SkinProbe 2.0

Development of a System for Low-Cost Measurement of Human Soft Tissues

by

Alistair Wick

M.Eng., University of Bristol, 2016

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

October 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

SkinProbe 2.0: Development of a System for Low-Cost Measurement of Human Soft Tissues

submitted by **Alistair Wick** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Dinesh K. Pai, University of British Columbia
Supervisor

James J. Little, University of British Columbia
Examining Committee

Abstract

We present "SkinProbe 2.0," a prototype system for low cost, high volume measurement of the physical properties of human soft tissues through direct contact and perturbation of the skin. Our solution encompasses a handheld device and associated cloud-based AI processing pipeline, and derives physically-representative values of stiffness and thickness directly from video. These input videos include images of the surface under contact, and of a "flexure," our novel apparatus for optical force measurement. Videos are captured using a smartphone embedded in the device.

Our system processes these videos, generating dense optical flow fields for selected frames, and passing these frames and flow fields through two bespoke Neural Networks: one providing estimated force readings, and one providing estimates of soft-body material properties in the contact vicinity.

We automate the collection of training data for our networks with robotics and a 3D-printed apparatus, along with custom-made silicone tissue phantoms, and a cloud pipeline for data collection, storage, and retrieval. This allows us to scale to thousands of samples in each training dataset, with minimal human involvement in collection, and a highly repeatable collection process.

We demonstrate the functionality of our measurement device, cloud pipeline, and force estimation system, and show promising material estimation results on our tissue phantoms. We further consider directions for future research in improving our system, both for handheld data collection, and for eventual usage on human subjects.

Lay Summary

We developed “SkinProbe 2.0,” a handheld probing device, which we hope to use in the future to measure how people’s skin moves and reacts to small amounts of pressure. Our device is currently a prototype, and not yet suitable for use on people. In the future, these results could let us use computers to help create customized clothing with a perfect fit, or allow us to automatically optimize prosthetics for an individual, making them more comfortable and secure. The device is low-cost and highly portable, and uses cloud computing and machine learning to help generate its results. We lowered the cost by basing the device around an ordinary smartphone, and using the phone’s camera for all our sensing needs – it observes movement of both the skin, and of a special spring-like device we call a “flexure,” which allows the camera to measure the pressure that is applied.

Preface

The UBC Sensorimotor System Laboratory's "Skincap" project is the direct predecessor to, and parent of this work. Within Skincap, we sought to develop and refine experimental techniques and tooling for the measurement of human soft tissues, with the goal of improving the physical accuracy of simulations for animations, and for tasks like computer-aided design of clothing. I participated extensively in this project, designing the version 1 Skin Probe (Section 1.4) and major portions of the associated capture software; other members of the lab collected data from human participants using this system, provided feedback on the design, and obtained results from human participants. This work led to our 2018 paper *The Human Touch: Measuring Contact with Real Human Soft Tissues* [34] (ACM Transactions on Graphics), on which I was one of many co-authors. This was the first publication describing and utilizing the SkinProbe V1. The SkinProbe 2.0 project would not have been conceived without the hard work of all the authors on that paper; that said, nobody other than Dr. Pai and I have made direct contributions to the development of SkinProbe 2.0.

As the name implies, SkinProbe 2.0 is positioned as a successor of, and potential replacement for the V1 probe. In Section 1.4, I outline the design and usage of this original probing system, and explore the shortcomings which led to a desire for this replacement version. Section 1.4.1 is based substantially on my own work in the paper discussed above [34], section 3, "Design and Fabrication of Skin Probe".

My supervisor, Dr. Pai, was largely responsible for conceiving this project, and provided valuable input throughout. I carried out the development of this system myself: I wrote the code, designed and fabricated parts for the prototype device, and carried out experiments with the system. Dr. Pai and I are the only contributors

to the work presented in this thesis, again excepting the previously-published V1 probe discussed in Section 1.4.

With the exception of Section 1.4.1, I wrote the entirety of this thesis as an original work; no other sections are based on any other sources, published or unpublished.

We did not require ethics board approval for this work, as no human subjects were involved in testing this prototype system.

Table of Contents

List of Tables

List of Figures

Glossary

AJAX Asynchronous JavaScript and XML, a set of tools and techniques which enables live interaction between client-side JavaScript and a web server.

API Application programming interface, a generic term for a set of technical commands and capabilities exposed by a software component.

AR Augmented reality, an interactive medium which mixes sensory input from the real world with generated virtual input.

AWS Amazon Web Services, a collection of cloud computing platforms and services offered by Amazon.

CAD Computer-aided design, the use of computer software in developing and modifying engineering designs.

CI Continuous integration, a development methodology where new code, models and data are frequently (continuously) loaded onto a production-like environment for automated and/or manual testing.

CNN Convolutional neural network, a DNN employing one or more convolutional layers – essentially, learned filters – for signal or image processing.

DNN Deep neural network, an artificial neural network with several stacked "hidden" layers, commonly used in modern machine learning tasks.

DRY Don't repeat yourself, a software development methodology where repetition of code and data is reduced or eliminated, providing a single location for editing each piece of "knowledge."

EM Electromagnetic, a term describing forces and fields around electrically charged particles, relating electricity and magnetism.

FEM Finite element method, a numerical method for discretizing problems such as fluid and soft-body simulations.

FPS Frames per second, the number of frames of video recorded or displayed per second.

GPU Graphics processing unit, a dedicated, highly parallel co-processor commonly found in modern desktops and servers, used to accelerate parallelizable tasks, such as image processing and large matrix operations.

GUI Graphical user interface, a visual, virtual, interactive interface consisting of display and interactive elements such as text, images, and push-buttons.

HTML HyperText Markup Language, an XML-like markup language used to define documents for display in a web browser.

HTTP HyperText Transfer Protocol, a common communications protocol for requesting and receiving documents from web servers.

IPC Inter-process communication, methods allowing for data exchange between separate software processes, possibly on different computers.

JSON JavaScript Object Notation, a widely-used, human-readable serialized data format. Often used to replace XML in client-server communication.

LSTM Long short term memory, a popular type of recurrent neural network which works well in practice.

MAE Mean absolute error, a distance metric between two sets of values, which sums their element-wise absolute differences:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - x_i|$$

MRI Magnetic resonance imaging, a method for non-invasive imaging of the interior of the body using powerful magnetic fields.

MSE Mean squared error, a distance metric between two sets of values, which sums their element-wise squared differences:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2$$

NSERC Natural Sciences and Engineering Research Council of Canada, a funding body for scientific research in Canada.

OOD Out-of-distribution [data], describing data which is anomalous or otherwise outside of the distribution of training data in a machine learning task.

PLA Polylactide, a polyester thermoplastic made with plant-based compounds, commonly used for 3D printing. Also known as "polylactic acid."

ReLU Rectified linear unit, a simple and popular activation function for artificial neural networks, essentially comprising $y = \max(0, x)$

RGB Red-green-blue, an additive color model where red, green, and blue principal components are added to create a range of colors, typically used for digital storage of color images.

RMSE Root mean square error, a common outlier-sensitive metric for regression model performance. Lower values indicate a better fit, with 0 being a perfect fit.

S3 Simple Storage Service, a bulk data storage service in AWS.

SVD Singular value decomposition, a factorization which separates a matrix M into the form USV^T.

Acknowledgments

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) Idea to Innovation (I2I) program and Vital Mechanics Research.

I'd like to thank Dinesh Pai for allowing me to join his lab, for his support and direction in achieving this milestone, and for allowing me the professional freedom to experiment with different techniques and technologies, as well as the personal freedom to maintain links with family and friends at home. I also wish to thank Jim Little for agreeing to be my second reader, and for his timely and helpful feedback.

To my labmates, past and present: Austin Rothwell, Pearson Wyder-Hodge, Jan Hansen, Seung Heon Sheen, Prashant Sachdeva, Egor Larionov, Ziheng Liang, Ye Fan, Matt Dietrich, Yuchi Zhang, Edwin Chen and others. Thank you for an enjoyable and enlightening time, and for all the discussions and help.

I've had many wonderful experiences here, for which I must thank the many friends I've made since arriving: Siddhesh Khandelwal, Giovanni Viviani, Hayley Guillou, Neil Newman, Kuba Karpierz, Yasha Pushak, Anna Scholtz, Clement Fung, Fabian Ruffy, Michael Oppermann, Nico Ritschel, Puneet Mehrotra, Jan Pilzer, Adam Hutter, and many more. You've hiked, biked, road-tripped and inflatable kayak-ed with me around this beautiful country, and helped me tour many a beautiful pub.

Living away from family and friends can be difficult, especially when you're a 9-hour flight from them. I couldn't have survived here without the support of the people I left behind. Thank you to my parents, my friends, and the family at home who supported my decision to explore life across the pond.

Zahra; thank you. I couldn't have done it without you.

Chapter 1

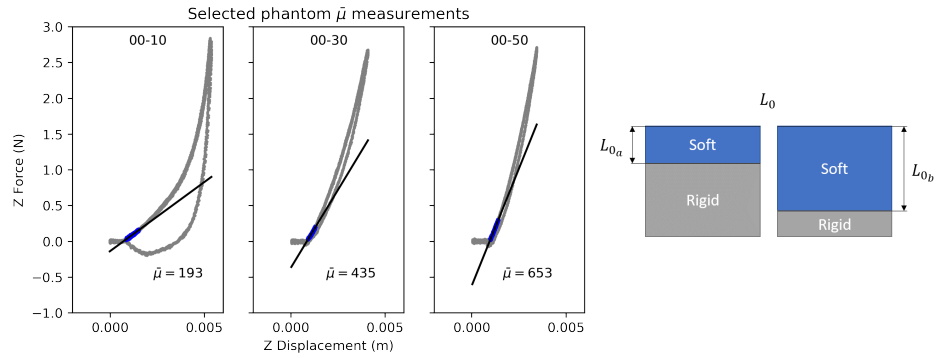
Introduction

1.1 Goals

With SkinProbe 2.0, we aim to simplify and accelerate the collection of large volumes of data on the physical properties of human soft tissues. We hope that this data will reveal scientifically and practically useful information on the distribution of these properties over wide populations, enhancing the modeling and simulation of both humans in general, and of individuals in particular. We break down these lofty goals into practical targets:

- Low cost of production and usage
- Portability – ability to collect data *"in the field"*
- Usability – little training or expertise required of experimenters
- Rapid data collection
- Accuracy – measure physically meaningful properties of soft tissues

With these goals in mind, we opted for a significantly different approach to the original V1 probe (Section 1.4). We base our solution around a consumer smartphone and cloud computing technology, with only the calibration of each device requiring specialized, immobile hardware. This minimalist approach is enabled by our use of a novel force-sensing "flexure," machine learning, and off-the-shelf



(a) $\tilde{\mu}$ Stiffness **(b)** L_0 Thickness

Figure 1.1: Visual descriptions of our material parameterization

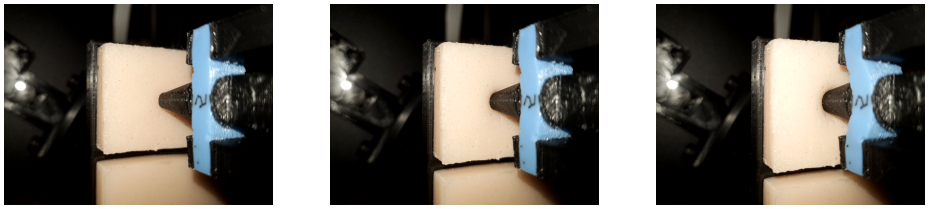
optical flow estimation, allowing us to infer material properties using *only* optical data. Usage in the field is supported with only the probe and a working internet connection. Our use of a consumer smartphone and consumer-grade 3D printing means the incremental cost of constructing a new probe is low. Each probe is operated through the embedded phone's touchscreen, so we achieve usability with interface elements which are easily recognizable to the millions of people who use smartphones every day. We achieve rapid data collection by requiring no external hardware, and by providing a cloud pipeline for processing new data, which provides rapid feedback to experimenters. The *accuracy* of our system is not currently production-ready, but we present our preliminary results as a proof of concept justifying further refinement of our techniques.

1.2 Overview

Here we provide a brief overview of the project's components and methods.

1.2.1 Material Parameterization

Estimating the properties of soft materials requires *defining* those properties, which we do here. We use a simple model which attempts to capture the *stiffness* and the *thickness* of the sample under consideration. We limit the model to two variables out of a desire to limit the necessary volume of data – specifically, the number of



(a) No contact **(b)** 1 N normal force **(c)** 3 N normal force

Figure 1.2: Raw video frames as captured by the probe's camera.

different material samples – necessary for training our material model.

We treat each sample as a depth-wise homogeneous block, backed by a rigid underlying surface, and define $\bar{\mu}$ as the slope of the force/displacement curve at the point of contact with the material – that is, for small displacements (Figure 1.1a). We define L_0 as the thickness of the material, in millimeters, to the (real or imagined) rigid underlying surface (Figure 1.1b).

Our "ground truth" material properties are measured with the same equipment used to generate training data for our system, with only minor modifications.

1.2.2 Data Capture

The smartphone in the probe serves three main purposes: it displays controls for registering a participant, and for starting and stopping recordings; it captures optical data (videos); and it communicates with the cloud pipeline – uploading data, and downloading results. Though modern smartphones are powerful computers in their own right, and are now capable of performing deep neural network (DNN) inference at a reasonable pace[1], doing so presents issues with regards to battery life and overheating in a portable application. We offload all significant processing "to the cloud," leaving to the phone's lesser computational powers the (still challenging) task of capturing, compressing and uploading high quality video.

The data we are interested in collecting are short videos of the probing shaft being pressed into the participant – we refer to a single capture, and resulting data from of one of these videos as a "touch." An experimenter starts recording, moves

[1] TensorFlow Lite (https://www.tensorflow.org/lite) permits deployment of fairly advanced image classification and object detection models on mobile devices.

the probe to touch the stationary participant, and applies a small amount of force (less than 4N) through the shaft. She then retracts the probe, and stops the recording, with the whole process having taken 3-6 seconds. See Figure 1.2 for several representative frames of captured video, but note that it shows contact with a silicone phantom, not a human subject.

The scope of experiments which could be carried out with the probe is broad, but in general, we foresee the device being used in multiple locations on each participant. Part of the utility of rapid measurements is to enable *many* measurements, generating a map of material values covering a significant region of the participant's body. In the future, we envision the probe's application software guiding the experimenter through captures for different locations on the body.

1.2.3 Estimation

The desired final output of our system, as applied to a particular recording, is simply a pair of material parameter estimates. As such, we want a material model "matEst" which accepts a frame sequence $\boldsymbol{x}$, and outputs a material estimate vector $\hat{\boldsymbol{m}}$ with our selected material properties:

$$\hat{\boldsymbol{m}} = \begin{bmatrix} \bar{\mu} \\ L_0 \end{bmatrix} = \text{matEst}(\boldsymbol{x})$$

Rather than attempting to tackle this problem as a monolithic challenge, our estimation pipeline breaks down the task into two major sub-problems: force estimation for frame selection, and material estimation using the selected frames. We sketch this overall pipeline in Figure 1.3. We attempt to solve these sub-problems with two distinct convolutional neural network (CNN) regressors, which we train from scratch. These are related to, but distinct from popular CNN discrete classification models – this class of models is famous for its excellent performance on object recognition tasks, where a particular image is categorized into one of several different classes. Our regression models instead output numeric estimate vectors, which are continuously-varying across the input space.

The video captured for each touch includes a view of both the subject's skin, and of the *flexure*, which includes the probing shaft. The flexure is a novel mechan-

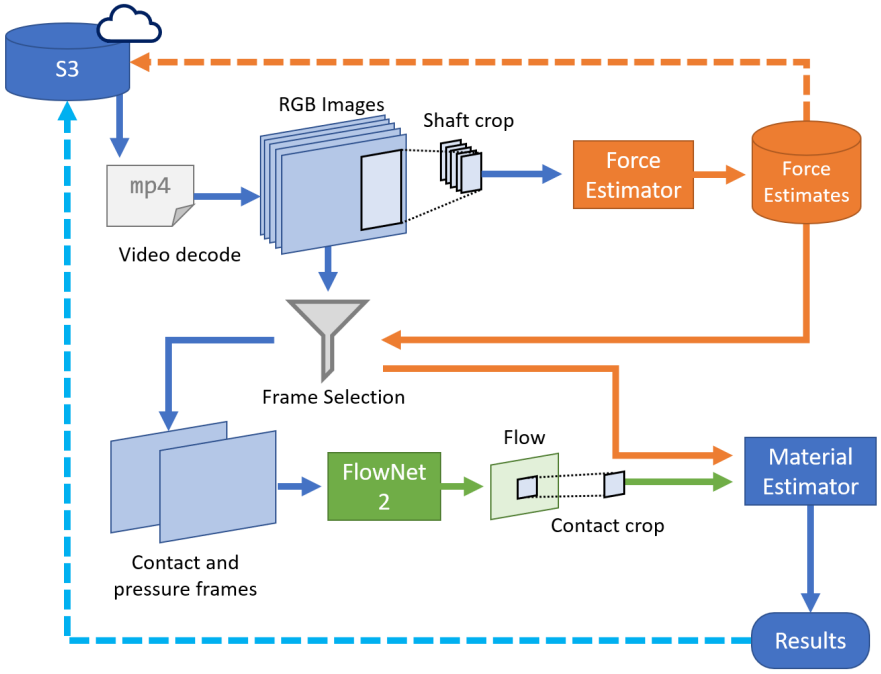


Figure 1.3: Our estimation architecture – raw video is fed frame-wise into the force estimator, which is used to select frames for optical flow. The generated optical flow is fed into the material estimator, along with estimated forces for the selected frames.

ical device, used to transform applied forces into visible deflection in a repeatable manner. We extract these forces using our "force model" (Figure 1.4), the first of our CNNs, which infers the force by observing this deflection in a cropped portion of the video. We apply our force model in an instantaneous frame-by-frame manner, ignoring the state of the flexure over time, as we found this provided sufficient accuracy while keeping the model relatively simple.

So, for a given frame of video x_i, our model outputs an estimate $\hat{\boldsymbol{f}}_{\boldsymbol{i}}$ of the true force vector $\boldsymbol{f}_{\boldsymbol{i}}$:

$$\hat{\boldsymbol{f}}_{\boldsymbol{i}} = \text{forceEst}(x_i) \approx \boldsymbol{f}_{\boldsymbol{i}}$$

Note that, for this prototype, we only estimate the normal force, so these force

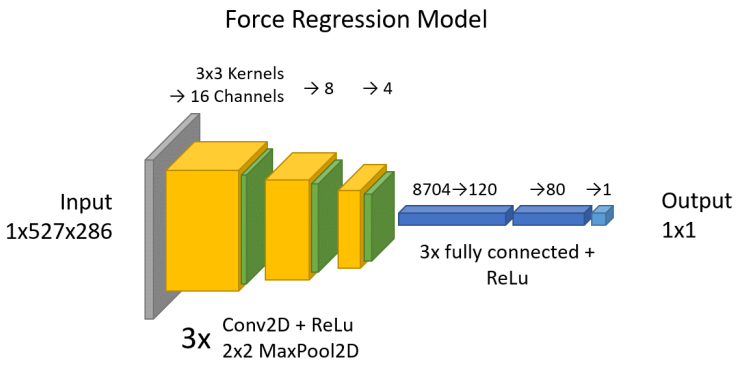


Figure 1.4: Structure of the force estimation model, showing the input, output and hidden layers of the DNN. Here, "channels" refer to the outputs, akin to filtered images, of each distinct CNN kernel in each layer. We use $x \rightarrow y$ as a shorthand for a layer with an input size of x channels/neurons, and and output size of y channels/neurons. Where x is omitted, it is the size of the last layer's output.

estimates are in fact scalar. However, the approach described here could be used to generate and process 3D forces.

Next, we compute the input for the material estimator, another CNN regression model. Given the estimated input forces $\hat{\boldsymbol{f}}_{\boldsymbol{i}}$, we select two frames of the video – a "contact" frame x_c, and a "pressure" frame x_p – with which to compute optical flows. These frames are selected as the estimated force magnitude crosses two separate thresholds: contact, at 0.15 N; and pressure at 2 N. We chose these values manually, aiming to reduce the effect of any noisy output from our force estimator, and to ensure significant visible deflection between the two frames.

Optical flow is a measurement of the apparent motion of objects, surfaces, or other features in a sequence of images, traditionally in pairs of sequential frames of video. We use *dense* optical flow between the selected frames, where an estimate of motion in image space is generated at every point (conceptually, at every pixel) in the image. For a pair of images, the output of this generation is a "flow," a dense field of 2D vectors indicating the motion in pixel units of the feature at that pixel's location, from the first to the second image in the pair.

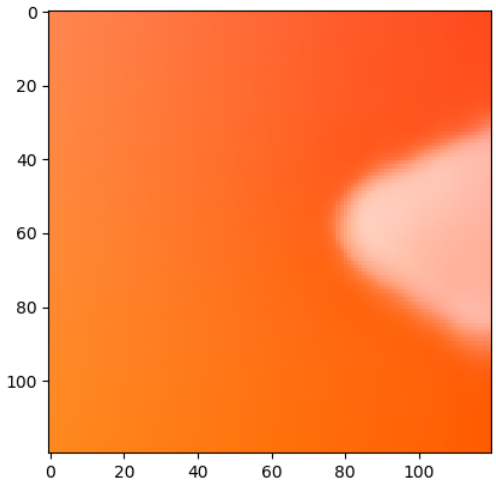


Figure 1.5: Color-mapped rendering of a 120×120 flow field input to the material estimator, with the tip of the flexure shaft – which has little relative motion to the camera – visible on the right of the image.

We compute our optical flow estimates (Figure 1.5) using an off-the-shelf solution – an open-source implementation of FlowNet2.0 [14], written in Tensorflow. We input the selected frames x_c and x_p for a touch into this estimator, obtaining a flow estimate $\hat{O}$:

$$\hat{O} = \text{flowNet2}(x_c, x_p)$$

The material estimator itself has broadly similar structure to the force estimator, though it is somewhat larger. It is a CNN regressor comprising three convolutional layers, and five fully-connected layers (Figure 1.6). The optical flow image $\hat{O}$ is cropped to a region near the point of contact with the flexure, and this cropped flow is input to our material estimator model, which outputs our final vector of estimated material properties:

$$\hat{\boldsymbol{m}} = \begin{bmatrix} \bar{\mu} \\ L_0 \end{bmatrix} = \text{matModel}(\hat{O})$$

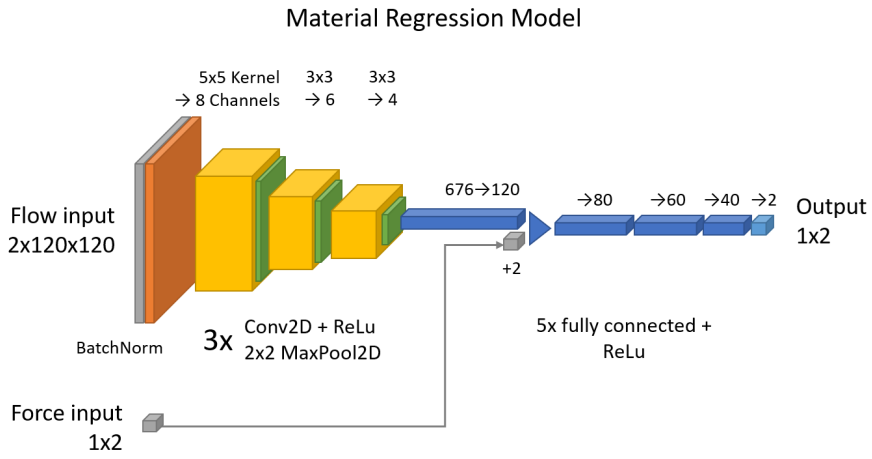


Figure 1.6: Structure of the material estimation model, showing the input, output and hidden layers of the DNN. The initial flow input is 2-channel as it is a field of 2D vectors (flows), while the two force inputs are the estimates $\hat{\boldsymbol{f}}_{\boldsymbol{c}}$ and $\hat{\boldsymbol{f}}_{\boldsymbol{p}}$ for the frames used to generate the flows.

1.2.4 Model Training

As our models for force and material estimation are DNNs, they must be trained to do anything useful. Since we have specific ground-truth data, our models are trained in a supervised manner. This means providing example inputs and *the corresponding expected outputs* – the difference between the calculated and expected results provides a loss, or error, which is minimized to train each model through backpropagation. This process typically requires large volumes of training data, enough to cover the expected distribution of real-world (test) data and outputs. Depending on the complexity of the model, this can mean thousands to millions of examples.

We provide large amounts of training data for our models with our robotic "automation rig," described in detail in Section 2.3. It holds the probe, and generates artificial touches against silicone tissue phantoms. We automate this data collection process with custom sequential control software for our robot, additionally controlling the probe's smartphone application over WiFi, to indicate the start and end of recordings. The rig integrates an industrial force/torque sensor, providing

reliable force readings, and the robot provides accurate position readings, which we also save with the training data.

We designed the tissue phantoms to provide a variety of stiffness and thickness results. We achieve this by varying the type of silicone resins used in the different phantoms, and by varying the depth of the material within each phantom using specially-designed trays.

We obtain the material training data in two ways: $\bar{\mu}$ by probing the tissue phantoms and analyzing the resulting force/displacement curves; and L_0 by simply analyzing the depth of each material sample across its surface, given the computer-aided design (CAD) model of the silicone tray.

1.2.5 Processing Pipeline

With the smartphone's storage limited to a few gigabytes, and processing taking place remotely, we opt to immediately offload all recorded videos to cloud storage. The phone application records to local `mp4` video files, which are then uploaded to a cloud storage bucket, where they are sorted according to the experiment's session ID and current recording number. The application then deletes the local temporary file.

During a live experiment, the material estimation processing takes place on a cloud computer equipped with a powerful graphics processing unit (GPU), which helps accelerate the large tensor operations involved in DNN inference. We also use this cloud machine for all optical flow processing, and the majority of the force estimation processing necessary for training and testing our force and material models. This provides some of the assurances of continuous integration (CI) – ensuring our development models function in the "production" cloud environment – and allows us to run the full FlowNet2 model, which has significant memory and compute requirements, and which requires special environmental configuration which proved difficult on our local workstations.

Our instance runs three custom services, and a Redis database application, which together make up our cloud pipeline:

- Redis database: an off-the-shelf, in-memory database providing fast accesses and asynchronous primitives for inter-process communication (IPC).

- Flask server: the instance's outward-facing interface. Provides HyperText Transfer Protocol (HTTP) endpoints for starting, monitoring, and retrieving the results of processing jobs.
- PyTorch service: loads and executes our force and material models on data which it pulls from our cloud storage.
- Flow service: loads the FlowNet2 model, and uses it to generate optical flow estimates on demand.

These services communicate on the instance to carry out the processing steps outlined across the previous pages of this section. This processing is initiated by the upload of a new video recording from the probe's smartphone application, and returns results back to the application upon completion, where they are displayed to the experimenter. All results are additionally stored securely in the cloud, in the popular (and widely-readable) JavaScript Object Notation (JSON) format.

We discuss this cloud pipeline in greater detail in Section 2.5.2.

1.3 Related Work

1.3.1 Regression Models

While the problems we attempt to solve with deep regression models are highly specific to our project, DNNs have been used extensively to solve a wide variety of computer vision problems in the last several years. Much of the focus of this work has been on supervised categorical learning – classifying one or more objects in an image [17], or semantically segmenting an image to its component parts [24] – but this is not the only type of vision task where deep learning has proven useful. The natural output of a neural network is a numeric vector, and while this can be interpreted (and optimized) as a likelihood distribution over categories, it can also be used for regression for many different target variables [1]. Regression from images may sound like an odd task, but is in fact very useful in a variety of domains. For example, human pose estimation from red-green-blue (RGB) images is a regression task, producing a vector describing a skeleton configuration. Outputs may come in the form of 2D joint locations [45], a heatmap of joint locations [48],

or, more recently, as a map of 3D limb orientations [25]. Depth estimation from images is also fundamentally a regression task, and has seen recent strides [7][23]. A common example of image-based regression, with some similarity to our force estimation task, is bounding-box estimation, where objects of interest are located in image space. Our force estimator works, at some level, in a similar fashion, by locating the flexure shaft in the frame. Examples include architectures with convolutions followed by fully-connected layers (as in our models) [12], and networks which regress an object mask output [43]. Many of these regression approaches propose advanced techniques which we do not make use of, such as multi-pass refinement, but our DNN architectures draw from these examples and more.

1.3.2 Force Measurement

The measurement of forces is an old and extremely widely-studied problem, which we could not hope to cover here. While our apparatus is novel, the basic idea of *optically* deriving a force estimate from the deflection of elastic material is not. Optical force measurement systems may employ mirrors and lasers to amplify apparent motion for detection – this technique is known as *optical beam deflection* [49][37][35], and produces highly accurate measurements from relatively small deflections. The features used for estimation in these techniques are generally more simple than images – generally, the position or brightness of spot(s) of light are transformed by a simple mathematical model to the final output. An advanced study of similar techniques can be seen in [13], where Hirose and Yoneda developed a packaged 6-axis optical force sensor, and discuss several methods of transforming light based on deflection.

Deflection-based force measurement has also been utilized without optical sensing components. Zhou *et al* [10] use a carbon-infused silicone rubber to create a wearable, capacitive tactile sensor, whose electrical capacitance changes as the material is deformed. Magnets placed in soft materials can be localized using hall-effect or magnetic field sensors to infer the forces applied to those materials [5][21][47], relating deflection to force output in similar manner to the optical methods discussed above. These magnetic and capacitive measurements suffer from environmental inaccuracies: for example, the human body affects the capac-

itance of objects it is near to or in contact with, and electronic devices generate electromagnetic (EM) interference which can affect the analog measurement of a magnet's position. In the case of powerful magnetic fields, such as those in a magnetic resonance imaging (MRI) machine, the distortion may be severe, and magnetic force sensors may become inoperable.

Our idea of monitoring *macro-scale* deflection for force detection is inspired in part by visual servoing, a robotic control technique where a robot's pose is monitored by camera in a closed-loop control system. This type of servoing has indeed been used for force control, as in Nelson *et al*'s work [29].

Many other forms of force sensing are common, but typically require specialized electronic hardware. This includes accurate, low-deflection strain gauges used in industrial force/torque sensors. We use this type of sensor for training our force model, and the original V1 probe also relied on this technology. These sensors require additional hardware for reading and interpreting analog electrical signals, are often tied to laptops or desktops, and have a very high unit cost (often $1,000 or more).

1.3.3 Physical Analyses and Simulation of Human Tissues and Other Materials

The human body is an exceptionally well-studied object, and this study has naturally extended to measurement and modeling of the dynamic response of human soft tissues. Our work ultimately aims to aid this analysis at a large scale, and so we pay homage to the enormous number of projects which have come before.

Much work has gone into *visually plausible* simulation of soft tissues, primarily for artistic applications in movies and, more recently, video games [9]. The behavior of soft, deformable materials is well-approximated by finite element methods (FEMs) (refer to [40] for a thorough overview), and recent projects have pushed towards highly detailed, anatomically-based human body models replete with muscles, tendons and fat. Fan *et al* used MRI data to model distinct muscles with targeted active and passive shapes [8]; Sueda *et al* matched the movement of muscles, tendons and bones in the hand to artistic animations [41]; and Si *et al* modeled a complete human musculoskeletal structure, conforming its motion to natural swimming gaits [39]. Additional examples of this approach may be found

in Lee *et al* 2009 [22] and Teran *et al* 2005 [44]. While structurally advanced, these approaches lack physical realism – they produce *visually* convincing results in many cases, but cannot meaningfully predict the *force* response of human tissues under direct contact, as they do not use real-world measurements of the behavior of these tissues.

Parameter fitting for these types of models has also seen significant work. Sifakis *et al* used dense motion capture markers, and a volumetric model, to estimate muscle and tissue properties within a male subject's face based on its motion. Similarly, Pons-Moll *et al* [36] used 4D, full-body motion capture data to train a statistical model predicting the future shape of a human body mesh, using its current physical state. This model produced realistic jiggles and bulges, but failed to generalize to the effects of external forces. Kim *et al* [15] extended this work with an FEM-simulated tissue layer over a statistically-modeled inner body, learning the physical material parameters from motion capture data, and allowing generalization to external forces. This work is appealing in its elegance, but it does not include data on compression behavior, and so cannot be relied on to reproduce this behavior accurately. It also optimizes a linear material model, which is unable to accurately simulate tissue compression. Wang *et al* [46] supply another example of parameter estimation in a linear model, using Kinect depth sensors to analyze the motion of complex (but inanimate) objects under external forces.

When attempting to capture compression behavior of soft tissues, contact measurement provides directly relevant data. In a work which heavily inspired this project, Pai *et al* [33] used a robotic probing system to capture the shape and physical behavior of objects such as plush toys. Kry and Pai [19] used motion *and* force capture to estimate the compliance behavior of joints (rather than soft tissues specifically) in the human hand – in this case, the contact was between the hand, and an object being grasped. Bickel *et al* [2] improved upon these techniques with a nonlinear model for heterogeneous materials. This approach was somewhat limited by their use of a basic, resistive force sensor in their probe. Miguel *et al* [28] estimated properties of cloth, skin, and internal anatomy with an advanced hyperelastic model. Finally, in our own lab's work, Pai *et al* [34] used direct capture of force-deflection behavior to parameterize a nonlinear FEM "sliding thick skin" layer over a rigid inner body. We captured this data using the SkinProbe V1, which

is discussed further in Section 1.4.

More generally, the haptic behaviors of complex structures – not limited to the human body – have been captured and modeled in several different ways. MacLean created the "Haptic Camera" [26] for automated, robotic capture of haptic environments, measuring and modeling a toggle switch's behavior. While actuated in only one dimension, this system was then able to convincingly "replay" the switch's behavior to a user. Pai and Rizun developed the "WHaT" [32] wireless haptic texture sensor, a handheld device incorporating a single-axis piezoelectric force sensor and multiaxis accelerometer, which allowed a user to measure surface properties like roughness. The handheld, wireless nature of the device made it easy to use on complex, three-dimensional objects – we hope our handheld, wireless probe will achieve much the same.

1.3.4 Tissue Phantoms

Beyond simply studying the body, significant research effort has gone into *recreating* the dynamic behaviors of human tissues in artificial analogs, referred to in the literature as *phantoms*. These are often designed for surgical training and practice; SynDaver® products have been used for surgical simulation [30], and range from single-material tissue and skin phantoms to complex, multi-layered phantoms with simulated fat, muscle, veins and skin. These commercial products are made with silicone and other gel-like materials, and aim to emulate the look and feel of real human tissues under surgical manipulations – palpation, cutting, stitching etc.

Other phantoms have been designed for calibrating and testing medical devices, such as ultrasounds [4]. Phantoms may also serve double-duty as training aids for these same devices, and need not consist of artificial materials – Sekhar *et al* [38] discuss the use of bovine liver and rib segments, along with pimento olive "tumors," as a phantom for liver biopsy training. We avoid the use of animal flesh and other organic materials in our phantoms for obvious reasons: our automated data capture sessions can last many hours, and we wish to make these sessions repeatable and mess-free.

The production of tissue phantoms has also seen some targeted research. Hall *et al* [11] characterized the force-displacement behavior of a variety of gelatin-

based materials using a motorized capture system. Derler *et al* [6] evaluated the frictional response of various skin phantoms on textiles, comparing the phantom's responses to that of real human skin.

1.4 SkinProbe 1.0

Version 1 of our SkinProbe was, naturally, the immediate predecessor to this project, and our goals and decisions for SkinProbe 2.0 were informed significantly by lessons learned designing and using version 1.

The original SkinProbe generated qualitatively different data to SkinProbe 2.0, providing force-displacement measurements and multi-view optical flows. We[2] locally optimized a *reality-based* model [31] we called "sliding thick skin" [34], with a thick FEM layer which slid smoothly over a parametric inner body.

1.4.1 Design

The SkinProbe[3] (Figure 1.7) was a custom in-house design, tailored for the SkinCap project, and produced with rapid prototyping techniques. We wrote custom software to operate the probe and record data, which ran on an attached "host" PC.

Hardware

The probe housed three miniature high-definition RGB cameras, a six-axis force/-torque sensor, three user input buttons, and a micro-controller board. The probe provided visual feedback to the experimenter during operation through a graphical user interface (GUI) displayed on a separate monitor, which could be controlled directly using the buttons on the probe body. The cameras were placed to provide a clear view of the skin patch around the probing point, with overlapping fields of view that eliminated occlusions around the probe, except at the point of contact.

The design underwent rapid iterative changes throughout the project. An initial prototyping phase with non-functional sample parts resulted in general decisions on the overall shape of the probe, including the pistol-grip design – which optimized grip angle for the probing locations we were considering – and rear place-

[2] In Section 1.4, "we" refers to the authors Pai *et al* [34], including me.

[3] Section 1.4.1 is based on my own work in [34] section 3: Design and Fabrication of Skin Probe.

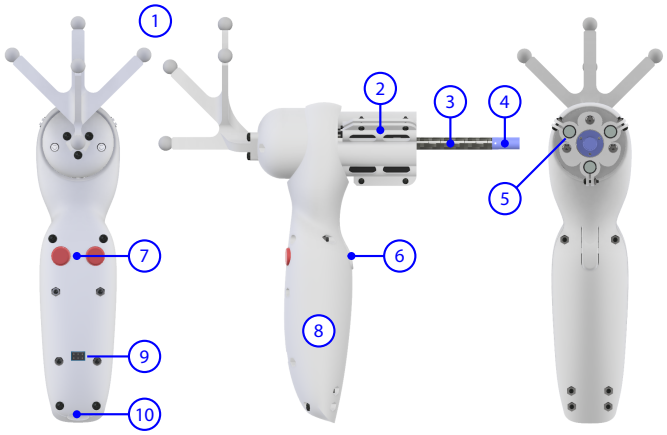


Figure 1.7: SkinProbe V1. 1) 3D tracking marker arrangement on the "tentacle"; 2) Single-piece probe "head," holding cameras and force sensor; 3) Carbon-fiber composite probe stem; 4) Quick-change, magnetic-lock probe tip; 5) Front-facing cameras; 6) Front trigger button; 7) Rear input buttons; 8) Ergonomic handle, housing micro-controller; 9) Externally accessible reprogramming port; 10) Exit point for probe cables, with internal strain relief. *Reproduced from [34] Fig. 2.*

ment of the marker tracking assembly, which kept the assembly out of the way of the probe cameras and experimenter. After this initial prototyping phase, the device saw five major iterations, with a general trend of increasing size and additional features, such as the addition of buttons for experiment control.

We optimized for ergonomics, durability over multiple experiments, and ease of use. This included use by a solo experimenter: the probe's input buttons allowed an individual to perform an initial calibration, then select and create recording "trials" with no other experimenters present. This was desirable as it improved participant privacy and comfort, and left other project members free for data processing tasks. With this in mind, and drawing inspiration from video-game console controllers, we laid out the probe buttons for easy operation with one hand. Two buttons were positioned for operation with the experimenter's thumb, on the probe's back – that is, the side facing the experimenter. The third button was integrated into a "trigger" on the rear of the probe, operated with a forefinger. In addition, we designed the probe's large, organically-shaped handle to be held comfortably with

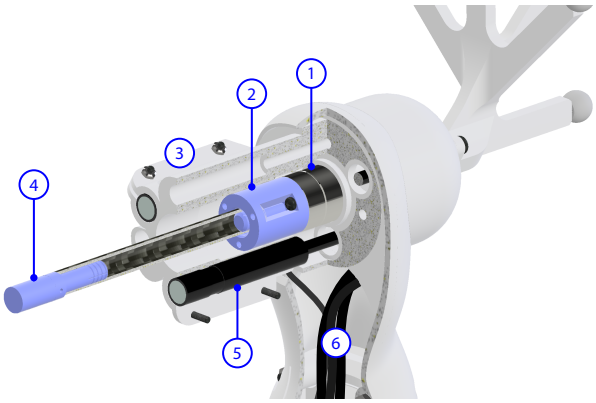


Figure 1.8: Cutaway view of SkinProbe V1's "head" section. 1) Embedded force/torque sensor; 2) 3D printed probe stem mount; 3) Integrated probe camera mounts; 4) Probing tip, also showing the internal channel housing the locking magnets; 5) Endoscope camera (one of three); 6) Camera and force sensor cable routing. *Reproduced from [34] Fig. 3.*

a one- or two-handed grip.

With a sensitive force transducer mounted back from the cameras (Figure 1.8), a relatively long shaft was needed to reach the probing point. This had to be lightweight, to minimize the influence of the shaft mass on the sensor; compact, to avoid occluding the camera's views of the skin patch; and extremely stiff, to ensure a rigid transformation between the probing location and the probe itself, even as forces were applied to the tip. After experimenting with metal shafts and tubes, we switched to a lightweight carbon-fiber shaft with a custom 3D printed mounting and a magnetically-locking, quick-change tip.

In an application where experiment hardware is in direct contact with members of the public, safety is an especially great concern. We kept contact forces for our experiments low – easily achieved with a human experimenter in control – and took care to avoid sharp edges and exposed electronics in the probe's design, minimizing risk of injury to all parties involved.

Software

The capture software for SkinProbe V1 (Figure 1.9) served primarily to monitor data streams from the various sensors and the Vicon motion capture system, and to

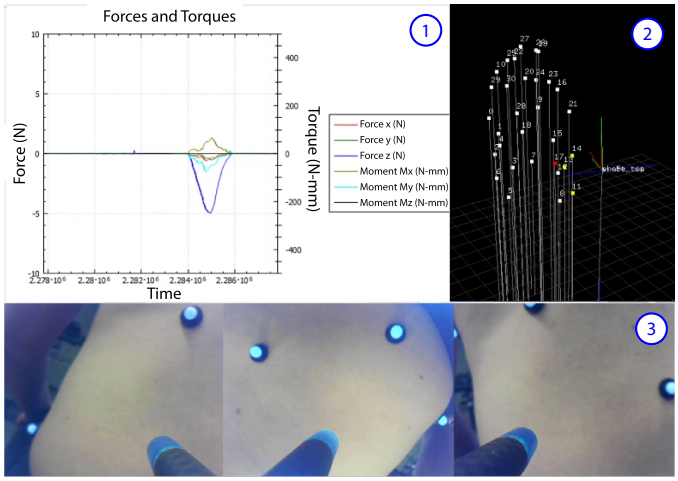


Figure 1.9: SkinProbe V1's capture software. 1) Live feed of forces and torques applied to the probe tip, showing a recent touch; 2) 3D visualizer window, displaying real-time positions of the participant markers and probe tip from a user-adjustable virtual camera; 3) Live split view from the three probe cameras, showing the shaft and tip in-frame, along with the participant's skin and attached markers. *Reproduced from [34] Fig. 4.*

consolidate these streams into recordings sampled at up to 120Hz. It also allowed the experimenter to execute automated optical flow post-processing on the captured video feeds. The software was controlled through a GUI written in C++ using the Qt 5 application library. An additional 3D view of the capture volume was provided through our visualizer application, a separate program written in Python and using OpenGL, which we used to display both live and pre-recorded data from the main application.

Another important role of the software was to calibrate the SkinProbe before experiments, and to store and re-use these calibrations as far as practicable. This included establishing the relative transformations between the probe's tracking "tentacle" and the tip, and between the tentacle and cameras – we considered these transformations to be relatively stable between sessions. The software also included the ability to *tare* the force sensor, and to establish the weight of the shaft and attached tip. This allowed us to accurately zero our force readings, leaving

only the externally-applied forces on the probe tip in our recorded data.

We used Vicon Blade software to monitor the Vicon system. Blade captured the 3D position of any detected markers in the capture volume, and provided robust tracking of the transformations of rigid "props" like the probe tentacle. Blade relayed this data via Vicon's proprietary streaming application programming interface (API) to our main capture application, where we implemented our own predictive marker tracking solution for the less-than-rigid bodies of our experiment participants.

1.4.2 Limitations

The most significant limitation of the V1 probe is the total system cost and complexity. An accurate, real-time, 3D motion capture system costs tens to hundreds of thousands of dollars. Full-body 3D scanners can cost hundreds of thousands of dollars. Accurate multi-axis force sensors range from thousands, to tens of thousands of dollars. Along with a powerful host PC, and thousands of dollars in required commercial software, these hardware components make for a system which is prohibitively expensive to create, use, and service.

These cost and hardware restrictions also make scaling the system infeasible. Only one participant may be run at a time, as the motion capture system constitutes an open area which cannot be shared for privacy reasons. The motion capture volume and body scanner are also immobile, requiring major effort to move, and extensive set-up at any new location. This means that participants must be able to travel to the laboratory to be tested, severely limiting both the number of available participants, and the geographic region from which they may be pooled. Duplicating the system in new locations would repeat the massive cost of a completely new set-up, and would require significant work from technicians to replicate the software environment and hardware connectivity of the original system.

Less serious limitations include convenience and duration issues. The system requires each participant to partially strip, and to wear a set of markers which must be carefully and professionally placed (this requires substantial practice). The system also requires a full body scan, with the markers. Errors in motion tracking, which we are fortunately able to detect, require the experiment to be paused and the

participant to stand for re-identification of the tracking markers. These additional steps beyond the core recorded touches introduce friction and use up time in the experimental process, frequently leading to session times of 90 minutes or more. This further limits the available pool of participants: in short, while individual measurements are rapid, the sessions are neither quick nor easy.

Finally, we found that the optical data gathered from the probe's trinocular vision system was of questionable quality. The high-resolution images tended to be grainy and out of focus. We also had little control over values such as exposure and frame-rate, which was typically 15 frames per second (FPS), and we struggled with the camera's lack of metadata output – for example, no accurate frame capture times were provided. The camera sensors were not well-seated inside their housings, tending to point slightly off-axis, and the cameras themselves were also notoriously unreliable, requiring frequent replacement. With this poor data, we found that our optimization process struggled to match virtual flows on simulated material to the real-world flow estimates our system generated, and generally resorted to using the much higher-quality force/displacement data generated with the probe's force sensor and the motion capture system.

Chapter 2

Methods

2.1 Probe Hardware

The probe itself is, of course, central to the project, providing the core data collection and experiment control capabilities of our system. Experimenters operate the probe through its touchscreen, and take measurements by gently pressing the probing tip into a participant or material sample. Results, including estimates of the forces exerted through the flexure, are generated in the cloud before being returned to the device, and displayed on the screen for experimenters to check.

Modern smartphones offer a wide variety of sensors and capabilities, including high-resolution multi-touch screens, high-speed internet over WiFi and 4G, very high-fidelity imaging through two or more cameras, accurate orientation sensing, GPS location, and, in newer devices, inside-out SLAM (typically used for augmented reality (AR) applications). They are perhaps the densest, best-value sensor suites available today, and are widely used and understood. We opted to use one in the probe to make the most of these capabilities. The probe is built around a consumer Android smartphone – we use a Samsung Galaxy S8[1], but only require a modern Android device with a back-facing camera, running our capture application (Section 2.5.1). Where the original SkinProbe (Section 1.4) used push-buttons, an electronic force/torque sensor, host PC, and external motion capture system, this new device contains and requires no electronics other than the smartphone – the

[1] https://www.samsung.com/global/galaxy/galaxy-s8/

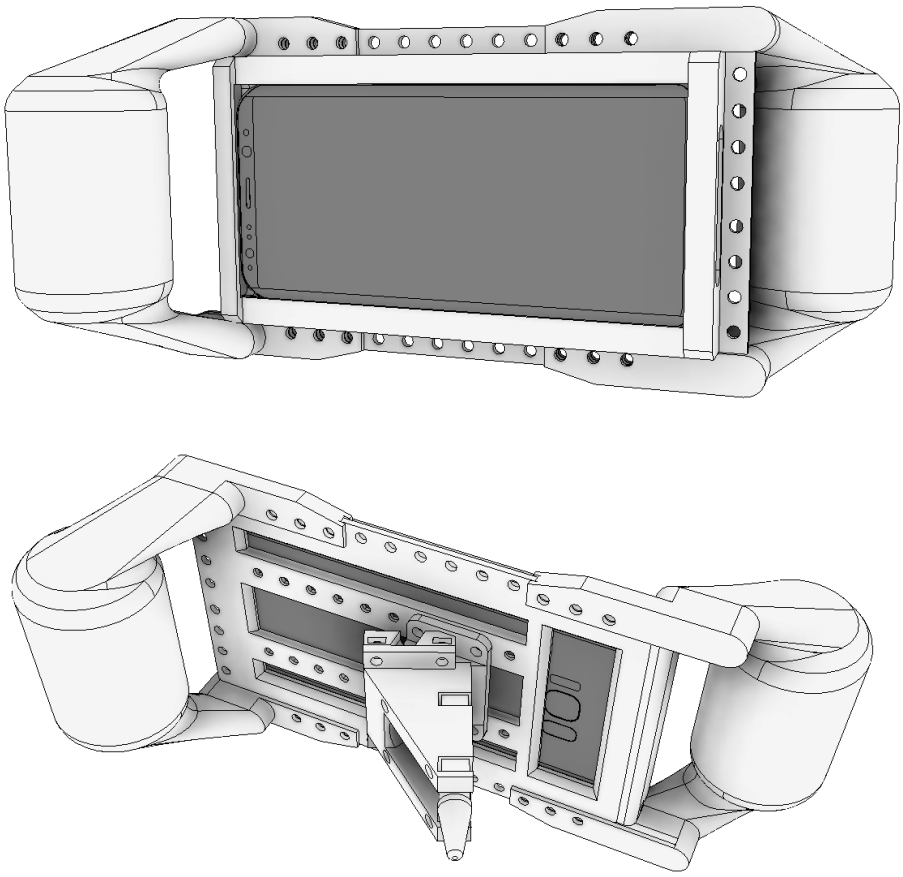


Figure 2.1: CAD model of the SkinProbe 2.0 device.

device is operated through its touchscreen, and relies solely on the back-facing camera for data capture.

A 3D-printed frame securely holds the phone, and provides mounting points for attaching the flexure and ergonomic handgrips (Figure 2.1). The frame also serves to allow mounting the device in the robotic automation rig, which we use for automated data collection (see Section 2.3). The probe's force-sensing flexure is fitted to the back of the device, attaching to the frame (Figure 2.2). The orientation of the flexure is adjusted relative to absolute normal by mounting it indirectly, on slanted angle adaptor pieces. We position the flexure to make it substantially

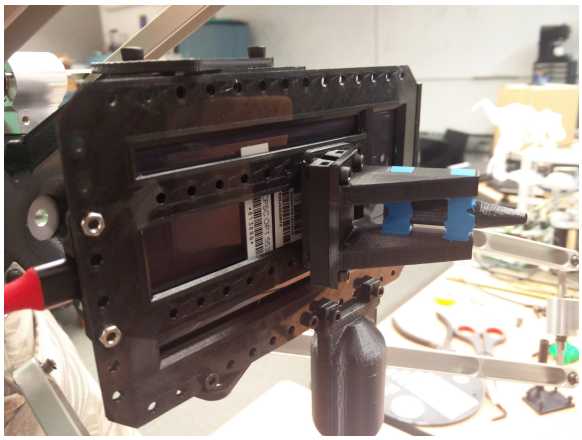


Figure 2.2: SkinProbe 2.0 rear, showing the frame, flexure mounting, and hardware attachment points.

visible to the phone's back camera, while not completely taking over its field of view. The various mounting points along the frame allow for adjustment of the flexure position, with further fine-tuning achieved by varying the flexure and angle adaptor designs.

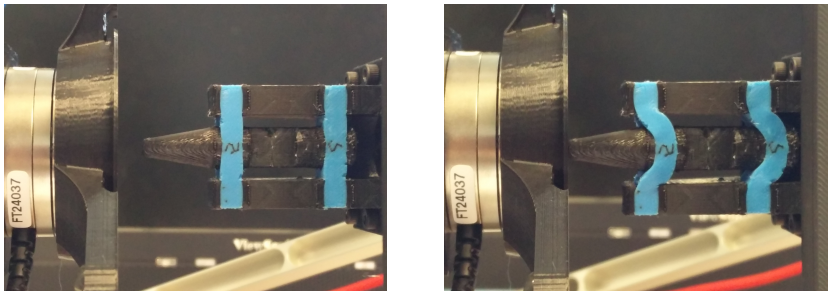
2.1.1 Flexure

The probe's flexure serves both to provide contact with the participant's tissue, and to translate the applied force into visible motion, detectable by the probe's main camera. We fabricated the flexure with a combination of 3D printing and silicone resin casting: the rigid, printed pieces serve as the probing shaft and frame; the silicone cross-bars connect the rigid pieces, and provide repeatable deflection behavior under force (Figure 2.3). We use some novel techniques to combine the silicone and printed plastic, made possible by our use of an Ultimaker 3[2] dual-extrusion printer (Ultimaker, Utrecht Netherlands).

The Ecoflex[3] 00-50 silicone rubber (Smooth-On Inc., Macungie PA) we selected for the flexure was chosen for its near-perfect elasticity and remarkable durability. We observed no plastic deformation on the silicone parts of the flexure

[2]https://ultimaker.com/3d-printers/ultimaker-3

[3]https://www.smooth-on.com/product-line/ecoflex/



(a) No load **(b)** 4N normal load

Figure 2.3: Side-on view of probe flexure under no load, and under 4N load.

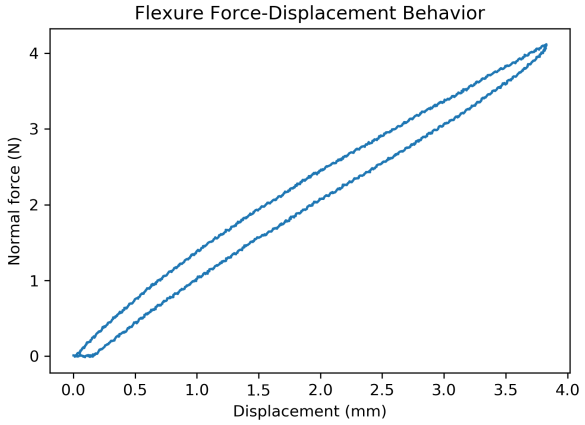


Figure 2.4: Probe flexure force-displacement behavior in the 0-4N range, with a 3mm/s contact speed. The lower line – where the plot loops back – shows some hysteresis, where the material takes some time to recover after being strained.

after tens of thousands of touches on our automation rig, with forces ranging up to 3 Newtons; however, we did observe plastic deformation after exposure to higher force levels (this is discussed further in Section 3.1). Additionally, we found that using this silicone yielded a flexure force-deflection response with a near-linear trend, and low hysteresis (Figure 2.4).

Single-Piece Fabrication

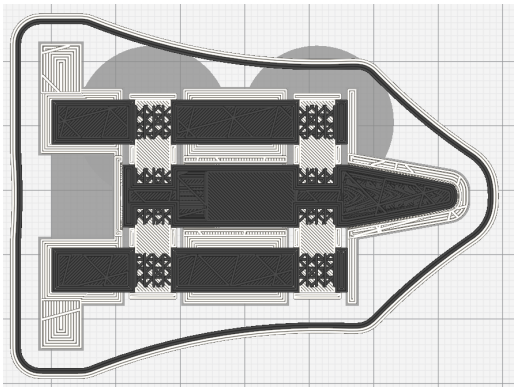
We developed a process whereby flexures are fabricated with a single 3D print, using integrated single-use mold pieces which are broken away after silicone casting. These mold pieces are made with Ultimaker Breakaway Material[4], which forms a mechanically breakable, but watertight bond with polylactide (PLA) plastic. We also use this material as removable support for the flexure's complex geometries, which is its original purpose. The silicone resin is poured directly into the printed part, through removable "risers" which allow resin ingress and air egress. Once the silicone has set, we remove the breakaway material. This conveniently leaves silicone areas with dimensions defined in our 3D design. We do not rely, for example, on accurately pouring the resin up to a level. It also removes the need to cut away leaked resin, which we found was a problem with rigid 3D-printed molds, where small gaps allowed the resin to leak from the part. Finally, this method simplifies the process of designing the molds, as it only requires defining a thin shell of breakaway material around the silicone volumes, rather than separate, solid molds which can be attached and removed intact.

Silicone does not adhere to 3D printed plastic parts when cast in contact with them, and in fact will not adhere at all to most materials. This is an attribute which makes it ideal for mold-making, but difficult to work with for our purposes. It also cannot be glued with standard compounds. We opt to mechanically join the silicone with our prints by integrating rigid 3D lattices, which the silicone resin flows into and around before curing, forming a multi-layered mechanical joint. We generate these lattices using Ultimaker's Cura 4.2[5] 3D printing slicer software, by setting the lattice volumes to print with no solid walls, and with an "infill" of parallel lines at alternating angles per-layer. These lattices are shown in the slicer in Figure 2.5a, and during a print in Figure 2.5b. They are completely obscured by the silicone post-curing (Figure 2.5d). Note that neither the dual-material breakaway mold design, nor the complex 3D bonding structures would be possible with traditional manufacturing techniques.

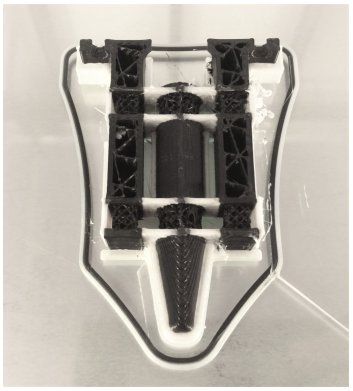
We calibrated the design of the flexure through rapid prototyping and iteration, starting with an initial concept where we used a simple strip of flexible 3D-printed

[4] https://ultimaker.com/materials/breakaway

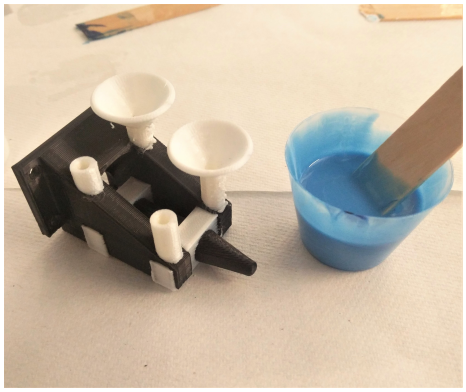
[5] https://ultimaker.com/software/ultimaker-cura



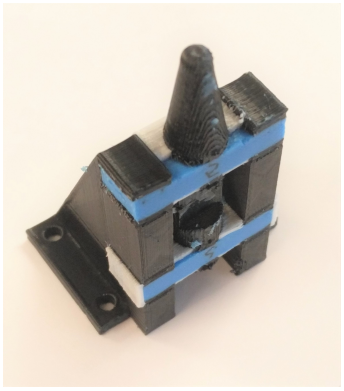
(a) Slicer cutaway



(b) During printing



(c) Silicone casting



(d) Post-processing

Figure 2.5: a) A cut-away view of the flexure in Ultimaker Cura 3D printing slicer software. Note the visible lattice areas in black plastic, and the white breakaway material. b) The flexure during printing, again with lattice structures and white breakaway material visible. c) Silicone ready for casting into risers. d) The breakaway mold material is removed after casting and curing, revealing the set silicone cross-bar (blue material).

plastic. Our final design provides some accommodation in all three axes of motion, and though we do not currently take advantage of this capability, we believe this will allow us to extract 3D force estimates in the future.

2.2 Estimators

Recall that two bespoke regression models are used in our system: the force estimator, which provides per-frame force estimates in Newtons, and the material estimator, which provides the final estimates of the physical material properties. The overall flow of data from raw video to material estimates is depicted in Figure 1.3.

We split the estimation pipeline in this way for several reasons. Force estimation allows us to impose constraints on the data. We can, for example, ensure that touches exceed a useful level of force for parameterizing the material. It acts as a kind of domain knowledge injection – we know that force estimation is at some level necessary for the task – and also allows us to perform frame selection for two-frame (rather than video sequence) optical flows, which keeps the individual models as simple feed-forward networks, and reduces the dimensionality of the input data to the material estimator. Given enough training data on handheld touches with human skin, we could expect a recurrent model, like a long short term memory (LSTM) network, to regress material properties *directly* from a video sequence of optical flow fields and corresponding flexure images. This is an appealing solution, but acquiring a sufficiently large sample of this in-distribution training data (and labeling it accurately) would be a huge undertaking. A single two-frame flow "image" between two points in a touch is far more constrained than the flow sequence from the full video of the touch, and our hope is that this allows robotically-collected data to be closer in distribution to the same data collected by an unsteady human hand.

The best-in-class results achieved on image-based tasks by other authors lead us to use CNNs as our regression models, for both force estimation and material property estimation. Both of our models have broadly similar structures and training methods, with minor tweaks for each task. We also apply some limited data augmentation, which we found improved robustness in practice.

Our models stack several convolutional layers, with each layer's output channels fed to the next layer's inputs. We apply max-pooling [3] between the layers, which is a common technique for improving the robustness of vision models, and has the added benefit of reducing the working set size with each layer. We append several fully-connected layers in an architecture resembling LeNet [20], with successively shrinking layer sizes collapsing down to the required output dimensionality – 1×2 for the material estimator, 1×1 for the force estimator. We use rectified linear unit (ReLU) activations on each layer, a standard choice for DNNs, and train with mean squared error (MSE) loss, another common choice for regression models. Our models are trained and run using PyTorch[6], utilizing GPU acceleration.

2.2.1 Force Estimator

The force estimation network generates the intermediate force estimates $\hat{\boldsymbol{f}}_{\boldsymbol{i}}$, which are used for frame selection in the overall pipeline. The estimates for the selected frames are also input to the material estimator.

The force network accepts a cropped, normalized, 286×527 grayscale image, and inputs this directly to its convolutional stack. We use 3×3 kernels and $2x2$ 2D max-pooling, expanding to 16 output channels in the first layer and dropping to 4 in the third and final convolutional layer. The output of the final convolutional layer is flattened to 1×8704, and fed into the network's fully connected layers, which run 3 deep and output a lone 1×1 normalized force estimate, representing the normal force applied to the flexure. We settled on these layer sizes and depths by trial and error, finding that new or larger layers did not improve the results, but did slow down training and evaluation times.

We expect the convolutional layers in the force estimator to act as powerful feature detectors, identifying elements in the images (and on the flexure) such as corners and boundary lines, with the layer outputs consisting of stacked 2D feature maps. We then expect the first fully-connected layer to focus not just on the magnitude, but on the *location* of feature activations in the convolutional output channels, with subsequent layers non-linearly regressing these feature movements

[6] https://pytorch.org/

into the normalized output forces. In essence, we expect the network to learn to track notable features on the flexure across the images, and use the positions of these features to infer the forces applied to the flexure. It is also reasonable to expect the network to identify image features which appear or disappear at certain force levels, such as specular highlights on the flexure shaft or silicone sections, or specific patterns on the silicone surface as it deforms.

2.2.2 Material Estimator

Recall from Section 1.2.3 that, from frames of video x_i, we select contact and pressure frames x_c and x_p, and, with the material model, estimate a material parameter vector $\hat{\boldsymbol{m}}$ using the optical flow between the two frames:

$$\hat{\boldsymbol{m}} = \begin{bmatrix} \bar{\mu} \\ L_0 \end{bmatrix} = \text{matModel}(\text{flowNet2}(x_c, x_p))$$

The selection of these contact and pressure frames runs as follows. Given the estimated input forces $\hat{\boldsymbol{f}}_{\boldsymbol{i}}$, the selection process picks the first of several consecutive frames where the estimated force magnitude has passed a manually chosen threshold value. With a window size of w frames, contact force threshold F_{tc} and n total frames in the touch, the chosen contact frame index c is:

$$c = \min\{i \in 0 \dots n-1 : \| \text{forceEst}(x_{i+j}) \| < F_{tc} \forall j \in 0 \dots w-1\}$$

With corresponding force estimate $\hat{\boldsymbol{f}}_{\boldsymbol{c}} = \text{forceEst}(x_c)$. The subsequent pressure frame index p is selected similarly, with F_{tp}. We set our threshold values at $F_{tc} = 0.15\,\text{N}, F_{tp} = 1.5\,\text{N}$, thus selecting frames x_c and x_p for flow calculation. We calculate an optical flow estimate using FlowNet2 for these frames (from x_c to x_p), and input this flow $\hat{O}$, along with the contact and pressure force estimates $\hat{\boldsymbol{f}}_{\boldsymbol{c}}$ and $\hat{\boldsymbol{f}}_{\boldsymbol{p}}$, to the material estimation process.

We crop the generated flow to a $2 \times 120 \times 120$ region around the contact point, a fairly small area in the overall image space which displays the most obvious surface displacements, and input this to the convolutional stack at the "start" of the material estimator network. We apply a large 5×5 convolutional kernel in the first layer, reasoning that features are fairly sparse in the smooth flow image. The sub-

sequent two layers apply standard 3×3 kernels. All three convolutional layers use 2×2 2D max-pooling, and we use fairly limited channel counts of 8, 6, and 4. The final convolutional output is flattened and passed through the first fully-connected layer (as in the force network), the output of which is then concatenated with the normalized force inputs. With all inputs accounted for, we gradually reduce the layer size through 4 more fully-connected layers to the final 1×2 normalized material parameter output. We normalize to avoid prioritizing the loss of one parameter over another – the scaling of $\bar{\mu}$ estimates in natural units is over $100\times$ greater than L_0. As in the force estimator, we use ReLU activations throughout.

2.2.3 Training

While neural networks with different overall structure and size perform differently, and are suited to different tasks, a newly-initialized network of any design will generally perform no better than random at its intended task. This is because the parameters, typically "weights" of the various connections in the net, must be tuned from a random initialized state to produce the desired behavior. This tuning is referred to as "training," and is analogous to training a person or animal to perform a task.

In the standard supervised learning paradigm[7], training a neural network involves repeatedly applying the network to its intended task using some set of "training data," and calculating a "loss" (essentially, the difference) between the desired results – the "ground truth" – and the network's output. Training aims to minimize this loss, and typically this is achieved through backpropagation. Backpropagation calculates a gradient for each learnable parameter in the network with respect to the loss, allowing an optimizer to be applied which "follows" these gradients to a greater or lesser extent. Some or all parameters are tweaked by a small amount with each update, in the direction indicated by the back-propagated gradients. Most training optimizers provide a user-tunable "learning rate," a so-called hyperparameter which adjusts the step size for tuning. Many optimizers provide additional hyperparameters, which alter their behavior in other ways.

Training is a time- and resource-intense process, with significant computational

[7] Other types of unsupervised training are possible, but are not directly relevant to this project.

requirements for both the forward and backward passes through the network, scaling with the network's size. Each set of updates utilizing all examples in the training set is referred to as an "epoch." A network may need hundreds, thousands, or even millions of epochs of training before its performance is optimal, and even then, it may perform poorly on real-world data for any number of reasons. A network may overfit to its training data, learning to recognize specific training examples or spurious noise rather than the "real," relevant information contained in the data. More general out-of-distribution (OOD) data issues can also occur, where the distribution of the training data does not properly match up with that of real-world test data.

As long as new training data is in-distribution, increasing and diversifying training datasets for neural networks has been observed to monotonically improve their real-world performance [42]. This drove our desire to capture large amounts of training data with reliability and repeatability, leading to the automated capture solution discussed in Section 2.3. We use this system to capture all of our training data for both models, and much of our validation data.

Dataset Processing

Storing, loading and utilizing large datasets for machine learning presents several difficulties. We currently store half a terabyte of training data on Amazon Web Services (AWS) Simple Storage Service (S3), and have encountered limits on disk, system memory, and GPU memory. We bypass these limits with custom dataset caching and loading classes, which cache requested data from S3 to local storage, unpack the data as needed, and support loading this data as PyTorch tensors. These tensors can then be moved into GPU memory for training or inference.

Our dataset loaders support several useful features:

- Checking the local hard drive cache, and only downloading missing or incomplete files.
- Partial dataset loading – request specific recordings, and only those recordings will be downloaded and loaded into memory.
- Lazy loading – on memory constrained systems or very large datasets, data

may be loaded from disc as needed, not kept in system memory.

- Unpacking `mp4` video files to `png` images with `ffmpeg`[8].
- Storing, loading, and displaying Middlebury-format optical flow `flo` files.
- As-needed optical flow and force estimate generation, starting and communicating with the cloud GPU machine as needed to obtain these estimates.

We use the same dataset loaders for local training and for live inference in the cloud, providing don't repeat yourself (DRY) guarantees that data will be loaded identically for training and "production." We also perform some limited data augmentation dynamically in these loaders.

Several other critical parts of the overall pipeline are implemented by the dataset loaders. This includes: the force-based frame selection logic discussed above; normalization and de-normalization of forces and material parameters for inputs, training targets, and outputs; image and flow cropping; data augmentation; and data selection (e.g. checking the minimum force was hit).

Force Training Data

The training data for our force network is captured by the robotic automation rig, discussed in Section 2.3. The data consists of the individual video frames from a series of touches on a rigid plate (Figure 2.6), with randomized target depths in the normal axis, and skew motions in the off-normal axes. Typically these training datasets number a few hundred touches, with tens of thousands of frames (each frame is a training example). This corresponds to just a few minutes of video at 30 FPS, but we found this was more than sufficient. Validation datasets are separately captured, with much the same programming and with the same hardware setup, but fewer touches – generally 30-50 touches, with 1000-2000 frames.

One of the difficulties with training and using DNNs is the unexpected behavior they can display, especially when testing with data slightly outside the training distribution. We found that while the probe's smartphone torch provided extremely consistent lighting in the flexure images, our initial force models were extremely

[8] https://ffmpeg.org/

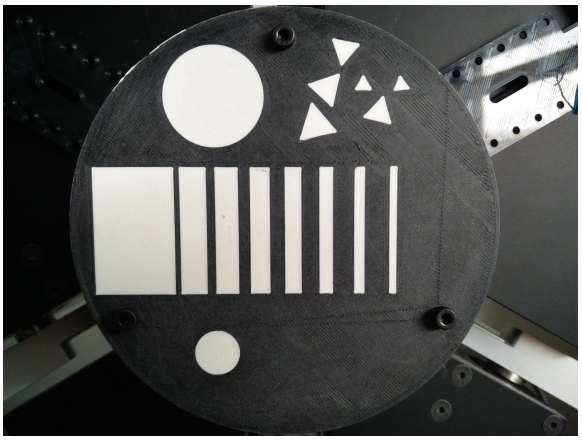


Figure 2.6: Rigid force training plate, with various shaded patterns to create a variety of backscattered lighting effects on the flexure.

sensitive to the effect of *backscattered* lighting changes – that is, changes in the albedo and texture of the material being probed. An especially bright or dark material would reflect light from the torch back on to the flexure, and could significantly bias the resulting force estimates: in fact, when training on the material phantom tray, the force model learned to directly correlate the brightness of reflected light to the output force. Bringing the probe closer to the surface increased the intensity of backscattered light, and the probe's proximity to the surface was roughly proportional to the level of force applied. We countered this unexpected ingenuity on the network's part by adding patterns to the force plate, which create a variety of backscattered lighting conditions when the plate is touched in different areas. We also made the input images grayscale, to prevent any dependence on the material's hue, and augmented the data with additive lightness variations across whole images. The model was then able to predict accurately on materials of different colors and patterns.

Another issue arose when we tested the force network after removing and remounting the flexure. This resulted in a small shift in the flexure's position in the image, moving the feature locations on which the network relied to make its predictions. Testing the force predictions under this scenario resulted in significant mis-predictions, and a massive increase in average loss. Given that the shift

was small, and given that stationary features existed in the images (the flexure frame), we reasoned that the network should be able to learn to recognize the *relative* position of the flexure shaft, and infer the force from that. This would make the estimates invariant to small relocations of the flexure in the image after being jogged out of alignment, or re-mounted. We achieved this with another layer of data augmentation, shifting and rotating the images by small, random amounts before cropping to the shaft area. This augmentation scheme draws heavily from the random-cropping augmentations used to great effect in CNN image recognition tasks.

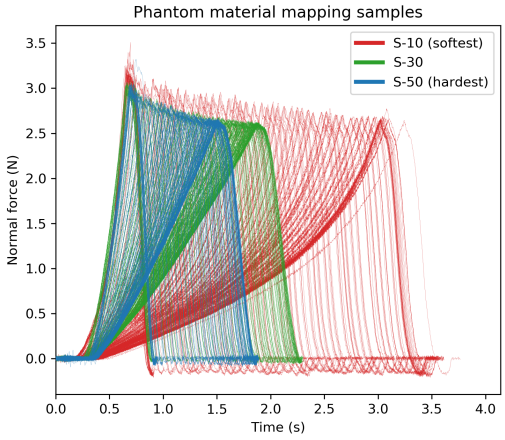
Material Training Data

Our material model training data is captured by the automation rig, in much the same way as the force training data. However, rather than touching the rigid force plate, we touch the phantom material samples – silicone slabs of variable thickness and stiffness, mounted in the phantom tray (see Section 2.3.2). In contrast to the force data, where every frame of video is a training example, each material example consists of a *complete touch*. So, to obtain the thousands of examples necessary for training, we have to collect videos of thousands of touches, making these material datasets comparatively large in terms of raw storage.

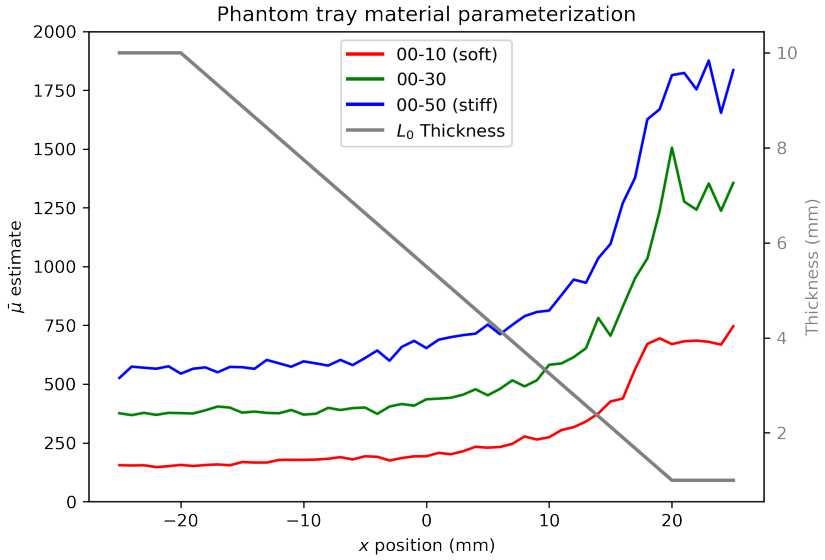
As discussed in Section 2.2.2, the actual inputs to the material network consist of a cropped optical flow between two selected frames from a touch, and the force estimator's output for those two frames. The training targets are also provided by the material dataset loader, and are drawn from material data gathered ahead-of-time on the phantom tray.

We obtain L_0 ground-truth data directly from our CAD models of the tray, but estimating $\bar{\mu}$ requires some additional work. Using a rigid probing shaft (Figure 2.7a), we sampled accurate force-displacement curves at small intervals across each phantom's surface, with repeated touches at each point. For each location, we then estimated the initial slope of this force-displacement curve – the slope *at contact* – by fitting a line to a short post-contact region in the aggregated data. This involved thresholding the force magnitudes to identify points between contact and a low target force (Table 2.1), and cropping the data to this region. We then ag-

(a) Rigid material probing shaft



(b) Samples obtained with the rigid shaft – $\bar{\mu}$ estimates are derived from these



(c) Parameterization of the material phantoms, including $\bar{\mu}$ and L_0

Figure 2.7: Capture apparatus, sampling results, and final parameterizations for material ground-truth data

Point	Force threshold
Contact	0.02 N
00-10 target	0.15 N
00-30 target	0.2 N
00-50 target	0.3 N

Table 2.1: Threshold forces for phantom $\bar{\mu}$ measurement for the different phantom materials

gregated cropped data from multiple repeats at each location, and fitted a line to this multi-sampled point cloud. We took the slope of this line as $\bar{\mu}$ for the location, with units in N/m. Figures 2.7b and 2.7c show the sampled forces, and resulting parameterizations for each phantom.

Returning to captures *with the probe*, a primary concern of ours when capturing the material training data was accounting for the differences between robotic, on-rails motion of the probe and the more fluid, unsteady human-held motion during actual experiments. Our solution is straightforward, and ties into the choice of a two-frame flow for our data – we simply randomize the direction of the probe's movement vector post-contact, within reasonable bounds, in the x and y axes. Ignoring hysteresis and other time-varying effects (like motion blur, which should be minimal), we reason that the optical flow between two points of a random walk – as in unsteady handheld movement – should be nearly identical to the flow between those same two points in a robotically linear motion. We additionally randomize the contact speed of the touches.

The touches are otherwise straightforward. We continue pushing into the material up to a target force of 3 N, quickly retract, and pause to stop any material hysteresis affecting subsequent touches (especially on the softer 00-10 phantom, which displays significant hysteresis). Samples of the material are distributed randomly along the x and y axes, and we perform the same number of touches (typically 1000-2000) on each of the three phantoms. We capture validation and testing datasets in the same way, simply reducing the number of touches to 200 per material.

Training Methodology

We trained our models on a local workstation equipped with an Nvidia GTX Titan X GPU, and packaged and uploaded our trained models to S3 upon completion. Both the force and material models were trained with similar techniques, using our custom data-loading code to formulate the raw video and force readings, captured by the automation rig, into usable input and target output data for training. Each instance of either network was trained with a particular, named, dataset and validated on another named dataset, usually much smaller and captured in the same way as the training set.

Our models were trained up to an epoch limit, which we varied based on each network's convergence rate. The material net was trained with relatively little data, and continued to converge over thousands of epochs, while the force network was trained with tens of thousands of examples, and converged in dozens to hundreds of epochs. We randomized the order of training examples in each epoch. We found that the networks converged well with a mini-batch size of 32, which is fairly standard in the literature. After each epoch, we ran the network (forward pass with no backpropagation) on the validation dataset, obtaining a loss score which we recorded and compared to the training session's best (lowest) score. We saved the network state whenever the best score was "beaten," and re-loaded this gold-standard network state at the end of training. This avoided the problem of worsening validation loss towards the end of training, as the networks overfit to their training data.

Packaging and uploading the trained networks involved naming them (usually by the training dataset, structure, and epoch count), gathering relevant meta-data (such as normalization parameters), and uploading the meta-data and PyTorch `state_dict` for the network to a `models` directory on S3. A model's state dictionary holds its trainable parameters, including the all-important connection weights.

We used the Adam optimizer [16], and found that learning rates of 5e-4 on the force network, and 1e-3 on the material network worked well for our tasks. We experimented with variable (during training) learning rates, with exponential decay and stepped functions, but found little success. We subsequently reverted back to

fixed learning rates, to reduce complexity and avoid them as a potential source of confusing errors. We also found that standard regularization techniques, such as batch-norm and dropout, tended to yield worse results. Dropout in particular appeared to completely prevent convergence in our tasks, so we stopped using these techniques.

2.2.4 Optical Flow Generation

As discussed in Section 1.2.3, optical flow is a high-level feature derived from image sequences – usually, image pairs – and describes the apparent motion of objects from one frame to another in 2D image space. Estimating flows given an image pair is a computationally difficult problem, especially when dealing with large displacements and with real images (which have artifacts like specular highlights) [18].

We generate optical flows for image pairs in the captured video of participant's skin, and these flows necessarily encode both the motion of the probe relative to the participant, and the deformation of the participant's tissues under contact. These separate but related forms of motion are left coupled – we leave the interpretation of the signal up to our material estimator network.

Our use of flows rather than the images themselves (or raw image pairs) is justified by a desire for robustness, and careful consideration of the problem. We do not believe that the texture and color of human skin is the most useful indicator of its (or the underlying tissue's) mechanical properties, and even if it were – say, by indicating the participant's age – we cannot expect to capture any relevant correlations when generating training data with our tissue phantoms. To do so would require advance knowledge of those correlations so that we might encode them in the phantom designs, and such knowledge could only be established using a system like the probe we are developing; a catch-22. Trying to capture this variation would also greatly expand the necessary range of tissue phantoms we had to develop, and the amount of training data we would have to collect for each probe. Finally, it would run the risk of introducing problematic bias – a tattooed participant would very likely display textures and colors outside the distribution of any conceivable training set, leading to (perhaps dramatically) incorrect estimates.

Ultimately, we hypothesize that the most relevant information for visually determining a soft material's physical properties is in its deformation, and we encode this belief into our system by the use of optical flows.

We use a publicly available[9] Tensorflow[10] implementation of FlowNet 2 [14], a cutting edge optical flow estimator, which is itself a particularly large and complex CNN regression model. We use pre-trained weights for this model, and run it alongside our own models on our cloud virtual machine. We do not modify or "fine tune" the model in any way, using it off-the-shelf.

2.3 Robotic Data Collection

Training a DNN from scratch, through back-propagation, requires a very large amount of labelled data, often thousands to millions of examples for robust performance. The amount of training data required for a Computer Vision task can often be reduced by using a "pre-trained" network, either in whole or in part, whose parameterized weights have already been tuned for some difficult task, such as object labeling in RGB images across hundreds or thousands of labeled classes. The idea is that typical images are made up of patches of common textures, colors, and shapes, and that a network trained on a widely-scoped task will necessarily be capable of identifying these common features. Re-training, or "fine-tuning" these networks to, for example, identify a different set of objects that those it was originally trained for, is far more tractable than the original training task working from randomly-initialized weights. For example, a DNN trained to recognize cars would have little trouble learning to recognize semi trucks: low-level texture features such as the presence of smooth body panels, asphalt road surfaces, and different types of skies in the images would often be present in both classes of image; higher-level features like tyres, grilles, and window panels could also be shared, recognized in the new dataset, and the mechanisms of recognition so reused.

The tasks we attack with DNNs bear little resemblance to these more traditional image classification and interpretation problems. We operate on close-up images of novel hardware, with a small, relatively fixed feature set, and 2D optical flow

[9] FlowNet2 Implementation: https://github.com/vt-vl-lab/tf_flownet2

[10] Tensorflow: https://www.tensorflow.org/

fields drawn from video footage of human skin. A standard image classification network like ImageNet would have little to say on the configuration of our probing flexure, and – as these pre-trained networks are often very large – would introduce a heavy computational burden.

Our use of DNNs in our force and material estimation models therefore necessitates the collection of a large volume of data – data which, incidentally, may vary qualitatively with each constructed probing flexure, and with each smartphone model used to build the probe. Manual data collection is a laborious task, slow and imprecise. Synthetic, software-generated data is another option, but comes with caveats. Rendering images which properly cover the true distribution – images which are, in other words, photorealistic, and calibrated to the properties of our physical camera system – would be a research project unto itself, and would additionally involve accurate physical simulation of the flexure and material under examination. We opt instead to use the universe as our real-time "simulation," using a robotic system coupled to a local server to reliably and repeatably generate large datasets.

2.3.1 Hardware Design

The robotic "automation rig" (Figure 2.8) is designed to produce data containing simple touches of some material sample(s) under consideration – typically a set of silicone phantoms, discussed further in Section 2.3.2. A *touch* consists of positioning the tip of the probing shaft above the sampling location, pressing the tip into the material (thereby applying some measurable force), and retracting. Force, position and video data are captured in real time as each touch is carried out.

Our rig is based on a Force Dimension Delta 3[11] robot platform, which provides 3 axes of motion. The robot's control libraries allow positioning the effector, and applying forces in 3D cartesian space. An ATI Mini40[12] force/torque sensor provides precise ground-truth force readings, and the probe itself (Section 2.1) is mounted on the robot's end-effector.

We arrange this hardware with two 3D-printed assemblies: a "base," which fastens securely to the robot's stationary frame, mounting the phantom tray (described

[11] http://www.forcedimension.com/products/delta-3/overview

[12] https://www.ati-ia.com/products/ft/ft_models.aspx?id=Mini40

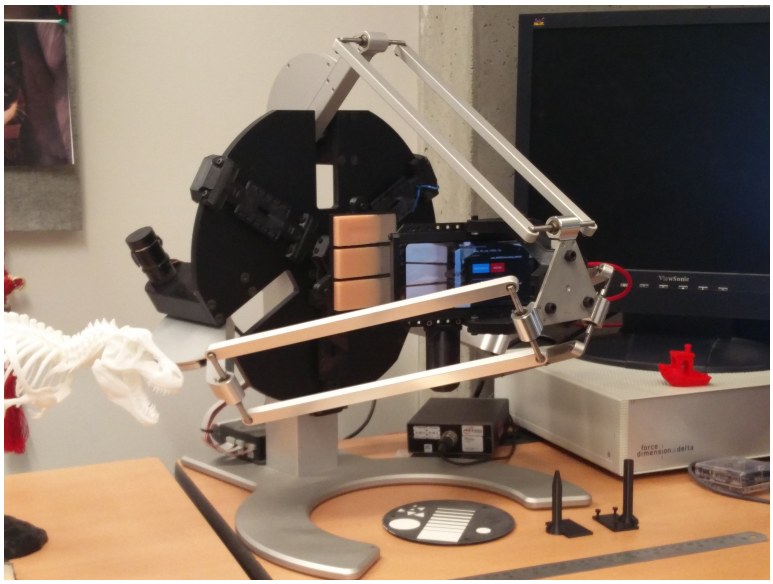


Figure 2.8: The "automation rig," with the delta robot platform, silicone phantom tray, and mounted probe visible.

in Section 2.3.2) through the in-line force sensor; and the probe mounting, a simple structural addition to the robot's end-effector, which facilitates fastening and removing the probe. We fabricate these assemblies using the same Ultimaker 3[13] 3D printer we use to fabricate the probe and flexure.

2.3.2 Tissue Phantoms

In lieu of a software simulation of human tissue, and without a human (or other mammal) to confine in our data capture apparatus, we revert to the next best thing: a set of silicone "tissue phantoms" (Figure 2.9), which attempt to mimic the soft, viscoelastic behavior of human tissue, and cover a range of material properties – mimicking, for instance, tensed or slack muscles; fatty belly tissues; or the thin skin-on-bone found around the clavicle.

We use a small selection of different casting silicones, covering a range of stiffness levels (Figure 2.10). These are Ecoflex[14] 00-10, 00-30, and 00-50 silicone

[13] https://ultimaker.com/en/products/ultimaker-3

[14] https://www.smooth-on.com/product-line/ecoflex/

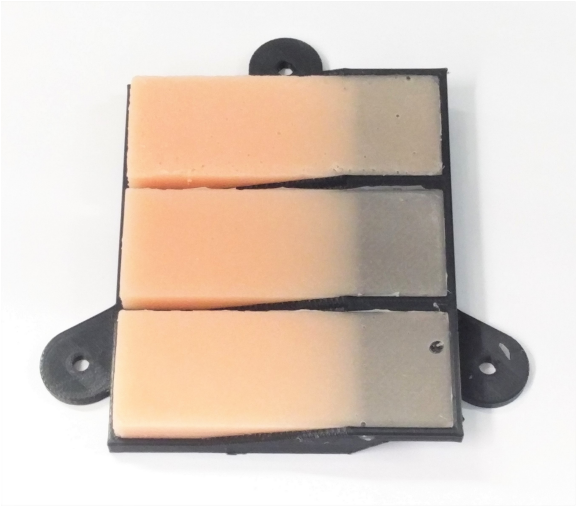


Figure 2.9: The assembled phantom tray. Silicone phantoms are incorporated in order of increasing stiffness, from softest at the top to stiffest on the bottom.

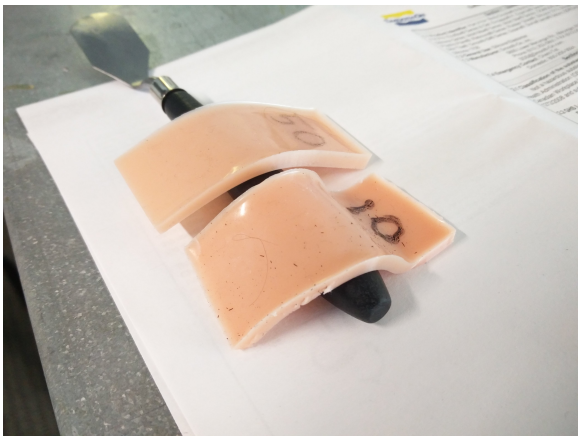


Figure 2.10: Two silicone phantoms cast outside the tray, demonstrating different levels of softness with variant resting configurations.

Figure 2.11: Cutaway view of a silicone phantom and its 3D-printed tray, used for material model training.

Figure 2.12: Close-up of the phantom tray before silicone casting, showing lattice base and removable (disposable) mold walls.

rubbers (Smooth-On Inc., Macungie PA), which we occasionally refer to as "S-10," "S-30," and "S-50" respectively.

These silicone resins are cast directly into the 3D-printed "phantom tray," which additionally varies the *thickness* of the soft materials beneath a uniform surface (Figure 2.11). This thickness then becomes one of the material properties learned by our material estimator, and creates additional variety in the measured $\bar{\mu}$ stiffness of the materials.

The phantom tray itself is designed with some of the same fabrication techniques as the probe flexure, discussed in Section 2.1.1. The *base* of each phantom volume is constructed of a dense, rigid, three-dimensional lattice of printed plastic (Figure 2.12). This lattice provides an extremely robust connection between the

Figure 2.13: Phantom tray during silicone casting, with mold caps installed.

silicone and the plastic tray, as the liquid silicone resin flows into and around the lattice before curing. As with the flexure, the walls of the casting volumes are made of disposable breakaway material, which is printed as an integrated part of the tray with an Ultimaker 3 dual-extrusion printer. The walls are broken away post-curing, leaving a smooth finish, and well-defined geometry for the phantoms. We ensure the top surface is perfectly level (and that the phantoms display some surface texture) with printed "cap" pieces for the molds, which slot in once the silicone resins have been poured (Figure 2.13).

2.4 Automation Server

The automation server operates and monitors the automation rig. This involves controlling the robot, reading values from the force sensor, remotely operating the probe's smartphone application, and providing a web-based graphical user interface (Figure 2.14) for user programming and control of automated capture sessions.

The architecture of this system is built around a custom Python Flask[15] server which provides API endpoints: these endpoints activate the server's various capabilities, can be polled for the system status, and generally provide the "back-end" functionality of the user interface. We run high-performance robot and force sensor data acquisition in background `C/C++` threads, which interface with our Python

[15]Flask documentation: https://flask.palletsprojects.com/en/1.1.x/

code.

2.4.1 Automation Programs

Automated recording sessions generate large numbers of recordings with our automation hardware, with limited or no user interaction beyond initial programming. To program the system, the user creates an "automation script," which controls the robot and probe – we discuss the functionality of this scripting system here.

Automation scripts are entered, edited, and run from our web UI. The scripts themselves are sent to the automation server for execution.

Recordings are grouped into named "sessions," and can be tagged individually with arbitrary metadata. Automated recordings hold not just the probe's videos, but also high-frequency measurements from the force-transducer, and calibrated position readings from the robot.

The programming of our automation system is built around Python scripting, which provides a familiar and exceedingly powerful environment for generating commands. It provides the ability to generate commands in loops and other high-level structures, and the ability to use Python's vast selection of libraries. The ability to use `numpy` proved particularly useful, as we are working primarily with numbers and vector positions. We can, for example, target random locations within a defined range by simply using `numpy`'s random number generation facilities.

Our scripts are written with special functions which emit commands for the automation server to execute through its sequential controller. We summarize the available commands in Table 2.2. These commands are collected into a complete sequence when the script is first run – that is to say, the entire program is executed ahead of time. We structure the programming this way – as opposed to dynamic scripts, which might be able to read and respond to the system state – as it allows us to perform basic correctness checks on the scripts before execution. If any part of the script produces an error (such as a divide by zero), the script fails to run, whereas a dynamically-executed Python script would only fail when it reached the point where the exception was generated. Since these automation programs can take several hours to run, a failure part way through (or at the end!) of a program would be a frustrating waste of our researcher's time. This structure also allows us

to perform more high-level correctness checks: for example, we can guarantee that all `start_record` calls are matched by a `stop_record` call. Finally, having a fixed-sized list of commands for each execution run allows us to display a progress bar, and a completion time estimate.

Python is a fully-fledged scripting language capable of, among other things, full filesystem access. When combined with the fact our scripts are entered through a web interface, this represents a potential security issue, as the scripts are executed remotely. We mitigate this by exposing the server only to the local network, and by requiring username and password authorization to access the UI, and all API endpoints.

There are some features here worth discussing, which go beyond the basic record/move/sleep commands. We use `await_input` to create partially automated scripts, where the user may move the probe manually for some recordings, or where the material tray or flexure should be swapped out (or any other aspect of the hardware manually reconfigured). We use this in combination with `enable` and `disable_motors` to move the probe to a target location, then allow the user to disengage the motors and start a recording with one press, before re-engaging and finishing the recording with a second press. We make extensive use of the ability to move up to a force limit, which helps keep data consistent, and prevents damage to the flexure and phantoms by limiting maximum strain.

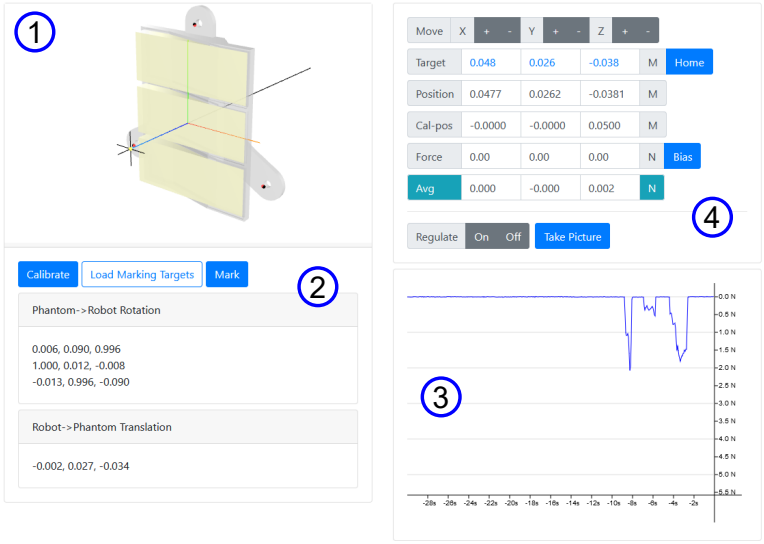
2.4.2 User Interface

Our web-based user interface (Figure 2.14) features a live readout of various metrics direct from the automation hardware. The interface is a HyperText Markup Language (HTML) webpage, accessed via a client browser. The page is served directly by the Flask application. We built the interface with Bootstrap user interface elements, and poll with asynchronous JavaScript and XML (AJAX) requests to load live data. The interface allows the user to monitor the robot's position – the position of the flexure tip, assuming no deflection – and the force being applied to the material tray.

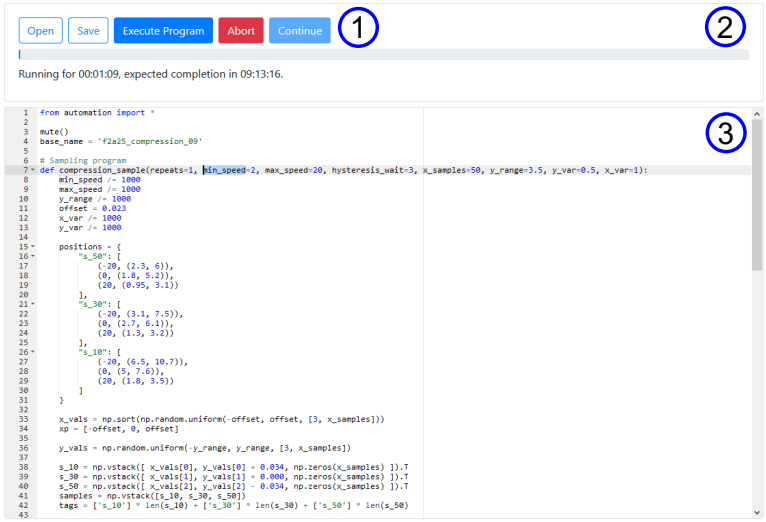
The user can monitor the robot's position through direct numeric readouts, which show both the raw robot-space position, and calibrated phantom-space po-

Command	Function
`start_session(s)`	Start a new session, named `s`. Recordings started after this command will be grouped under this session. Must be matched by a subsequent `end_session` call.
`end_session()`	End the current session, packaging and uploading the recorded data to S3.
`start_record(m)`	Start a new recording, attaching the provided metadata object `m` – typically a Python dictionary, which must be serializable to JSON. Must be matched by a subsequent `stop_record` call.
`stop_record()`	Stop the current recording, holding while data is saved and video is uploaded.
`sleep(t)`	Stop processing new commands for `t` seconds.
`move(p[,s[,t[,f]]])`	Move the robot to phantom-frame position `p` with speed `s`. The command (but not necessarily the move) ends when the robot is within threshold distance `t` of the target, or normal-force `f` is exceeded.
`sync()`	Synchronize the server and smartphone clocks, obtaining a new Δ_t measurement as described in Section 2.5.1.
`bias()`	Bias the force sensor readings to a new zero-point.
`set_headless()`	Disable smartphone recordings. Used for material force-displacement measurements for ground-truth data.
`mute()`	Disable saving and upload of any data (outputs a warning). Used for developing and testing scripts.
`await_input(m)`	Display a message `m` and wait for user input – the user may click "Continue" in the UI, or "Go" in the app to continue execution.
`enable_motors()`	Enable the robot's motors.
`disable_motors()`	Disable the robot's motors.

Table 2.2: Control commands available in our Python automation programming environment, and their functions.



(a) Direct monitoring and control elements. 1) Interactive 3D visualization of flexure tip position relative to phantom tray CAD model. 2) Position transformation calibration (controls and results) 3) Live normal force graph. 4) Direct system control - "regulation" enables or disables the motors, and movement controls allow manually moving the robot.



(b) Programming controls. 1) Program execution controls, and program save/load (on client machine). 2) Program execution progress bar. 3) Interactive editor with syntax highlighting, multi-select.

Figure 2.14: The automation server's web-based user interface.

sition. The position is additionally displayed in an interactive WebGL view. This view shows the phantom tray using its 3D CAD model, and represents the idealized tip position as a small yellow marker with intersecting lines. These lines extend out from the marker point to provide additional 3D reference for its location relative to the phantom tray. The user can shift their view of the tray and tip by rotating, panning, and dollying[16] the virtual camera with mouse controls. Additional markers serve as calibration points: if the reference and measured markers overlap completely, the calibration is perfect (see Section 2.4.3 for more details on the calibration process).

Forces are monitored in two ways: through direct numeric readouts, which show both the raw force reading and a moving average; and through a live HTML canvas graph, which shows only the normal component of the moving-average force reading. This graph is scalable in the Y-axis, using the mouse wheel. We found no existing JavaScript packages which worked well for providing this type of live, scrolling view, so we developed a custom solution which is functional, but basic. Our graph places one "reading" (response from the automation server) at each screen pixel, which corresponds to an approximately 30-second long window. The main issue with this approach is that slow or intermittent responses, which can occur when the page is de-focused, will unpredictably and unevenly shift and scale measurements in the time axis. It also means the graph cannot show high-frequency changes: the sensor itself is polled at approximately 300 Hz, but the UI retrieves a moving-averaged force reading at only 15 Hz.

Direct control of the robot is enabled through push-buttons in the UI. The robot can be moved in small increments along each axis by adjusting its "target position" (displayed above the position readouts) with the "+" and "−" buttons. It can also be brought immediately to a safe position with the "home" button. Finally, the user can completely disable the robot's motors by switching off position regulation; re-enabling regulation will cause the robot to hold its new position, so that the end-effector can be placed by hand and left stationary.

[16] Dollying refers to moving a camera forwards and backwards along its line of sight. This is distinct from "zooming," where the field of view is changed on a static camera to enlarge or shrink objects in the frame.

2.4.3 Calibration

The Delta-3 robot's proprietary control software calculates the cartesian position of its end-effector by applying forward kinematics with its measured servo positions. This is a helpful capability, which we rely on heavily, but we wish to obtain the tip position (translated from the end-effector), and want this position relative to the phantom tray. This makes the automation programs easier to write (with a coordinate origin in the tray center) and more repeatable – re-assembling or re-attaching the phantom tray mounting in a slightly different position should not change the results.

Our calibration process revolves around measuring three fixed points X on the phantom tray in the robot's coordinate frame, which have known positions Y in the phantom's coordinate frame. For calibration points, we use the three bolts which attach the phantom tray (or other material tray) to the force sensor assembly. A rigid "calibration probe" may be used to manually register these bolt's positions in 3D space by following these steps:

1. Replace the flexure with the calibration probe
2. Support the probe with one hand, and disable the motors using the `Regulate/Off` UI control
3. Move the probe so that the calibration probe slots over the top bolt
4. Click `Calibrate` to mark the calibration point
5. Repeat steps 3 and 4 for the remaining two bolts, in clockwise order
6. Return the probe to a safe position, and re-engage the motors with `Regulate/On`

When the three positions have been sampled, we automatically calculate and save a new calibration. This involves constructing a transformation composed of a rotation matrix R, and translation vector $\boldsymbol{t}$. This transformation maps points from the robot frame to our phantom frame. Let $\bar{x}$ be the mean of the measured positions,

and $\bar{y}$ be the mean of the phantom frame positions. We can then calculate the transformation, using the singular value decomposition (SVD) solution [27] to Wahba's problem to find the rotation matrix. Note that we weight all points equally:

$$\begin{aligned} B &= \sum_{i=1}^{3} (x_i - \bar{x})(y_i - \bar{y})^T \\ B &= USV^T \\ R &= U[\text{diag}(1,1,\det(U)\cdot\det(V^T))]V^T \\ \boldsymbol{t} &= \bar{y} - R \times \bar{x} \end{aligned} \tag{2.1}$$

We can then transform any point x in the robot's coordinate frame (such as a position reading) to and from the corresponding phantom frame position y as follows:

$$\begin{aligned} y &= R^T \times (x - \boldsymbol{t}) \\ x &= R \times y + \boldsymbol{t} \end{aligned} \tag{2.2}$$

2.5 Experiment Software

Our software runs during data capture sessions, facilitating the collection of samples in the form of video and raw data files. The probe runs our Android Application, gathering and uploading video data, while a virtual machine in the cloud runs a suite of server applications which the probe interacts with. Additionally, in automated capture sessions, a local server runs our robotic automation rig (Section 2.3), provides a web interface for controlling and monitoring these automated capture sessions, and operates the probe via local WiFi.

2.5.1 Android Application

The Android application (Figure 2.15) runs on the probe's smartphone, and serves to capture the video data on which our material predictions are based. We operate the application in two ways: with standard touchscreen controls, visible in the

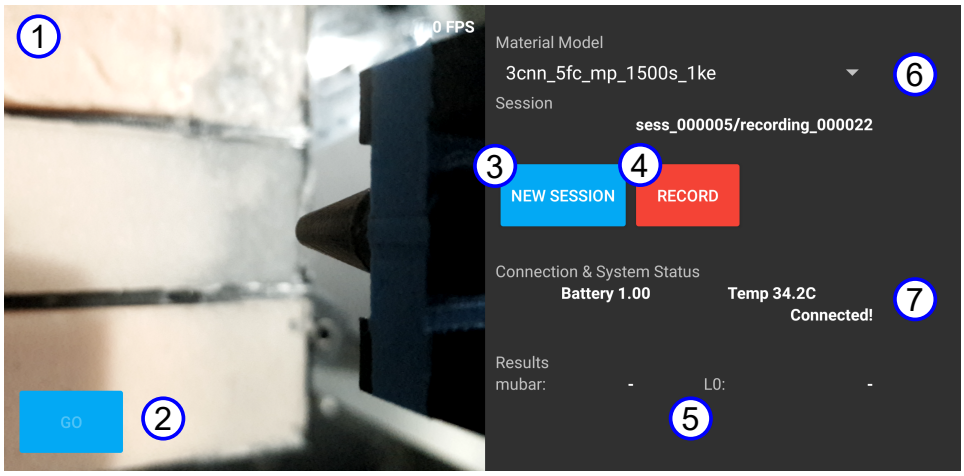


Figure 2.15: Screenshot of the probe's smartphone application, with the phantom tray visible in the camera preview (the probe is mounted in the automation rig). 1) Live camera preview. 2) Action button for partially automated captures. 3,4) Session control buttons, for creating new sessions and starting and stopping manual recordings. 5) Material estimation results display. 6) Material estimator model selection (spinner). 7) Phone system status: battery level and temperature.

screenshot; and remotely, through the automation server discussed in Section 2.4. The application uploads captured videos directly to S3, and is additionally able to issue commands to, and display results from the cloud server discussed in Section 2.5.2.

Video Capture

We ensure consistent video footage for training and inference by manually setting the camera's exposure, focus, and color-adjustment values. We additionally enable the flashlight for video recordings, which provides highly consistent lighting in our images – within the 6 cm the probe's camera operates at, the flashlight drowns out nearly all ambient light sources. We reduce exposure times to account for this, which has the added benefit of reducing or eliminating motion blur artifacts. The manually-set values we use are summarized in Table 2.3. We additionally set the color transformation and color correction gains by reading out values from an auto-

	Preview	Capture
FPS	30	30
Focus distance	55mm	55mm
Auto-focus	No	No
Auto-exposure	No	No
Optical stabilization	No	No
Video stabilization	No	No
Torch	No	Yes
Exposure time	10ms	3ms
ISO Sensitivity	1500	70

Table 2.3: Probe application camera settings[17] for Samsung Galaxy S8 rear camera.

corrected image taken with our other settings held constant.

Videos are recorded as 1024×768 resolution `mp4` files with a high quality setting for compression, and no audio. We (optionally) upload the video files to the current session's directory on S3, and immediately remove them from the phone's local storage post-recording to avoid exhausting the available capacity. Uploading requires a functional internet connection, using either mobile data or WiFi. Failed uploads are re-attempted, and in automated capture sessions, no new recordings will be made until upload has completed.

Automation

For automated capture sessions, the phone application maintains a background TCP connection with the automation server over a local WiFi network, with an additional layer of detection for dropped connections, and the ability to resend dropped messages. Messages are exchanged in a simple JSON format, and consist primarily of simple instructions for the phone from the server, and confirmation from the phone that execution of those instructions completed. This confirmation is necessary as some actions, like starting a recording, can take several seconds.

[17] Refer to Android Camera2 CaptureRequest documentation for additional information on the meaning of these settings: https://developer.android.com/reference/android/hardware/camera2/CaptureRequest

We thus established robust, asynchronous communications between the phone and server.

When operating in tandem with the automation server and robotic platform, accurately recording the timing of image captures becomes essential. Even a small delay could lead to images of the flexure and material samples being systematically mis-labeled with force and position readings, which are recorded separately by the automation server. To correct for this, we establish a precise time offset between the phone and server system clocks, assuming similar latency in both communication directions. We calculate a time delta between the two devices by recording the start time t_0 on the server, requesting a timestamp t_p from the phone, and recording the response time t_1, again on the server. The time delta is then:

$$\Delta_t = t_p - t_0 - \frac{t_1 - t_0}{2}$$

Where at any time t:

$$t_{phone} = t_{server} + \Delta_t$$

We repeat this process 200 times, and take the average Δ_t. We found that the clocks drift without frequent synchronization – on the order of 0.01s an hour – and this drift can build up enough to affect synchronization of force and position readings with individual frames of video. We avoid drift between the two clocks by repeating this synchronization process frequently during longer capture sessions, which can last 12 hours or more.

Operating a smartphone for extended periods is somewhat outside such a device's intended usage, and presents its own difficulties. Recording video is an CPU- and power-intensive task, and we found that the phone's battery would often drain despite being plugged in to the AC adaptor. Evidently, the adaptor did not provide enough power alone, so the phone's battery had to make up the difference. This high power usage would, after some time, completely drain the battery, causing an uncontrolled shutdown. It also created a secondary issue: power use in electronics invariably translates to heat, and the phone's overheat protection would throttle CPU use and, eventually, shut down the device. We solved these issues in software, by monitoring the phone's battery level and temperature, completely disabling the

camera whenever these metrics passed critical levels. This allowed the phone to cool and/or charge until the metrics passed above a second set of target levels. The application maintains a state machine with transition queueing to ensure that, for example, starting a new recording is deferred until recordings are resumed, and that the camera is not inadvertently disabled during an active recording.

Live Experiments

In a live experiment, the application's role changes in several ways. A connection to the automation server is, naturally, no longer required, and the application becomes fully controlled by the experimenter, through the touchscreen.

The experimenter can start a new "session," and then make an unlimited number of recordings in each session. "Live" recordings are treated differently to automated ones by both the phone application and server suite, in large part due to the lack of external sensing hardware – there are no force sensor or robot position readings to store, and no automation server to synchronize with or send updates to. Instead of the automation server generating and uploading a metadata `json` file for each recording, as in automated touches, the phone application generates this file. It includes accurate capture times for each video frame, and the start and stop times of the recording. These files are uploaded individually, rather than packaged in a `zip` archive as in automation sessions. Videos are uploaded as normal, albeit to a different S3 folder.

Once each live touch has been completed (when recording is stopped), and the resulting video has been uploaded, the phone application sends an HTTP request to our cloud instance, demanding a new material estimate for the recording. The request specifies the session, recording, and *model* to use to generate the results. The model to use is selected by the experimenter through a drop-down UI, which is populated by listing the available models on S3. The job request occurs on an independent background thread, which then continues to monitor the progress of the material estimation job in the background, polling the instance with additional requests. Multiple simultaneous requests are supported, whether from a single probe device or many – jobs are simply queued by the instance, and several monitoring threads may run at once within the phone application, though it may not be

clear which set of results corresponds to which recording; currently, only one set of results is displayed at a time. The application alerts the user when estimation has completed (or failed) via a UI update. The resulting $\bar{\mu}$ and L_0 estimates are shown in the results display within the application (Figure 2.15) as soon as they are available, and are also stored on S3 for later processing.

Older "live" recordings may be subsequently reevaluated on-demand using newer, or different models. New estimates are indexed by the model used to create them, not overwritten. This reevaluation is not currently possible from the phone application itself, which provides no ability to browse or play back old recordings, or to examine old results – however, these capabilities are perfectly feasible with software updates to the application.

2.5.2 Cloud Server Suite

We perform live, GPU-intensive computations for the SkinProbe 2.0 system on an AWS `P3.2xlarge` instance, which we operate on-demand. The instance performs three main tasks: force inference, optical flow calculation, and material parameter estimation. The force and flow tasks are generally rolled into a material estimation calculation – as discussed in Section 2.2, force estimates and optical flow fields for a given recording are necessary for generating these final material estimates. However, we split these tasks semantically so we can maintain the capability to perform them separately. This allows us to test separately, and to automatically obtain force estimates and flows from the cloud instance for training.

The three tasks are split across two main `systemd` services on the instance, with an additional service providing the public-facing API endpoints using Flask[18], and an automatic update service which runs at startup, pulling in new code. Finally, we also run a standard Redis in-memory database on the instance, which we use for high-performance IPC, with synchronous polling and robust queueing.

Workflow for Live Experiments

During a live experiment, the smartphone application communicates with the cloud instance to obtain results. Upon receipt of a request for new material estimates,

[18]This service should not be confused with the local automation server, which also uses Flask

the Flask server places the job (and its associated parameters) on a Redis queue, and creates a new entry in the database representing the job's status and results. The PyTorch server, which synchronously[19] polls from this queue, receives the job and begins processing it. Meanwhile, the Flask server responds to the phone application with a job ID, which can then be used in subsequent queries to monitor the process status.

As discussed in Section 2.2, generating a material estimate involves first estimating forces applied throughout the video, selecting frames for optical flow, generating that flow, and executing the material model with the flow as input. The PyTorch service executes most of this process, communicating with the flow server through Redis to trigger flow generation (the flows are then available on the local disc).

Some implementation details complicate the pipeline we have previously discussed. As the video data is initially uploaded to S3, and is not immediately available locally, the server must download the new data, and unpack it to an image sequence on the instance's virtual storage – we achieve this using the same data loading libraries we use for training and testing our regression models. The force and material models requested for the job may also not be available locally; in this case, they are dynamically loaded from S3, where they are stored in their fully-trained state. When processing material estimation requests, which require both a material and force model, the downloaded material model's metadata then specifies the particular force model to download and use for the force estimation step. As only one pass through each dataset is required, we use our data loader's *lazy loading* ability to keep memory usage in check.

The optical flow service runs in a separate process to the material and force estimation, and exists to operate the Tensorflow FlowNet2 model. It uses Redis for inter-process communication in precisely the same way as the estimation service: when optical flow pairs are required by the material data loader in the main estimation process, a new flow job is posted to a Redis queue, requesting those flow pairs. The flow service polls this job from its queue, loads the images from disc, estimates the optical flow fields, and saves them back to disc in Middlebury `flo`

[19] In this context, synchronous means *blocking*. The PyTorch service waits idle until a job becomes available on the queue.

format. Once all pairs are processed, the flow service updates the job to indicate that the results are available for loading by the estimation process.

After completing the estimation pipeline to obtain normalized results, we denormalize the results to real units using each model's normalization parameters. The generated force and material estimates are then gathered and stored on the job's Redis entry, and the job is marked as "complete." We finally add the generated estimates to the session's estimates files, indexed by the model used to create them, and (re-)upload them to S3. These files constitute a basic database of completed estimates, which can then be downloaded for offline analysis.

Throughout the job's processing, which takes several seconds, status updates are written to the previously-created Redis database entry. These updates may be read asynchronously by the smartphone – or any other – client by simply polling a status HTTP endpoint with the job ID. The client obtains final results in the same way – once they are made available on Redis, the generated estimates are returned in the status response, for display to the experimenter through the smartphone application UI.

Chapter 3

Results

Here we outline and discuss the results of our work, analyzing the accuracy of the probe's flexure-based force estimation, and of the generated material results for the synthetic tissue phantoms – the same phantoms used to train the probe's neural nets. We establish the prototype probe's functionality under controlled conditions, and with this limited set of materials, and further evaluate performance of the system when probing is done by hand.

3.1 Force Predictions

The overall design of our pipeline for SkinProbe 2.0 means that accurate force predictions are an important foundation for accurate overall results. Poor force estimates could add noise or, worse, bias to the final material estimates by leading to incorrect frame selection – with a target of 1 N, underestimating the force and mistakenly selecting frames at a true applied force of 1.2 N could lead to a $\bar{\mu}$ underestimate, as the observed deflection would be greater than at the levels the material estimator was trained for.

We therefore tested the force predictor experimentally, to validate its performance in isolation from the rest of the pipeline. We captured data for these experiments using the robotic automation rig, generating probe touches on either the rigid force-training plate, or the phantom tray. We programmed for touches with various angles of attack, speeds, and maximum forces – these variables were ran-

	Loss (Newtons)			
Dataset	MSE (L2)	MAE (L1)	RMSE	R^2
Training	4.44e-03	0.0432	0.066	0.9977
Validation	2.74e-03	0.0359	0.071	0.9972
Test	5.08e-03	0.0422	0.069	0.9964
Soft contact	7.43e-03	0.0592	0.086	0.9931
Leather contact	1.03e-02	0.0791	0.101	0.9916
Guided handheld	8.90e-03	0.0655	0.094	0.9927
Freely handheld	3.75e-03	0.0426	0.061	0.9961
Post-stress	1.10e-01	0.229	0.332	0.9185
Post-stress corrected	6.04e-03	0.0492	0.078	0.9955
Post-stress retrained	4.81e-03	0.0409	0.070	0.9961
Leather in direct sunlight	3.24e-01	0.445	0.569	0.7676

Table 3.1: Force estimator network MSE and mean absolute error (MAE) losses, root mean square error (RMSE) and R^2 metrics. The losses and RMSE are measured in Newtons, which the network outputs directly.

domized across the different samples. We further evaluate the model in handheld usage, with "randomization" of touches carried out by the experimenter.

We summarize the accuracy of our force estimation results numerically in Table 3.1, and expand on these results in detail in the remainder of this section.

3.1.1 Performance on Training and Validation Data

We first examine the performance of the force model on its own training and validation datasets. We plot individual touch recordings from the datasets for clarity - note again that the force readings are calculated per-frame, so while some of these plots run over time, each estimate is calculated independently.

The net performs well on its own training data (Figure 3.1), with 87% of the training samples yielding predictions within $\sim 0.1\,\mathrm{N}$ of the measured ground truth. This is not surprising – in fact, neural networks often over-fit to their training data, picking up on artifacts and specific patterns which may appear there, but which are absent in real-world data. We might expect the network to do better here than it has, and indeed leaving the network to train does improve results on the training

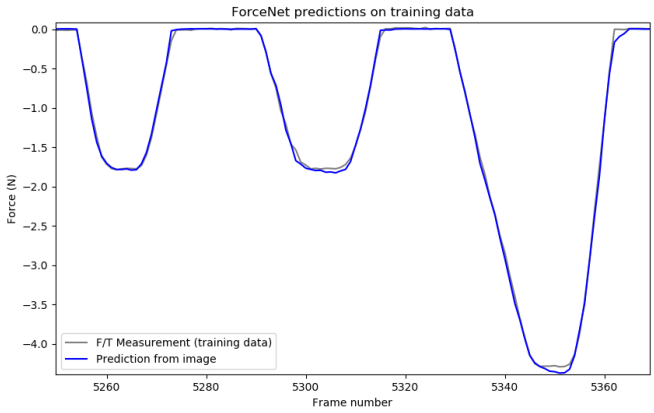


Figure 3.1: As expected, the force network predicts forces in its training set extremely accurately; results of several consecutive touches are shown, with different maximum depths (maximum force levels) and contact speeds.

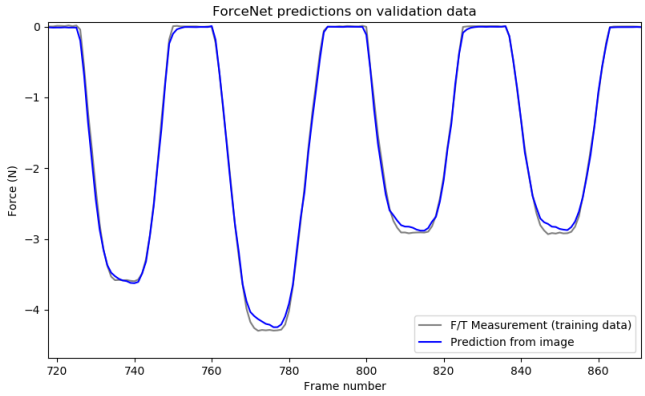


Figure 3.2: Performance of the ForceNet on its own validation dataset, which is captured in the same way as the training set. The net is not trained on this data, but the network state which performs best on this data is selected. The results here very nearly match performance on training data.

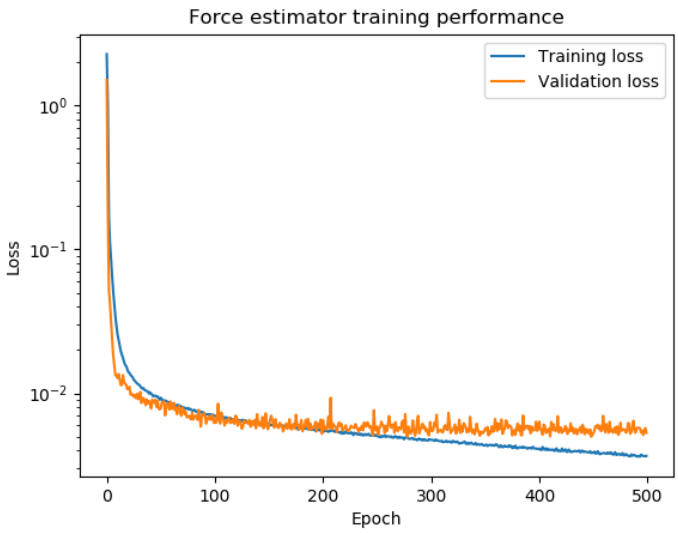


Figure 3.3: Progression of the average per-item training and validation losses while training the force estimator. Note the logarithmic y-axis. Training loss continues to drop long after validation loss has plateaued, demonstrating over-fitting – to counter this, we save and use the network which performs best on validation data.

set to the point of near-perfect accuracy (Figure 3.3). Since we save the network during training at the point it performs best on its *validation* data (which may occur relatively early in training), this effect is not seen in these results – performance on the training data is imperfect. We show how the network performs on its validation data in Figure 3.2.

In fact, comparing the MAE for training and validation data shows that this particular network performs better on its *validation* set than on its own training set, with 0.0432 N MAE for training data, versus 0.0359 N for validation data. 91% of validation data predictions fall within 0.1N of the ground-truth.

This improvement over training data performance is slightly surprising, but is perhaps explained by outlier predictions in the training set. It seems that fewer such "difficult" cases appear in the validation set, which is far smaller, and which may simply miss these types of outliers by chance. Another explanation is stochasticity in the validation performance – examining the loss graph in Figure 3.3 shows that

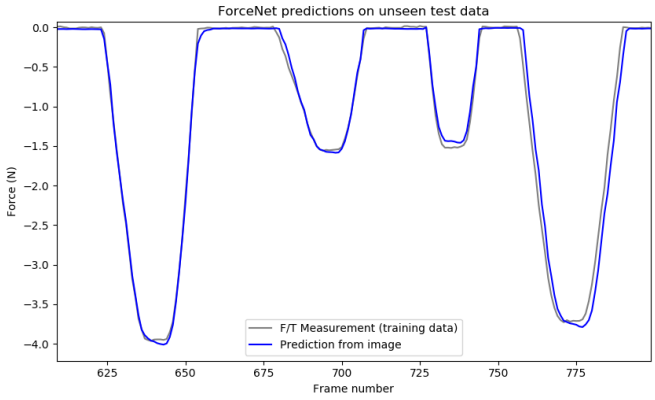


Figure 3.4: Performance of the ForceNet on an unseen test dataset, which was captured in the same way as force training and validation data. The net was not trained or validated on this data.

the validation loss fluctuates noisily, even long after it plateaus. If the performance happens to peak at an early point (while training data performance is still worse), this network will be saved and kept. In this case, the network would be over-fit to the *validation* data.

3.1.2 Performance on Test Data

The force network also performs well on test data – data not seen in any way during training[1] – with some caveats as our test data moves away from the distribution of training and validation data. This first test set is a small collection of rigid plate touches, captured in the same way as the network's validation data. We show the network's results on this data in Figure 3.4.

The network accurately tracks the overall shape of the force curves, with no apparent noise and with only minor inaccuracies. The MAE of 0.0422 N compares favorably with results on the training and validation datasets, with the network performing only slightly worse than on its validation data, and marginally outperforming its training data results.

[1] As discussed above, our networks could be slightly over-fitted to their validation data, as we explicitly keep the network weights which yield best performance on that data.

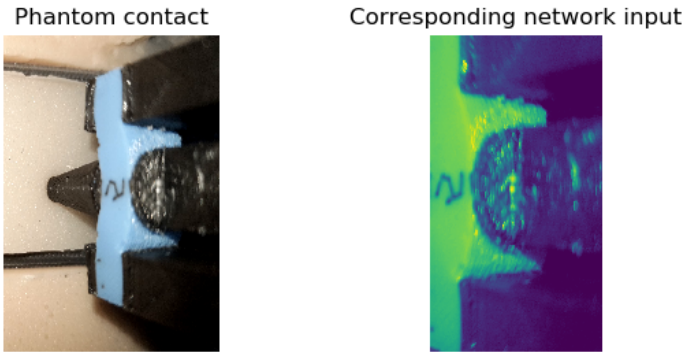


Figure 3.5: Testing force estimation in contact with a phantom.

3.1.3 Performance on Novel Test Data

Testing the neural net only on data captured robotically, in a fashion identical to its training data, is somewhat misguided. In our system, the force predictions are intended to eventually be used in contact with soft human tissue, which displays a variety of colors and textures that may inadvertently be shown to the network through reflections, and other optical artifacts. Motion of the probe when held by a human, or when in contact with different materials, may also differ qualitatively from robotic motion in contact with a rigid plate. These different types of motion could reveal flexure configurations which are not reached (or not commonly reached) in the robotic training data. In the remainder of this sub-section, we test the network's resilience to different contact materials, surface aesthetics, and modes of motion.

First, we capture data in contact with one of the flesh-colored silicone phantoms (Figure 3.5). Here, the network performs well in general (Figure 3.6), but struggles with cases displaying unusual (with respect to the training data) configurations of the flexure. Summarizing performance over 30 touches yields an MAE of 0.0592 N, which is worse than the tests on a rigid backing, but this averaged error does not tell the whole story. On mid-to-high force touches, the network performs as normal, but it struggles to predict on some of the lower-force touches, pathologically continuing to predict near-zero forces at levels which are normally

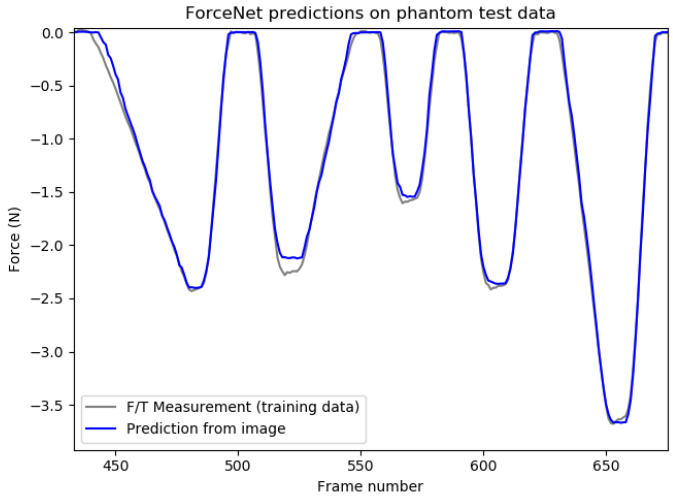


Figure 3.6: Performance of the ForceNet on a test dataset containing touches with a silicone phantom. These touches are otherwise similar to those carried out on the rigid force plate for training.

easily detectable. Figure 3.7 shows one touch where this effect was particularly dramatic.

In this case, the network barely deviates at all from a prediction of 0 N force, despite clear contact being made. Examining the data tells us why – Figure 3.9 shows the frame with the highest error, and compares it to one with no force applied. The two frames are visually very similar, making a zero-force prediction understandable. This touch involved a shallow contact with significant lateral motion – the probe was moved to the left during contact, pulling the flexure tip to the right, away from the camera, and exerting nearly as much lateral force as normal force (which is what the network predicts). The network is not trained with data like this, for the simple reason that this type of motion results in sliding on the rigid plate; the flexure cannot be moved into a configuration with proportionally high lateral force without a grippy, or slanted surface to press against. The best way to train the network to predict accurately on touches like this would be to include similar data, with large lateral forces, in the training set. This would require a force plate with a grippy surface, presenting a shortcoming in the rigid

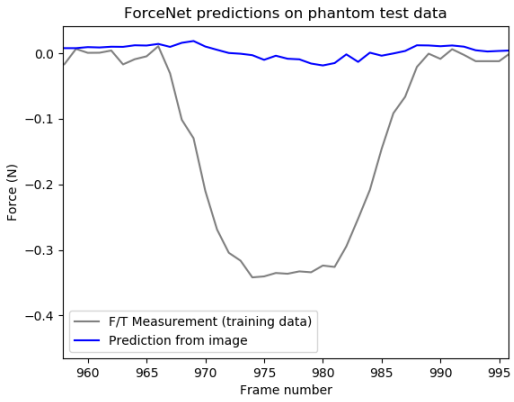


Figure 3.7: Poor force estimation on a pathological case, in contact with a soft silicone phantom.

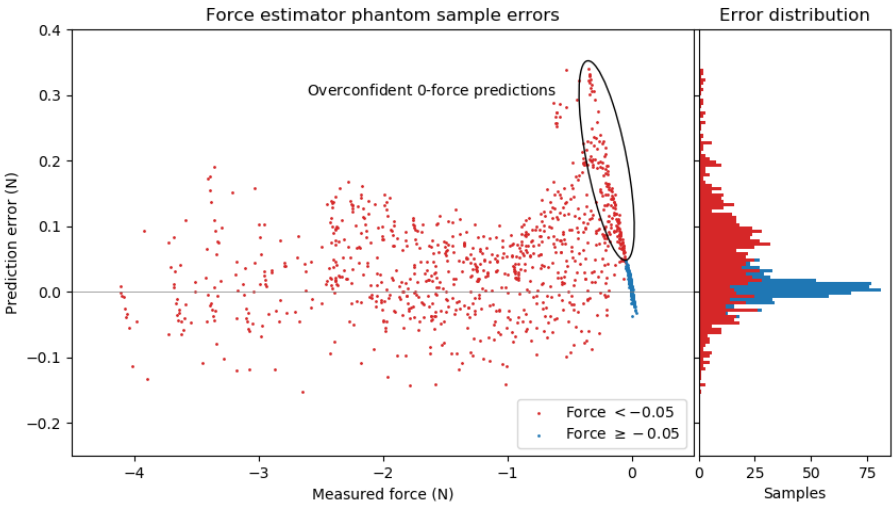


Figure 3.8: Contact force errors, with distribution, in the phantom force test. We separate estimates at near-zero measured force from those at higher force levels to highlight the network's relatively high accuracy near zero, and the network's tendency to incorrectly predict zero force even at fairly high force levels – up to $-0.4\,\mathrm{N}$.

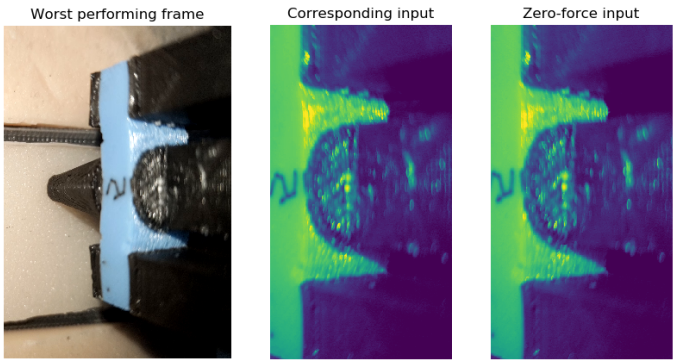


Figure 3.9: Poorly estimated frame of force network input, compared to a frame with no force applied. Here, the network predicted $-0.003\,\mathrm{N}$, for a frame with a true measured force of $-0.342\,\mathrm{N}$.

plate design, and an opportunity for future improvement.

Luckily for our purposes, we do not consider this type of contact typical for the touches used for probing, which largely move directly into and out of the target – this "blind spot" in the network's training should not affect the intended use case.

Examining the distribution of prediction errors directly (Figure 3.8) shows us that the network has a strong preference for predicting a zero force level. This is troubling, but is likely a simple reflection of a skewed distribution in the training data. The network learns that images with no force applied are extremely common, and while this is true, feeding the network a proportionally large number of these cases – as we currently do – may be counterproductive for training and prediction performance, as they are all extremely similar. These zero-force frames are also clearly not captured during touches, making them of little interest to us.

We captured an additional dataset in contact with leather, laid over the same phantom as above. The leather provides a far darker, more specular surface (Figure 3.10), as well as different contact dynamics. Examining the prediction graph (Figure 3.11), we see that the network follows the true force curve, but often does so with an erroneous offset. In general, it appears to have a troubling tendency to

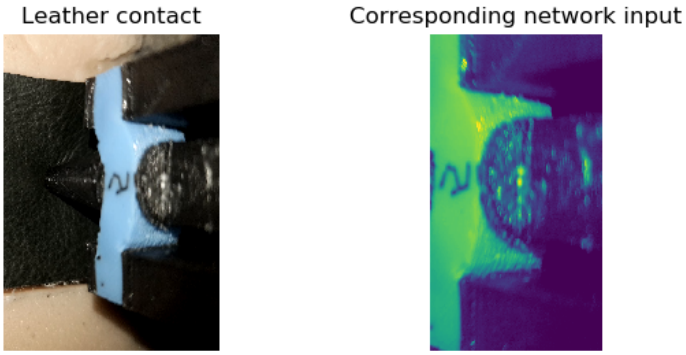


Figure 3.10: Testing force estimation in contact with a leather strip (laid over a phantom).

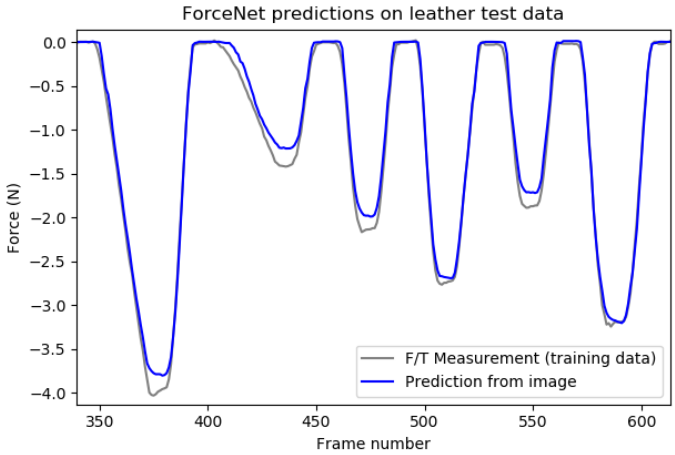


Figure 3.11: Performance of the ForceNet on a test dataset containing touches with leather, overlaid on a silicone phantom. These touches are otherwise similar to those carried out on the rigid force plate for training. Note the tendency to underestimate the force magnitude.

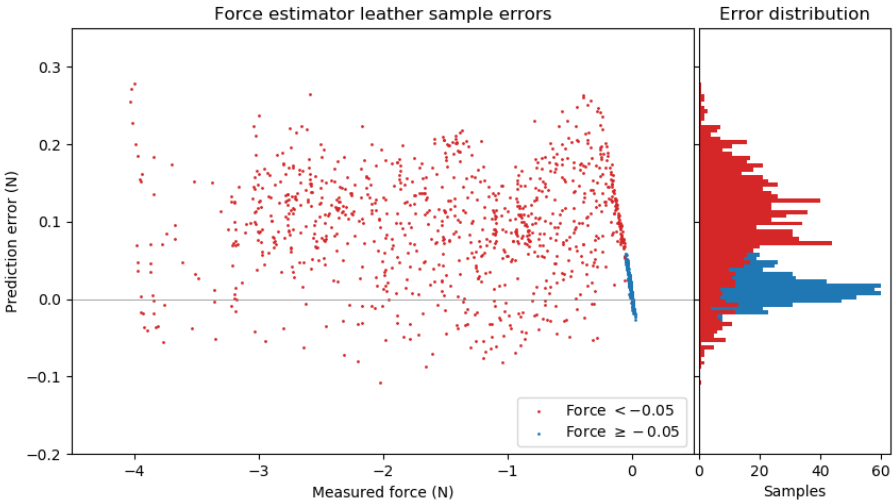


Figure 3.12: Contact force errors, with distribution, in the leather strip test. We separate estimates at near-zero measured force as before, highlighting the relatively poor (and skewed) results at higher forces in this case.

underestimate the magnitude of the applied force. We show the error distribution for the leather dataset in Figure 3.12.

We speculate that this may be due to differences in cast light from the leather altering the pattern of features detected by the network, when compared to the training data using the lighter rigid force plate. As the rigid plate is in fact designed to create different levels of cast light, this demonstrates another shortcoming of the plate's design. The plate's surface is fairly diffuse, so it scatters the light from the smartphone's flashlight back across the probe and flexure, while the leather absorbs or specularly reflects the light.

Remaining with the theme of robustness to lighting conditions, we captured an experimental data set in direct sunlight (Figure 3.13). These data represent a significant departure from those seen in the training set, as the sunlight is bright enough to overwhelm the phone's flashlight on the contact surface (the same leather used in the previous test). As such, the network makes significant errors in its prediction, but interestingly is not completely thrown off – a strong signal remains

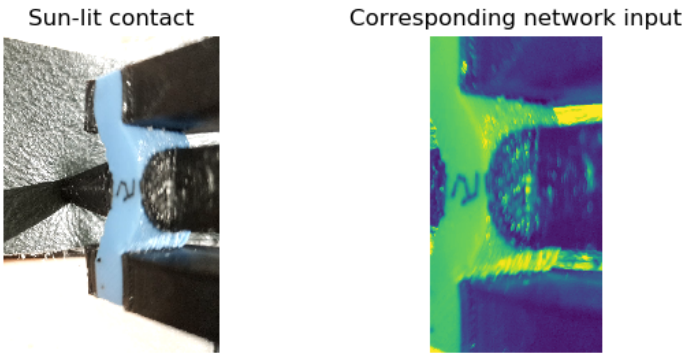


Figure 3.13: Testing force estimation *in direct sunlight*, in contact with leather.

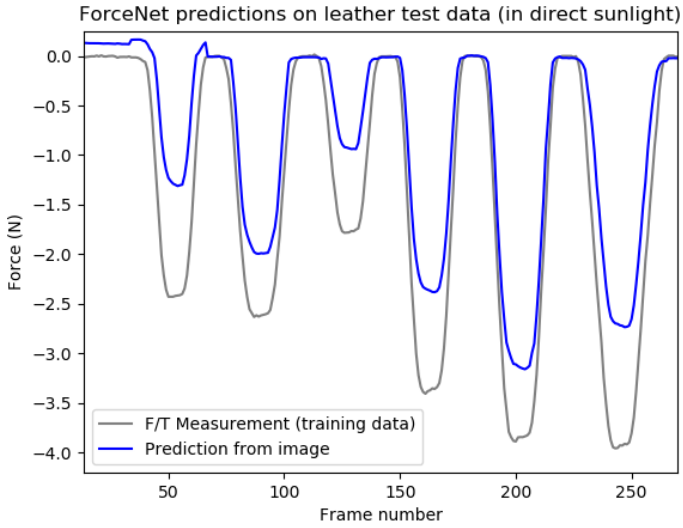


Figure 3.14: Performance of the ForceNet on a test dataset containing touches with leather *in direct sunlight*. We do not expect (and do not see) accurate results here, but a strong signal remains.

Figure 3.15: Moving the probe by hand in its mount.

(Figure 3.14), and the predictions are stable from frame to frame, if not particularly accurate. That said, the MAE of 0.445 N is unacceptable for real-world usage. Further work, likely including new force training data, would be needed for the system to function in the presence of such a bright light source.

Handheld Usage

As discussed above, types of motion outside the training data's range can present a challenge for the force predictor, and so handheld motion of the probe could prove difficult for the force predictor to follow. Here, we captured datasets comprising assorted hand-guided and fully handheld touches on the silicone phantoms.

The first dataset has the probe mounted on the automation rig, but with the motors disabled. All motion of the probe is created directly by hand, using a handle on the probe's base, but the rig holds the *orientation* of the probe perfectly steady. We made several touches on different phantoms and at various force levels, up to approximately 4 N, and sometimes retracting and re-inserting the probe tip during a touch (a type of motion not seen in training data).

Results here were promising (Figure 3.16), with metrics very comparable to robotic touches on the same phantoms. While predictions worsened when the probe remained in contact with the phantoms, predictions during the compression phase

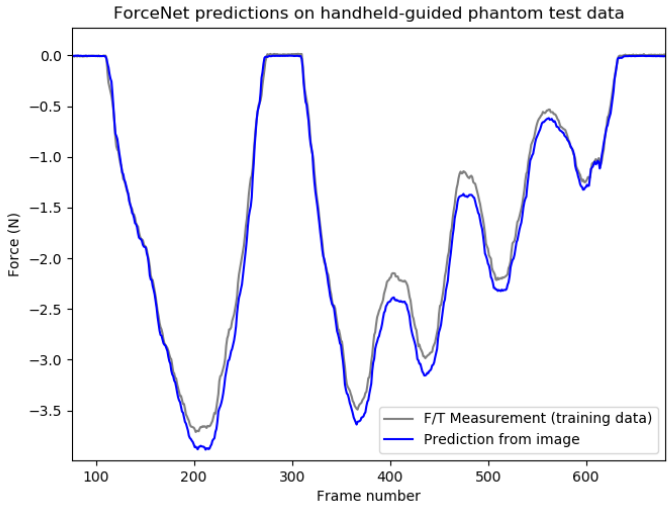


Figure 3.16: ForceNet predictions for phantom touches, made by hand but with the probe mounted in the automation rig (with the motors disabled).

of each touch were excellent, and it is these predictions which are critical for the material estimation process. We suspect this degradation in performance during touches is due to unaccounted hysteresis in the flexure itself, and could potentially be accounted for with a more advanced model – one which takes into account the flexure's dynamics. This could be a similar model with multiple image frame inputs, or even a "tacked-on" post-processing step accepting consecutive force estimates, and outputting a revised, dynamics-aware estimate.

We followed up by detaching the probe from its mounting entirely, and recorded freely handheld touches, intended to resemble material estimation touches by an experimenter: slow-to-medium contact speed, up to approximately 3 N. In this scenario, the flexure and predictor network also performed very well (Figure 3.17), with an MAE of 0.0426 N; no worse than in robotic testing. The only meaningful failure in this test is the system's obvious inability to detect negative (or, strictly speaking, *positive*) normal forces; as with other shortcomings we have observed, it is not trained to do this, as the rigid plate does not display the 00-10 silicone's distinctive tackiness. We reason this shortcoming is unimportant for our use case,

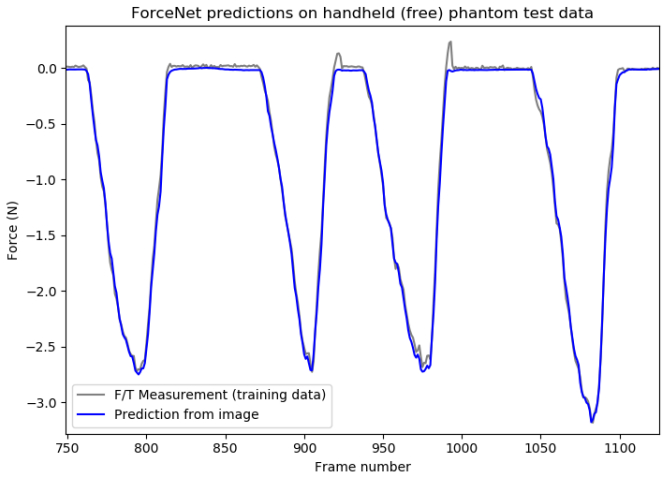


Figure 3.17: ForceNet predictions for phantom touches, made fully by hand – the probe was not mounted or in any way attached to the automation rig, though the phantom tray was (in order to detect forces).

as we are interested in material behavior under compression, rather than these brief periods of tension.

Post-Stress Test Data

While we account for slight shifts in the mounting position of the flexure using data augmentation, the force prediction process does not account for plastic deformation ("settling") of the flexure after extended use. The data set in Figure 3.18 was captured on the rigid plate after approximately 6,000 touches at relatively high force levels, up to 7N. While the zero-point has not majorly shifted, the network significantly over-predicts forces at higher levels of deflection, sometimes overestimating by a full Newton. Overall accuracy reduced to an MAE of 0.229 N, over an order-of-magnitude decrease in performance.

Currently the only way to solve this problem in actual usage is to re-train with a more recent data set, which yields results at the normal level of accuracy, returning to an MAE of 0.0409 N. However, in Figure 3.19, we show that the error is approximately a function of the estimated force. It could therefore be feasibly corrected

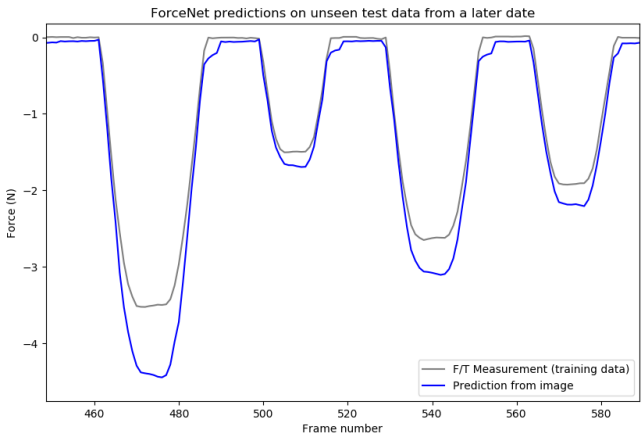


Figure 3.18: Performance of the ForceNet on an unseen test dataset captured after thousands of post-training touches, recorded in the same way as force training and validation data. The net was not trained or validated on this data.

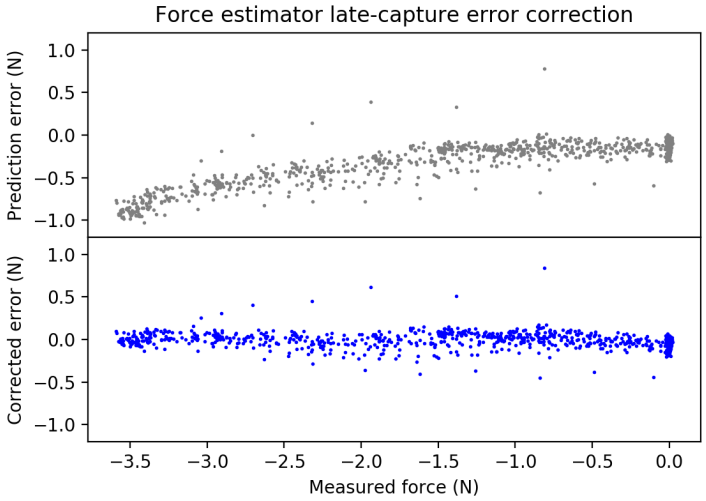


Figure 3.19: Force network residuals on the post-stress dataset, with and without a polynomial correction. Raw prediction errors are shown above in gray, while corrected errors are shown below in blue.

using a simple calibration process, transforming the network output to match reality using a small dataset. We optimized a fourth-degree polynomial correction on the output of the force model, and show that these "corrected" errors are only marginally worse than a retrained network. With a corrected MAE of 0.0492 N, this correction may be sufficiently accurate for our methods – though this metric, taken alone, could display some over-fitting of the correction to this particular case.

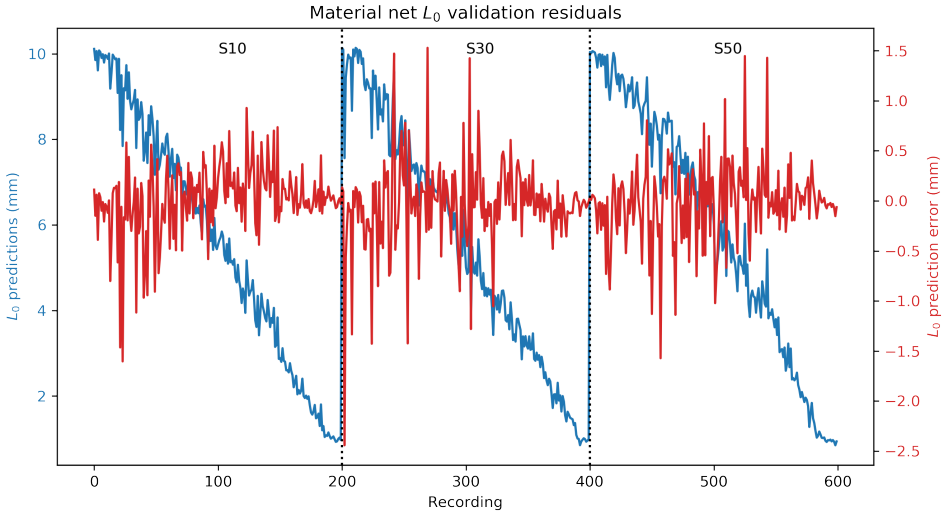
We believe the problem of mechanical changes to the flexure during extended use may indeed be solvable with a calibration process, which would remove the need for large-scale force data collection for re-training. In any case, this problem only occurs after the flexure is exposed to force levels outside its range of use – we did not observe this problem when limiting touches to $3-4$ N, even after thousands of touches.

3.2 Material Estimation

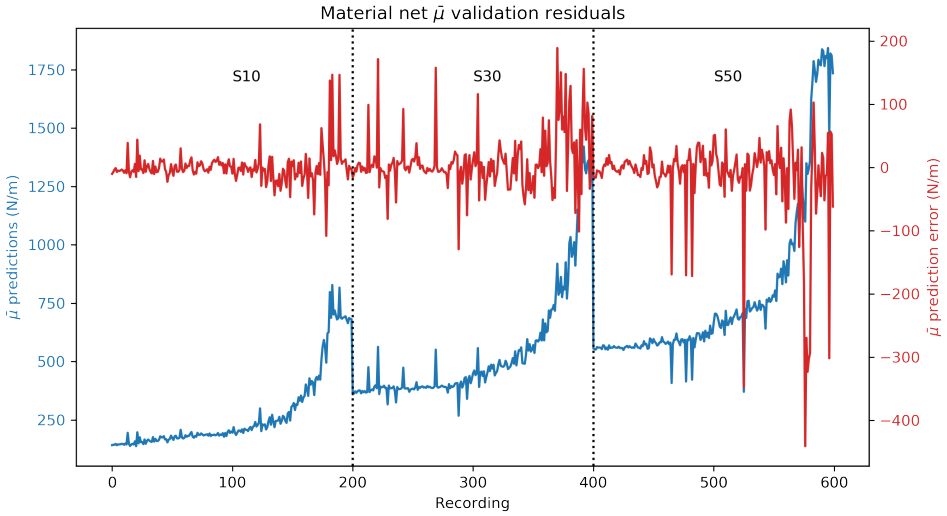
3.2.1 Validation and Test Data

Here we examine the accuracy of the material network's estimates, beginning with the training and validation data performance. Recall that robotically-collected material data takes the form of a scan of hundreds to thousands of touches across each of the three material phantoms, in order of increasing stiffness. Each full touch is a single data-point for the material network, resolving to a single flow input. For each touch, the material network provides a material estimate vector containing $\bar{\mu}$ and L_0 predictions. Touches take place left-to-right on the phantoms, going from the deep towards the shallow side of the phantoms, are randomly spaced with a uniform distribution, and randomly offset in the Y (up/down) direction. Refer to Section 2.2.3 for additional details on the material data collection.

The material network performs well on its validation data, though notably worse than on its training data, demonstrating the overfitting common to many DNNs. Results are somewhat noisy for both dimensions of material property estimates, but the underlying signal is very clear, and the network rarely makes significant mis-predictions on this data. L_0 predictions are shown in Figure 3.20a, and $\bar{\mu}$ predictions in Figure 3.20b. Table 3.2 provides useful metrics for the per-



(a) L_0 results



(b) $\bar{\mu}$ results

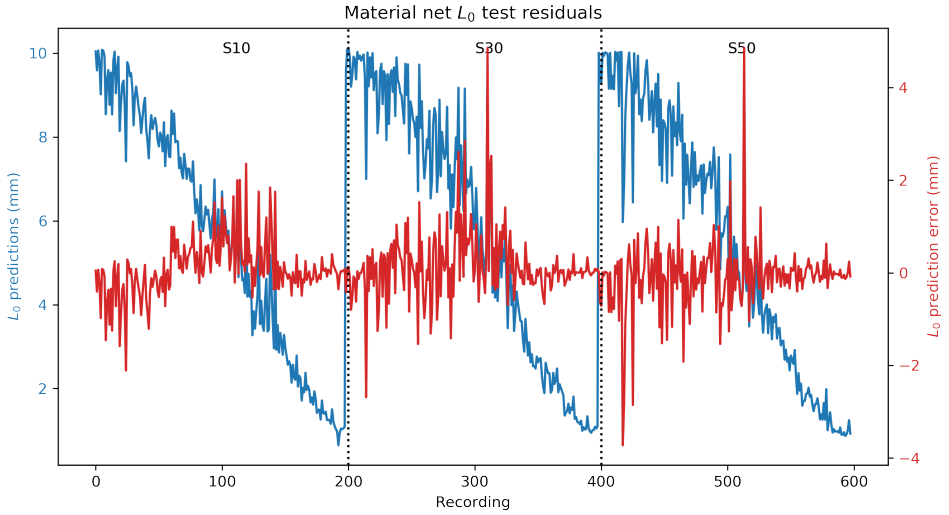
Figure 3.20: Material network performance on its own validation set, predicting a) L_0 material depth and b) $\bar{\mu}$ surface stiffness. Estimates are shown in blue, and corresponding estimate error (to ground truth) is shown in red.

	MAE	RMSE	R^2
Training			
L_0	0.050 mm	0.064 mm	0.9995
$\bar{\mu}$	6.909 N/m	9.035 N/m	0.9994
Validation			
L_0	0.259 mm	0.389 mm	0.9816
$\bar{\mu}$	25.468 N/m	51.601 N/m	0.9782
Test			
L_0	0.438 mm	0.706 mm	0.9447
$\bar{\mu}$	35.793 N/m	69.765 N/m	0.9662

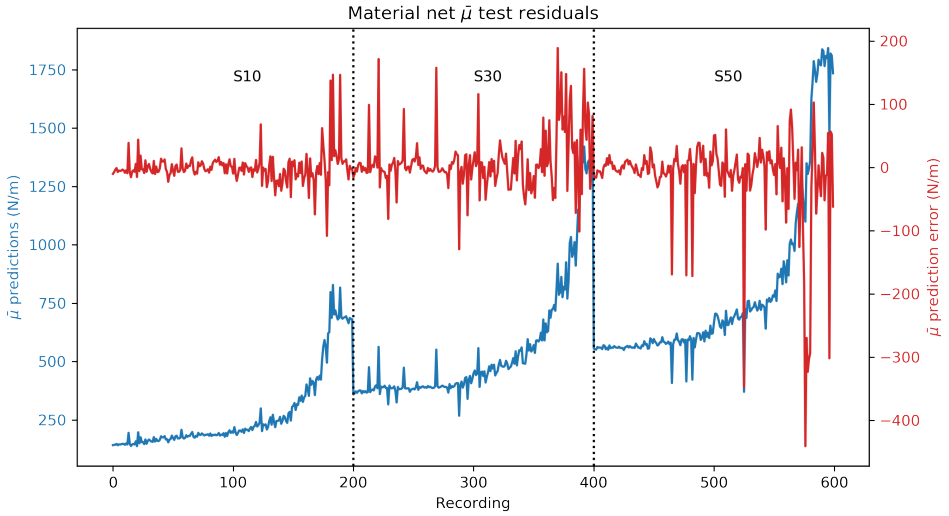
Table 3.2: Material estimator MAE, RMSE and R^2 metrics for the two different material properties. Recall that L_0 measures thickness, and $\bar{\mu}$ measures stiffness. Note that training error is, by several measures, significantly lower than validation error, which is itself lower than the test error.

formance on both properties. Promisingly, we can see that most L_0 predictions fall within half a millimeter of the true value, and that the handful of outliers fall within 1.5mm. The majority of $\bar{\mu}$ predictions here are also excellent, but the few outliers are fairly pronounced, and there are some regions where predictions are incorrectly skewed.

Testing with data unseen by the network during training shows performance which is worse again than the validation performance (Figure 3.21), but still promising. In this case, most L_0 predictions fall within two millimeters of the true value – slightly, but not majorly worse than on validation data – and we once again observe a handful of more extreme outliers. The majority of $\bar{\mu}$ predictions are again excellent, but we see significant outliers and some regions where predictions are incorrectly skewed – the predictions seem to be particularly bad around the point the phantom flattens out at 1mm, where the flexure could simply be getting deflected down the slope unpredictably. L_0 results seem to be most accurate in the final thin section of each phantom, while the opposite appears to be true for the $\bar{\mu}$ predictions.



(a) L_0 results



(b) $\bar{\mu}$ results

Figure 3.21: Material network performance on an unseen test dataset, predicting a) L_0 material depth and b) $\bar{\mu}$ surface stiffness. Estimates are shown in blue, and corresponding estimate error (to ground truth) is shown in red.

(a) Front

(b) Top view, showing flexure orientation

Figure 3.22: The dismounted prototype, ready for freehand usage.

3.2.2 Handheld Comparison Experiment

We set up an experiment to analyze the probe's true performance on the phantoms, and to help eliminate variables as sources of error. We collected three datasets, with different methodologies:

- **Handheld-Guided:** The probe was left mounted in the automation rig, but the robot's motors were disabled (near-zero resistance), and the probe was held and moved by the experimenter. This kept the probe constrained to a fixed orientation, but left motion in the three cartesian axes to the experimenter's hand movements.
- **Handheld:** The probe was completely detached from the automation rig, and used freehand (Figure 3.22). The phantom tray was also detached, and placed flat on a table (not in the standard "wall-mounted" orientation). All motion of the probe was left to the experimenter, including rotation, though the experimenter attempted to maintain a perpendicular angle of attack, as before.
- **Robotic:** The probe was mounted and robotically articulated in much the same way as was used to generate training and validation data. This was meant to serve as a "control."

Each set of three "locations" corresponds to a different phantom – 00-10, 00-30 and 00-50 from left to right in the figures. We sampled each phantom with 5 repeats at each of 3 locations – on either end, at the extremes of thickness, and

	MAE	RMSE	R^2
Robotic			
L_0	0.316 mm	0.481 mm	0.9829
$\bar{\mu}$	34.524 N/m	65.161 N/m	0.9856
Guided			
L_0	0.377 mm	0.627 mm	0.9709
$\bar{\mu}$	101.470 N/m	232.215 N/m	0.8165
Handheld			
L_0	1.179 mm	1.713 mm	0.7827
$\bar{\mu}$	217.282 N/m	359.104 N/m	0.5612

Table 3.3: Material estimator MAE, RMSE and R^2 metrics for L_0 and $\bar{\mu}$. Note that these metrics represent only 45 samples per dataset, far fewer than the other test results discussed above.

in the center. We chose this coarse sampling to enable us to target touches with reasonable accuracy with the probe handheld.

The material estimates for these datasets (Figure 3.23) followed a general trend of decreasing accuracy from robotic, to guided, to handheld results (Table 3.3). The robotic measurement accuracy was comparable to the previously-discussed test results for the material estimator, and we observe robotic estimates generally clustered around the ground truth values for both material properties. Estimates on the handheld-guided dataset were also fairly accurate – while performance in the $\bar{\mu}$ dimension suffered, predictions for tissue thickness were only slightly worse than those with the robotically-captured data. Estimates with the freely handheld dataset were notably less accurate than either dataset with the probe mounted, with worse performance in both parameter dimensions.

There are several possible explanations for the poorer results with hand-guided and fully handheld readings: these are necessarily limited to differences between the test data and the training data, and corresponding differences in downstream results, including the force predictions and flows. We have established that the force predictions remain accurate even as the probe is manipulated by hand, and the phantoms contacted in these tests have not changed from those the material estimator was trained on – so, we suspect differences in the probe's *motion* between

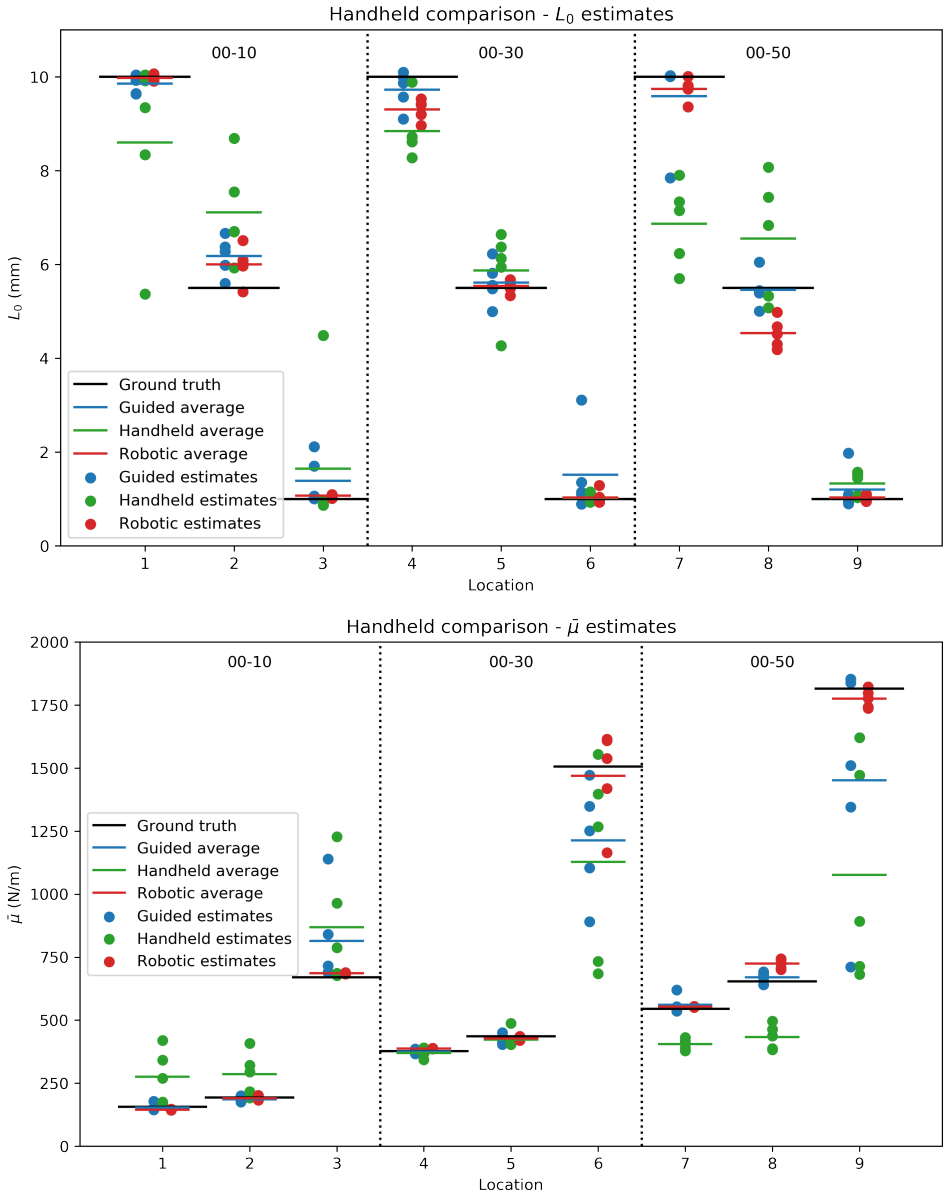


Figure 3.23: Material estimator results for robotically- and manually-captured datasets.

the training and test data to be the cause of the mis-predictions seen here.

As with any handheld object, handheld articulation of the probe can create small "tremor" motions, which the material network does not see in training – while we attempt to account for this in training by dynamically moving the probe while in contact with the phantoms, we may not have accounted for all possible variations. Another possible difference is that the speed of a handheld touch may vary through the duration of the touch, while in the robotic touches seen in training, the speed is consistent throughout each touch. When fully handheld, dynamic rotations may also occur relative to the phantom surface; rotation is not present in the training data, robotic data, or in the handheld-guided data, as the probe's orientation is fixed: On its mounting, the probe moves only in the 3 cartesian axes permitted by our robot.

In this case, experimenter inaccuracy may be an additional cause of some amount of the error seen in these graphs. Small differences in the location of each touch on the phantoms could lead to measurement of substantially different material properties, as both measured properties can change rapidly across each phantom. This experiment assumes that the experimenter has contacted the phantoms accurately in each touch, at the designated positions. This is necessary for the handheld data, as we have no simple way of measuring the probe's position once it is removed from its mounting. However, this leaves room for human error, particularly at locations 1, 4, and 7 in the middles of the phantoms.

Interestingly, L_0 predictions appear to be best at the shallower end of each phantom, where $\bar{\mu}$ predictions are worst. This may be an issue of differentiation in the data – visually, there is little difference between pressing into a very thin strip of soft silicone, and pressing into the same depth of stiffer silicone; what *is* obvious is that both strips are thin. The $\bar{\mu}$ estimates for the thin end of each phantom appear to jump between the 00-10, 00-30 and 00-50 stiffness values for 1mm thickness, demonstrating both that the network has over-fit to the particular properties of the three phantoms, and that there remains uncertainty even in differentiating just these three cases. The material estimator also seems to struggle with the deeper L_0 predictions, especially for 00-50: in this stiffer phantom, there may be less surface-level behavioral changes due to increasing depth than for the softer phantoms, particularly the 00-10.

Chapter 4

Conclusions

4.1 Contributions

Our primary contributions are the hardware and software making up our system:

- We built a low-cost prototype probe device capable of accurate force sensing from purely optical data, and limited detection of material properties in contacted surfaces.
- We developed application software for the probe's smartphone, which controls camera parameters, communicates with cloud services, and provides both remote-control capabilities and a user-operated graphical user interface.
- We built a robotic system for automated capture of training data for our estimators, developed control software for this platform permitting custom user programs, and developed cloud storage clients for storing and retrieving this data in useful formats.
- We deployed a cloud pipeline for rapid, on-demand estimation and storage of forces, optical flows, and material data for one or more probe devices, using GPU-accelerated machine learning models.
- We characterized the performance of our prototype through a series of laboratory tests.

To summarize, we believe the notable contributions of this project include the use of a smartphone and "flexure" to estimate forces and material properties with only optical data, the cloud pipeline for estimating those results, and the robotic system used to help train the estimators.

4.2 Project Goals

We set out to create a system with these targets in mind:

- Low cost, both to produce and to use
- Portable
- Usable with little training
- Rapid data collection
- Accurate, useful results

Our device is certainly low-cost, especially when considered in the context of the SkinProbe V1. Building a new probe requires a one-off investment in a dual-extrusion 3D printer, which is a professional, but relatively affordable device costing in the region of $4000. Training estimators for a new probe requires a 3-axis robot and force sensor, together costing no more than $10,000. Training also requires access to a GPU-equipped workstation or cloud computer. The incremental cost of a SkinProbe 2.0 device is then no more than $1000, the bulk of which is the cost of a new smartphone. This compares to an initial cost of up to several hundred thousand dollars for the V1 probe, which, unlike our prototype, could not be duplicated for use in parallel – a new host computer and motion capture setup would be required at each measurement facility.

Using the device for experiments with live feedback imposes a cloud hosting cost of approximately $5 per hour, which pays for GPU instance rental. This cost could be reduced by using non-GPU instances, or lower-tier GPUs for inference, though estimation speeds are likely to suffer as a result. Note that this cost could be effectively reduced if multiple probes were in use at once – a single cloud instance, with a fixed hourly price could support multiple "client" probes.

Use of a smartphone makes our system both user-friendly and portable. Our current phone application is usable for the limited trials we have run, with very straightforward touchscreen controls – essentially just two buttons for experimental data capture, which any smartphone user should be comfortable with. This software could also be easily expanded to serve many additional roles, changed to be made more user-friendly (for example, by the addition of in-app tutorials), or even localized to different local languages. Carrying out touches is not demanding, simply requiring a steady hand. The probe's two-handed grip makes manipulating it easy.

We support experimental data collection with only an active internet connection, making use on mobile data networks possible – an experimenter would be able to travel to other locations (or countries!) to gather data with human subjects, solving one of the major drawbacks of the V1 probe. There remain some limitations around working environment: we have seen that force estimation fails in direct sunlight, and we can reason that strong shadows cast on a participant's skin would interfere with optical flow generation. For these reasons, our system can likely only be used effectively in an indoor environment, without strong direct lighting (e.g. from floodlights). That said, there are no other constraints on the ambient lighting conditions – our use of the phone's flashlight ensures consistent illumination, so long as the flashlight source is not overwhelmed.

Collection of data is indeed rapid, though the response time for the cloud-based generation of results is not ideal, typically taking $\approx 10\,\mathrm{s}$. While new recordings can be made during processing (data collection need not be stopped), real-time feedback would allow experimenters to know if their experimental collection is working as they go. The majority of this processing time is spent uploading, downloading, and unpacking the video files, and this time could be significantly reduced by reducing the size of the videos. For this prototype, we took a highly conservative approach to video compression, using high-bitrate, high-quality encodings to decrease the possibility of compression artifacts tampering with our results.

The final, and perhaps most important goal is accuracy. Our force-estimation pipeline works well in practice, including when handheld and in contact with novel materials – though performance does degrade in some cases, we believe these predictions are still accurate enough to be useful, and hope that the minor remaining

issues with the force estimation are solvable with further work. Processing images to estimated forces is fast, and the network is small and simple enough that, in the future, it could feasibly be run on a mobile device, providing instant feedback to experimenters.

However, in this project, the force estimations are only a means to an end; our material estimation results are, for now, not reliable enough to be useful in practice, and our system is not yet usable on humans. This is clearly a significant drawback, but we do not see this challenge as insurmountable: merely deserving of further research and development on the probe's software, primarily its estimation algorithms. Transitioning to use on human subjects may also require further development of the training phantoms, use of human data for training, or both.

4.3 Future Work

As this is a prototype system, there are many aspects of our work that could be improved upon or expanded.

4.3.1 Force Sensing

The probe's flexure deflects in more than just the normal direction – our intention was to support force estimation in three dimensions, and we believe this should be feasible. Supporting this would be a matter of expanding the force estimation model to a three-dimensional output, verifying the results, and scaling each dimension of the network output targets appropriately to ensure suitable levels of accuracy in all three axes. This would also likely require replacing the rigid force plate with a grippy surface, allowing significant lateral forces to be applied during training.

Another notable drawback of our force sensing system is its lack of awareness of the flexure's dynamics from frame to frame. Improving the force estimation model to account for these dynamics, including hysteresis, would allow the accuracy of our force estimates to be further improved, especially in cases of sustained contact with the subject. This could be achieved with a time-sequence model, such as an LSTM, or even with a simpler model working as a post-process on a sequence of single-frame estimates.

Robustness could be improved by adding darker patches to the rigid plate, tackling the problem seen in the leather contact test, or potentially by using a more traditional pre-processing pass on the flexure images to negate the effects of different lighting conditions. There may also be some benefits to adding high-contrast, "trackable" features to the flexure shaft and frame, which the network (or a more traditional computer vision pipeline) would be able to pick out more robustly.

4.3.2 Material Estimation

Our current material estimation system has several limitations, not least its lower accuracy in handheld use. Our estimates tend to draw from the observed distribution of phantoms seen in training, regardless of the actual properties of the material being tested – we see this in the tendency for estimates to "jump" between the groupings for the different silicone types. This clearly is of little use for analyzing the properties of human soft tissues, and so we expect making this model generalize may require a significantly wider variety of phantom data for training, including phantoms designed to better approximate human tissue. One approach for gathering this data could be to generate it synthetically, covering a very wide range of material properties, with the use of optical flows and force data as inputs serving to remove the need for photorealistic image rendering – flows from a roughly textured skin rendering should be very similar to the real-world equivalent. Another would be to construct a wider variety of physical phantoms, training with these, and potentially fine-tuning with data captured from humans. Capturing human data would require using either the V1 probe, or a hybrid motion-tracked V2 probe to establish ground-truth measurements.

We would also suggest expanding the range of material properties estimated. Material force-displacement behavior could be parameterized in far greater detail than our basic stiffness and thickness model. Terms could including volume preservation and anisotropic properties.

4.3.3 Human Trials

SkinProbe 2.0 is ultimately intended to be used on humans, and so this is a major direction for future work. Building new probes, and bringing them into the field for

human trials would allow data collection at a large scale, for application in diverse fields such as character animation, medicine, and clothing and prosthesis design and customization. New probes could be equipped with mobile data SIM cards for use away from dedicated research facilities.

4.3.4 Cloud Pipeline

As mentioned above, generation of estimation results is hardly real-time: there is significant room for improvement in the *speed* of our cloud pipeline. Reducing the file size of our videos would be a shortcut to reducing these processing times, and it is likely that lower-bitrate video encodings would still work well, especially if our estimation networks were trained using videos encoded in the same way. If our current bitrate of 20-30,000 Kbps could be successfully reduced to a more typical rate of 2-4,000 Kbps for our 1024×768 resolution, we could see $5-10\times$ speed increases in this transfer and unpacking portion of the results generation. There are also other straightforward optimizations which could be made: by unpacking videos in-memory, we could perform force estimation directly on image frames as they are decoded, rather than waiting for all frames to be written to disc before reading them back into memory one-by-one. Developing high-speed video upload direct to the cloud instance would remove the interim step of S3 upload and download; we would then be able to transfer videos to S3 from the instance, ensuring the same level of data retention as before. Better still, if the phone could *stream* video to the instance, frames could be processed for force estimates even before the recording was completed. At that point, frame selection and material estimation could be completed even before the experimenter retracts the probe!

The cost of operating the cloud pipeline could be reduced by switching to instances with less-powerful, or no GPUs, though this could present technical challenges for running the FlowNet2 model.

4.4 Final Thoughts

This prototype is our first step towards large-scale measurement of human tissue properties. Our system is not yet ready for prime time, but we have demonstrated functionality in several key areas, and developed robust, low-cost hardware ready

for further software improvements. On the technical side, we hope this work inspires other researchers to use smartphones, optical force sensing, and cloud computing in their own projects.

Bibliography

[1] V. Belagiannis, C. Rupprecht, G. Carneiro, and N. Navab. Robust optimization for deep regression. In *Proceedings of IEEE International Conference on Computer Vision*, December 2015. → page 10

[2] B. Bickel, M. Bächer, M. A. Otaduy, W. Matusik, H. Pfister, and M. Gross. Capture and modeling of non-linear heterogeneous soft tissue. *ACM Transactions on Graphics*, 28(3):89:1–89:9, July 2009. ISSN 0730-0301. doi:10.1145/1531326.1531395. URL http://doi.acm.org/10.1145/1531326.1531395. → page 13

[3] Y.-L. Boureau, J. Ponce, and Y. LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of 27th International Conference on Machine Learning*, pages 111–118. ICML, 2010. → page 28

[4] R. O. Bude and R. S. Adler. An easily made, low-cost, tissue-like ultrasound phantom material. *Journal of Clinical Ultrasound*, 23(4):271–273, 1995. doi:10.1002/jcu.1870230413. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/jcu.1870230413. → page 14

[5] J. J. Clark. A magnetic field based compliance matching sensor for high resolution, high compliance tactile sensing. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 772–777 vol.2, April 1988. doi:10.1109/ROBOT.1988.12152. → page 11

[6] S. Derler, U. Schrade, and L.-C. Gerhardt. Tribology of human skin and mechanical skin equivalents in contact with textiles. *Wear*, 263(7):1112 – 1116, 2007. ISSN 0043-1648. doi:https://doi.org/10.1016/j.wear.2006.11.031. URL http://www.sciencedirect.com/science/article/pii/S0043164807003535. 16th International Conference on Wear of Materials. → page 15

[7] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information*

Processing Systems 27, pages 2366–2374. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5539-depth-map-prediction-from-a-single-image-using-a-multi-scale-deep-network.pdf. → page 11

[8] Y. Fan, J. Litven, and D. K. Pai. Active volumetric musculoskeletal systems. *ACM Transactions on Graphics*, 33(4):152:1–152:9, July 2014. ISSN 0730-0301. doi:10.1145/2601097.2601215. URL http://doi.acm.org/10.1145/2601097.2601215. → page 12

[9] S. Glassenberg and M. Yaeger. Gastro ex: real-time interactive fluids and soft tissues on mobile and vr. In *ACM SIGGRAPH 2018 Real-Time Live!*, page 3. ACM, 2018. → page 12

[10] X. Guo, Y. Huang, X. Cai, C. Liu, and P. Liu. Capacitive wearable tactile sensor based on smart textile substrate with carbon black/silicone rubber composite dielectric. *Measurement Science and Technology*, 27(4):045105, 2016. → page 11

[11] T. J. Hall, M. Bilgen, M. F. Insana, and T. A. Krouskop. Phantom materials for elastography. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 44(6):1355–1365, Nov 1997. ISSN 0885-3010. doi:10.1109/58.656639. → page 14

[12] D. Held, S. Thrun, and S. Savarese. Learning to track at 100 fps with deep regression networks. In *Proceedings of European Conference on Computer Vision*, pages 749–765. Springer, 2016. → page 11

[13] S. Hirose and K. Yoneda. Development of optical six-axial force sensor and its signal calibration considering nonlinear interference. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 46–53 vol.1, May 1990. doi:10.1109/ROBOT.1990.125944. → page 11

[14] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2462–2470, 2017. → pages 7, 39

[15] M. Kim, G. Pons-Moll, S. Pujades, S. Bang, J. Kim, M. J. Black, and S.-H. Lee. Data-driven physics for human soft tissue animation. *ACM Transactions on Graphics*, 36(4):54:1–54:12, July 2017. ISSN 0730-0301. doi:10.1145/3072959.3073685. URL http://doi.acm.org/10.1145/3072959.3073685. → page 13

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. → page 37

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. → page 10

[18] T. Kroeger, R. Timofte, D. Dai, and L. Van Gool. Fast optical flow using dense inverse search. In *Proceedings of European Conference on Computer Vision*, pages 471–488, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46493-0. → page 38

[19] P. G. Kry and D. K. Pai. Interaction capture and synthesis. *ACM Transactions on Graphics*, 25(3):872–880, July 2006. ISSN 0730-0301. doi:10.1145/1141911.1141969. URL http://doi.acm.org/10.1145/1141911.1141969. → page 13

[20] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998. → page 28

[21] C. Ledermann, S. Wirges, D. Oertel, M. Mende, and H. Woern. Tactile sensor on a magnetic basis using novel 3d hall sensor - first prototypes and results. In *Proceedings of IEEE International Conference on Intelligent Engineering Systems*, pages 55–60, June 2013. doi:10.1109/INES.2013.6632782. → page 11

[22] S.-H. Lee, E. Sifakis, and D. Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Transactions on Graphics*, 28(4):99:1–99:17, Sept. 2009. ISSN 0730-0301. doi:10.1145/1559755.1559756. URL http://doi.acm.org/10.1145/1559755.1559756. → page 13

[23] Z. Li, T. Dekel, F. Cole, R. Tucker, N. Snavely, C. Liu, and W. T. Freeman. Learning the depths of moving people by watching frozen people. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 2019. → page 11

[24] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 2015. → page 10

[25] C. Luo, X. Chu, and A. Yuille. Orinet: A fully convolutional network for 3d human pose estimation. In *Proceedings of British Machine Vision Conference*, 2018. → page 11

[26] K. E. MacLean. The 'haptic camera': A technique for characterizing and playing back haptic properties of real environments. *Proceedings of Haptic Interfaces for Virtual Environments and Teleoperator Systems*, pages 459–467, 1996. → page 14

[27] L. Markley. Attitude determination using vector observations and the singular value decomposition. *Journal of the Astronautical Sciences*, 38: 245–258, 11 1987. → page 51

[28] E. Miguel, D. Miraut, and M. A. Otaduy. Modeling and estimation of energy-based hyperelastic objects. *Computer Graphics Forum*, 35(2): 385–396, 2016. doi:10.1111/cgf.12840. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12840. → page 13

[29] B. J. Nelson, J. D. Morrow, and P. K. Khosla. Improved force control through visual servoing. In *Proceedings of American Control Conference*, volume 1, pages 380–386 vol.1, June 1995. doi:10.1109/ACC.1995.529274. → page 12

[30] L. Nicholas, K. Toren, J. Bingham, and J. Marquart. Simulation in dermatologic surgery: A new paradigm in training. *Dermatologic Surgery*, 39(1pt1):76–81, 2013. doi:10.1111/dsu.12032. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/dsu.12032. → page 14

[31] D. K. Pai. Robotics in reality-based modeling. In *Robotics Research*, pages 353–358, London, 2000. Springer London. ISBN 978-1-4471-0765-1. → page 15

[32] D. K. Pai and P. Rizun. The what: A wireless haptic texture sensor. In *Proceedings of Haptic Interfaces for Virtual Environments and Teleoperator Systems*, pages 3–9. IEEE, 2003. → page 14

[33] D. K. Pai, K. van den Doel, D. L. James, J. Lang, J. E. Lloyd, J. L. Richmond, and S. H. Yau. Scanning physical interaction behavior of 3d objects. In *Proceedings of ACM SIGGRAPH*, pages 87–96. ACM, 2001. → page 13

[34] D. K. Pai, A. Rothwell, P. Wyder-Hodge, A. Wick, Y. Fan, E. Larionov, D. Harrison, D. R. Neog, and C. Shing. The human touch: Measuring

contact with real human soft tissues. *ACM Transactions on Graphics*, 37(4): 58, 2018. → pages v, xi, 13, 15, 16, 17, 18

[35] J. Peirs, J. Clijnen, D. Reynaerts, H. V. Brussel, P. Herijgers, B. Corteville, and S. Boone. A micro optical force sensor for force feedback during minimally invasive robotic surgery. *Sensors and Actuators A: Physical*, 115 (2):447 – 455, 2004. ISSN 0924-4247. doi:https://doi.org/10.1016/j.sna.2004.04.057. URL http://www.sciencedirect.com/science/article/pii/S0924424704003917. The 17th European Conference on Solid-State Transducers. → page 11

[36] G. Pons-Moll, J. Romero, N. Mahmood, and M. J. Black. Dyna: A model of dynamic human shape in motion. *ACM Transactions on Graphics*, 34(4): 120:1–120:14, July 2015. ISSN 0730-0301. doi:10.1145/2766993. URL http://doi.acm.org/10.1145/2766993. → page 13

[37] P. Puangmali, H. Liu, L. D. Seneviratne, P. Dasgupta, and K. Althoefer. Miniature 3-axis distal force sensor for minimally invasive surgical palpation. *IEEE/ASME Transactions on Mechatronics*, 17(4):646–656, Aug 2012. ISSN 1083-4435. doi:10.1109/TMECH.2011.2116033. → page 11

[38] A. Sekhar, M. R. Sun, and B. Siewert. A tissue phantom model for training residents in ultrasound-guided liver biopsy. *Academic radiology*, 21(7): 902–908, 2014. → page 14

[39] W. Si, S.-H. Lee, E. Sifakis, and D. Terzopoulos. Realistic biomechanical simulation and control of human swimming. *ACM Transactions on Graphics*, 34(1):10:1–10:15, Dec. 2014. ISSN 0730-0301. doi:10.1145/2626346. URL http://doi.acm.org/10.1145/2626346. → page 12

[40] E. Sifakis and J. Barbič. Finite element method simulation of 3d deformable solids. *Synthesis Lectures on Visual Computing: Computer Graphics, Animation, Computational Photography, and Imaging*, 1(1):1–69, 2015. → page 12

[41] S. Sueda, A. Kaufman, and D. K. Pai. Musculotendon simulation for hand animation. *ACM Transactions on Graphics*, 27(3):83:1–83:8, Aug. 2008. ISSN 0730-0301. doi:10.1145/1360612.1360682. URL http://doi.acm.org/10.1145/1360612.1360682. → page 12

[42] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. *CoRR*, abs/1707.02968, 2017. URL http://arxiv.org/abs/1707.02968. → page 31

[43] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems 26*, pages 2553–2561. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf. → page 11

[44] J. Teran, E. Sifakis, S. S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics*, 11(3): 317–328, May 2005. doi:10.1109/TVCG.2005.42. → page 13

[45] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660. IEEE, 2014. → page 10

[46] B. Wang, L. Wu, K. Yin, U. Ascher, L. Liu, and H. Huang. Deformation capture and modeling of soft objects. *ACM Transactions on Graphics*, 34 (4):94:1–94:12, July 2015. ISSN 0730-0301. doi:10.1145/2766911. URL http://doi.acm.org/10.1145/2766911. → page 13

[47] H. Wang, G. De Boer, J. Kow, A. Alazmani, M. Ghajari, R. Hewson, and P. Culmer. Design methodology for magnetic field-based soft tri-axis tactile sensors. *Sensors*, 16(9), 2016. ISSN 1424-8220. doi:10.3390/s16091356. URL https://www.mdpi.com/1424-8220/16/9/1356. → page 11

[48] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, June 2016. → page 10

[49] Y. Zhou, B. J. Nelson, and B. Vikramaditya. Fusing force and vision feedback for micromanipulation. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1220–1225 vol.2, May 1998. doi:10.1109/ROBOT.1998.677265. → page 11