Training robots with machine learning: Juggling & Throwing

Based on

Robot juggling: Implementation of memory-based learning

TossingBot: Learning to throw arbitrary objects with Residual Physics

Training robots with machine learning

- Compared to digital tasks: *slooow*
- Starting new trials: not trivial
- Sometimes: Need fast predictions / decisions
- But also: highly useful

What does a robot-task look like?

- Input from environment
  - Sensor data
  - Camera
  - …
- Issue command what robot should do
  - Turn motor to position x
  - Throttle/increase thrust by x
  - …
- Commands lead towards goal
  - Can be very vague
  - Need to *understand* environment

What kind of model to use?

parametric

- Mathematical function with *finite* set of free parameters
- *Global* function fitting
- Don’t remember data

Examples:

- Linear regression
- Neural networks

non-parametric

- Mathematical function with *unlimited* set of free parameters
- *Local* function fitting
- Remember data

Examples:

- N-nearest neighbor
- Kernel regression

Locally weighted regression (LWR)

- Non-parametric (memory-based)

- Estimate local linear models for different points

- Offers various statistical tools to:
  - Assess reliability of lookups
  - Optimize quality of lookup
  - Handle noise and corrupted data

Locally weighted regression (LWR)

- Unweighted regression:
  - Find solution to equations $y = X\,\beta$ (solve $X^T X\beta = X^T y$)
  - X: m x (n+1) matrix
    - m = # data points
    - n = # input dimensions
  - Prediction of query point x_q:
    - $\hat{y}_q = x_q^T \beta$
- Problem: each point is equally weighted
- Solution: Weight by distance

Locally weighted regression (LWR)

- Introduce distance to query point x_q
- For each stored data point: $d_i^2 = \sum_{j=1}^{n} s_j \left(X_{ij} - x_{q_j}\right)^2$
- Weight for every point: $w_i = f\left(d_i^2\right)$

- Simple weighting function: $w_i = \frac{1}{d_i^k}$
- Better scheme: $w_i = exp\left(\frac{-d_i^2}{2k^2}\right)$

Locally weighted regression (LWR)

- For each stored data point (index i)
  - Calculate distance to query point
  - Calculate weighting, based on distance
  - Multiply row in X and y with w_i

- Apply regression to weighted matrices
- Additionally: *ridge regression*
  - Classic regression: $X^T X\beta = X^T y$
  - Ridge regression:$(X^T X + \Lambda)\beta = X^T y$

LWR - Comparison

[1]

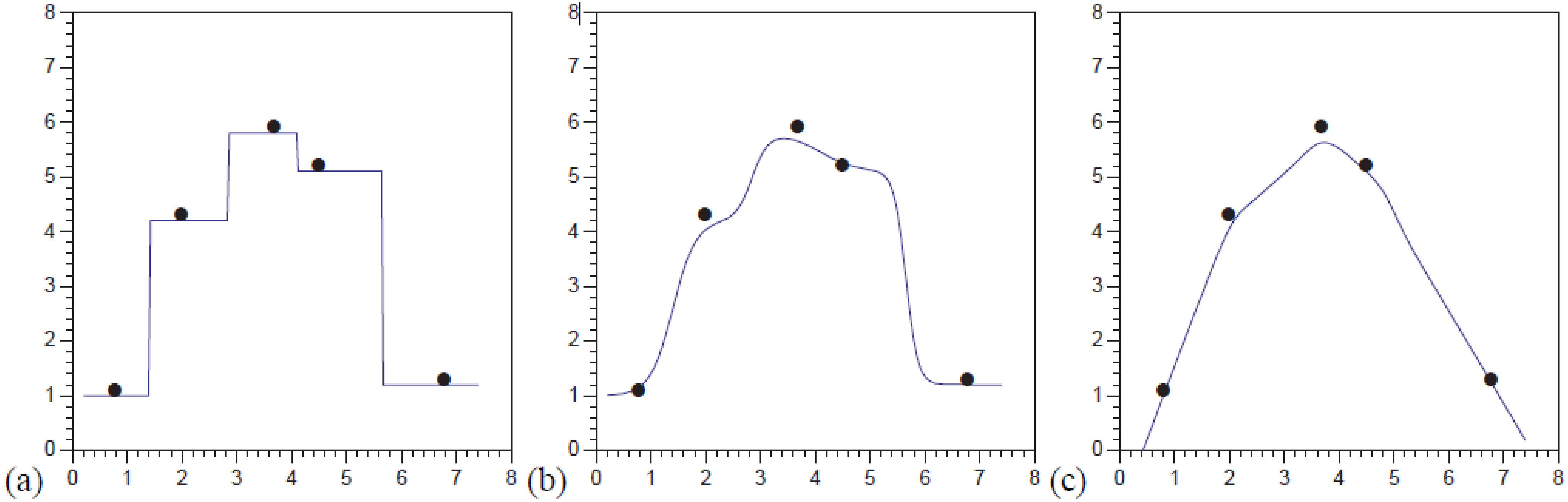


Figure 1: Characteristic performance of three different nonparametric function approximation techniques: (a) nearest neighbor; (b) weighted average; (c) locally weighted regression

Exploration

Problems:

- High-dimensional space
- Sparse data

Even worse:

- Robots are **slow**
- Some regions may be costly / unsafe

→ Random search not feasible

Shifting Setpoint Exploration Algorithm (SSA)

- Approach: slow and steady
- Break task down into two parts
  - Fast timescale: Keep system controlled at fixed certain points
  - Slow timescale: Shift setpoints towards goal

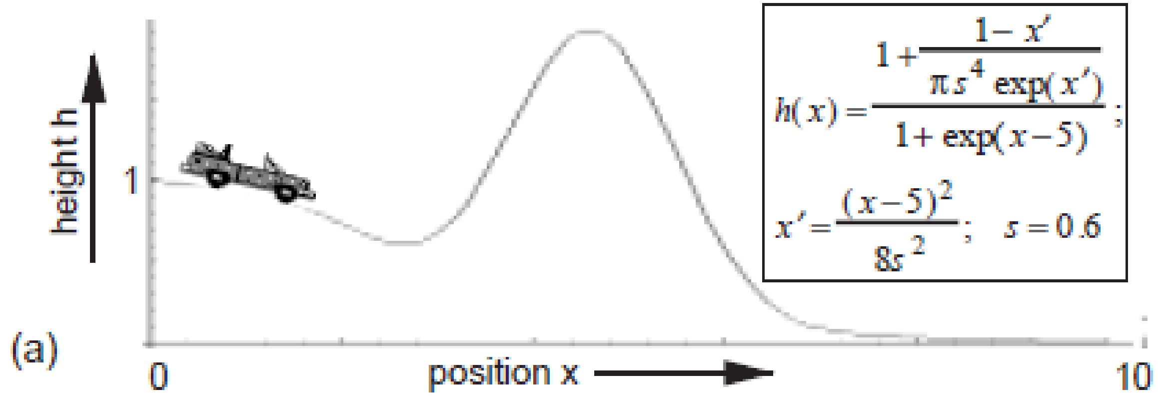
→ Exploration around setpoints ensures confidence in that region

→ Shifting moves the system slowly but steadily towards target, learning along the way

SSA - Example

- Car driving along mountain road
- Task:
  - drive at constant *horizontal* speed $\dot{x}_{desired}$ from left to right
  - Minimize fuel consumption

- Interaction:
  - Noisy feedback of x and $\dot{x}$
  - Control thrust F at 5Hz



[1]

SSA - Initialization

1. Start at random location

2. Execute a few random trials

3. Search for point with highest confidence → Declare as setpoint
   $\left(x_{S,in}^{T},\ F_S,\ x_{S,out}^{T}\right)^{T}$

4. Try to reach setpoint from each new trial

SSA - Procedure

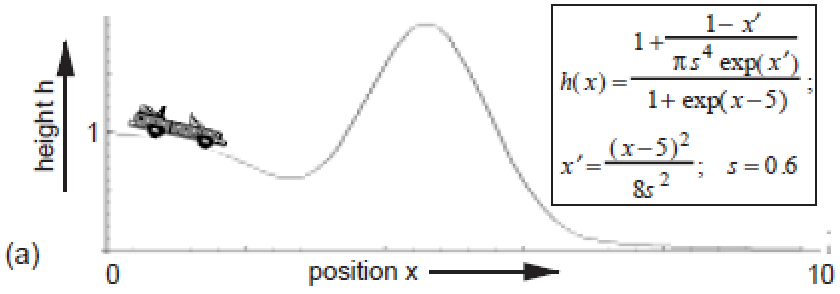
1. Learn to reach setpoint until certain confidence

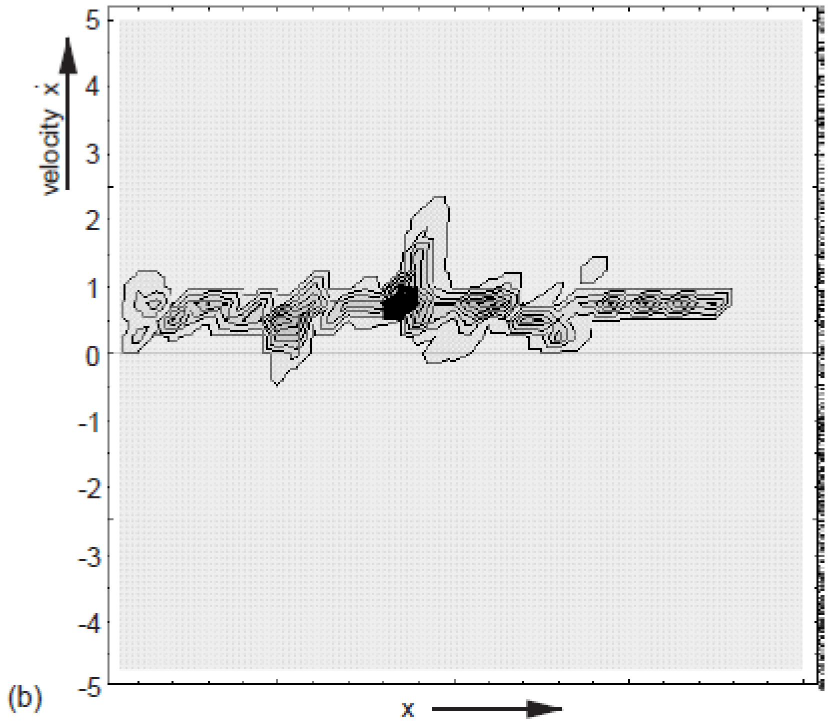
2. Take derivative of $x_{desired} - x_{S,out}$ w.r.t. command F_S

3. Calculate correction ΔF_S and update: $F_S = F_S - \Delta F_S$

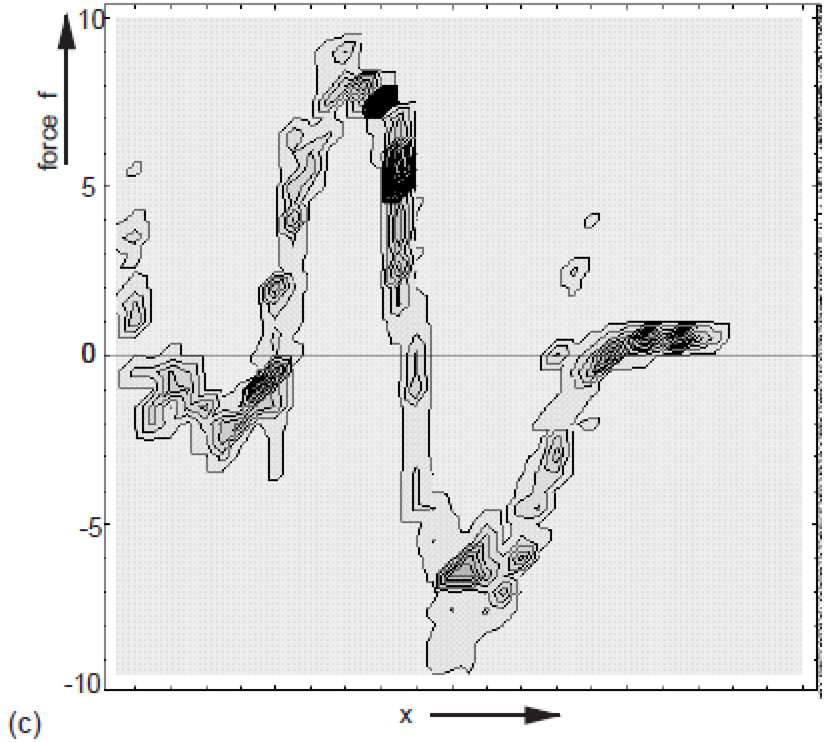
4. Assess fit at updated setpoint
   - If quality above some threshold: continue with 1.
   - Else: Terminate

SSA – Example result





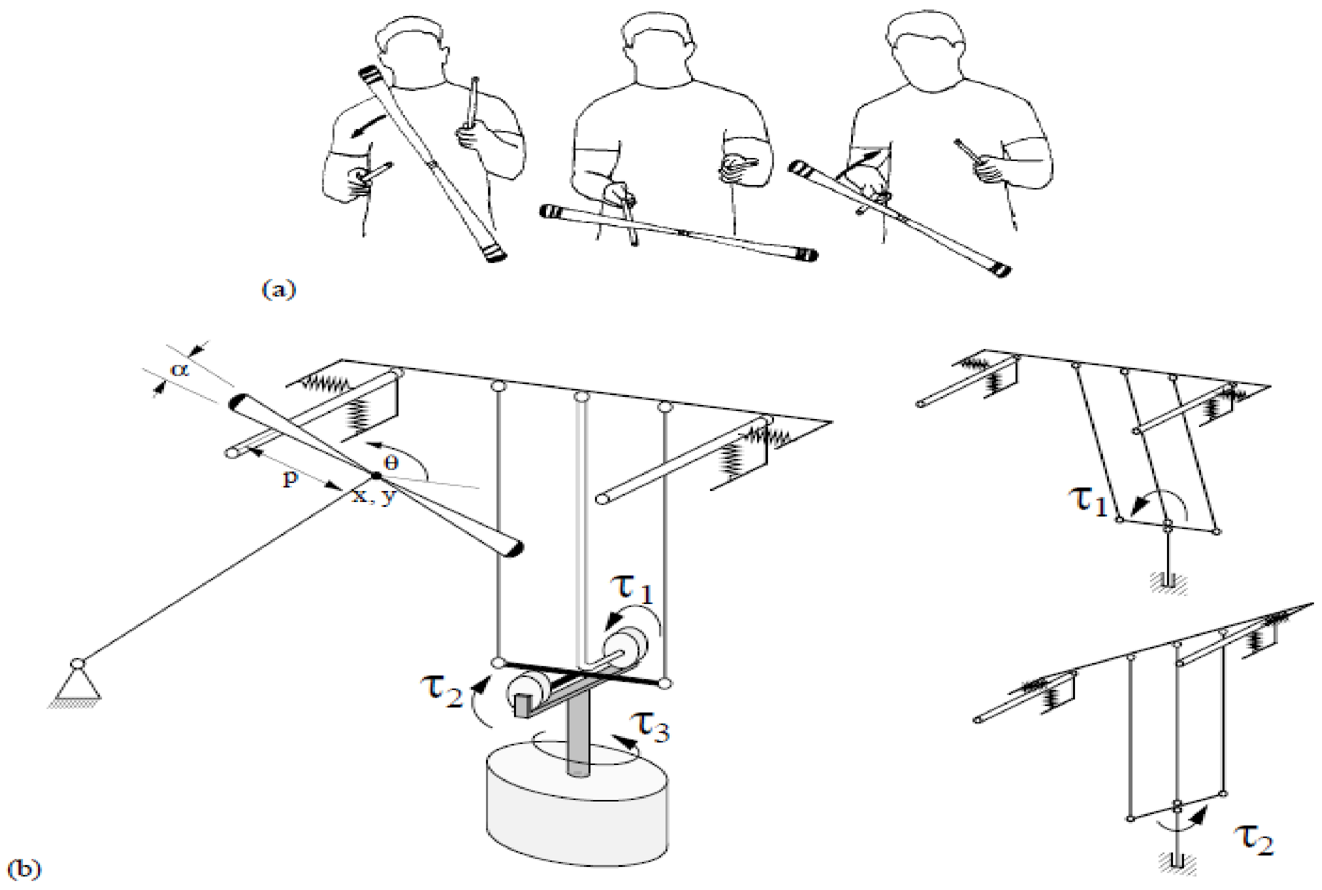
Phase space



[1]

Position-action space

Robot juggling ("Devil sticking")



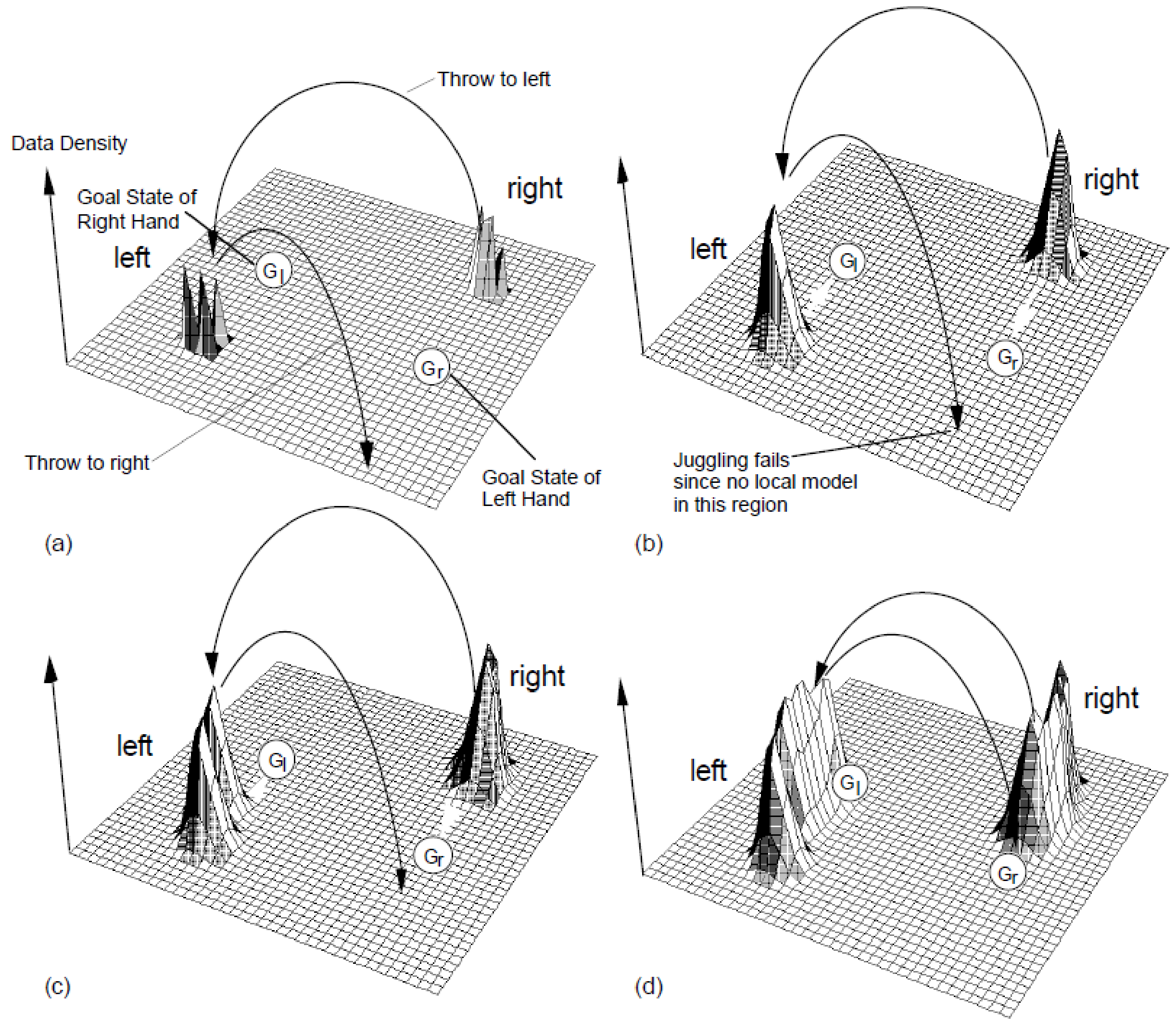
[1]

Task description

- Task state vector
  - Impact state with other hand stick at nominal position
  - $x = \left(p, \theta, \dot{x}, \dot{y}, \dot{\theta}\right)^T$
  - After stick leaves hand: estimate impact with other stick from flight trajectory

- Task command
  - Displacement of hand stick from nominal position $(x_h, y_h)^T$
  - Center stick angular velocity threshold $\dot{\theta}_t$
  - Throw velocity vector $\left(v_x, v_y\right)^T$
  - $u = \left(x_h, y_h, \dot{\theta}_t, v_x, v_y\right)^T$

- Each throw generates experience vector $\left(x_k^T,\ u_k^T,\ x_{k+1}^T\right)^T$


Data Density
Goal State of
Right Hand
left
right
Throw to left
Throw to right
Goal State of
Left Hand
(a)
left
right
Juggling fails
since no local model
in this region
(b)
left
right
(c)
left
right
(d)

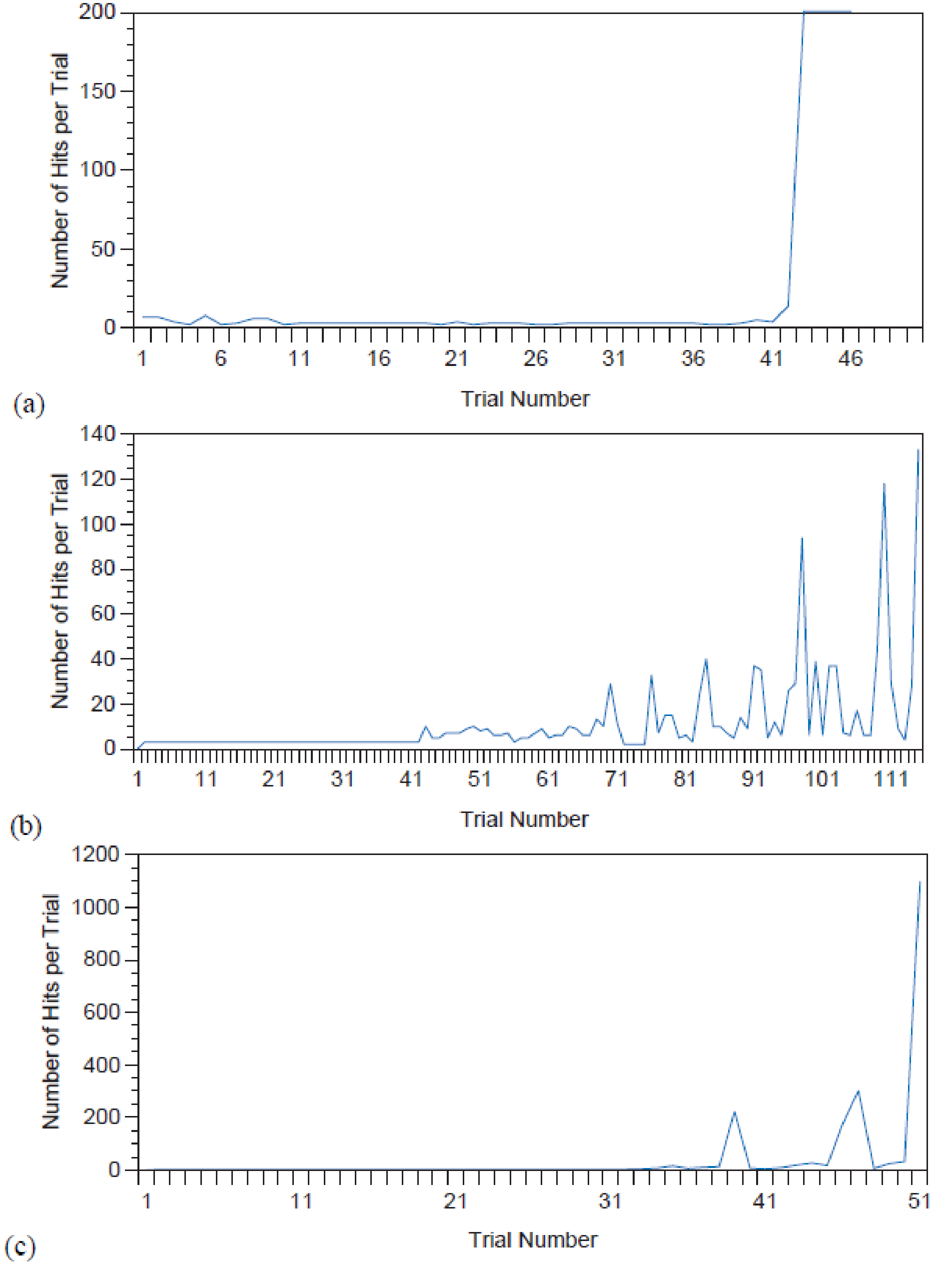
[1]

Devil sticking: Learning curves

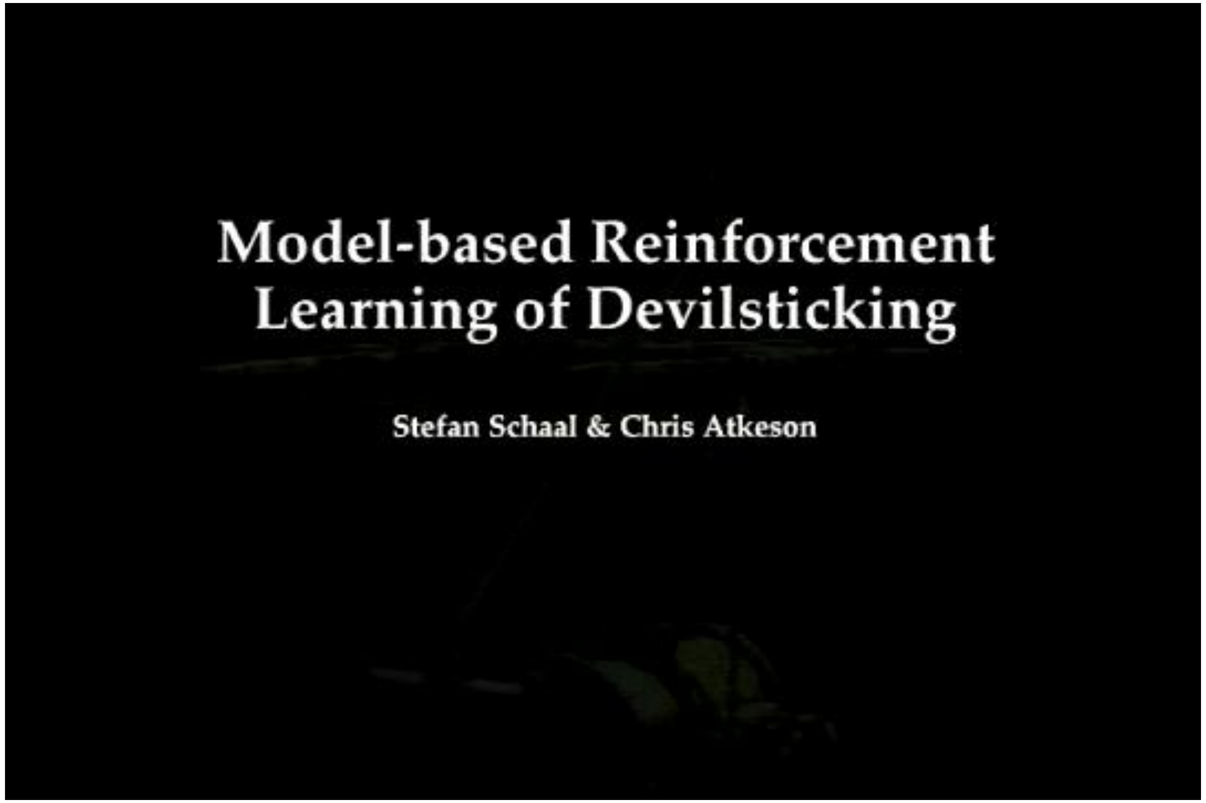
a) Simulation results

b) Real robot results

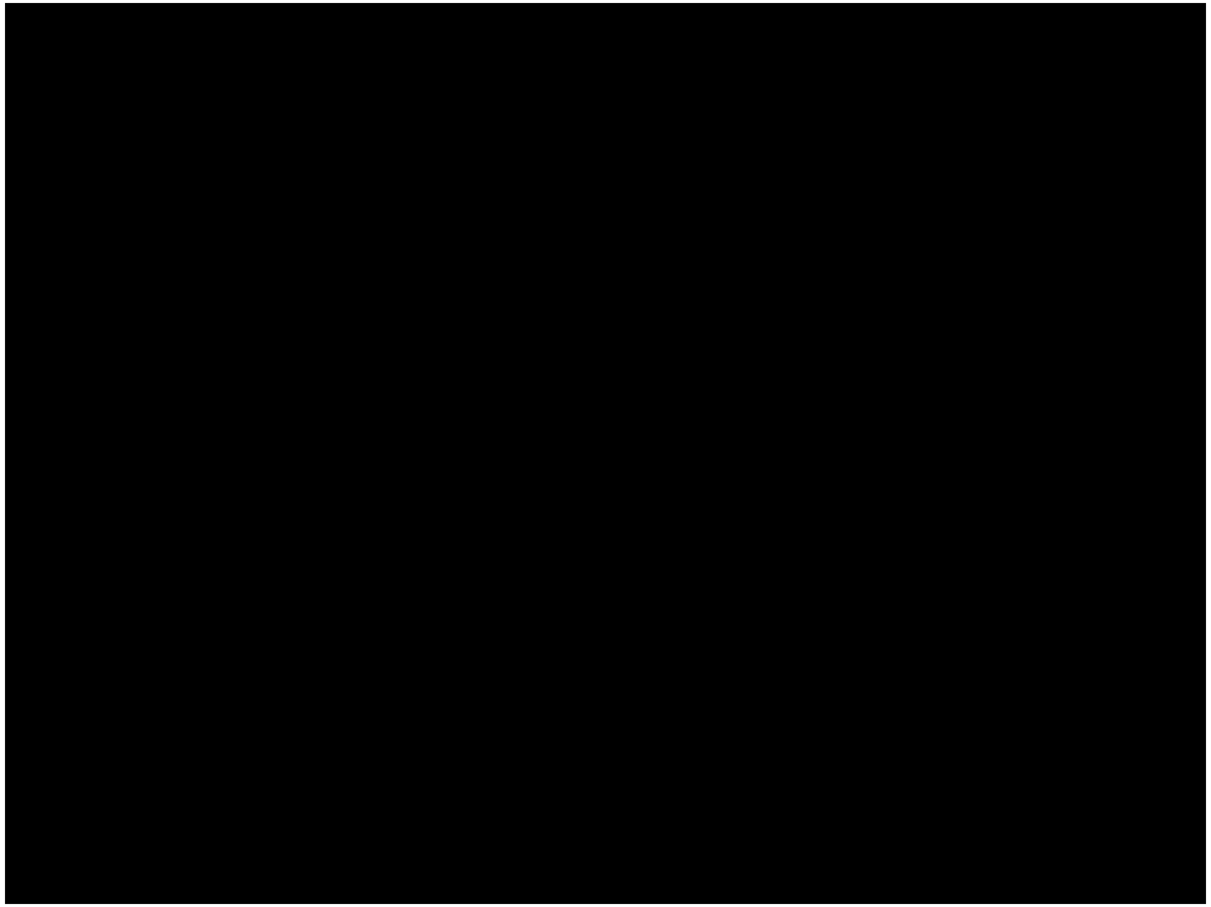
c) Real robot results with small random noise in commands



[1]


Model-based Reinforcement
Learning of Devilsticking
Stefan Schaal & Chris Atkeson

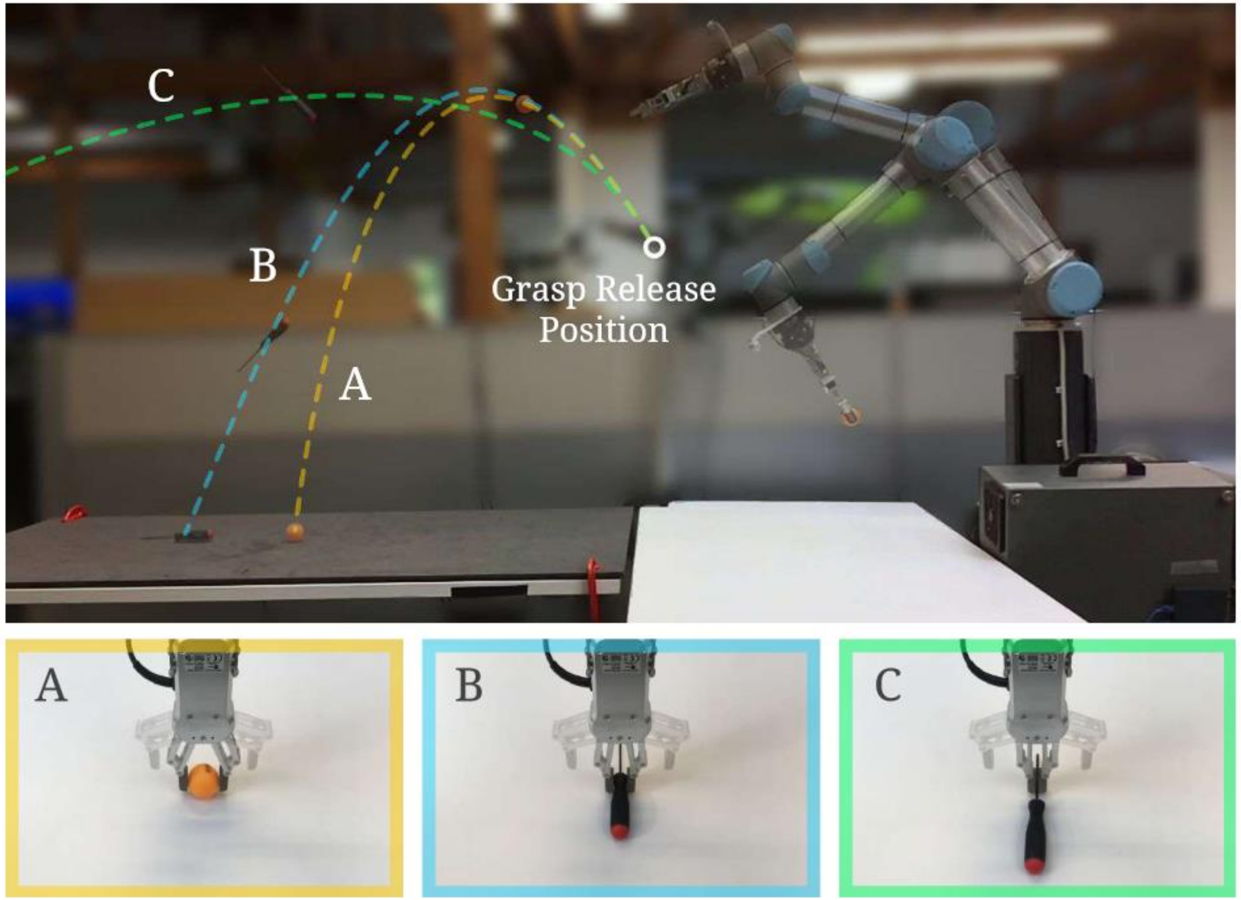
[2.1]



[2.2]

More recent: TossingBot (March 2019)

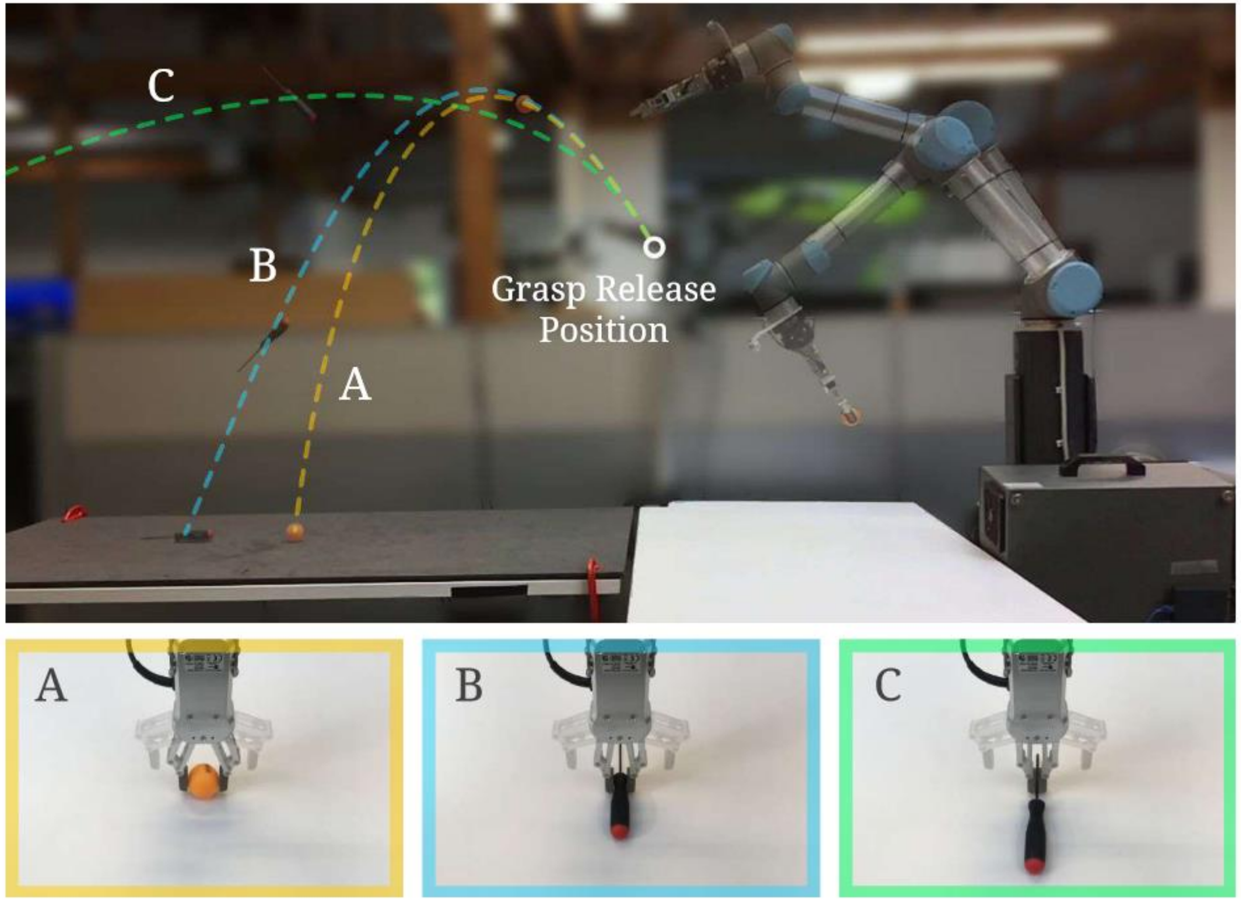
- Teach robot arm to grab **and** throw arbitrary objects



[3]

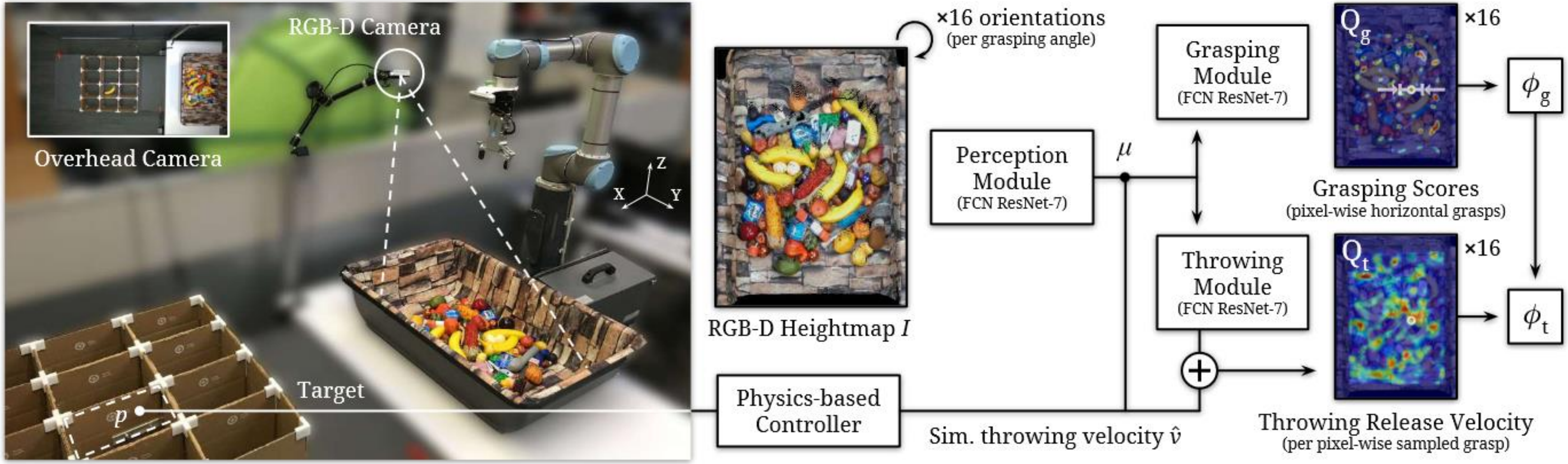
More recent: TossingBot (March 2019)

- Teach robot arm to grab **and** throw arbitrary objects
- 500+ mean picks per hour
- Generalization to new objects

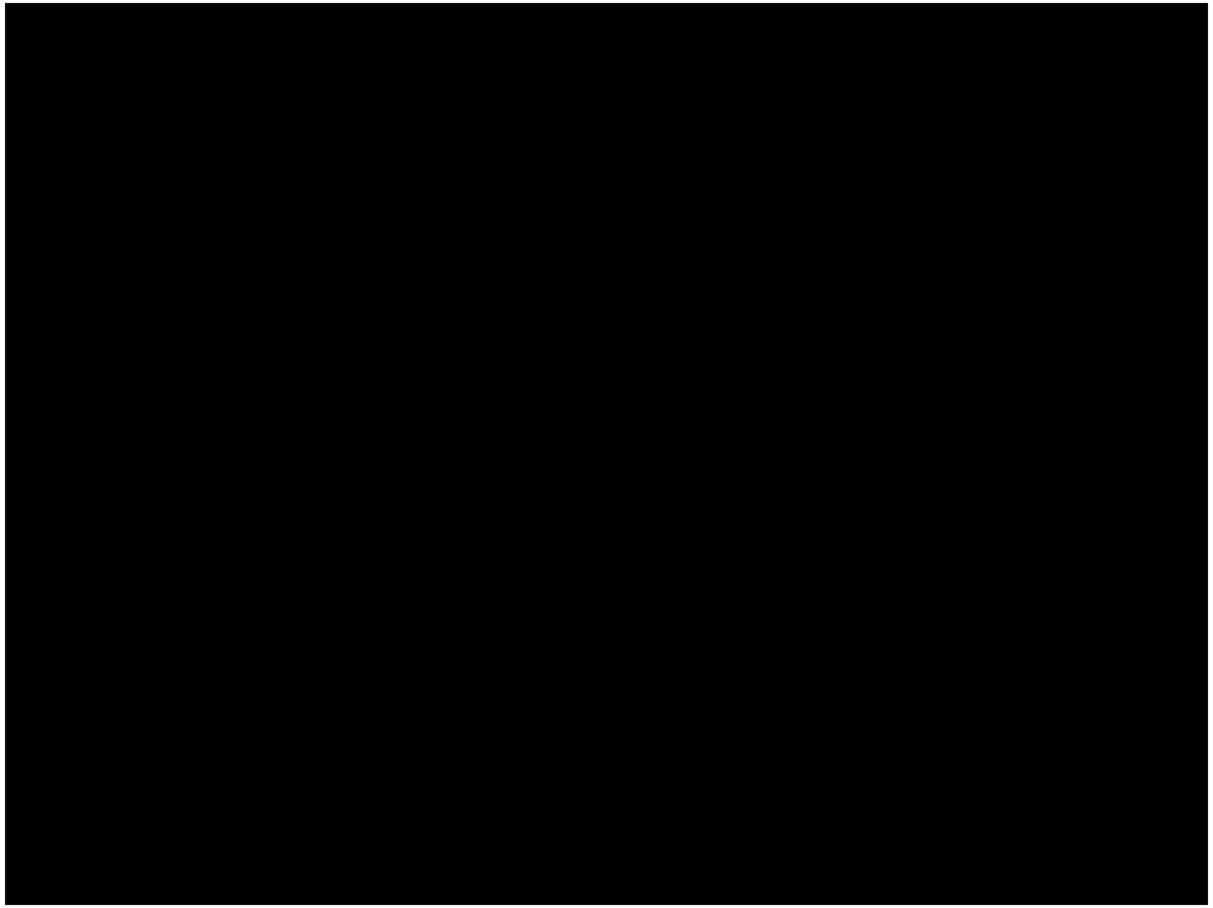


[3]

TossingBot - Structure



[3]

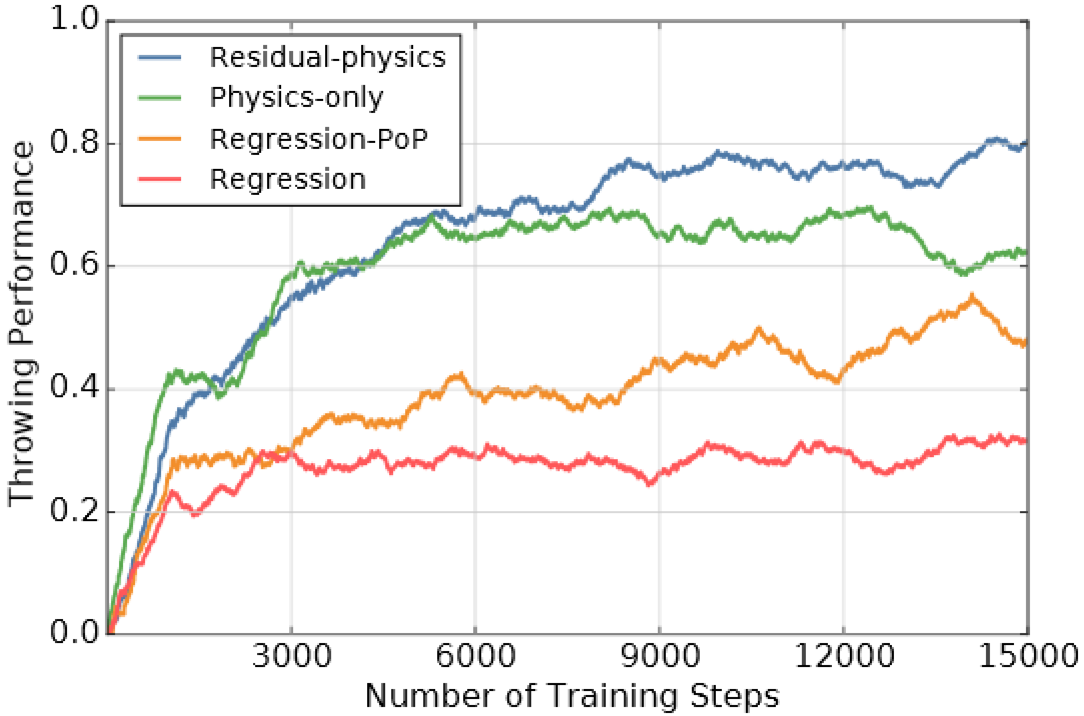


[4]

TossingBot - Results

GRASPING AND THROWING PERFORMANCE IN REAL (MEAN %)

Method	Grasping Seen	Grasping Unseen	Throwing Seen	Throwing Unseen
Human-baseline	–	–	–	80.1±10.8
Regression-PoP	83.4	75.6	54.2	52.0
Physics-only	85.7	76.4	61.3	58.5
Residual-physics	86.9	73.2	**84.7**	**82.3**



[3]

Conclusion

- Robot learning has long history with various methods
- Challenging task, additionally constrained by physical limitations
- LWR successful approach, already at early years
  - Has limitations: More data increases lookup-time
- Today: new approaches using Deep learning and hybrid methods

Sources

(1) Stefan Schaal and Christopher G. Atkeson: “Robot Juggling: An Implementation of Memory-based Learning”

(2) https://www.youtube.com/user/cga1959/videos (Chris Atkeson)
   (1) A Robot Learning Devil Sticking https://www.youtube.com/watch?v=KZdBBKgOyBg
   (2) 3 ball juggling and devil sticking by a robot https://www.youtube.com/watch?v=pKJEbs64Y2o
   (3) Sarcos Dextrous Arm one ball paddle juggling https://www.youtube.com/watch?v=rFHjHUqyp-I

(3) “TossingBot: Learning to Throw Arbitrary Objects with Residual Physics” https://arxiv.org/abs/1903.11239

(4) https://tossingbot.cs.princeton.edu/

(5) www.cs.cmu.edu/~cga/bighero6 (build-baymax.org)

(6) Wikipedia: Big Hero 6 (film) https://en.wikipedia.org/w/index.php?title=Big_Hero_6_(film)&oldid=902898498

Bonus slides

Fun fact

- Chris Atkeson's work was inspiration

[5]

[6]

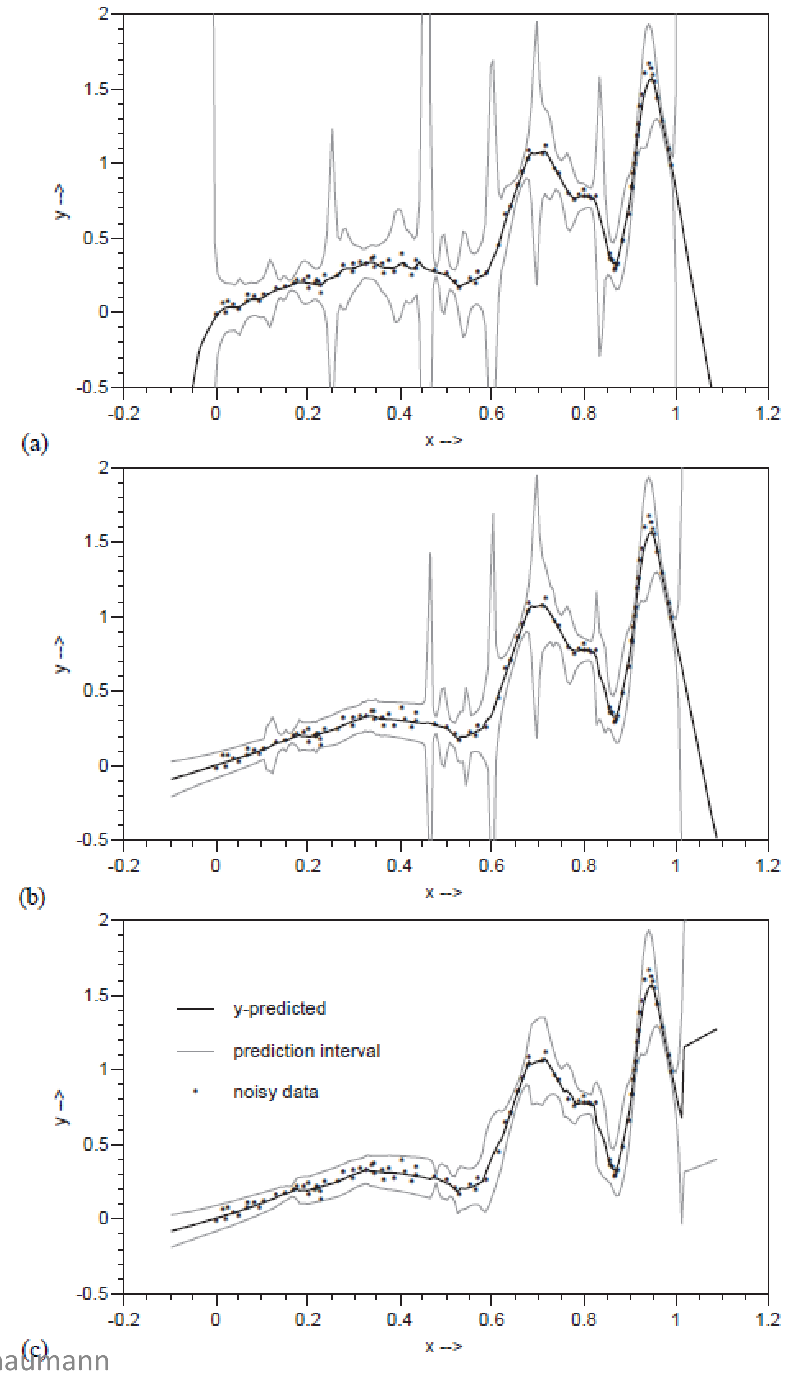
Implementation of LWR

- 33 MHz Intel i860 microprocessor
- Peak computation rate: 66 MFlops (effective comp. rate: 20 MFlops)
  - $n = 10$ inputs, $o = 5$ outputs
- Lookup time ≈ 15 ms on database of $m = 1000$ points

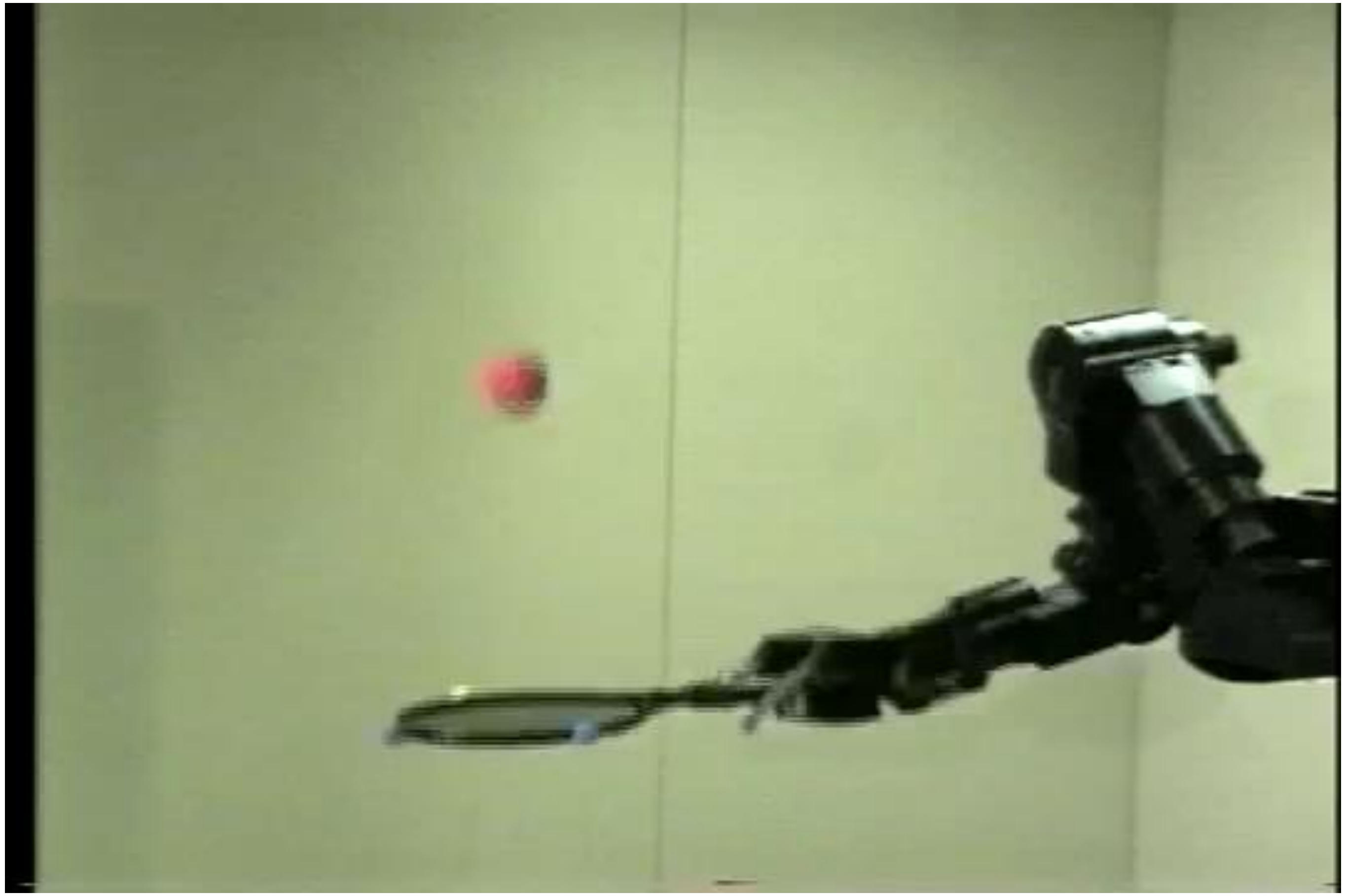
Optimizing LWR

The LWR fit was optimized using different measures:

a) Global cross validation
b) Local cross validation
c) Local prediction intervals



[1]



[3.3]