Learning manipulation with multi-fingered robot hands from human demonstration

Philipp Ruppel

University of Hamburg
Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics
Technical Aspects of Multimodal Systems

June 26, 2019

Overview

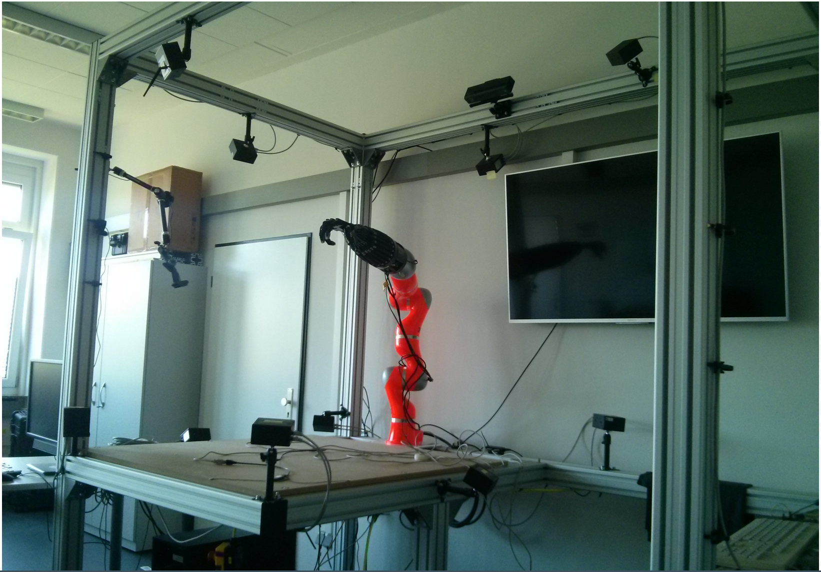
1. Hardware setup
2. Motion tracking
3. Pneumatic robot control
4. Learning from humand demonstration
5. Learning + kinodynamic online trajectory optimization
6. Videos
7. Future work

CML Demo in September

- Open "medicine bottle"
- Humanoid robot hand
- Learning from human demonstration
- Machine learning / neural networks
- Crossmodal Learning

Hardware setup

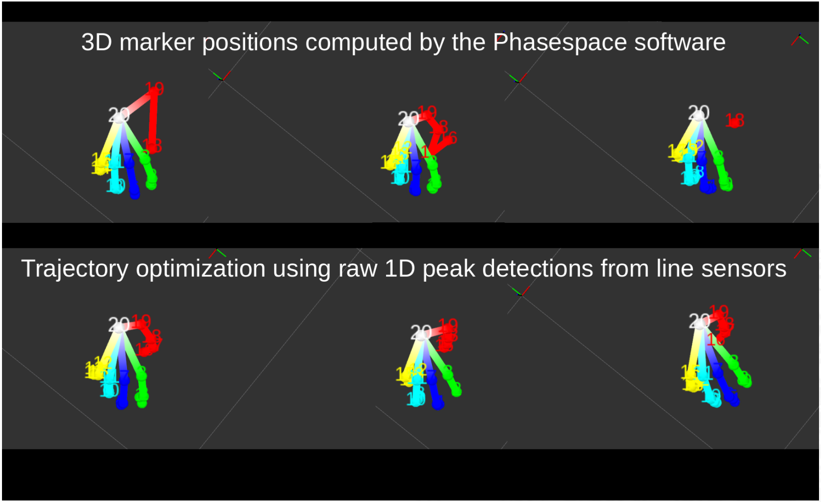
- KUKA LWR arm, pneumatic C5 hand, Phasespace Impulse X2



Phasespace

- Phasespace Impulse X2
- Line Cameras
- High FPS, up to 960 Hz
- Output:
  - 3D marker positions
  - 6D rigid body poses
  - 1D marker positions on line sensors
- 3D positions inaccurate
- Custom reconstruction

Tracking


3D marker positions computed by the Phasespace software
Trajectory optimization using raw 1D peak detections from line sensors

Tracking glove and bottle

Pneumatic Robot Control

Previous

- Position error => Proportional controller => Valve commands
- Hardware support on Shadow hand valve boards
- Unstable under contact

New

- Position error => P controller => Forces
- Forces => P controller => Valve commands
- Current state-of-the-art method
- Stable under contact
- No hardware support, run in software

Hand / Bus System

Previous

- Hardware: PC => Ethernet => Second PC => Parallel port => Converter => CAN bus => Shadow hand
- Software Software: ROS + network client + network server + Shadow software
- Too slow and unreliable to run controllers in software

New

- Hardware: PC => USB => CAN bus
- Software: Roscontrol + custom driver
- Fast enough to run controllers in software

Learning from humand demonstration

Sub-tasks

- Record demonstrations (see above)
- Learning
- Transfer from human to robot

- First record demonstrations and pre-process data, then learn
- How to combine learning and transfer?

Learning + Transfer, 1/5, Retargeting + Policy Cloning

- First map marker positions to matching robot states (IK or trajectory optimization), then learn robot joint angles
- Problem: Redundancies
- Problem: Accurate non-linear regression using neural networks

- Policy cloning: $state2 = policy(current\_state, observations)$
- Problem: unstable, errors accumulate over time, but network has only been trained on single time steps

Learning + Transfer, 2/5, Learned Transfer

- Also learn human-to-robot mapping from demonstrations
- E.g. robot assumes random poses, human imitates them, invert and learn human-to-robot mapping through supervised learning
- Popular approach in literature
- Problem: requires huge amounts of training data
- Problem: usually inaccurate

Learning + Transfer, 3/5, Reinforcement Learning

- End-to-end reinforcement learning
- Input marker positions, learn joint angles or velocities
- Network could learn redundancy resolution
- Rewards across multiple time steps => robust policy
- Could improve policy autonomously
- Reinforcement learning currently slow and inefficient (even in simulation), good differentiable robot simulators not (yet) available (future work?)
- Network would have to learn technical details about a specific robot
- Physics equations already exist / why learn them?
- Want to focus on (higher-level) manipulation problem

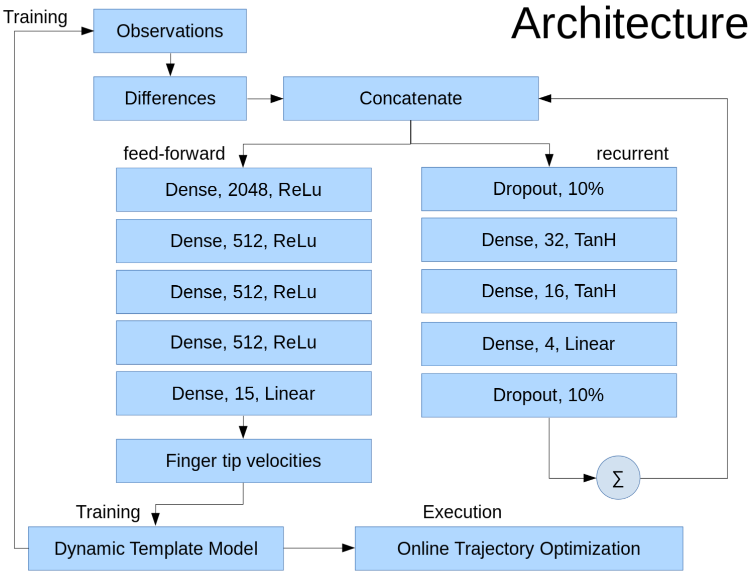
Learning + Transfer, 4/5, Inverse Optimal Control

- Learn a cost function + trajectory optimization
- Assume that human actions are close-to-optimal according to a reward function, try to learn reward function
- Usually requires accurate models of humans and objects
- "Inverse Reinforcement Learning"
- Often solved using (inefficient) reinforcement learning methods ("inverse reinforcement learning")
- Ill-posed problem: many different reward functions could explain a specific action
  - Strong regularization / sparsity
  - How to represent the cost function?
  - Neural network = very general but many variables, hard to find a meaningful cost function from few examples
  - Huge number of demonstrations usually required for non-trivial tasks

Learning + Transfer, 5/5

- Simplified differentiable template model: 5 points = finger tip positions, learn 3D cartesian velocity commands
- Online trajectory optimization
- Compromise between 3 and 4: Position goals = simplified template model (3) or learned cost function (4)
- Less ill-posed than general IRL, differentiable, can be solved efficiently
- Does not have to learn technical details about a specific robot, network can focus on high-level aspects related to manipulation, learned policies mostly robot-independent
- Consider multiple time steps and use differentiable model to propagate gradients back in time to improve robustness (s. option 1)
- Prediction + trajectory optimization for redundancy resolution
- Fast trajectory optimizer needed, online, many DOF: 20 (hand) + 7 (arm)

Architecture



Training

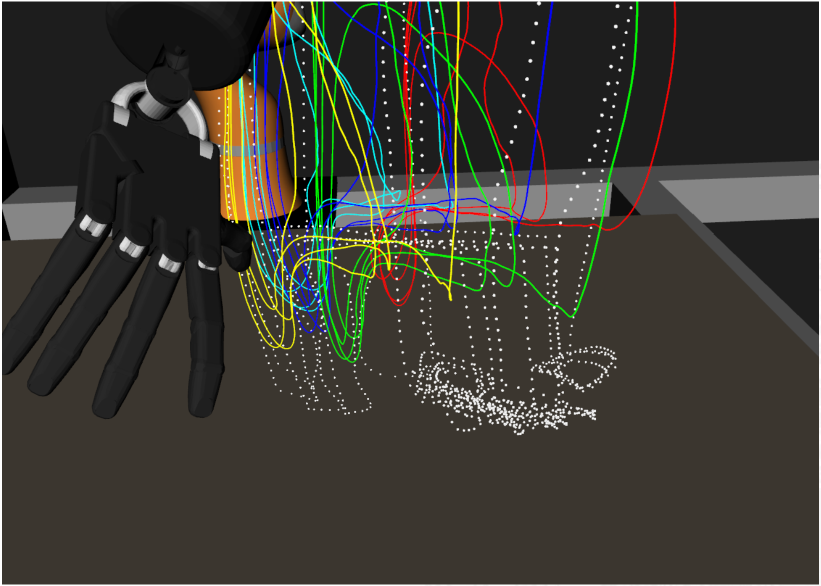
- Train with simplified dynamic template model on complete trajectories
- Minimize mean absolute error between predicted trajectories and demonstrations
- Reset robot states to recorded tracking data at randomly selected time steps
  - Many resets = faster learning, but less stable policy
  - How to choose reset probability?
  - Per-trajectory reset density $= c^{rand()}$
  - Per-sample reset probability $= rand() * density$
  - => Parameter choice simple (exponential, from almost zero to almost 1)

Pose Invariance and Augmentation

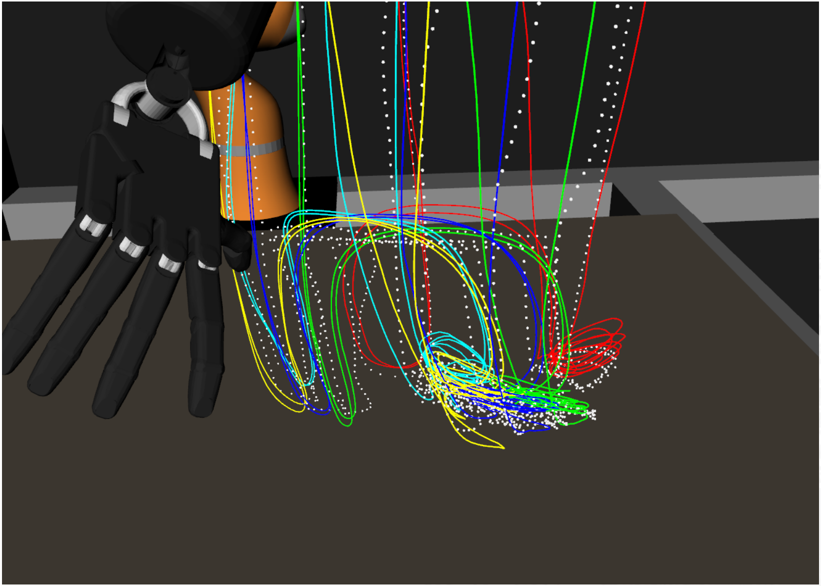
- Input: relative finger-to-object vectors
  => position invariance
- Random rotations (augmentation)
  => rotation invariance
- Random hand/object offsets (augmentation)
  => Learn to control hand pose relative to object
  => Generalize approach motions
- Arbitrary random mutations (augmentation)
  => Learn to control finger poses relative to each other

- Per-trajectory random scaling: $c^{rand()}$
- Per-sample offsets: $gaussian() * scaling$

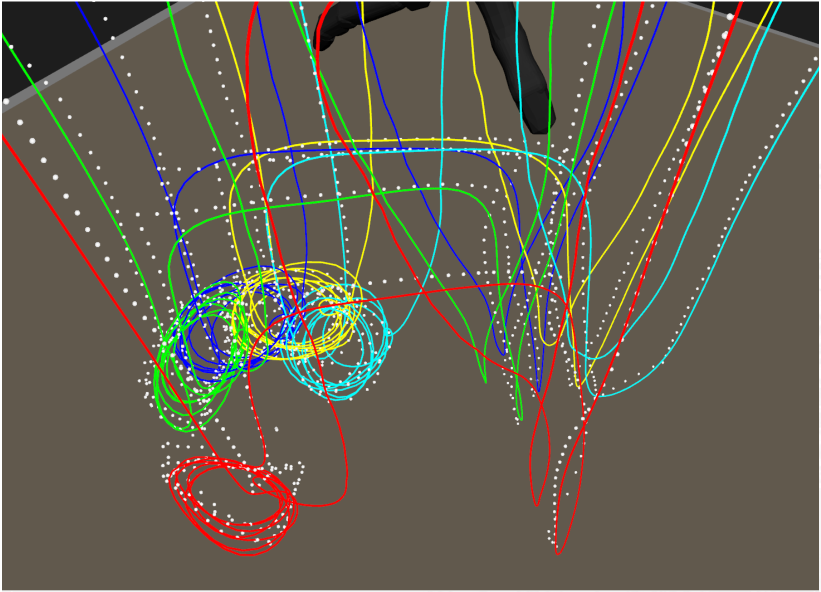
Learning



Learning



Learning



Trajectory Optimization

- Online
- Kinematics + dynamics
- Many degrees of freedom: 20 (hand) + 7 (arm)
- Doing it efficiently - interesting problem
- Custom trajectory optimizer
- Goal programming interface
  - PositionGoal (finger tips)
  - MinimalAccelerationGoal (smooth trajectories)
  - CollisionAvoidanceGoal (2x: hard + soft w. padding)
  - etc.

Trajectory Optimization

- Sequential Quadratic Programming
  - Locally approximate non-linear trajectory optimization problem via linearly constrained quadratic equations
  - Solve QP (many existing methods available)
  - Repeat until convergence
  - One of the standard methods for solving non-linear problems
- Several different solvers
  - Interior-point method (usually fastest, as expected)
  - Penalty method (ok for simple problems)
  - Active-set method (interesting but in practice often slow)
  - Projected Gauss-Seidel (sometimes faster for highly under-determined problems)
  - EnsembleQP: Run different methods in parallel, take first valid solution

Videos

Video

Videos

Video

Videos

Video

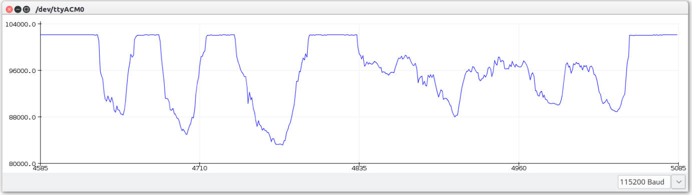
Videos

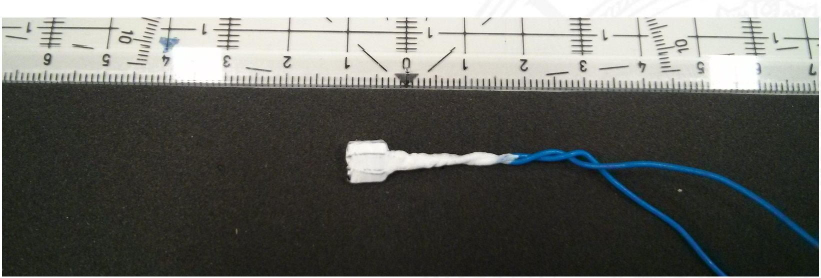
Video

Future Work - Bottle Opening

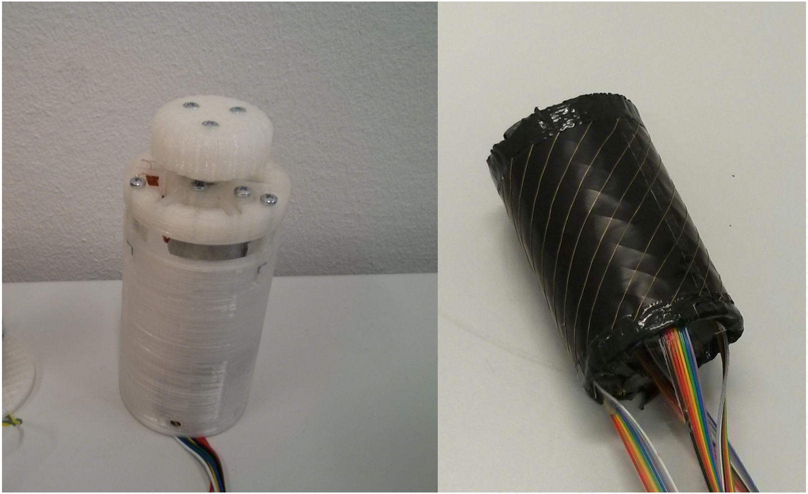
- Try new recurrent network on robot
- How fast can the robot do it?
- Object detection
- Different bottle types and sizes
- Different strategies?
- Collect data from different test subjects
- Crossmodal learning, instrumented objects, tactile glove

- Tactile sensor



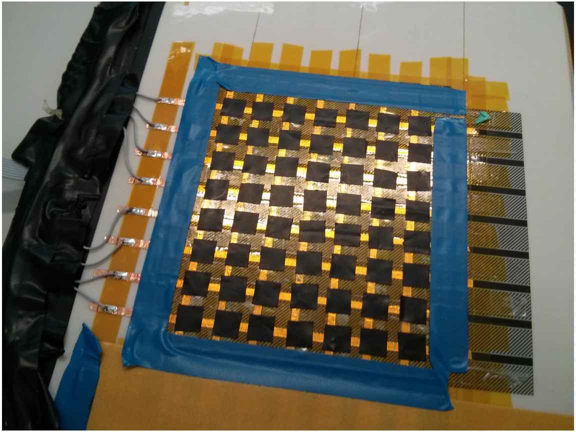


- Torque sensor, motor, tactile matrix



Tactile Sensing

- Tactile matrix



Tactile Sensing

- Tactile glove prototype

Future Work - Trajectory Optimization

- Dual interior points for collision prevention
  - Interior-point methods known for decades
  - Currently popular robot trajectory optimizers often use penalty-like formulations
  - Dual interior points = simultaneously solve for objectives and constraint feasibility, starting point does not have to be feasible
- Weights AND hard priorities
  - Collisions > position goals > smoothness
- Special contact models for planning (e.g. Contact Invariant Optimization, results currently unrealistic + bad scalability)
- Exploit problem structure
  - Hessian (H) / model (M) and problem (P): $H = M^T P^T P \; M$
  - P: usually $O(n)$ sparse, maybe pre-multiply
  - M: worst case $O(n^2)$ for n links but sparse $O(n)$ internal structure (kinematic tree)
  - Auto-diff through kinematic tree inefficient (rotations)
  - High-performance matrix replacement

Future Work - Learning From Simulation

- Discover manipulation strategies autonomously
- Standard RL slow
- Run in simulation, still slow
- RL = useful for unknown environments
- Simulated environments exactly known (even with pseudo-random "domain randomization")
- Simulation + learning = single well-defined optimization problem (vs. RL algorithm), but non-convex
- Supervised learning = usually convex? (exception: sparse regularization)
- Many learning problems in robotics hard-to-solve non-convex optimization problems (manipulation + locomotion)
- Convex = "easy", non-convex = "hard"

Future Work - Learning From Simulation

- Can we convexify robot simulation?
- Quite complex, full and provable convexification unlikely
- Find something that works in practice
- In machine learning: neural networks not provably convex, but can be trained through convex optimization in practice
- Soft contact models
- Relax physical consistency, simultaneously optimize for rewards and physical consistency
- Walking = fly from A to B, then learn to move legs via soft ground model = learning to walk via convex optimization !!!
- Manipulation?
- Soft contact models + physical consistency relaxation -> solve efficiently through convex optimization