Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN Faculty
Department of Informatics

Adaptive Pouring of Liquids with a Robotic Arm

Master Thesis Colloquium

Jeremias Hartz

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

08. Mai 2018

Outline

Pouring liquids

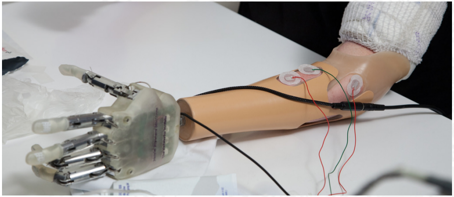
Industrial Applications

- Dangerous liquid/environment
- High precision/efficiency required

Oil changes could be automated[1]

Home Applications

- Cooking
- Cleaning
- Feeding



Intelligent prosthetic arms could pour on its own[2]

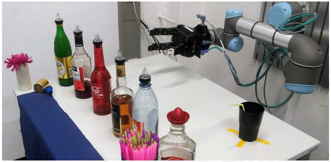
[1] https://upload.wikimedia.org/wikipedia/commons/6/68/SIGAUS_aceite.jpg

[2] https://c1.staticflickr.com/4/3831/12326026754_d979df9a14_b.jpg

Motivation

Mixing Cocktails

- Masterproject 16/17 at TAMS
  - Minimal pouring amount too high
  - Not esthetically pleasing enough



Robot Bartender at TAMS

Robotic arms

Advantages

- Extremely versatile
  - Different tasks executable
  - Liquid containers replacable
  - Move by themselves

Pouring alternatives, precise but not as flexible[1]

[1] https://c1.staticflickr.com/8/7417/9346264212_a0b2b05781_b.jpg

Related Work

Using Force Sensors

- Learning by demonstration
- Generating dynamic pouring model

Force-based learning using Parametric Hidden Markov Models [Rozo, 2013]:

- Input: force, joint states at time t
- Output: joint states at $t+1$
- – Not tested with real liquid
- – Trained only by remote control
- – Retraining for different bottle shapes needed
- + Fast once trained

Using Liquid Simulation

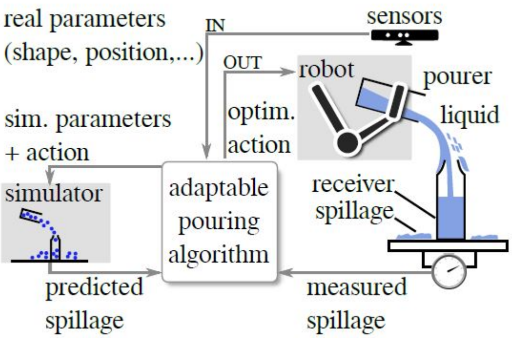
- Models of liquids, prediction of deformation over time
- Generating paths with liquid constraints
- – Exact models of liquid containers needed

Algorithm for planning a collision-free trajectory for pouring [Pan, 2016]:

- Trajectories are generated and optimized
- Final trajectory:
  $E^*(Q^C) = c_{obstacles}(Q^C) + c_{smoothing}(Q^C) + c_{liquid}(Q^C)$
- Liquid body trajectory: $Q^L = (q_1^T q_2^T ... q_N^T)^T$
- – Evaluation of Q_L with $N = 1000$ almost 1h

Learning spillage minimization[Lopez, 2017]:

- Combination of simulation and live feedback



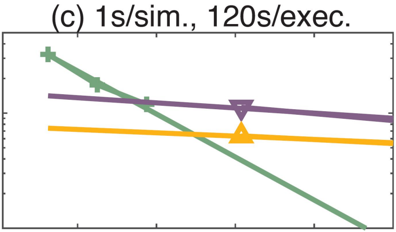
Model of Approach

- Simplified liquid simulation for computation saving

Related Work

1. Calibration of simulation parameters (Particle number, cohesion)
   - Pour with real robot
   - Measure real spillage
   - Adjust parameters to match sim. spillage
2. Pour, measure spillage, optimize

- Simulation
- Spillage feedback
- Both



Spillage over iterations

– Always initial amount = poured amount

Related Work

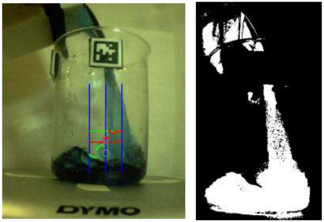
Precise Dispensing of Liquids Using Visual Feedback [Kennedy, 2017]:

- Using camera and Apriltag to measure poured liquid
- No simulation

Input parameters for controller:

- Circular/rectangular shape/opening
- Angle
- Liquid amount poured

– Only transparent container
– Only colored liquid



Liquid amount measured by image processing

Related Work

Learning to pour from video demonstration [Sermanet, 2017]:

- Time-Contrastive Networks
- 2 Videos: 1st and 3rd person view
- Learning from video comparison
- Trying to imitate movements
- Optimizing through reinforcement learning
  - Not very precise
  - Requires a lot training data for generic solutions

Mathematical Approaches Used

For optimization, often used in robotics in learning algorithms:

- Bayesian optimization [Sermanet, 2010]
  Unknown function (or too costly to calculate)
  Single data points available / computable

1. Generate (random) functions going through few data points
2. Merge into one function use more data points for optimization
   (Gaussian process often used for 1. and 2.)
3. Point selection for training at areas of interest

For simulating liquids:

- Navier-Stokes equations

Tools Used

- Hardware
  - UR5 Robotic Arm
  - Force-Torque Sensor
  - USB Camera
  - Bottles as pouring containers
  - Glasses as receiving liquid containers
- Software
  - Linux Ubuntu 16.04
  - C++ 11 and 14
  - ROS version: Kinetic
  - Simulation: Rviz
  - Arm movement: Moveit
  - Pouring interface: tams_ur5_bartender

Limitations

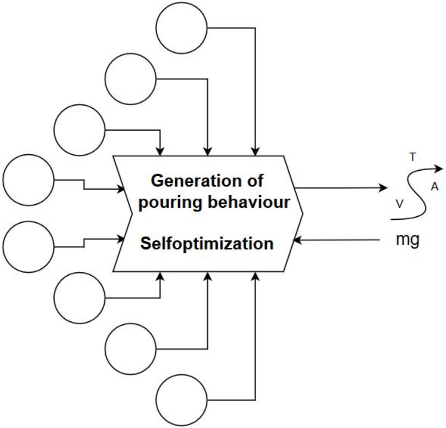
- No liquid simulation
  - Complex and resource costly
  - Simplified not precise enough
  - Exact shape of containers needed
- No liquid detection by camera
  - Liquids are mostly transparent
  - Liquid containers are often non-transparent
- Only bottle shapes

Goals

- Human-like movements
- Pouring exact amount specified (ml)
- No spillage
- Adapting to parameters without needing retraining

Adaptive pouring

Automatically adjust output to a set of input parameters:



General idea

Input: Circles (Parameters), mg (real weight values for continuous improvement)

Output consisting of time, velocities, accelerations, a path

Concept and Input Parameters

Adaptive pouring

Automatically adjust output to a set of input parameters:

1. Location of bottle and glass
2. Height of bottle and glass
3. Collision Objects
4. Amount of liquid to be poured
5. Amount of liquid inside bottle
   - Weight of empty bottle
   - Total weight of bottle
6. Pouring type (normal/high)
7. Viscosity (syrup/water)

Input Parameters

Parameters' measurement

1. Location of bottle and glass

→ 2d camera image
→ Feature detection (find_object_2d) for identifying and locating
→ AprilTag for locating camera itself in respect to robot arm

Input Parameters

Parameters' measurement

1. Location of bottle and glass
2. Height of both liquid containers (bottle and glass)
3. Collision Objects

→ Camera (infrared sensor/ feature detection)

Parameters' measurement

1. Location of bottle and glass
2. Height of both liquid containers (bottle and glass)
3. Collision Objects
4. Amount of liquid that should be poured

$\rightarrow$ Direct/indirect user input
$\rightarrow$ Database: specified cocktail recipe

Parameters' measurement

1. Location of bottle and glass
2. Height of both liquid containers (bottle and glass)
3. Collision Objects
4. Amount of liquid that should be poured
5. Amount of liquid inside bottle
    - Weight of empty bottle
    - Total weight of bottle

→ Database: identified by feature detection
→ Scale
→ Force-torque sensor

Input Parameters

Parameters' measurement

1. Location of bottle and glass
2. Height of both liquid containers (bottle and glass)
3. Collision Objects
4. Amount of liquid that should be poured
5. Amount of liquid inside bottle
6. Pouring type (normal/high)

→ Direct user input
→ Random high for human likeness, given $pouringamount > x$

Input Parameters

Parameters' measurement

1. Location of bottle and glass
2. Height of both liquid containers (bottle and glass)
3. Collision Objects
4. Amount of liquid that should be poured
5. Amount of liquid inside bottle
6. Pouring type (normal/high)
7. Viscosity (syrup/water)

→ Database: identified by feature detection
→ Force-torque sensor

Trajectory Type

- Trajectory for entire arm movement (not only pouring angle)
- Start with trajectories demonstrated by humans
  - Find general function
  - Find how function has to be changed with different inputs

Resulting questions

Can the robot arm play back human trajectories?

- How can a robot try to play them back?
  - How can human trajectories be recorded?
    - What exact information has to be recorded?

Human Trajectory Recording Setup

Needed Information

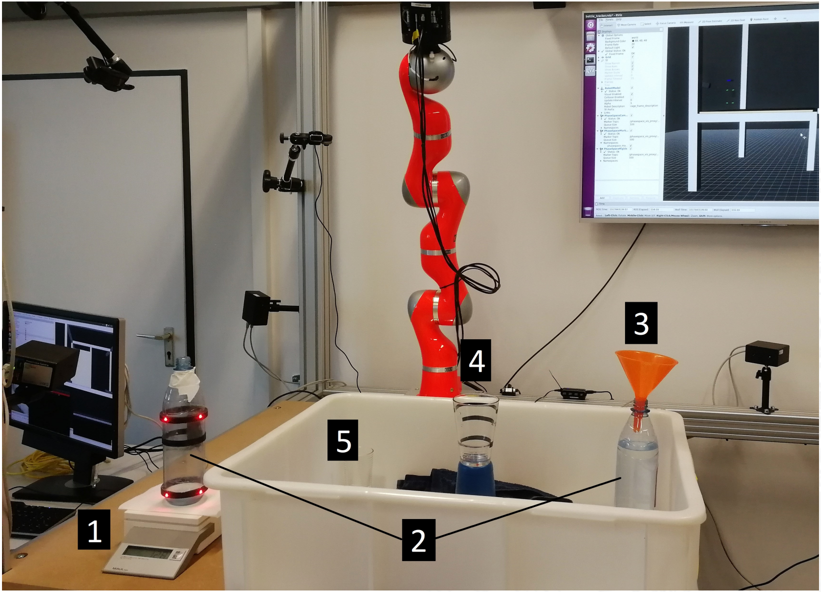
- Bottletop tracking
- Recording topics to rosbag:
  - Bottle position and rotation
  - Bottle weight
  - Video images for monitoring

Used Tools

Tracking cage + trackable markers

1. USB-Scale
2. 2 Bottles
3. Funnel
4. Glass
5. Container

Trajectory Recording Setup



Trajectory Recording Configurations

Configuration	Pour up to Marker X	Spout	Slow	High
1	1			
2	2			
3	3			
4	2		x	
5	3		x	
6	3			x
7	1	x		
8	2	x		
9	2	x		x

Recorded configurations

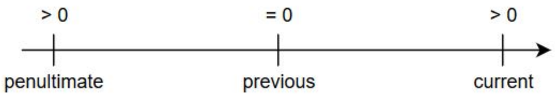
13 rosbags total
Emptied 39 bottles
208 Valid samples so far (from 12 bags)

Automated Sample Extraction

How to recognize one pouring sample?

- Scale topic
  - Stable weight:
    - Weight stays the same X(=2) times (Rate: 1HZ)
    - Same weight = +/- 0.5 gram
  - 3 stable weights = current, previous, penultimate

Extract sample if no weight in between:

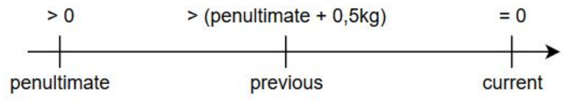


Condition on which a pouring sample ends

Automated Refill Detection / Error Finding

How to identify bottle refill?

```
trajectory.size() > 0 //1st fill is not a refill
```



Condition on which the bottle has been refilled

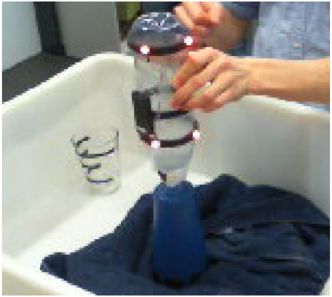
How to identify a failed sample?

- Bottle out of range (-x values)
- Bottle not tilted enough ($< 45^\circ$)
- No deletion, just mark as failed

Transform Poses

Transformation of bottle points

- Goal: transform data into glass frame
  1. Record glass position

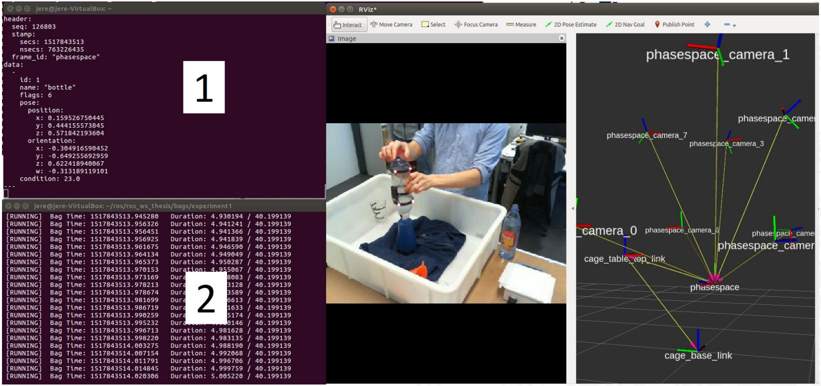


Recording glass pose

  2. Retrieve glass position
  3. Transform bottle points

Retrieve Glass Position

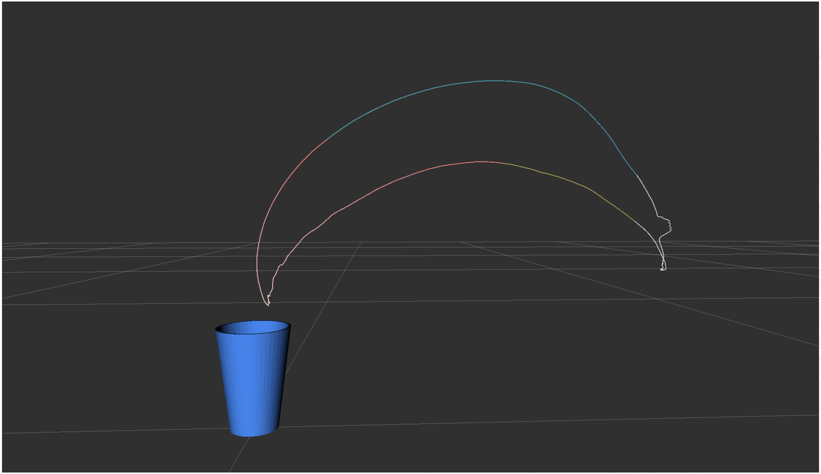
1. rostopic echo /phasespace_ros/rigids
2. rosbag play setup_glass.bag



Reading glass frame pose for new frame of origin

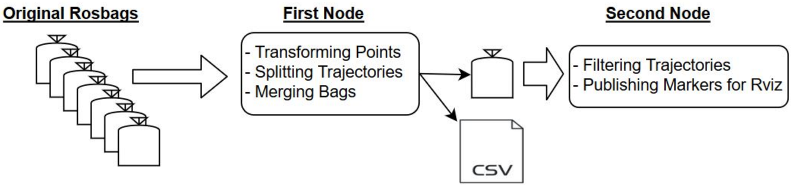
Transformation in Rviz

- Transformed points as visualization markers
- Inserted glass mesh into URDF



Implementation

2 Nodes for analysis



Processing of recorded rosbags

Input Parameters

First Node

- Path of bag folder
- Array of bag names

Second Node

- Path to bag
- Min/max pouring/intial amount
- Min pouring angle to display

Message Type for Comparison

New .msg type for better filtering:

Trajectory.msg

```
tams_pour/PoseStampedArray stampedPoses
std_msgs/Bool valid
std_msgs/Bool person
std_msgs/Bool high
std_msgs/Bool slow
std_msgs/Bool bottleSpout
int32 initialAmount
int32 pouredAmount
```

Setting properties:

- person, high, slow and bottleSpout: configuration info
- initialAmount, pouredAmount and valid: fully automated

Comparison Methods

Colors indicating changes in:

- Angle
- Poured amount
- Initial amount
- Time (relative vs total)
- Motion direction

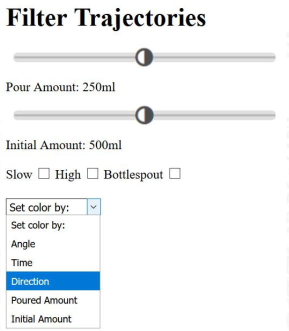
Filter trajectories by:

- Initial amount
- Poured amount
- Other configuration properties

Implementing Live Comparing

User interface for live filtering

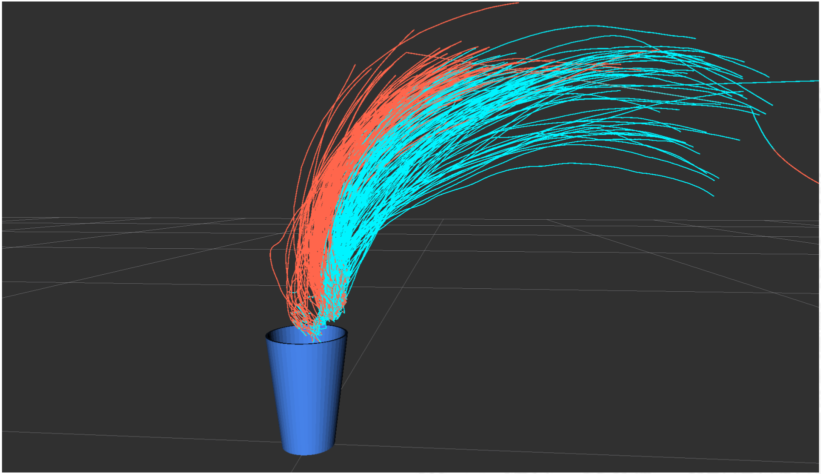
- Rosbrigde: Connection between JavaScript and C++
- Action server for communication:
  → Get available trajectories
  → Filter and display through rviz
- Selectable list of trajectories
- Slider for single point bottle mesh
- Integrate arm movement testing
  → Reuse tams_ur5_bartender



First filter interface idea

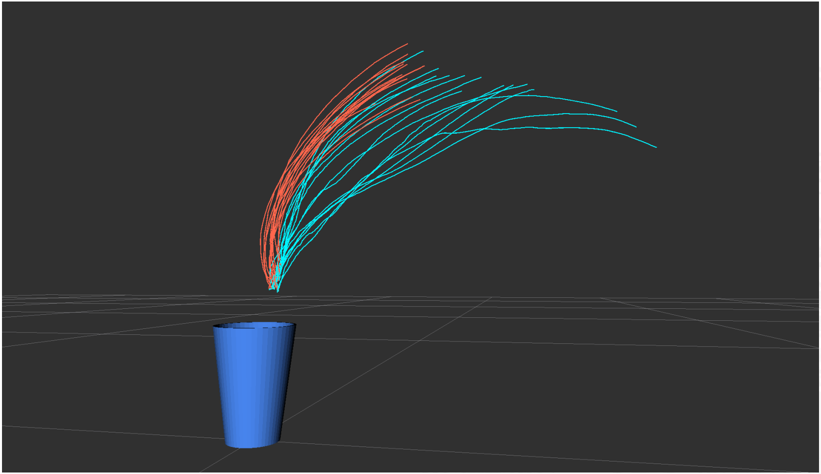
All Samples "Regular" Configuration

All Samples from Configurations 1-3 (103)



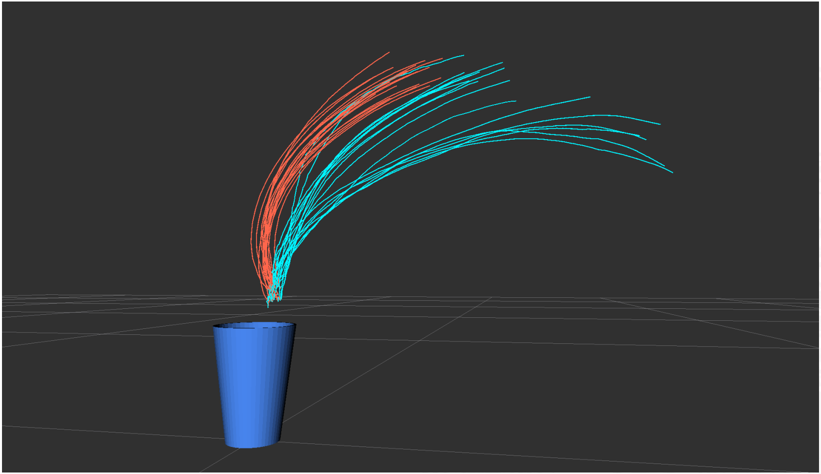
Filter 1st Glass-Marker

Pouring: 0-70 ml (40 ml avg. - First Glass-Marker)
Start Amount: 600-900 ml, Samples: 14



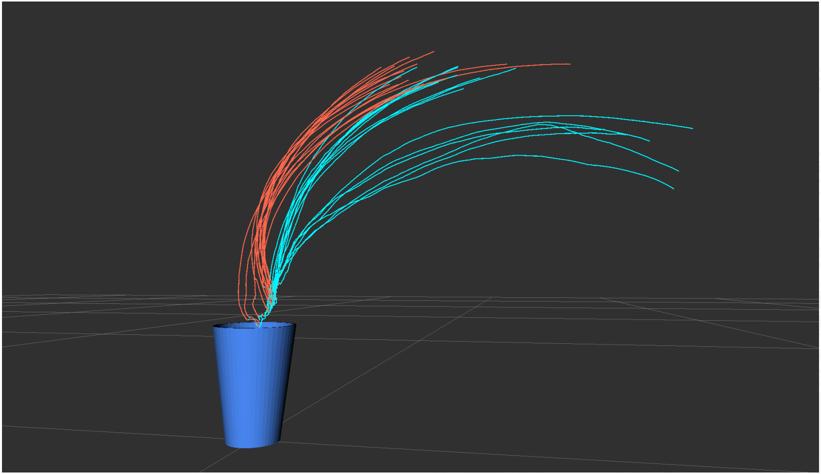
Filter 1st Glass-Marker

Pouring: 0-70 ml (40 ml avg. - First Glass-Marker)
Start Amount: 300-600 ml, Samples: 17



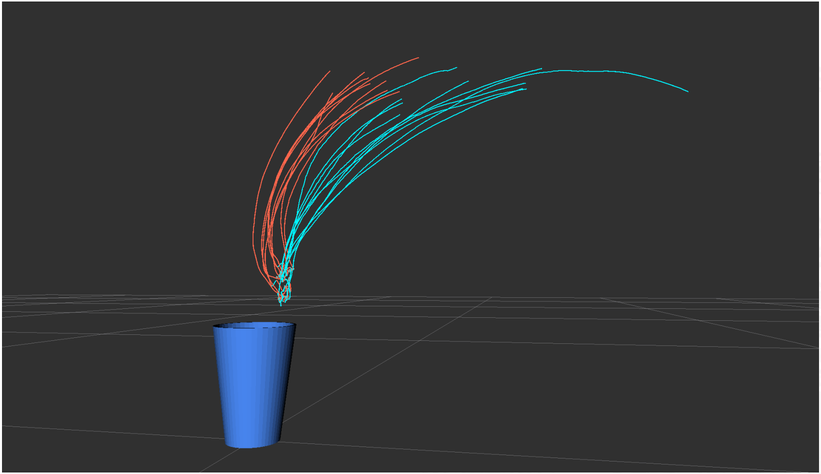
Filter 1st Glass-Marker

Pouring: 0-70 ml (40 ml avg. - First Glass-Marker)
Start Amount: 0-300 ml, Samples: 15



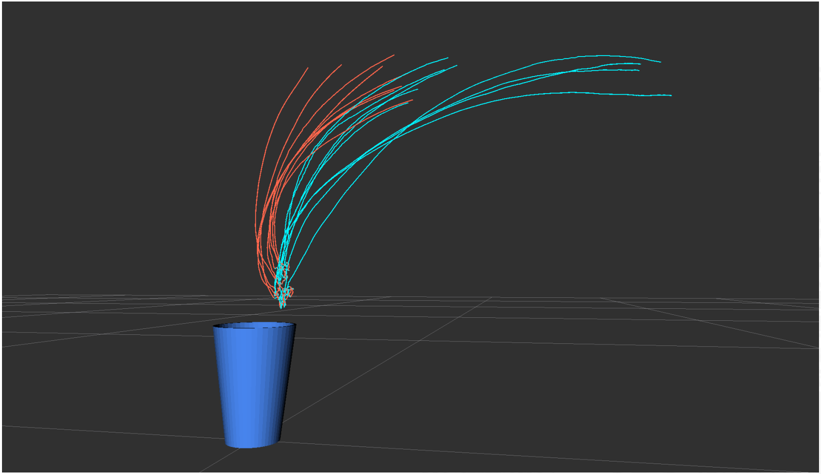
Filter 2nd Glass-Marker

Pouring: 70-220 ml (190 ml avg. - Second Glass-Marker)
Start Amount: 600-900 ml, Samples: 10



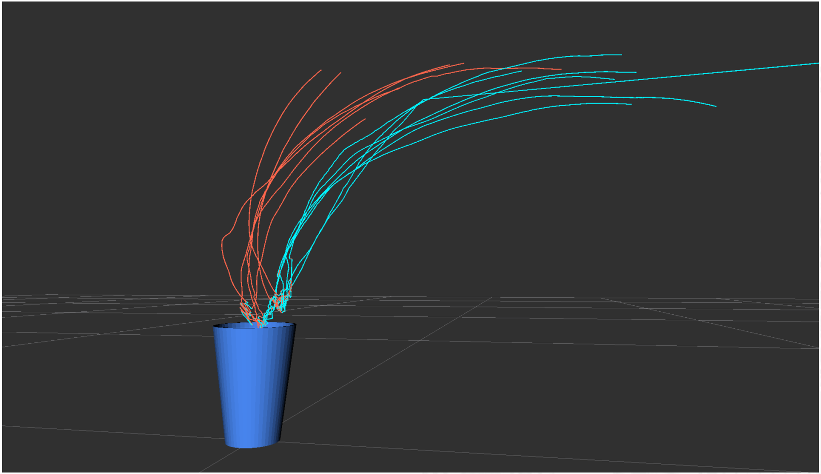
Filter 2nd Glass-Marker

Pouring: 70-220 ml (190 ml avg. - Second Glass-Marker)
Start Amount: 300-600 ml, Samples: 09

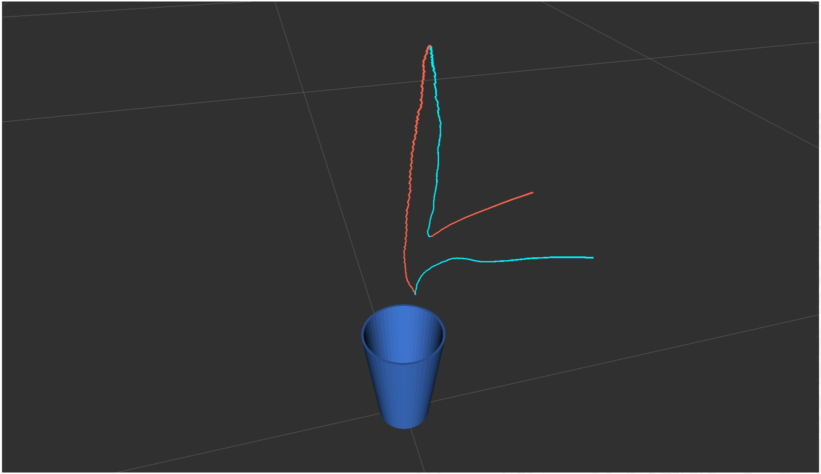


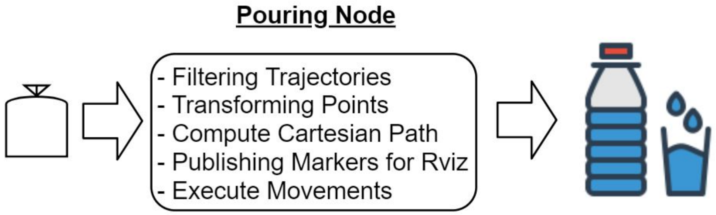
Filter 2nd Glass-Marker

Pouring: 70-220 ml (190 ml avg. - Second Glass-Marker)
Start Amount: 0-300 ml, Samples: 07



Typical Sample from "High" Configuration

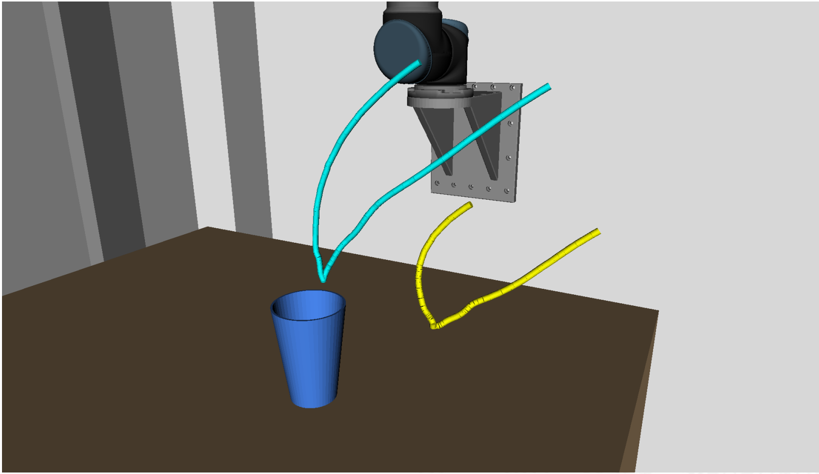




Input Parameters

- Path of bag
- Filter: Every X point
- Min pouring angle to traverse
- Min distance between filtered points
- Min angle distance between filtered points
- Max distance between computed points

Bottle trajectory and transformed trajectory for UR5 end-effector



Problems

- Trim trajectories to pouring part
  $\rightarrow$ filtering out all angles below X° in regards to the glass
  Another approach:
  Pouring start/end detection with force-torque sensor
  $\rightarrow$ Training needed
- Move arm to first point of trajectory
  IK-Solution not always found
  $\rightarrow$ Test Bio-IK Solver
  Constraints needed for testing with real liquids
  $\rightarrow$ Integration into tams_ur5_bartender[1]

[1] https://github.com/TAMS-Group/tams_ur5_bartender

Smoothing Trajectory

- Arm can not move in original speed - speed limits
- Jitter in original trajectories
- Only traverse through every 5/10/20/50/100 point
  $\rightarrow$ Still not fast enough
- Only 3 points: Start, lowest, end
  $\rightarrow$ Smooth, but still not fast enough

Arm Speed Limits

- Moveit computes trajectory with maximum speed
  $\rightarrow$ Changing speed limits:

```
$ roscd  tams_ur5_setup_moveit_config/config
$ xdg-open joint_limits.yaml
```

Set *max_acceleration* and *max_velocity* to max (around 3.0)
Set *has_acceleration_limits* on all UR5 joints to *true*

- Given waypoints: 110
- Computed 97.27% in 11.23s
- Computed waypoints: 107
- Relative human time: 4.30s

Before (max speed = 0.5)

- Robot time: 8.41s

After (max speed = max)

- Robot time: 5.03s

Adjust to Original Duration

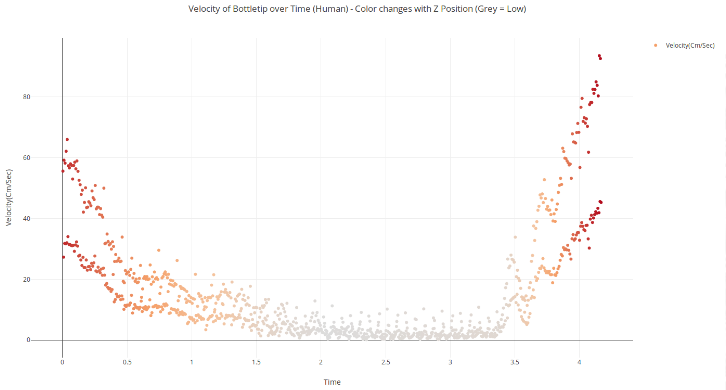
Optimize waypoint filtering until robot time $=$ original time

- Too fast in main pouring part of trajectory
- Not smooth even with $\sim$10% of original points
- No visible improvement after using *Quaternion.slerp*()

Analyzing Original Speeds

Other approach:

- Split trajectory based on original speed and compute separately



Velocity over Time in Original Trajectory

Manual Cartesian Speed Changing

Setting new speeds for each joint:

- Time difference of original trajectory
- Joint distance to next angle
- Set Velocity/Acceleration

```
jointVel = jointDistToNextPoint / originTimeDiff;
jointAcc = jointVel / originTimeDiff;
```

- Problem: Newly generated points
  $\rightarrow$ Arm stops: Acceleration above max
- Limit acceleration, adjust velocity accordingly
  $\rightarrow$ Time behavior better, not smooth

Biggest challenges:

- Smooth trajectory while imitating human velocity profile
- Extracting parameters of pouring trajectory that have to be changed with different inputs
  → Dynamic motion primitives

Final thoughts:

- Pouring is a wide field with many subtopics -
  all has to be put together for a complete pouring task
  → Working demo will be priority
- Human trajectory recording framework can be used on for Machine Learning

Thank You

References

[Rozo, 2013] L. Rozo and P. Jiménez and C. Torras (2013)
Force-based robot learning of pouring skills using parametric hidden Markov models
9th International Workshop on Robot Motion and Control 13(7), 227–232.

[Pan, 2016] Zherong Pan and Chonhyon Park and Dinesh Manocha (2016)
Robot motion planning for pouring liquids
Proceedings International Conference on Automated Planning and Scheduling, ICAPS 16(1), 518–526.

[Lopez, 2017] Tatiana Lopez-Guevara and Nicholas K Taylor and Michael U Gutmann and Subramanian Ramamoorthy and Kartic Subr (2017)
Adaptable Pouring: Teaching Robots Not to Spill using Fast but Approximate Fluid Simulation
Proceedings of Machine Learning Research 17(11), 77–86.

References (cont.)

[Kennedy, 2017] M. Kennedy and K. Queen and D. Thakur and K. Daniilidis and V. Kumar (2017)
Precise dispensing of liquids using visual feedback
2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 17(9), 1260–1266.

[Sermanet, 2017] Sermanet, Pierre and Lynch, Corey and Chebotar, Yevgen and Hsu, Jasmine and Jang, Eric and Schaal, Stefan and Levine, Sergey (2017)
Time-Contrastive Networks: Self-Supervised Learning from Video
arXiv preprint arXiv:1704.06888.

[Sermanet, 2010] Eric Brochu and Vlad M. Cora and Nando de Freitas (2010)
A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning
http://arxiv.org/abs/1012.2599.