic setup

valve positions.

We record the pressures at the pressure taps indicated in fig. 8.1. We then construct a pneumatic system inside our PDE-based pneumatic simulator that matches the setup. The following parameters are treated as unknowns to be estimated by our method:

1. Exact position of pressure taps.
2. Pneumatic line cross-sectional area.
3. Viscosity of the fluid and non-linear viscosity terms as outlined in section 5.1. We have found that using additional terms that are non-linear with respect to velocity help preserve the fidelity of the simulation.
4. Temperature related terms (Ambient Temperature, conductivity of walls, heat conductivity) as outlined in section 5.2.
5. Actual size of valve opening as a function of voltage.
6. Dimensions of internal valve chambers.

8.1.2 Optimization method

The collected data consists of multiple minute-long trajectories. It would be impractical to expect our model to start at the beginning and predict in open-loop the entire trajectory given the control signal. This task is in general impossible in robotics since even small amounts of noise or modelling errors will lead to divergence very quickly. It is not needed, as any interesting robotic controller will have a feedback mechanism so we only care about near future prediction.

To generate predictions for the model being identified we perform multiple restarts at regular intervals ranging from $100ms$ to $500ms$, with very little difference in that range. We initialize the system at the given time using the recorded sensor data, then perform a roll-out using the recorded control signal and try to minimize the difference between the recorded pressure values and the estimated values for all N sensors, for the roll-out horizon T.

$$\min_{\mathbf{P}} \sum_{i=1..T} \sum_{j=1..N} (p_{ij} - \hat{p}_{ij})^2, \qquad s.t. P_k \in (P_k^{min}, P_k^{max}) \tag{8.1}$$

Initializing the system is in general a hard problem with no unique solution. Our PDE-based model is multi-dimensional with three numbers (ρ, u, P) per each discretized cell. We have access to only few pressure sensors located along the line. To give values for all cells we linearly interpolate and extrapolate the pressure values along the pneumatic line and assume zero velocity and room temperature. This is only true when the system is at rest, while our restarts start at random points throughout the execution. Truly solving this issue requires an estimation framework built on top of our PDE model, which is beyond the scope of this work. The current initialization that we use is inferior as such interpolated values, produces mini shockwaves in the pipe, but their magnitude is small and it does not affect the parameter estimation.

Since our optimization problem is in the form of constrained non-linear least squares we use Gauss-Newton optimization combined with a barrier method to account for the limits on the parameters. The limits exists for two reasons: Some parameters have natural physical limit (areas and lengths cannot be negative). Other times it's beneficial to constrain certain parameters to a range where the simulation is stable.

Lastly, we find the gradient and Jacobian of the constructed least squares optimization problem via finite differences, and use that to construct pseudo-Hessian and complete the Gauss-Newton implementation.

8.1.3 System ID for the Thin-Plate model

In order to have a fair comparison with the prior models we need to estimate their parameters as well. As the Thin-Plate model is simpler and does not have a notion of many pressure locations, we aim to predict the pressure at the farthest location (fig. 8.1). The model assumes there are two pressure sources (same as our PDE-based model). We measured those sources to be $0.635MPa$ (the high pressure source) and $0.101MPa$ (ambient room pressure).

Also the Thin-Plate model assumes we are filling a vessel, which in this case is just a long pipe, so we set the volume to be equal to the total volume of the pipe. The last point is actually irrelevant since the only parameters that we estimate (valve-opening vs. applied command voltage) can subsume the volume as a multiplicative factor.

With that said we use the exact same optimization scheme for the Thin-Plate model as for ours.

8.2 System Identification Results

We tested our model on two variations of the described system (fig. 8.1). In one case the length of the pipe from the valve to the end was $2.5m$ and in the other case $5m$. While the second case might seem long, such a setup is equivalent to using a shorter pipe with smaller cross-sectional area. We did not have a thinner pipe at our disposal to test with. This is applicable to certain scenarios where space is constrained and thinner tubes have to used. The cross-sectional area of our downstream tubes is roughly $6mm^2$. The exhaust tube is roughly $20cm$ long.

8.2.1 Step response

First we look into how well do we predict the pressure when a sudden change in the valve position happens. This is just a sanity check to verify that our model truthfully captures what is going on inside the tube.

Fig. 8.2 shows the results. Of notice here is the fact that we correctly capture the drop in source pressure (the dark blue sensor) as well as the proper ramp-up of the pressure immediately after the valve. Interestingly the pressure at the middle and at the end are fairly similar which shows that the biggest drop-off happens in the beginning, before a steady state flow is established.

Another interesting phenomenon (that we see later as well) is the initial knee-like response of the green and magenta values. This is a real phenomenon that is a result of initial shock-wave propagation. Even though our PDE model is capable of modeling such waves in this

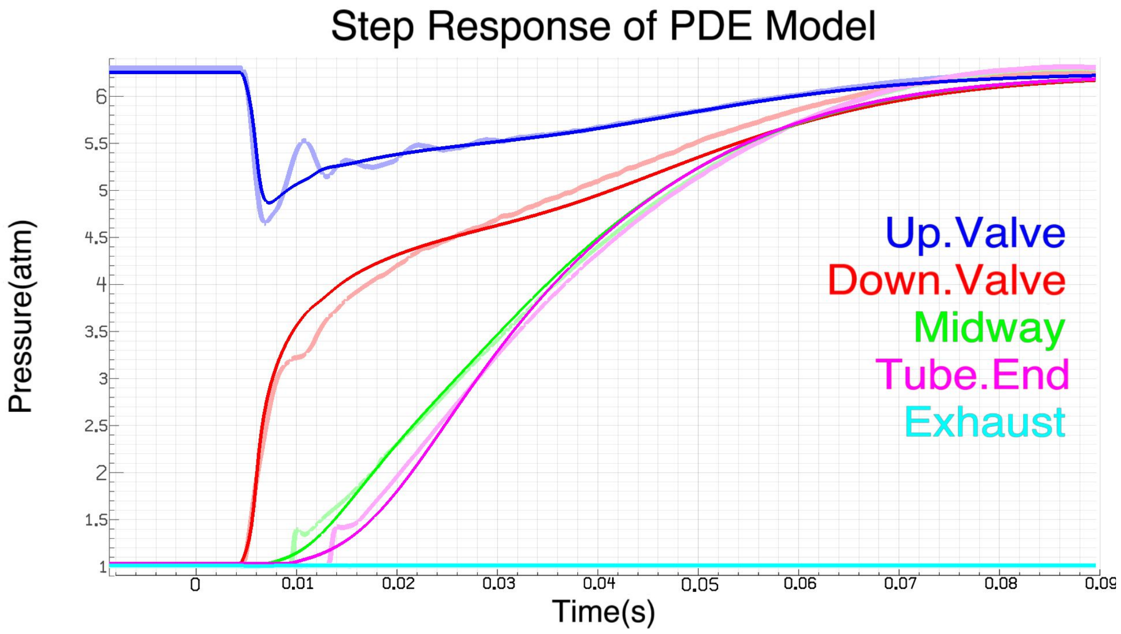


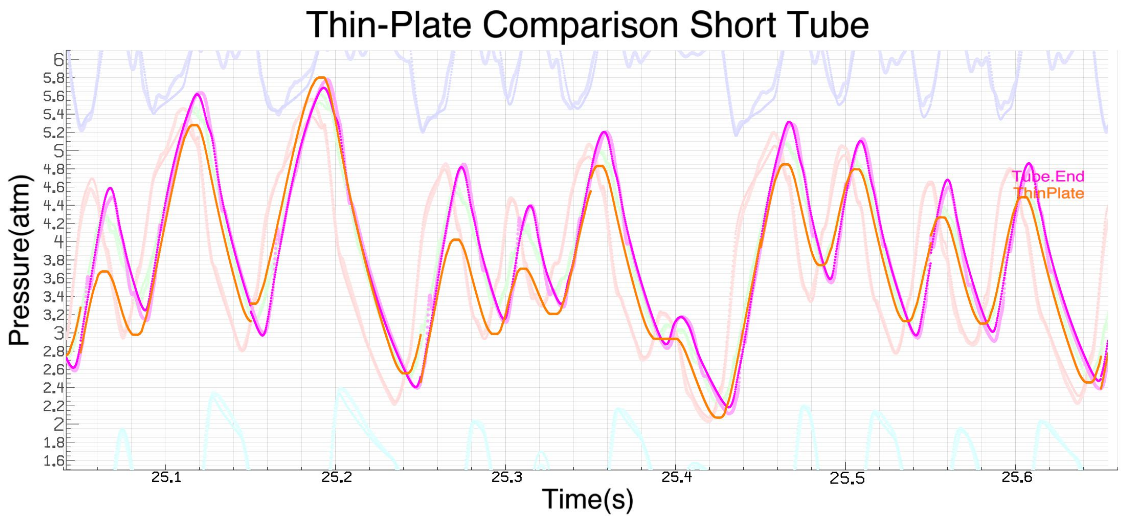
Figure 8.2: Pressure simulation inside a pneumatic tube with a sudden step control from fully open towards the ambient pressure to fully connected to the high-pressure source. The command happens at $t = 0$. There is about $5ms$ due to internal valve spool motion that we model correctly. The various sensors estimates are displayed in the corresponding saturated colors while the actual sensor values are displayed in the muted version of the color. The exhaust stays at ambient pressure the whole time because the system is not letting air out.

case we do not model that small feature very well. With a low enough viscosity parameters we correctly captures such effect, but overall the simulation is vastly under-damped. With viscosity settings that make the overall behavior correct we smear such small shock-waves. It is unclear whether such phenomena are important, and with a better PDE model, which is beyond the scope of the current work, such behaviors can be be potentially solved.

8.2.2 Results for random pressure trajectory

Here we investigate the more interesting regimes and compare predictive performance of our PDE-based model with the Thin-Plate model. Figure 8.3 shows the behavior on both the short and the long tube. It can be seen that our model matches the sensor values much better than the Thin-Plate model, while it misses both the position as well as the amplitudes of the features of the pressure waveform. The difference is exacerbated in the case of the long tube with the Thin-Plate model almost getting out of phase from the real values. This shows that our method is much better at tracking high-bandwidth behaviors.

Figure 8.4 shows the rollout behavior of just our method, but this time on all the sensors (as explained in fig. 8.1). Of note is that the the predictive power degrades to a lesser extent when we move onto the harder case (longer tube).



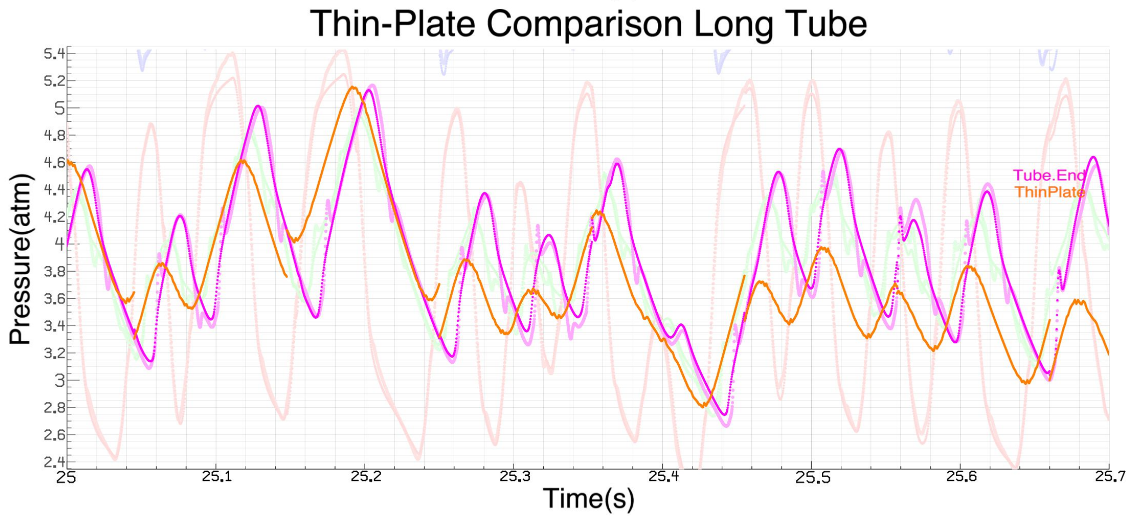
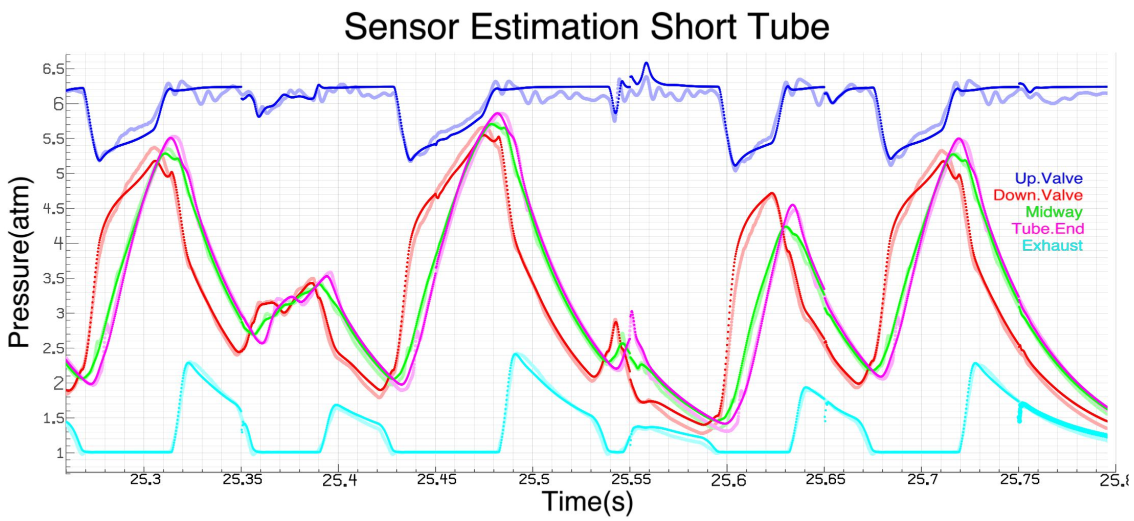


Figure 8.3: Comparison of the rollout behavior of the PDE-based model (in saturated magenta), with the performance of the Thin-plate based model (orange line). The reference pressure they are estimating is in light magenta and is hard to see because the PDE-based model overlaps it very well. Top graph is with the short tube, while bottom graph with the long tube. New rollout starts every 100ms.



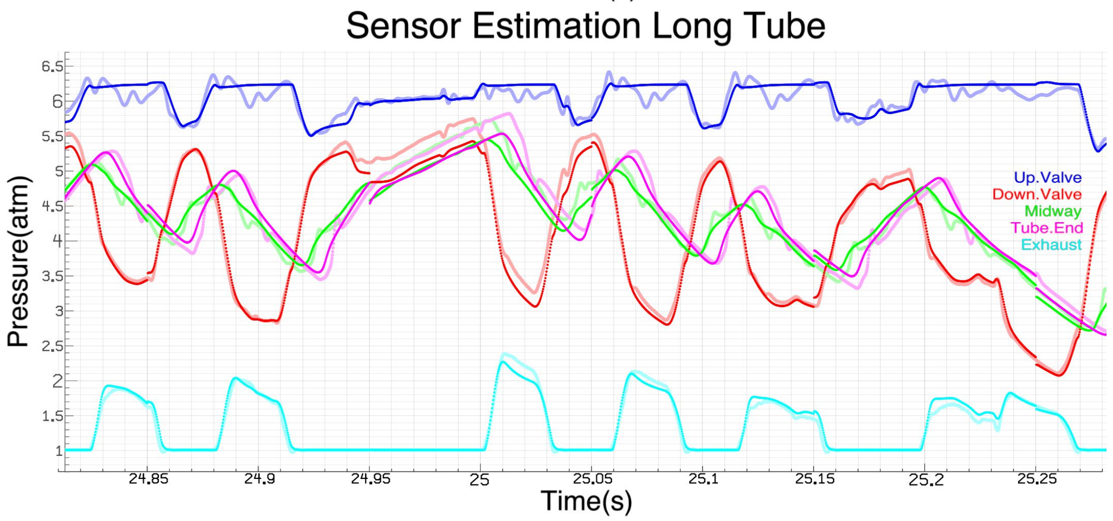


Figure 8.4: Comparison of sensor value predictions for all 5 sensors explained in fig. 8.1 of our PDE-based model with the actual sensor data. We correctly capture the behavior at all points in the pneumatic line, including the exhaust as well as the input port. Notice that with the longer pipe and some of the downstream sensors, there are some unexplained spikes of the pressure. They are result of mini shockwaves that happen with quick air motion that we induce and we cannot model them due to our system being overdamped.

Chapter 9

FROM SIMULATION TO REAL HARDWARE

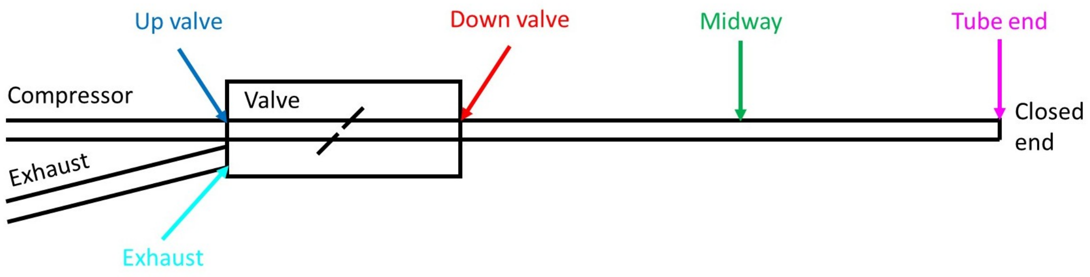


Figure 9.1: Setup for the basic pressure control task. We have proportional control over the valve and can decide in which direction and by how much to open it with the goal of achieving a desired pressure trajectory at the end of the tube (the magenta pressure sensor).

9.1 Controlling pressure

In this section we look into most basic system to establish whether our approach has benefits over prior methods such as the Thin Plate model. The simple system we are interested in is controlling the pressure at the end of a pneumatic line using proportional valve connected to two sources of pressure (low at ambient pressure, and high, coming from a compressor). This setup (shown in fig. 9.1) is the basic building block of any pneumatic system and the performance on it a the first indicator whether a method is viable.

We use the system identification process outlined in chapter 8. Then we used the Reinforcement Learning procedure outlined in chapter 7, entirely in simulation, using either the Thin-Plate model or our PDE-based model. The resulting policy is directly applied to the hardware, with care being taken to calibrate the pressure sensors to the best of our abilities and use the same calibration for both system identification as well as testing.

One of the advantages of a PDE based model is the much bigger state space. It is an advantage since a bigger state space provides higher number of observations as an input to the policy as well as the Value Function estimator which is an important component in training policies using reinforcement Learning. We therefore use all of the available pressure sensors during the training of the PDE-based model. We should note that the Thin-Plate model has only the notion of a single state variable which is the target pressure tap at the end of the tube, so we only used that one when training the Thin-Plate model.

One limitation is the absence of an observer for our model, which means that at the moment we are limited to using the sensor values that are provided by the physical sensors only. Hypothetically, given a state estimator, we would know the full state of the system at all times and will be able to use any number of virtual sensors. Given the accuracy of our simulation as shown in chapter 8, such an observer is definitely possible, though it was not pursued during this work.

9.1.1 Experimental Setup

The testbed consists of two different tubes of different lengths, the same as those mentioned in chapter 8. We use a $2.5m$ long tube as well as $5m$. While the longer tube might seem impractical at first, it can simulate situations where a much thinner tube is used.

The task is to achieve a desired pressure trajectory at the end of the tube. The agent is given information (as part of the observation vector) about the desired state at the moment as well as in the next $100ms - 200ms$. While this additional information is non-standard it makes sense, because any planner will have an idea of what its agent is going to do in the future, as opposed to being purely reactive and only responding to immediate feedback. The ability of an agent to make sense and plan through such future goals using its internal model is very important.

We use two different kinds of target pressure trajectories:

1. A smooth trajectory which is a sum of several sine waves of various frequencies and amplitudes

2. A jagged trajectory which is a sequence of random pressure targets at random time points, connected with a line.

Figure 9.2 shows the two types of trajectories that we have been using. Generally we keep

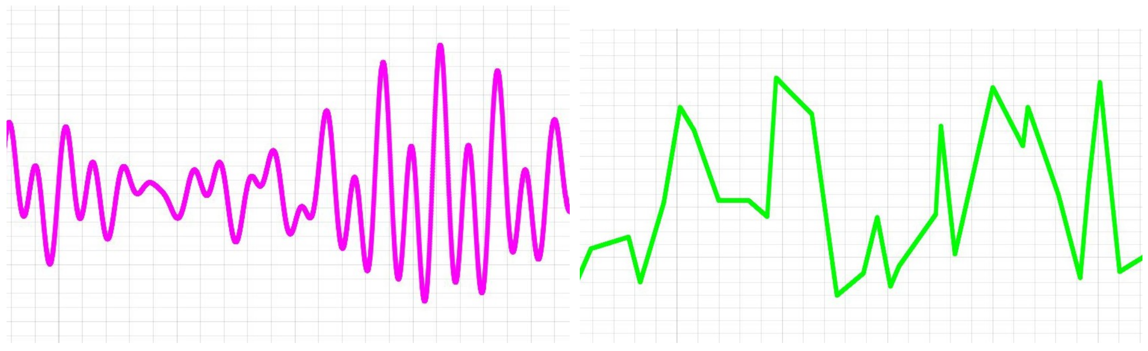


Figure 9.2: Sample of the two different types of trajectories used in our experiments. On the left is the smooth trajectory type and on the right the jagged.

the total amplitude roughly equal to the pressure range of the system, while we vary the frequency with faster tasks being harder.

9.1.2 Results

For each of the reported scenarios we perform 5 evaluation runs on the hardware and show an example run in the graphs, while showing the average RMSE (root mean squared error) in the tables.

Figure 9.3 shows the performance of both models on the short tube. Clearly our model behaves better both quantitatively and qualitatively. The jagged trajectory profile induces significant oscillations in the Thin-Plate model, while our PDE-based model can correctly model the system behavior and not enter unstable behavior. The difference is even more pronounced with the long tube shown in figure 9.4.

All the results are summarized in table 9.1.

9.1.3 Usefulness of extra state features

Here we investigate whether the presence of the extra features afforded by our model help the performance of the agent. The answer is yes (see fig. 9.5). The RMSE across all 5 runs (tested on the short tube with smooth target) for the agent trained with the extra features

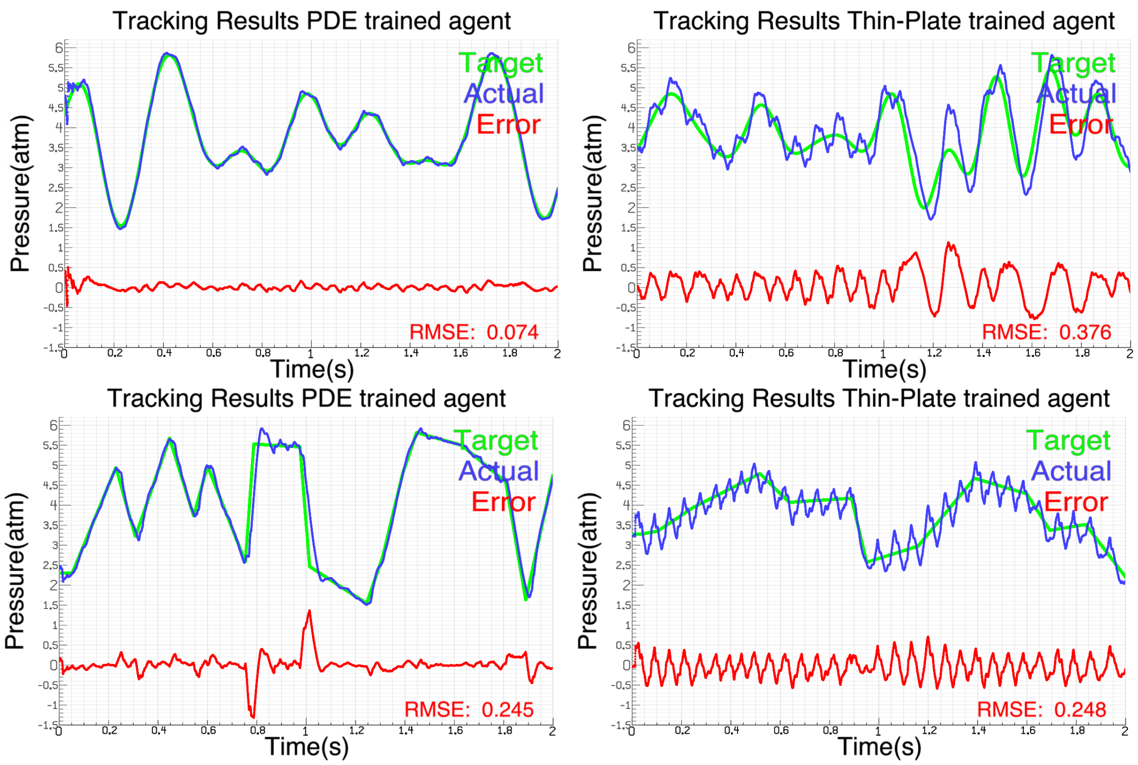


Figure 9.3: Sample behavior for the short ($2m$) tube. Top rows show behavior on the smooth target trajectory, while the bottom on the jagged target trajectory. Left column is our PDE-based model, while the right is the Thin-Plate based model. Clearly, the oscialltory behavior of the Thin-Plate model can be seen in the figures.

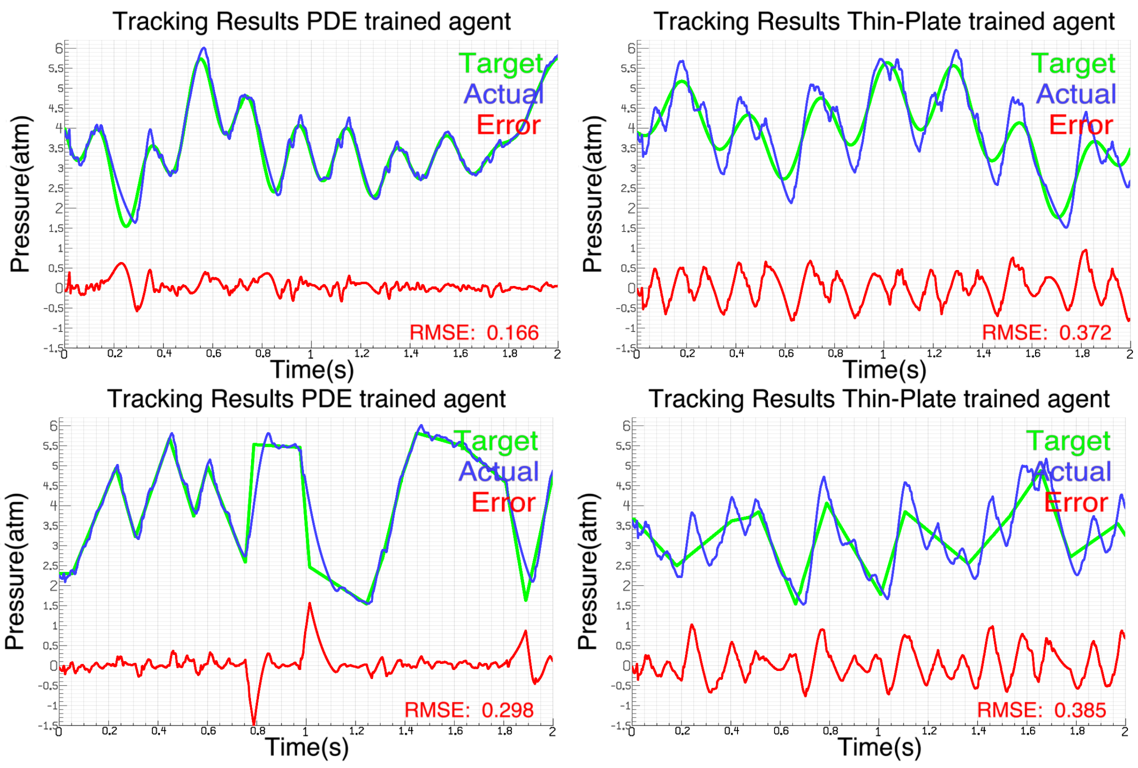


Figure 9.4: Sample behaviors on the long $5m$ tube. Top rows show behavior on the smooth target trajectory, while the bottom on the jagged target trajectory. Left column is our PDE-based model, while the right is the Thin-Plate based model. Here we see even more pronounced oscillations of the Thin-Plate trained agent. While the tracking error of the PDE trained agent increases, it is still both qualitatively and quantitatively better than the Thin-Plate model. Of note are the the big error spikes in the PDE-based model: They do not reflect our model's deficiency but an actual physical limitation of the system. It has only limited velocity at which it can move air, which is exacerbated at the lower end of the range (lower pressure) as the air dynamics are slower.

Task	RMSE PDE Model (atm)	RMSE Thin Plate (atm)
Short Tube Smooth Target	0.090	0.319
Short Tube Jagged Target	0.181	0.315
Long Tube Smooth Target	0.173	0.460
Long Tube Jagged Target	0.225	0.392

Table 9.1: Pressure Tracking Results for the two types of tasks and two pipe lengths.

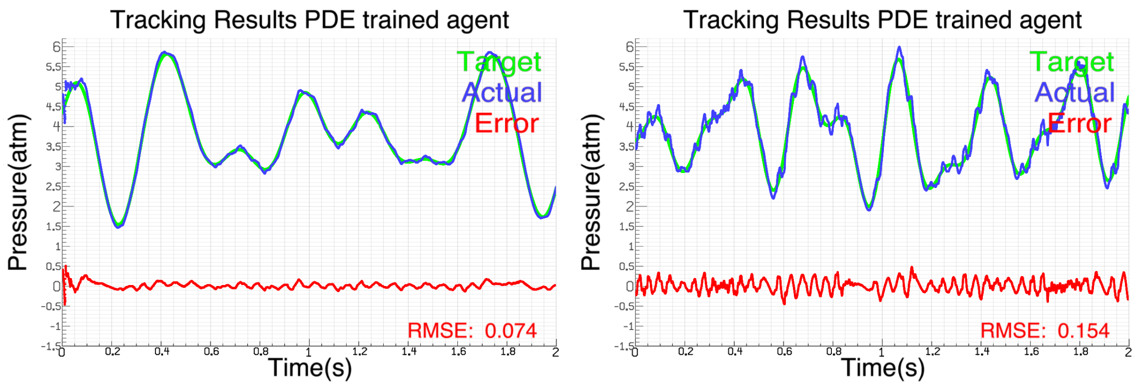


Figure 9.5: Left image shows the behavior when trained and using all the available sensor information, while the right image shows the behavior when only using the target pressure tap as our sensor in both training and execution.

is $0.090atm$, while the RMSE for the one without the extra features is $0.140atm$. While the performance is not as good as when using all the sensors, it is still quite better than the Thin-Plate agent, which shows that the Agent learned through Reinforcement Learning is still able to make use of the better model, even without making use of the extra data.

9.2 Controlling 1-DOF robot

In this section we apply our PDE-based model on a 2-DOF hardware system, shown in fig 9.6. It is a pneumatic test-bed by the Japanese Robotics company Kokoro. Section 9.2.1 describes the hardware, the system identification that we did on it, as well as the learning procedure. Section 9.2.2 sets the baseline performance with a PID controller. Section 9.2.3 shows the results of the from doing full-system learning and subsequent testing. Even though those experiments were not successful enough per our standards, we include them, as well as some other less great results to contrast them with our successful method and draw conclusion from that. Section 9.2.4 introduces our hybrid control and learning model and section 9.2.5. Finally section 9.2.6 offers discussion of some of the lessons we learned.

For our experiments we used only the second DOF, while clamping the first at a given value. Nothing in our setup prevents the method to be applied to full 2DOF system, and we leave that for future work.

9.2.1 Hardware and Software setup

Kokoro Robot

We will refer to the robot as Kokoro from now on. Kokoro is a 2-DOF robot with one rotational and one linear pneumatic actuator (shown in figure 9.6), arranged in a linear chain. The linear actuator is responsible for moving the second joint and is connected via a crank system to the end-effector, making the second DOF a rotational one as well. The resulting two axes are orthogonal to each other. The dimensions of Kokoro's parts were measured to the best of our abilities, with particular focus on the slider-crank mechanism, as it determines the force-to-torque ratio for the second joint, which varies by a factor of 2 across the range of the joint. The final model uses 2 rotational degrees of freedom, with a conversion formulas (or short iterative algorithm for the slider mechanism) to convert angles to cylinder positions, which is needed for modeling the pneumatic part of the system.

We label the two joints as α and θ. The resulting dynamics model is then computed to

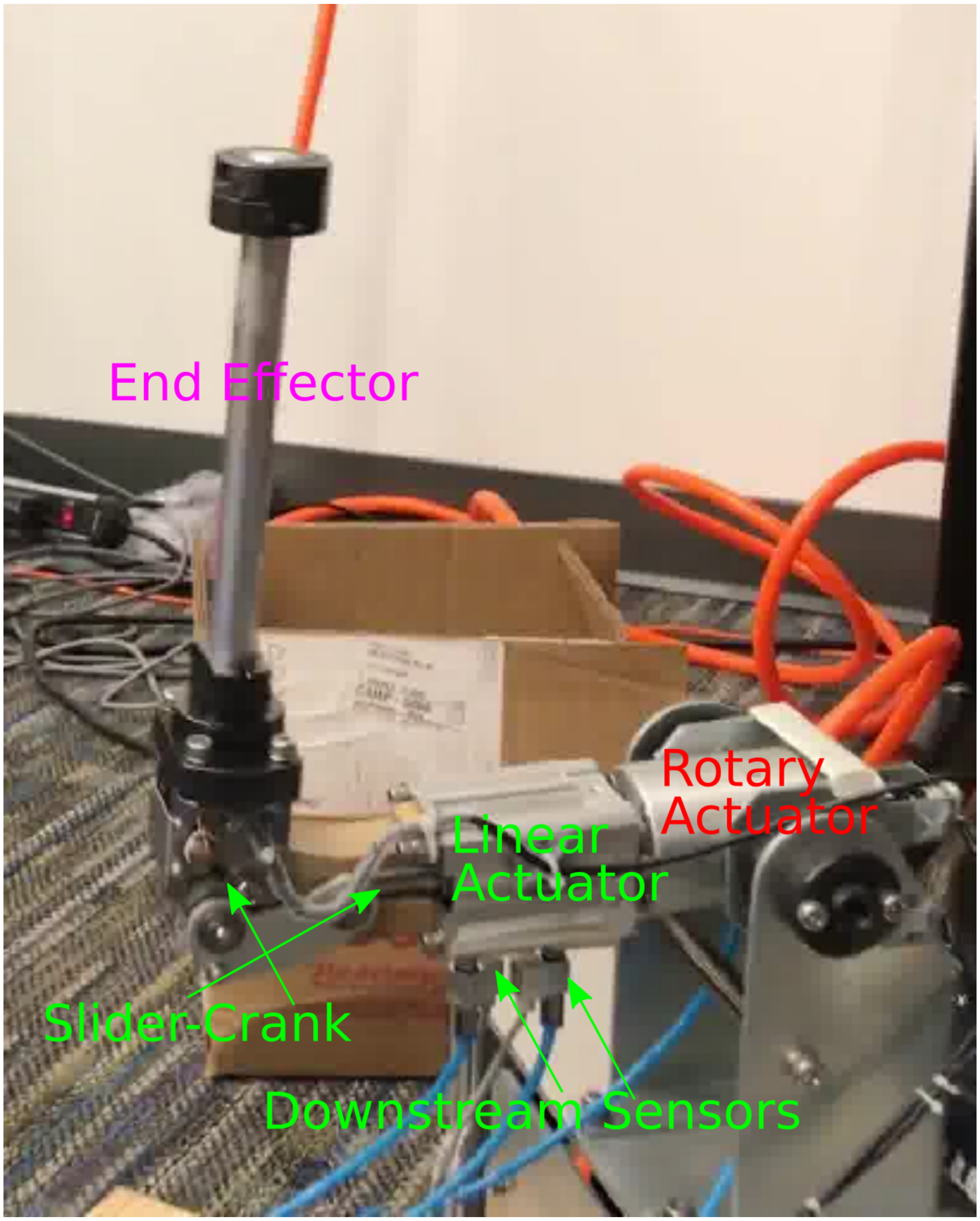


Figure 9.6: Picture of the Kokoro Robot. On the bottom are shown 2 of the 4 pressure sensors that we use. The other 2 are located next to the valves outside of the image.

be:

$$\begin{aligned} (\omega_1 + \cos^2\theta\omega_2)a_\alpha &= \tau_\alpha - \dot{\alpha}\delta_\alpha + 2\dot{\alpha}\dot{\theta}\sin\theta\cos\theta\omega_2 + \cos\theta\sin\alpha lgm_1 \\ \omega_2 a_\theta &= \tau_\theta - \dot{\theta}\delta_\theta - \dot{\alpha}^2\cos\theta\sin\theta\omega_2 + \sin\theta\cos\alpha lgm_1 \end{aligned} \tag{9.1}$$

where τ is the applied torque after conversion from actuator force, l is the position of the center of mass of the end-effector, $g = 9.81m/s^2$ is gravity, δ is the joint damping, and m_1, ω_1 and ω_2 are the rest of the inertial parameters (masses and moments of inertia respectively for joints 1 and 2).

We leave some of the non-measurable part of Kokoro for the system identification procedure. These include damping parameters for the two joints as well as the mass and position of center of mass of the second joint and moment of inertia of the first joint. As for the pneumatic part of the setup we used the same setup as with the previous section's (9.1) short tube of $2.5m$.

System Identification

The system identification process is exactly the same as in section 9.1, with the addition of the Kokoro's dynamics parameters. The dynamics computation for Kokoro was integrated into our Pneumatic solver as an extension to reduce the need for round-trip from the GPU back to CPU, which were the main cause for slowdown throughout all the simulations we have done with our system.

For the source of data we used a basic PID controller to drive the robot throughout its range of motion with frequencies similar to those encountered during learning and testing afterwards.

The setup uses two pneumatic lines and two valves for each chamber of the actuator. This time we only used two pressure sensors per pneumatic line: one just before the pipe entering the chamber and one just after the valve (figure 9.6).

Reinforcent Learning

For all the subsequent learning we use Reinforcement Learning to learn the policy/agent. We use the same framework as described in 7. The interesting part is the reward that we give as feedback to our agents, as well as the state variables (or derivatives thereof) that they have access to both in training as well as in testing.

The task is to follow a desired trajectory, of the same class as the ones described in 9.1 spanning the joint range.

The reward is a function of how close we are to the target at each moment in time: $-|position - target|$. Notice the negative sign, since positive reward is what the agent learns to optimize. We also add a sharper spike near 0 (of the form $-\log \max(|error|, error_{min})$ to promote further adherence once the gross tracking is achieved. We found that greatly helps tracking in section 7.

More interesting is the amount of type of observation we allow the agent to use. We have to make sure it is derived from sensors that are available at testing run-time, since we do not perform state estimation at the moment. In this scenario we use the two pressure sensors explained in section 9.2.1. Since we do not perform any pressure tracking, we only use these sensors as they are, without forming error terms.

Regarding the joint position terms we add the error terms for the joint position: $|target_{0ms} - position|$. In general we also add future error terms: $|target_{i\delta} - position|$, where $i \in [0, 4]$ and $\delta = 40ms$. This helps the agent optimize its current behavior, so it can better track future targets. While unorthodox, this strategy is exactly what a planning algorithm might do, so we do not consider that cheating. This allows our PDE-based model to shine, since its internal state can be better manipulated so as to move better in the future.

When the agent needs to achieve desired force, as introduced in section 9.2.4, we employ the same scheme as for future target joint positions.

Moreover, to increase the resolution of observations we use positional encoding as introduced in section 7.2. This allows our agent to better resolve and reason small errors.

To prevent overfitting to perfect behavior in simulation we only use two levels of positional enconding with base 2 up to 2^2.

In general, what data we put into the observation vector varies across the subsequent experiments and more is not always better. Each experiment details which set of data are used.

9.2.2 Baseline controller

As a baseline we use a PID controller derived from the PID controller described by Todorov et al. in [34]. The basic formula is:

$$u = k_p e - k_d \dot{e} + k_i g(\int e) + k_{pressure}(P_1 A_1 - P_2 A_2) \tag{9.2}$$

where $g()$ is smart integrator that clamps and resets the accumulated error. P and A are the pressures and areas of the two chambers. This is the only difference with the controller from [34]. They assume A is the same for both chambers, and since the result is the actual force acting on the piston, that assumptions is wrong.

The resulting control signal u is sent as $5+u$ and $5-u$ to the two valved correspondingly as per [34], since they have opposite effect on the resulting force and 5 is the neutral position.

The results are shown in figure 9.7. As is expected the controller exhibits delays and overall cannot accurately follow the desired target. This is inherent in part to reactive controllers that do not take future targets into account. However, the only way to take future targets into account is to have a model, so it is not accessible to a basic PID controller.

Table 9.2 shows the numerical results for all controllers. The apparent latency is estimated by calculating the optimal amount to shift one graphic so it aligns with the other. The adjusted RMSE is the error estimate after correcting for latency. The smooth target task does not benefit from latency correction, whereas the sawtooth task improves by a factor of two. We will use such table for comparison for each method we present.

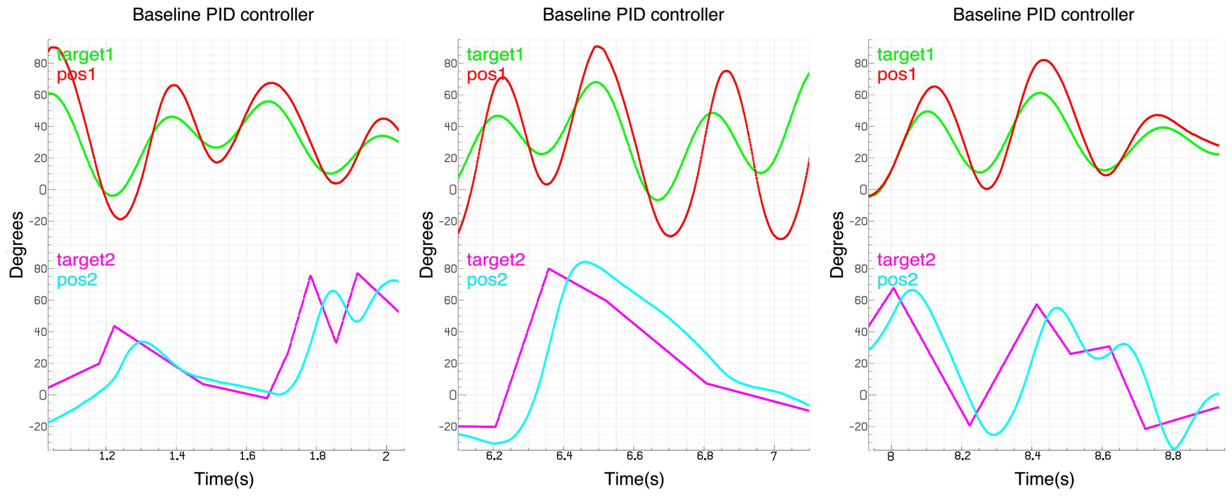


Figure 9.7: Three excerpts from executing the PID controller. Shown are both types of targets: the smooth (top) and sawtooth (bottom). Both delays are overall inaccuracies are clearly visible.

9.2.3 Full system experiment results

.

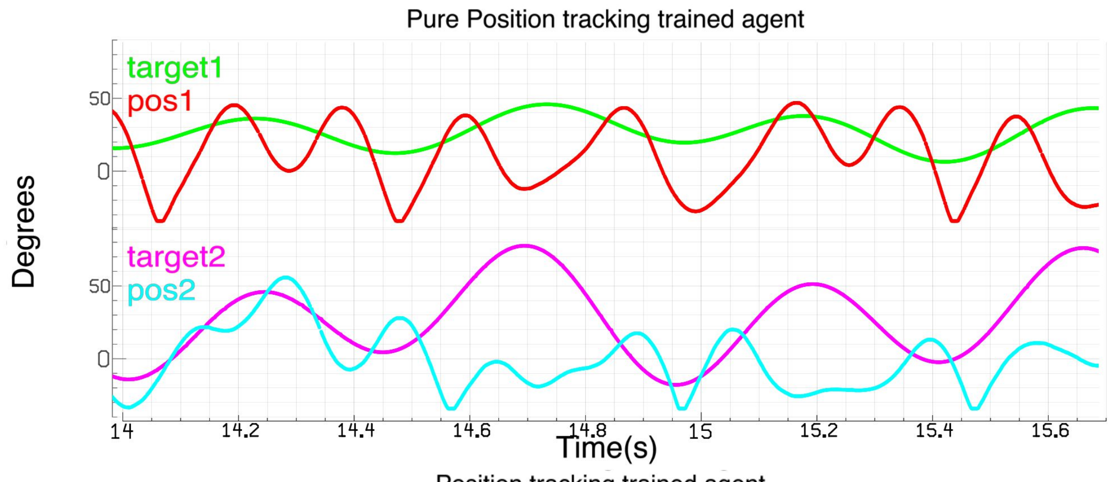
Basic positional tracking

In this experiment we just ask the agent to stay as close to the desired trajectory (giving only position-based reward signal) as well as position-based observations, plus the pneumatic internal state as explained in section 9.2.1.

Figure 9.8a show the performance relative to a target trajectory. The resulting policy was bad so we did not attempt on the sawtooth target at all.

Adding required force hints

Similarly to the last section in this experiment we ask the agent to stay as close to the desired trajectory but this time we add to the observation the force-based features. In essence that



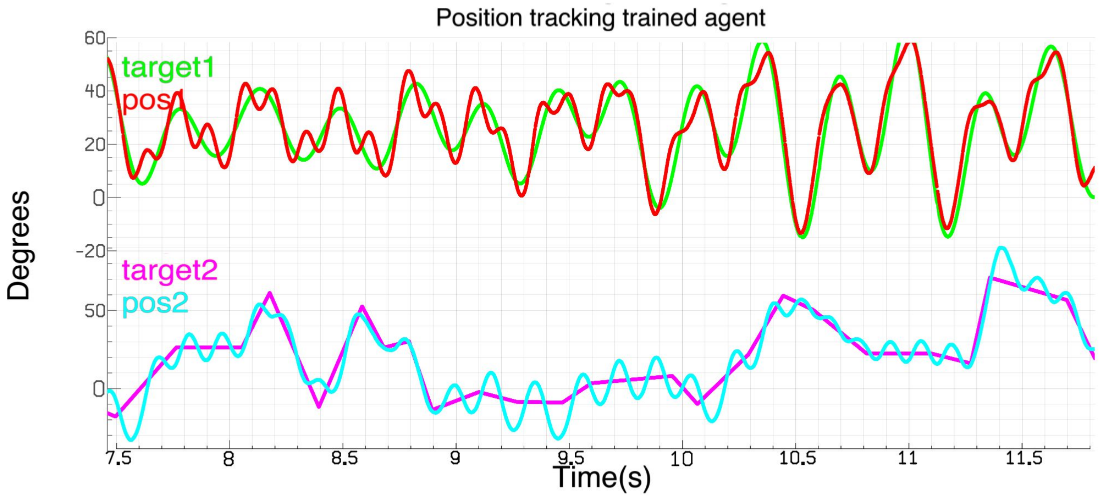


Figure 9.8: **(a)TOP:** Excerpt from evaluating an agent trained on purely tracking goal. **(b)BOTTOM:** Excerpt from evaluating an agent trained on tracking desired position with added force feedback hints.

could be used by the agent as a hint but we do not reward it for staying close to the required force.

Figure 9.8b show the performance relative to a target trajectory. The behavior is much better but clear oscillations still remain. Of note is the much lower latency compared to the baseline PID controller (table 9.2).

9.2.4 Hybryd Model

We already saw in section 9.1 that our PDE-based model is capable of producing a target pressure. In pneumatic actuation pressure and force are directly proportional, therefore we should expect the same framework to be able to produce required force profile.

We know for a basic fully actuated robot, we can use PID controller to almost perfectly track a desired trajectory, provided we can control the force input, especially if we know the inverse dynamics model of our robot (a third degree pneumatic system can no longer be considered fully actuated).

We implement a hybrid model, where we use the strengths of each type controller: a PID controller computes the overall trajectory for our tracking agent, as well as the piston positions and required forces at each timestep. We then give that information to the lower level RL-trained PDE-based pneumatics model and ask it to follow the required forces given the positions of the pistons for the near future. The near future is needed as per section 9.2.1.

Figure 9.9 show the algorithm in action. Given the current position and the desired target trajectory for the future, the PID controller comes up with a plan to get close to and join the desired trajectory. The bottom portion of the graphs show the current and past forces of the actuator as well as the required force at the actuator to achieve the plan shown in the top part of the graphs. As can be seen in the first two subplots the plan can change drastically when the agent is away from its goal. The third graph shows the behavior when the agent is adhering to the plan so there's continuity between the target and past required forces.

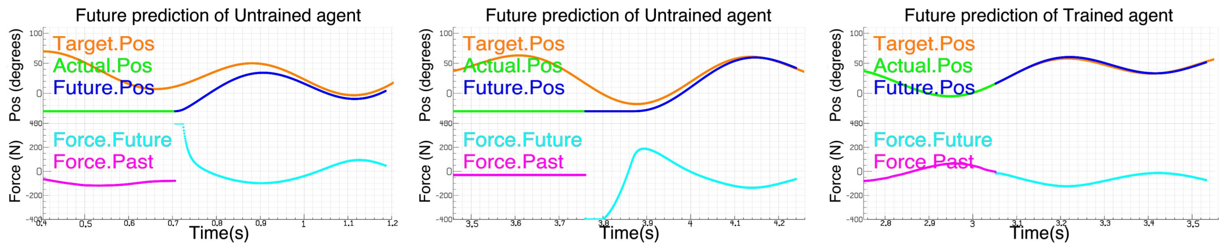


Figure 9.9: Three examples of PID high level setting targets for low level controller. The left two show an untrained agent that cannot track its desired position accurately. The right one is the same agent fully trained successfully tracking.

Training

The next question is how to train such an agent that can track desired future force given the future location of the piston inside the actuator. There are several approaches to this problem.

The most involved approach employs a setup similar to the one used in testing. There is a high-level PID controller trying to track a desired target position, computing the required future force as well as the expected piston position. However, we also add reward for staying close to the actual target position as well as staying close to the required force. Naturally we also include the position features into the observation vector. The expectation is that following both provides stronger training signal.

The second idea is the same but without the extra position reward nor position features, but still with the PID controller in the loop.

The third idea is to train an agent to follow a desired force trajectory given position trajectory of the piston that is completely detached from the intended task. This does not require knowledge of Kokoro at all nor does it employ a PID high level controller.

We compare the performance of these ideas in the next section.

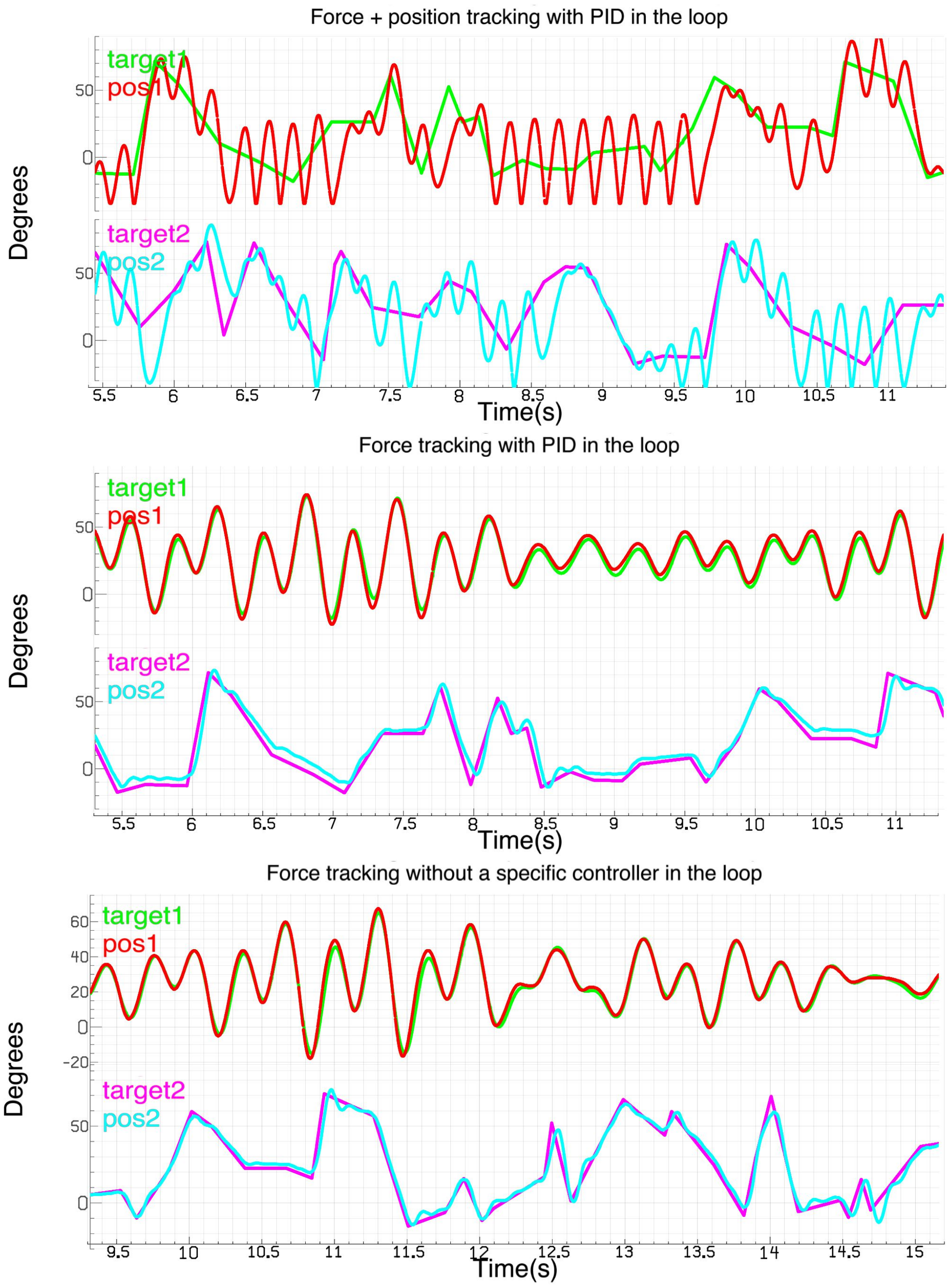


Figure 9.10: **(a)TOP:** Excerpt from evaluating an agent trained on tracking both desired force plus desired position. **(b)MIDDLE:** Excerpt from evaluating an agent trained only on required force to achieve desired trajectory, computed via a PID controller. **(c)BOTTOM:** Excerpt from evaluating an agent trained on tracking desired force given piston positions generated at random via our trajectory generating method, without using any controller in the loop.

	Metric:	RMSE	Apparent Latency	Adjusted RMSE
PID Controller	Smooth	13.7°	20ms	11.6°
	Sawtooth	17.9°	63ms	9.4°
Pure Position Tracking	Smooth	33.6°	41ms	33.3°
	Sawtooth			
Position Tracking + Force Hints	Smooth	6.0°	$-4ms$	6.0°
	Sawtooth	8.3°	9ms	8.2°
Hybrid(PID) Force + Position Tracking	Smooth			
	Sawtooth	19.7°	34ms	18.8°
Hybrid(PID): Force Tracking	Smooth	4.0°	$-3ms$	3.8°
	Sawtooth	8.6°	26ms	7.3°
Hybrid: Force Tracking	Smooth	2.7°	$-5ms$	2.3°
	Sawtooth	4.4°	16ms	3.4°

Table 9.2: Numerical results for all the controllers.

9.2.5 Hybryd Model Results

Tracking both force and position

In this experiment we make the agent learn to follow a desired future force. We also add rewards for positional tracking, including all the corresponding features. In this case we use a PID controller as the driving source for the target force.

Results are shown in figure 9.10a. Lots of oscillations are present so we skipped testing on the smooth target.

Force-only tracking with PID controller

Similar to last section, except we do not add any position-related features but still use a PID controller in the training phase.

Results, show in figure 9.10b. Behavior is much better than the previous methods, but still lacking compared to the last experiment.

Uncorrelated training trajectories

Lastly, we exclude the PID controller from the training phase, letting the agent learn to follow completely unrelated force trajectory with mismatched piston position trajectory. That mismatch is actually making the problem quite harder, since the two trajectories are usually correlated, while with this method we use independent trajectories.

Results are shown in figure 9.10c. These are our best results and numerical details can be found in table 9.2. The reason for the improvement is the much wider range of training trajectories that we expose our agent during training. When a PID controller is in the loop the agent learns to narrowly follow only a successful set of commands that appears in the late stages of the training process, since then our agent learns to perfectly execute its commands. This however makes it less robust to a real world scenario where we might deviate from the correct trajectory due to other modeling errors.

9.2.6 Conclusion

We found that Sim-2-Real with a third order system is quite challenging. Interestingly in all of our experiments using position related features and rewards worsens the performance. We think this is a result of overfitting to very a specific successful mode of operation in training and then failing due to getting out of successful regime due to slight modeling errors. This is exacerbated by the system being third order which complicated the relationship between the control signal and the position quite a lot.

Results were always improved when we added either force related features or rewards.

This might be a result of having only first order system between control and desired force and that helps the controller.

The best results were achieved using the Hybrid Model which emphasized force tracking the most, and even within it the best results were obtained by not learning to track position, but only rely on the required force set by a simpler PID controller.

Even within this set of methods, the better agent was the one that did not use a PID controller in the training loop and was training to follow arbitrary trajectory in force space. This, again, is probably due to not overfitting on a specific successful regime of operation, but instead exploring the space thoroughly.

The most important conclusion we draw from these set of experiments is that overfitting with sim-2-real is a real danger, or in other words, mode mismatch between training and testing on the real hardware. One successful way of attacking this problem is proper separation of the system into multiple levels, where each can be tackled with a tool that is most appropriate for it.

Chapter 10

CONCLUSIONS

Contact modeling

When it comes to contact modeling consistency is what really matters. While our method estimates certain parameters like friction and contact softness, the models that employ these parameters are merely idealizations of the true contact phenomenon which is still far from being modelled truthfully. In fact, a model can always get more complex in its attempt to resolve with even finer resolution, seemingly to no end. Instead, what we really strive is to be able to tell a consistent story, one that is useful for the task at hand. Consistent here means an internal representation of the environment together with our robot itself, that is derived from the sensor data and is able to explain those sensor data. That story necessarily includes some model parameters and whether they correctly map to textbook formulas is not as important as whether they help make the story more consistent.

Regardless of the choice of parameters and state variables, we show that for highly nonlinear phenomena, where small state changes lead to huge changes in dynamics, separately identifying both is impossible and a joint solution is needed. We show that our method is robust to varying the dataset and the resulting parameters are close to each other.

This idea has implications about real-time control. It might not be possible to always have a model, given that model parameters do not always have basis in reality, especially when a robot encounters a new environment. The processes developed in this thesis might need to be applied in real time, which is enabled by the framework we created.

Pneumatic actuation

Pneumatic actuation has several superior characteristics over other types of actuation. It has higher force per volume or mass for the actual actuator, as well as it is naturally compliant, without relying on gearboxes that isolate the force producing element from the end-effectors. It has some similarity to hydraulic actuation, however, hydralics lacks the compliance and are quite messy and dirty potentially. Although the actuator itself is fairly small, its control mechanism as well as pressure source are quite the opposite, usually big and heavy, which is why they are often located in quite different parts of the robot (or even outside of it) and therein lie some of the challenges of using them.

Naturally pneumatic actuation is assumed to lead to a third order system, which means we have direct control over the gradient of the force. This is roughly correct, but the separation of control mechanism and actuators makes this much more complicated, as the valve does not directly output towards the actuator. The control that we send flows through the connecting pipes and is delayed and attenuated before it reaches the actuator. These are the challenges we attempt to solve with our PDE-based model.

Historically fluid dynamics simulations and solutons via Partial Differential Equations have been out of the realm of real-time robotics, due to inherently being computationally intensive. We show that that is a barrier that can be overcome if we make the right assumptions, such as using one dimensional simulation rather than full 3D simulation. Especially nowadays with the advent of massively parallel computing devices real time fluid dynamics simulations are possible as we have shown.

Throughout the development of our method we discovered that the most essential aspects of our model are the abilities to simulate the effective delay of control commands as well as flow attenuation due to various viscosity effects and the length and finite cross sectional area of the pipe. One might wonder then: Would it be possible to achieve the same fidelity results with some delay mechanism and a simple attenuation rule. A delay mechanism requires us to have some expanded state space to remember previous commands. We also extend the state space by discretizing the pipes and cylinders, and that yields exactly the same effect as a delay mechanism would. The added extra computational cost, while not insignificant is purely local computation, that is efficiently solved by a GPU for example and the pure flow computation cost is on the order of 2% to 5% of the total time, when training an agent via Reinforcement Learning. There is an added benefit of solving the correct fluid equations as well.

Moreover as we show in section 6.1. we do not need extremely fine state discretization. While most of the experiments shown use around 100-200 cells to model the pneumatic system, it might be possible to manage with around 30 or less, though we leave such optimizations as future work.

Running these simulations on a massively parallel devices such as a GPU, has the benefit of being able to run hundreds of simulations of the same system in parallel, which is very useful for certain model predictive control schemes, such as Model Predictive Path Integral. We found great success with Reinforcement Learning, where it is common in training to execute multiple simulations in parallel, so we propose the MPC direction as future work.

Sim2Real

Transferring a learned behavior in simulation to a real hardware system (Sim2Real) has been very challenging problem. We show that in the case of controlling the pressure at the end of a pneumatic line (which is a basic building block for a pneumatic controller as we showed earlier) our method surpasses the prior work in terms of performance, allowing us to reach the system's maximal performance in some cases with a direct Sim2Real.

When moving to an actual robot, with pneumatic actuation, things are more complicated. An agent learned with the fully combined model (body dynamics + pneumatic actuation), while producing overall correct behavior exhibits big oscillations. This is a result of the learning algorithm overfitting to a specific successful regime of operation, therefore unable to handle both deviations from the plan or certain modeling errors.

While those are in theory corrected by encouraging more exploration, and using ensemble of models, we found that those ideas are not sufficient to overcome this issue. The richness and complexity of the combined model result in more overfitting opportunities. A PID controller, is quite simple and with proper tuning is very stable.

Being able to split a system into independent levels, in other words, building abstraction layers is one of the most important concepts in dealing with complex systems. Pure separation is rarely possible but in the case of pneumatic robots we have shown that it is a reasonable approach. Our PDE-based model together with a learned agent can successfully serve as a lower level controller that faithfully executes commands coming from a high level controller, in this case a simple PID controller for a fully-actuated 2DOF system.

Pure separation is not achievable of course. A high level controller might request a behavior that is impossible, for example changing the pressure too fast is physically impossible. However, encoding such a simple constraint within a high level controller is much easier to implement than having to jointly devise a plan for the low level pneumatics as well as the high level dynamics, as is the case when using fully combined model.

Another, perhaps bigger, problem with such an approach is the specific regime in which you train the low level controller. If one uses the high level controller as a driving source when learning the lower level controller, then the learned agent will only know how to follow orders coming only from a successful execution. When transferring to a real work scenario, the high level controller is going to encounter a new mode of operation, which will drive the low level controller into an unknown regime and fail. This is exactly what we observed with our experiments.

The best approach for hybridizing such a system is to let the low level controller train independently, on its fullest range of state so that at run-time it will not be surprised by an impossible task. Moreover we show that our PDE-based model is able to perform the task in such scenarios and can be used as the basis of a low level pneumatic controller.

Conclusion

With the advent of data driven methods (non-parametric or unstructured), one can forget that physics can in fact be useful. Better physics models in robotics do exist and do lead to better control and estimation. While a perfect model will never exist, and there will always be a room to combine physics with a data-driven approach, we have demonstrated improvements in both contact modeling as well as state of the art pneumatic actuation modeling.

BIBLIOGRAPHY

[1] Shun-Ichi Amari. “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2 (1998), pp. 251–276.

[2] J. E. Bobrow and B. W. McDonell. “Modeling, identification, and control of a pneumatically actuated, force controllable robot”. In: *IEEE Transactions on Robotics and Automation* 14.5 (1998), pp. 732–742. DOI: 10.1109/70.720349.

[3] Maxime Chalon, Jens Reinecke, and Martin Pfanne. “Online in-hand object localization”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 2977–2984.

[4] R. Courant, K. Friedrichs, and H. Lewy. “Über die partiellen Differenzengleichungen der mathematischen Physik”. In: *Mathematische Annalen* 100 (Jan. 1928), pp. 32–74. DOI: 10.1007/BF01448839.

[5] Evan Drumwright, Dylan Shell, et al. “An evaluation of methods for modeling contact in multibody simulation”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 1695–1701.

[6] Tom Erez et al. “An integrated system for real-time Model Predictive Control of humanoid robots”. In: *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*. 2013.

[7] G Gilardi and I Sharf. “Literature survey of contact dynamics modelling”. In: *Mechanism and machine theory* 37.10 (2002), pp. 1213–1239.

[8] Sergei Godunov. “Different Methods for Shock Waves”. In: *Moscow State University* (1954). URL: http://self.gutenberg.org/articles/eng/Sergei_K._Godunov.

[9] N. Gulati and E.J. Barth. "Non-linear pressure observer design for pneumatic actuators". In: *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM* 1 (Aug. 2005), pp. 783–788. DOI: 10.1109/AIM.2005.1511078.

[10] Navneet Gulati and E.J. Barth. "A Globally Stable, Load-Independent Pressure Observer for the Servo Control of Pneumatic Actuators". In: *Mechatronics, IEEE/ASME Transactions on* 14 (July 2009), pp. 295–306. DOI: 10.1109/TMECH.2008.2009222.

[11] L.E. Humphrey. *Hagen-Poiseuille Flow from the Navier-Stokes Equations*. Claud Press, 2012. ISBN: 9786201150461. URL: https://books.google.com/books?id=uGJkLwEACAAJ.

[12] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems* 14 (2001).

[13] V. Kalikmanov. *Statistical Physics of Fluids: Basic Concepts and Applications*. Theoretical and Mathematical Physics. Springer Berlin Heidelberg, 2013. ISBN: 9783662045374. URL: https://books.google.com/books?id=2YEBswEACAAJ.

[14] Michael C Koval et al. "Pose estimation for contact manipulation with manifold particle filters". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 4541–4548.

[15] V. Kumar et al. "Real-time behaviour synthesis for dynamic Hand-Manipulation". In: *Proceedings of the International Conference on Robotics and Automation (ICRA 2014)*. 2014.

[16] K. Lowrey et al. "Physically-Consistent Sensor Fusion in Contact-Rich Behaviors". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS)*. 2014.

[17] K. Lowrey et al. "Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system". In: *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. 2018, pp. 35–42. DOI: 10.1109/SIMPAR.2018.8376268.

[18] Ben Mildenhall et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis”. In: *CoRR* abs/2003.08934 (2020). arXiv: 2003.08934. URL: https://arxiv.org/abs/2003.08934.

[19] Michael Montemerlo et al. “FastSLAM: A factored solution to the simultaneous localization and mapping problem”. In: *AAAI/IAAI*. 2002, pp. 593–598.

[20] Igor Mordatch, Zoran Popović, and Emanuel Todorov. “Contact-invariant optimization for hand manipulation”. In: *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*. Eurographics Association. 2012, pp. 137–144.

[21] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*. Fluid Mechanics and Its Applications. Springer International Publishing, 2015. ISBN: 9783319168746. URL: https://books.google.com/books?id=GYRgCgAAQBAJ.

[22] B.R. Munson, D.F. Young, and T.H. Okiishi. *Fundamentals of Fluid Mechanics*. Wiley, 2005. ISBN: 9780471675822. URL: https://books.google.com/books?id=_cGzQgAACAAJ.

[23] Richard A Newcombe et al. “KinectFusion: Real-time dense surface mapping and tracking”. In: *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE. 2011, pp. 127–136.

[24] J.M. Powers. *Lecture Notes on Gas Dynamics*. Joseph Michael Powers. URL: https://books.google.com/books?id=3G6JggTnjekC.

[25] Aravind Rajeswaran et al. “Towards generalization and simplicity in continuous control”. In: *arXiv preprint arXiv:1703.02660* (2017).

[26] Edmond Richer and Yildirim Hurmuzlu. “A High Performance Pneumatic Force Actuator System: Part I—Nonlinear Mathematical Model”. In: *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR* 122 (Sept. 2000). DOI: 10.1115/1.1286336.

[27] Mark Schmidt. *MinFunc*. 2005.

[28] Tanner Schmidt, Richard Newcombe, and Dieter Fox. “Dart: Dense articulated real-time tracking”. In: *Proceedings of Robotics: Science and Systems, Berkeley, USA* 2 (2014).

[29] Tanner Schmidt et al. “Depth-Based Tracking with Physical Constraints for Robot Manipulation”. In: *IEEE International Conference on Robotics and Automation*. 2015.

[30] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).

[31] John Schulman et al. “Trust region policy optimization”. In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[32] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018. ISBN: 9780262039246. URL: https://books.google.com/books?id=6DKPtQEACAAJ.

[33] Yuval Tassa et al. “Modeling and identification of pneumatic actuators”. In: *2013 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2013* (Aug. 2013), pp. 437–443. DOI: 10.1109/ICMA.2013.6617958.

[34] E. Todorov et al. “Identification and control of a pneumatic robot”. In: (2010), pp. 373–380. DOI: 10.1109/BIOROB.2010.5627779.

[35] Emo Todorov. “Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 6054–6061.

[36] E.F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. Springer Berlin Heidelberg, 2009. ISBN: 9783540498346. URL: https://books.google.com/books?id=SqEjX0um8o0C.

[37] Diederik Verscheure et al. “Identification of contact dynamics parameters for stiff robotic payloads”. In: *Robotics, IEEE Transactions on* 25.2 (2009), pp. 240–252.

[38] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson education, 2007.

[39] Eric Wan, Ronell Van Der Merwe, et al. "The unscented Kalman filter for nonlinear estimation". In: *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. IEEE. 2000, pp. 153–158.

[40] F.M. White. *Fluid Mechanics*. McGraw-Hill series in mechanical engineering. McGraw Hill, 2011. ISBN: 9780073529349. URL: https://books.google.com/books?id=egk8SQAACAAJ.

[41] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[42] Tingfan Wu et al. "STAC: simultaneous tracking and calibration". In: *IEEE/RAS International Conference on Humanoid Robots (HUMANOIDS)*. 2013.

[43] Li Emma Zhang and Jeffrey C Trinkle. "The application of particle filtering to grasping acquisition with visual occlusion and tactile sensing". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 3805–3812.