



Introduction to:

# Stable Estimator of Dynamical Systems (SEDS)

*Mohammad Khansari*

*PhD in Robotics,  
MSc in Aerospace Eng.*

*Laboratoire d'Algorithmes et Systemes d'Apprentissage (LASA)  
École Polytechnique Fédérale de Lausanne (EPFL), Switzerland*

February 12<sup>th</sup>, 2013



# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# Introduction

## ➤ Generating robot point-to-point movements

Shaping behavior, Easy to use

- Optimization/Planning

*(Lozano-Prez & Wesley, 1979;  
Kavraki et al., 1996;  
Kuffner & LaValle, 2000  
Toussaint, 2009)*

- Potential/Navigation functions

*(Khatib, 1986;  
Koditschek, 1987;  
Kim & Khosla, 1992;  
Feder & Slotine, 1997;  
Lindemann & LaValle, 2005)*

Reactivity, Robustness

- Imitation Learning

*(Andersson, 1989;  
Schaal & Atkeson, 1994;  
Inamura et al. 2002;  
Calinon et al., 2007;  
Coates et al., 2008)*

- Movement Primitives

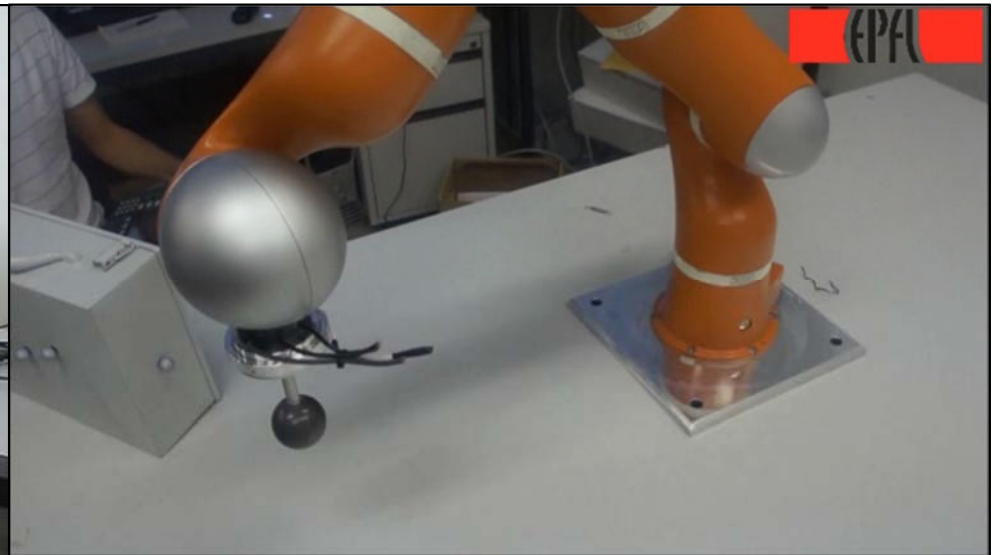
*(Ijspeert et al., 2002;  
Hersch et al., 2008;  
Pastor et al., 2009;  
Bitzer & Vijayakumar, 2009;  
Kober et al., 2010)*

Modularity, Convergence



# Objective

- We seek to design an all-encompassing framework with:
- Instant Adaptation to dynamic environments (reactive)
  - Inherent robustness to perturbations (robust)
  - Guaranteed convergence to the target (stable)
  - Ability to customize the robot motion (customizable)

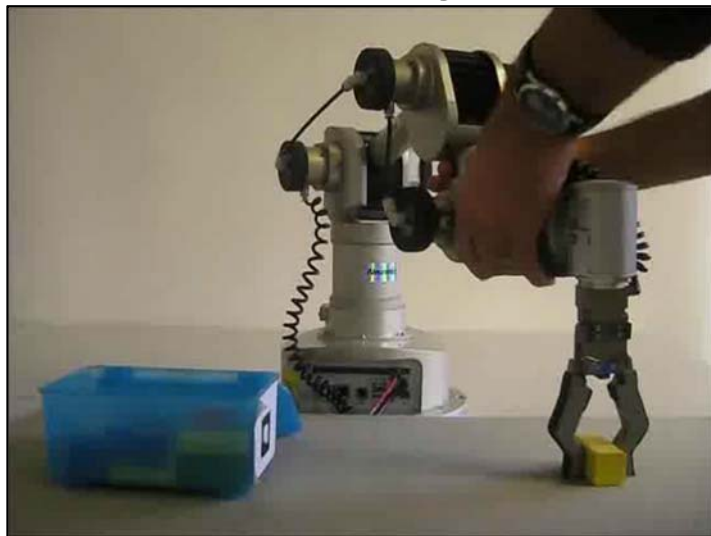




# Approach

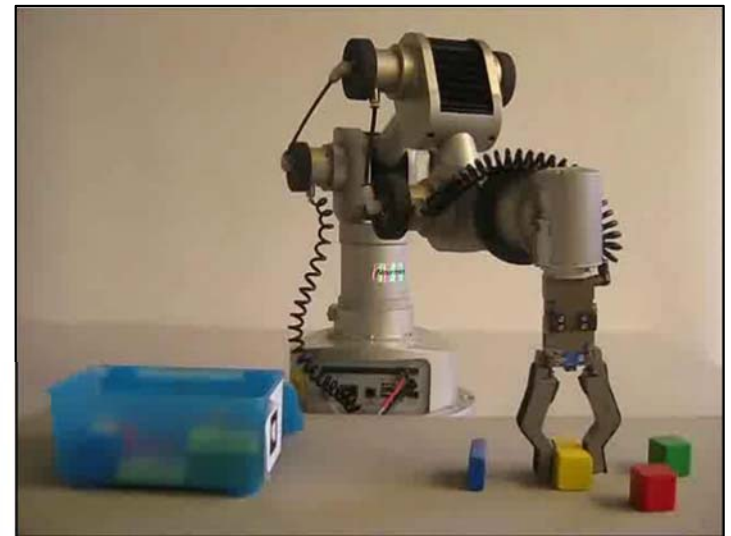
- SEDS is based on two main pillars:  
*Imitation Learning & Dynamical Systems*
- Imitation Learning:
  - Provides an intuitive way to transmit skills

Teaching



Model?  
Learning?

Reproduction





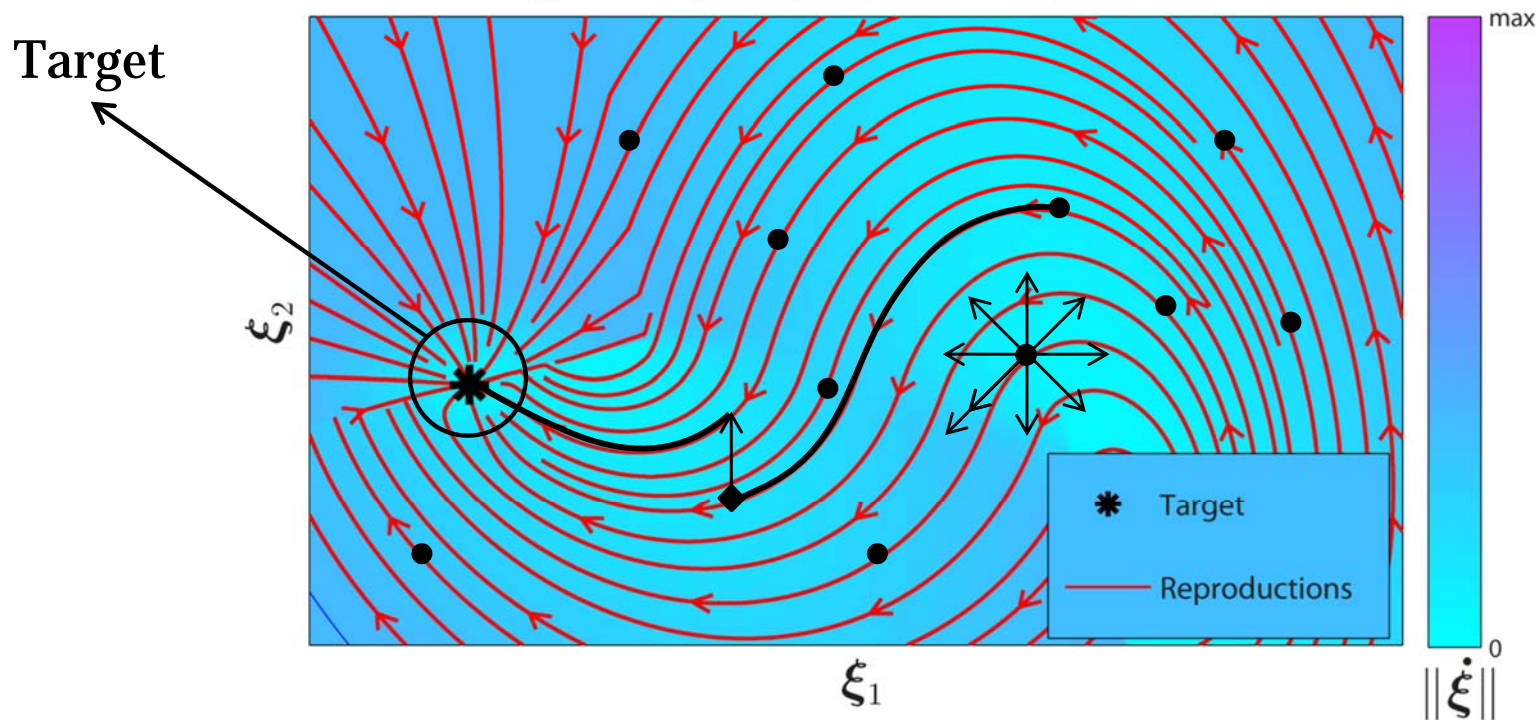
# Approach

Dynamical Systems to encode robot motions:

$$\dot{\xi} = f(\xi)$$

└─┬─> **Multidimensional Kinematic Variable:**  
e.g. End-effector position/orientation,  
joint angles

Streamlines of a *globally asymptotically stable* autonomous DS

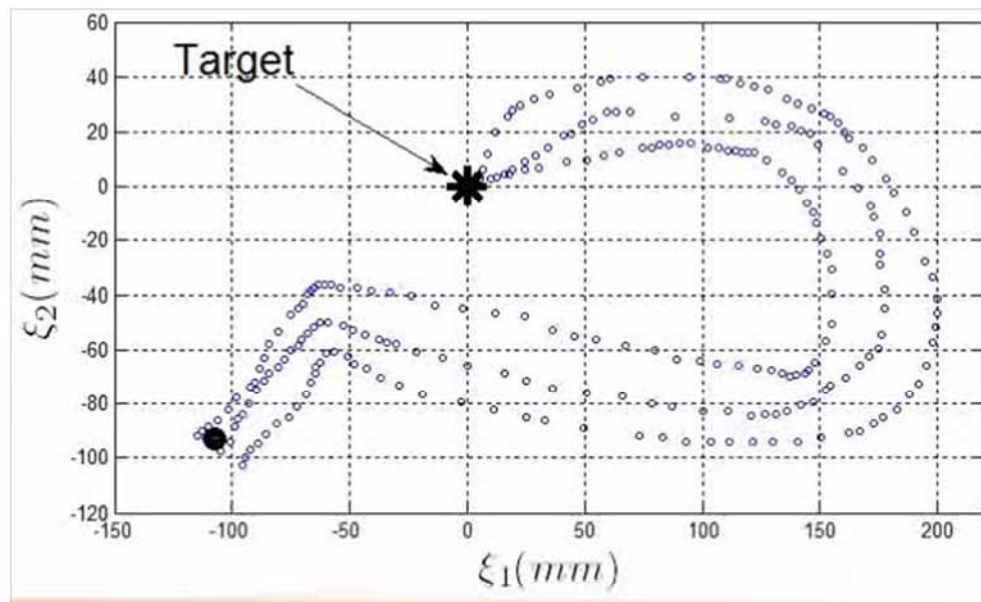




# Challenge

## Challenge:

- How to obtain such a model?
- Take a probabilistic approach (imitation learning)
- Existing regression techniques do not ensure stability



How to build a globally stable DS from demonstrations?

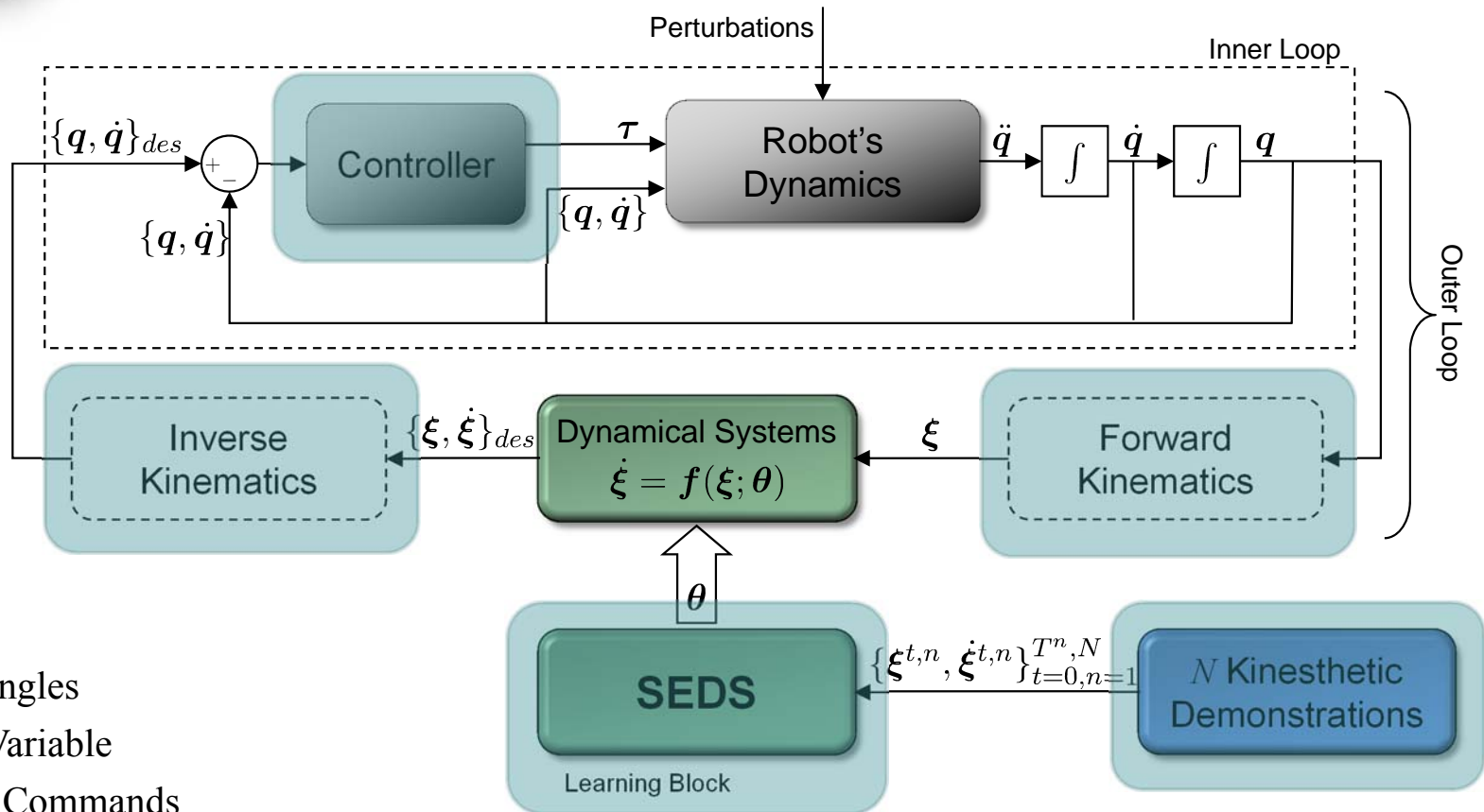


# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# SEDS\*: System's Architecture



$q$  : Joint Angles  
 $\xi$  : State Variable  
 $\tau$  : Motor Commands

➤ Instant refinement/adaptation of the motion

\* *Khansari-Zadeh and Billard (2011), TRO*



# SEDS: Function Encoding

- Probabilistic estimation of DS
- SEDS uses Gaussian mixture regression

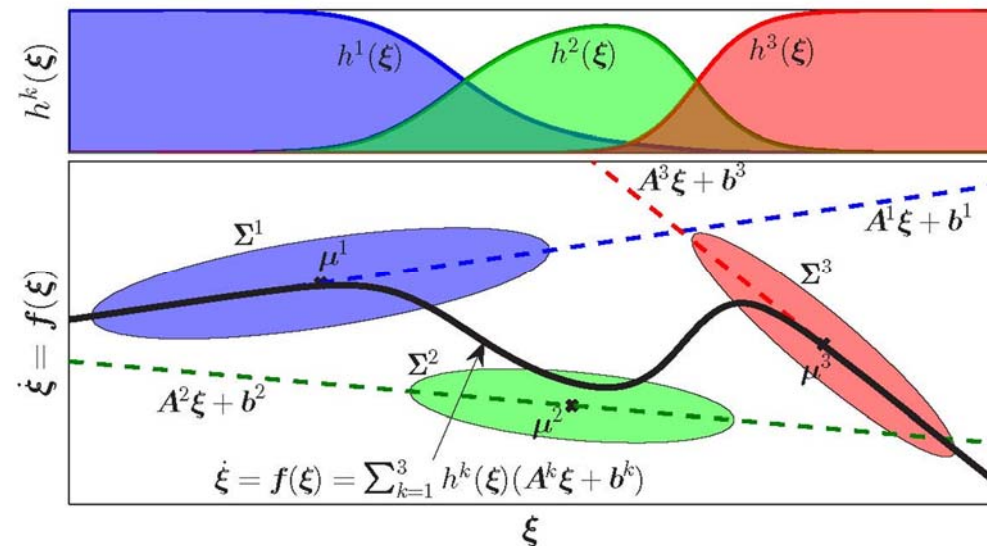
$$\dot{\xi} = f(\xi; \theta) = \sum_{k=1}^K \underbrace{\frac{\mathcal{P}(k)\mathcal{P}(\xi|k)}{\sum_{i=1}^K \mathcal{P}(i)\mathcal{P}(\xi|i)}}_{h^k(\xi)} \underbrace{\left( \Sigma_{\xi\xi}^k (\Sigma_{\xi\xi}^k)^{-1} \xi + \mu_{\xi}^k - \Sigma_{\xi\xi}^k (\Sigma_{\xi\xi}^k)^{-1} \mu_{\xi}^k \right)}_{A^k \xi + b^k}$$

$$\Rightarrow \dot{\xi} = f(\xi; \theta) = \sum_{k=1}^K h^k(\xi) (A^k \xi + b^k)$$

Learning parameters:

$$\theta = \{\pi^1 \dots \pi^K, \mu^1 \dots \mu^K, \Sigma^1 \dots \Sigma^K\}$$

Defining value of  $\theta$   
under stability constraints





# SEDS: Stability Theorem

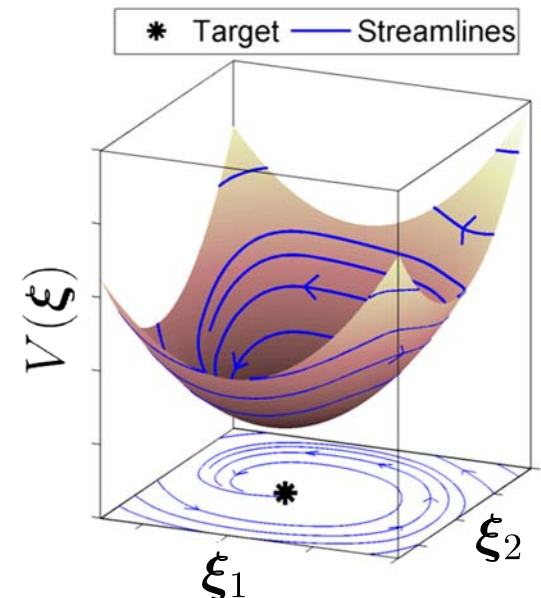
- Stability conditions:

$$\begin{cases} \text{(a)} \quad \mu_{\xi}^k = \Sigma_{\xi\xi}^k (\Sigma_{\xi}^k)^{-1} (\mu_{\xi}^k - \xi^*) \\ \text{(b)} \quad \Sigma_{\xi\xi}^k (\Sigma_{\xi}^k)^{-1} + (\Sigma_{\xi}^k)^{-1} (\Sigma_{\xi\xi}^k)^T \prec 0 \end{cases} \quad \forall k = 1..K$$

These stability conditions only depend on the *GMR parameters*

Can be quickly verified without performing any numerical analysis

- Derived from Lyapunov stability theorem
- Based on a quadratic energy function





# SEDS: Learning Algorithm

**Input:**  $N$  demonstrations  $\{\xi^{t,n}, \dot{\xi}^{t,n}\}_{t=0, n=1}^{T^n, N}$

**Solve:**

$$\min_{\theta} J(\theta) = \frac{1}{2\mathcal{T}} \sum_{n=1}^N \sum_{t=0}^{T^n} \underbrace{(f(\xi^{t,n}; \theta) - \dot{\xi}^{t,n})^T (f(\xi^{t,n}; \theta) - \dot{\xi}^{t,n})}_{\text{Minimizing discrepancy between demonstrations and reproductions}}$$

**Subject to:**

Minimizing discrepancy between demonstrations and reproductions

From stability conditions

$$\left\{ \begin{array}{l} \text{(a)} \quad \mu_{\dot{\xi}}^k = \Sigma_{\dot{\xi}\xi}^k (\Sigma_{\xi}^k)^{-1} (\mu_{\xi}^k - \xi^*) \\ \text{(b)} \quad \Sigma_{\dot{\xi}\xi}^k (\Sigma_{\xi}^k)^{-1} + (\Sigma_{\xi}^k)^{-1} (\Sigma_{\dot{\xi}\xi}^k)^T \prec 0 \\ \text{(c)} \quad \Sigma^k \succ 0 \\ \text{(d)} \quad 0 < \pi^k \leq 1 \\ \text{(e)} \quad \sum_{k=1}^K \pi^k = 1 \end{array} \right. \quad \forall k \in 1..K$$

Imposed by the nature of the mixture model



# SEDS: Learning Algorithm

**Input:**  $N$  demonstrations  $\{\xi^{t,n}, \dot{\xi}^{t,n}\}_{t=0, n=1}^{T^n, N}$

**Solve:**

$$\min_{\theta} J(\theta) = \underbrace{-\frac{1}{\mathcal{T}} \sum_{n=1}^N \sum_{t=0}^{T^n} \log \mathcal{P}([\xi^{t,n}; \dot{\xi}^{t,n}] | \theta)}_{\text{Maximizing the likelihood}}$$

**Subject to:**

Maximizing the likelihood

$$\left\{ \begin{array}{l} \text{(a)} \quad \mu_{\dot{\xi}}^k = \Sigma_{\dot{\xi}\xi}^k (\Sigma_{\xi}^k)^{-1} (\mu_{\xi}^k - \xi^*) \\ \text{(b)} \quad \Sigma_{\dot{\xi}\xi}^k (\Sigma_{\xi}^k)^{-1} + (\Sigma_{\xi}^k)^{-1} (\Sigma_{\dot{\xi}\xi}^k)^T \prec 0 \\ \text{(c)} \quad \Sigma^k \succ 0 \\ \text{(d)} \quad 0 < \pi^k \leq 1 \\ \text{(e)} \quad \sum_{k=1}^K \pi^k = 1 \end{array} \right. \quad \forall k \in 1..K$$



# SEDS: Learning Algorithm

**Input:**  $N$  demonstrations  $\{\xi^{t,n}, \dot{\xi}^{t,n}\}_{t=0, n=1}^{T^n, N}$

**Solve:**

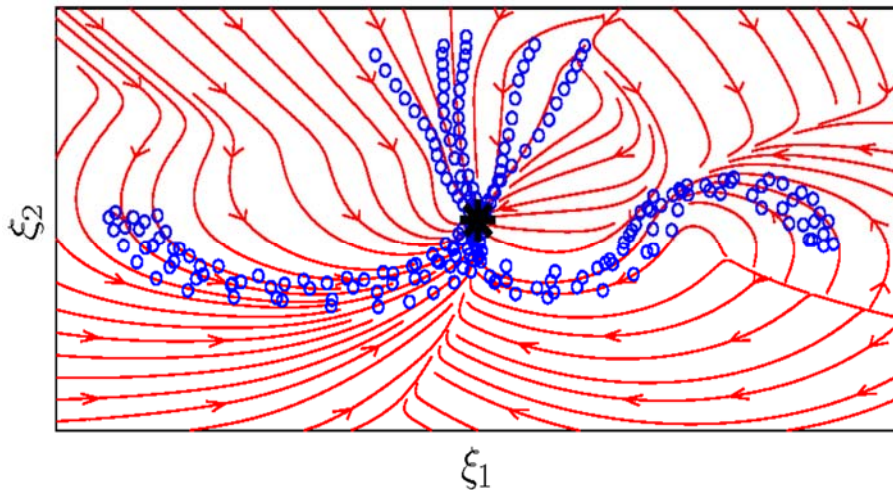
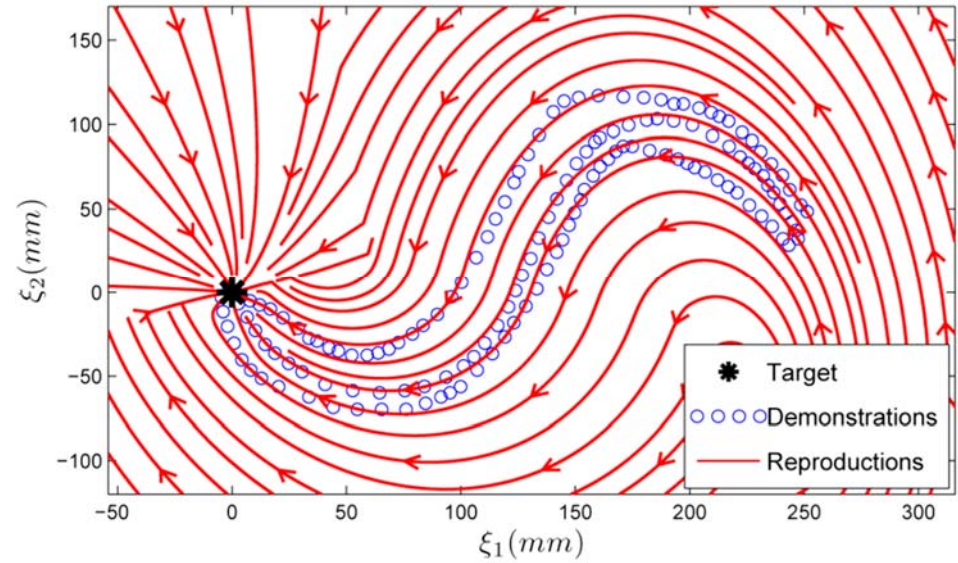
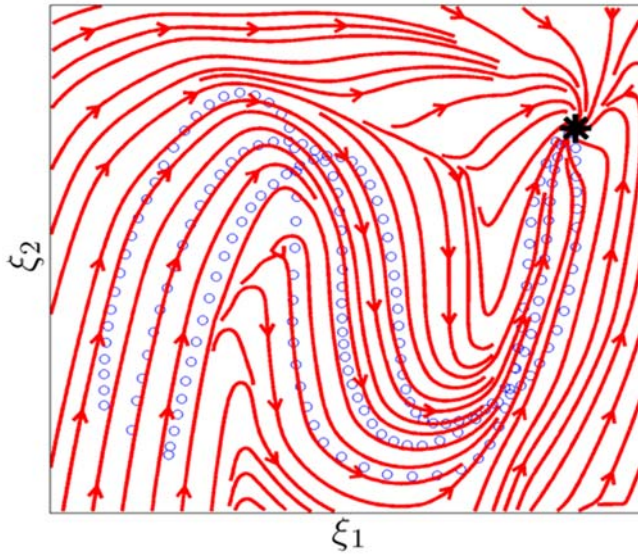
$$\min_{\theta} J(\theta) = \underbrace{-\frac{1}{\mathcal{T}} \sum_{n=1}^N \sum_{t=0}^{T^n} \frac{(\dot{\xi}^{t,n})^T f(\xi^{t,n}; \theta)}{\|\dot{\xi}^{t,n}\| \|f(\xi^{t,n}; \theta)\|}}_{\text{Minimizing the angle between demonstrations and estimation}}$$

**Subject to:**

$$\left\{ \begin{array}{l} \text{(a)} \mu_{\dot{\xi}}^k = \Sigma_{\dot{\xi}\xi}^k (\Sigma_{\xi}^k)^{-1} (\mu_{\xi}^k - \xi^*) \\ \text{(b)} \Sigma_{\dot{\xi}\xi}^k (\Sigma_{\xi}^k)^{-1} + (\Sigma_{\xi}^k)^{-1} (\Sigma_{\dot{\xi}\xi}^k)^T \prec 0 \\ \text{(c)} \Sigma^k \succ 0 \\ \text{(d)} 0 < \pi^k \leq 1 \\ \text{(e)} \sum_{k=1}^K \pi^k = 1 \end{array} \right. \quad \forall k \in 1..K$$



# SEDS: Theoretical Examples



- Integrating different motions into one single dynamics
- Enabling the robot to switch from one behavior to another



# SEDS: Robot Experiment

- Putting a sugar block into a cup of coffee





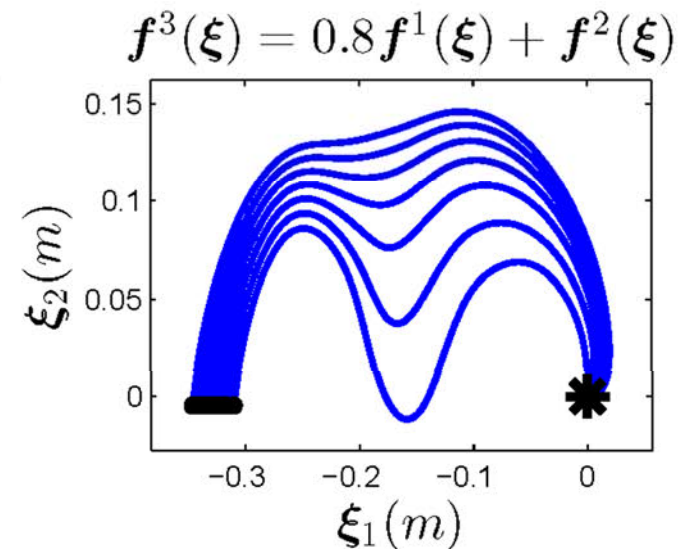
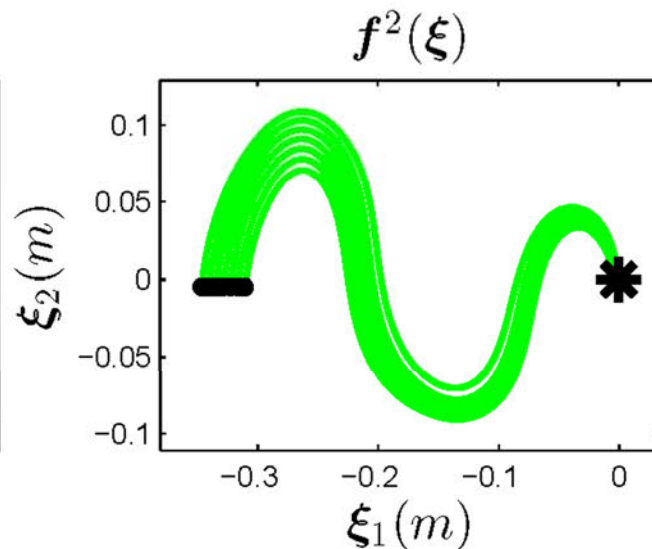
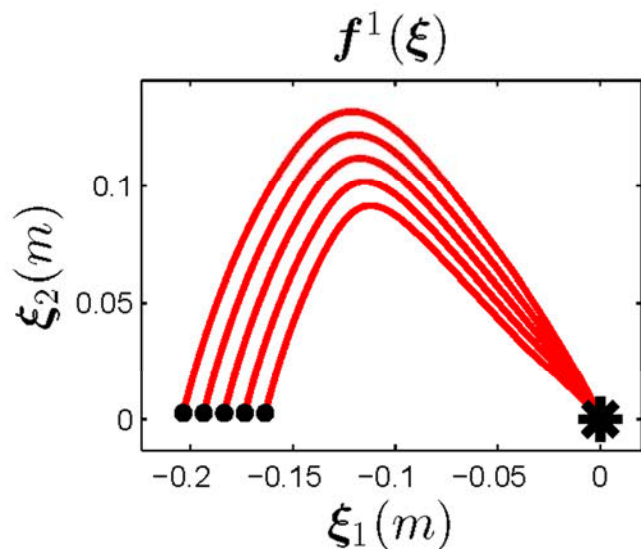
# SEDS: Mathematical Properties

➤ If  $\dot{\xi} = f(\xi)$  is SEDS, and  $\alpha > 0$  is a real number

➡  $\dot{\xi} = \alpha f(\xi)$  is also SEDS

➤ Consider  $M$  SEDS functions:  $\dot{\xi} = f^m(\xi)$ ,  $i \in 1..M$

➡  $\dot{\xi} = \sum_{m=1}^M \alpha^m f^m(\xi)$  is SEDS, provided  $\alpha^m > 0$

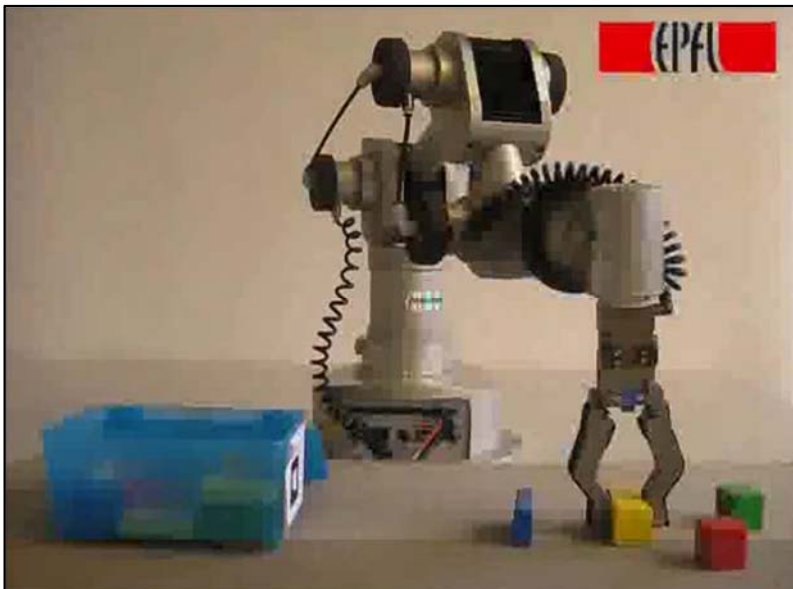




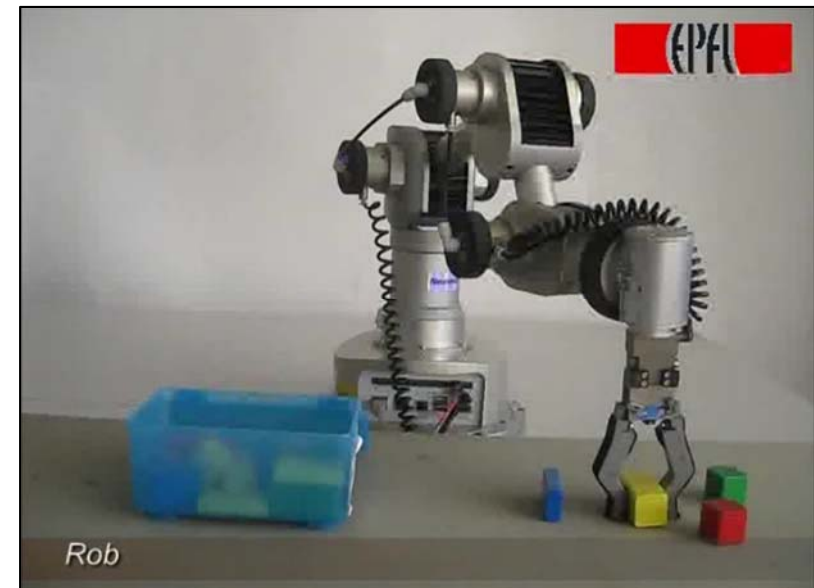
# SEDS: Robot Experiment

- Putting blocks into a container
  - End-effector's position control
  - On 6-DoF Katana-T robot

Generalization of the task



Instant adaptation to changes

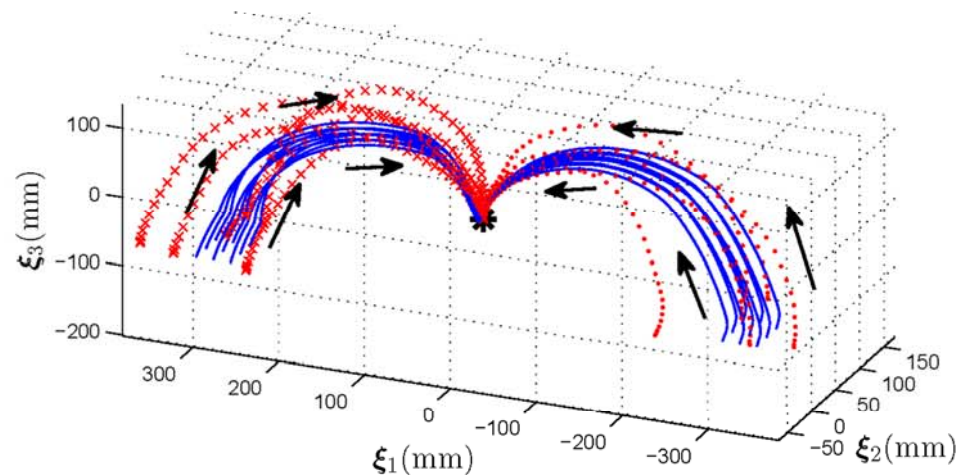
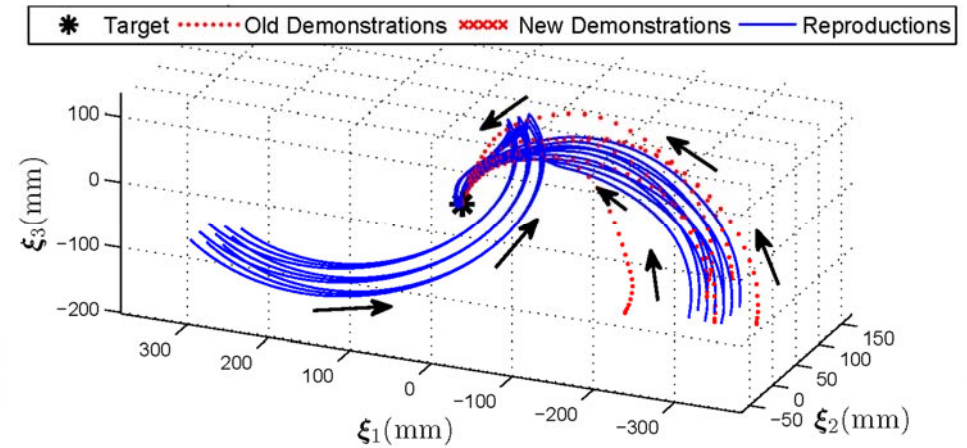




# SEDS: Robot Experiment

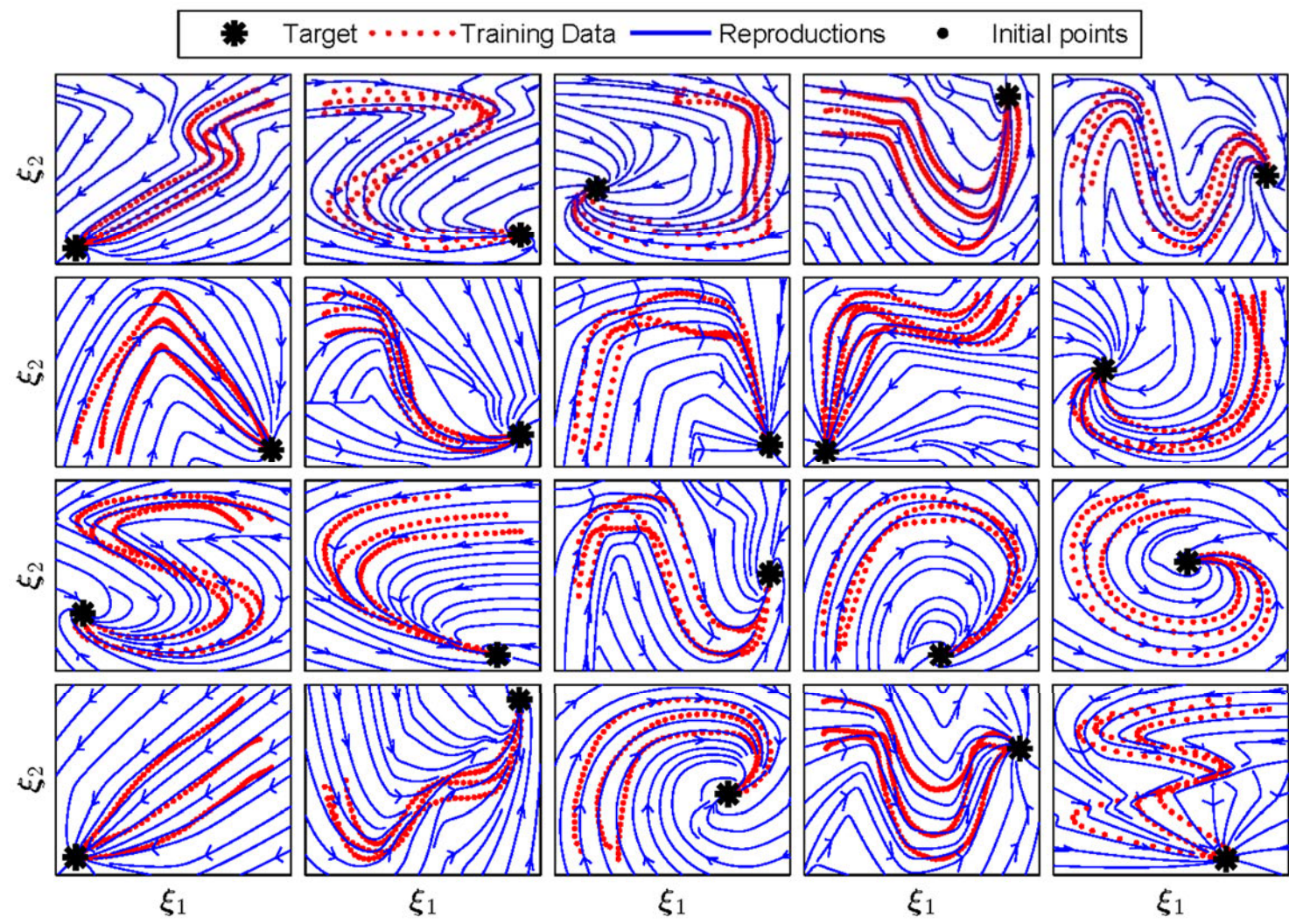
➤ What would happen if we go extremely far from demonstrations?

- Convergence to the target is ensured
- There is lack of information
- The robot behavior can be corrected
- Retraining is required after adding new data points





# Experiments with a library of 20 human's 2D hand motions





# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# SEDS Library

- SEDS source code is available online:

<http://lasa.epfl.ch/sourcecode>

- SEDS Package include:

- GMR\_lib: Computes  $\xi$  at a query point  $\xi$  for a given mixture model
- SEDS\_lib: Learn a stable DS model from the provided demonstrations (depend on GMR library).
- SEDS\_Cpp\_lib: A ROS package to use SEDS in realtime control
- Extensions: Extensions to SEDS developed by myself or others
- Sample models: A library of 24 human handwriting motions
- Example files: showing how to use SEDS



# SEDS: MATLAB Package

Data  
Collection

Storing the set of demonstrations trajectories in a SEDS compatible format

Data  
Preprocessing

```
[x0 , xT, Data, index] = preprocess_demos(demosTraj,demosTime,...  
                                          tol_cutting);
```

SEDS:  
Initialization

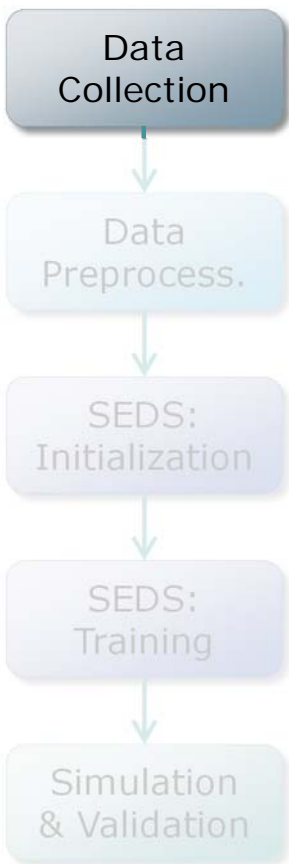
```
[Priors_0 , Mu_0, Sigma_0] = initialize_SEDS(Data,K);
```

SEDS:  
Training

```
[Priors Mu Sigma]=SEDS_Solver(Priors_0 ,Mu_0,Sigma_0 ,Data,options);
```

Simulation &  
Validation

```
[x xd]=Simulation(x0_all ,xT,fn_handle ,opt_sim);
```



Initial point  $\leftarrow$   $\xrightarrow{\hspace{10em}}$  Final point

$$\mathbf{X} = [\boldsymbol{\xi}^0 \quad \boldsymbol{\xi}^1 \quad \dots \quad \boldsymbol{\xi}^T]$$

$$\mathbf{t} = [t^0 \quad t^1 \quad \dots \quad t^T]$$

$\xrightarrow{\hspace{10em}}$   
Time increases

$$\begin{array}{ll} demosTraj\{1\} = \mathbf{X}^1 & demosTime\{1\} = \mathbf{t}^1 \\ demosTraj\{2\} = \mathbf{X}^2 & demosTime\{2\} = \mathbf{t}^2 \\ & \vdots \\ demosTraj\{N\} = \mathbf{X}^N & demosTime\{N\} = \mathbf{t}^N \end{array}$$



# Preprocessing Data

```
[x0 , xT, Data, index] = preprocess_demos(demosTraj,demosTime,tol_cutting);
```

Demonstrations  
Trajectories

Trimming  
Threshold

Demonstrations  
Time

Data  
Collection

Data  
Preprocess.

SEDS:  
Initialization

SEDS:  
Training

Simulation  
& Validation

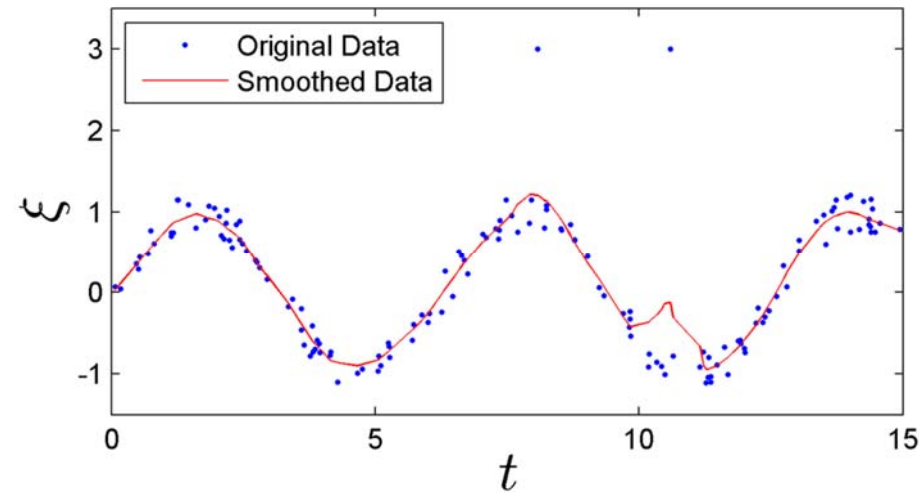
## ➤ Functionality:

- Smoothing trajectories
- Computing the velocity
- Trimming demonstrations
- Transforming all demonstrations to the target frame of reference
- Merging all demonstrations into a single matrix



# Preprocessing Data

- Trajectory smoothing:
  - It is not obligatory, but useful!
  - Uses MATLAB built-in *smooth* function.



`[x0 , xT, Data, index] = preprocess_demos(demosTraj, demosTime, tol_cutting);`

- Velocity Computation:

- Fixed time step:

$$\dot{\mathbf{X}} = \frac{1}{\delta t} [(\boldsymbol{\xi}^1 - \boldsymbol{\xi}^0) \quad (\boldsymbol{\xi}^2 - \boldsymbol{\xi}^1) \quad \dots \quad (\boldsymbol{\xi}^T - \boldsymbol{\xi}^{T-1}) \quad \mathbf{0}]$$

- Variable time step:

$$\dot{\mathbf{X}} = \left[ \frac{\boldsymbol{\xi}^1 - \boldsymbol{\xi}^0}{t^1 - t^0} \quad \frac{\boldsymbol{\xi}^2 - \boldsymbol{\xi}^1}{t^2 - t^1} \quad \dots \quad \frac{\boldsymbol{\xi}^T - \boldsymbol{\xi}^{T-1}}{t^T - t^{T-1}} \quad \mathbf{0} \right]$$

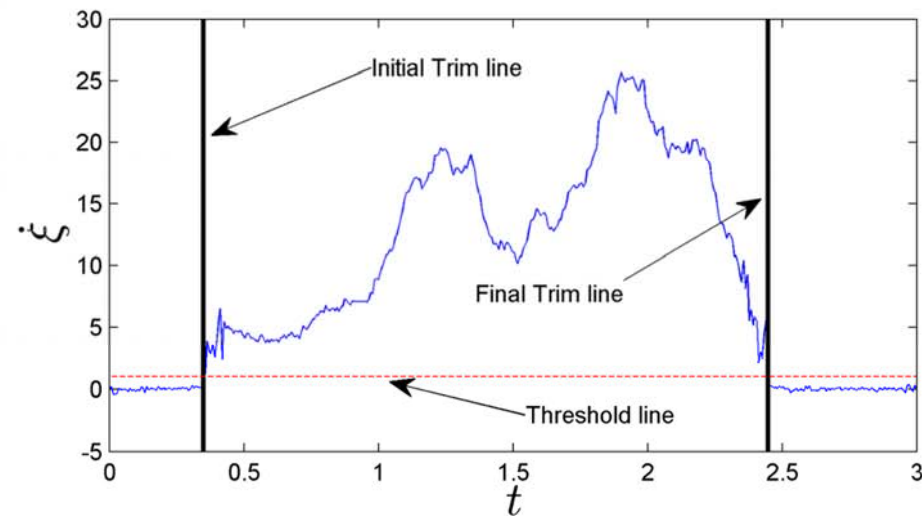
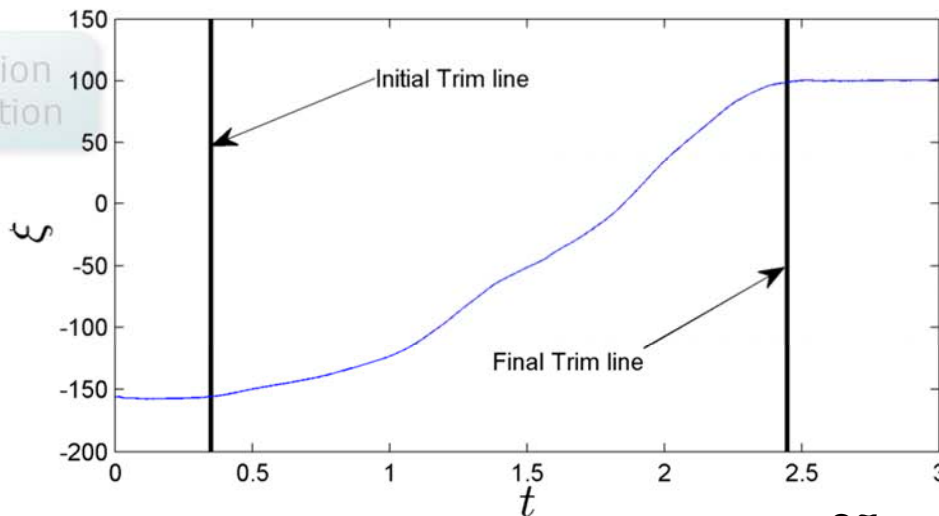
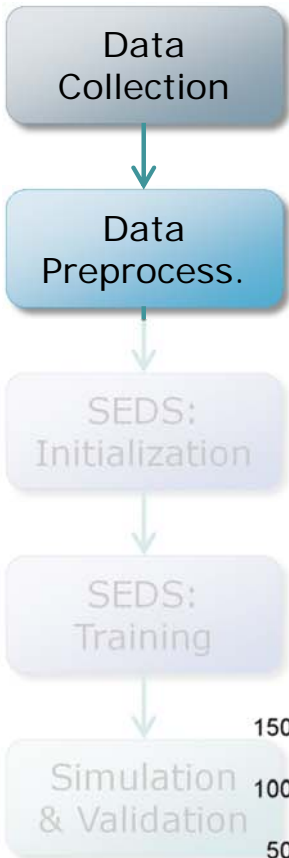


# Preprocessing Data

```
[x0 , xT, Data, index] = preprocess_demos(demosTraj, demosTime, tol_cutting);
```

## ➤ Demonstrations Trimming:

- Essential to get the best performance
- Many useless points at the beginning and end of demos
- Without trimming
  - ➔ Put more priority to the initial and final parts
- This step should be done after smoothing
- To trim, the user needs to define a threshold.





# Preprocessing Data

- Transformation to the target frame of reference:
  - Target will be transformed to the origin, i.e.  $\xi^* = \xi^T = 0$
  - Significantly reduces the required computation power
  - Makes all demonstrations consistent!
  - Do not impose any limitation

$$\tilde{\mathbf{X}}^m = [(\xi^{0,m} - \xi^{T,m}) \quad (\xi^{1,m} - \xi^{T,m}) \quad \dots \quad (\xi^{T-1,m} - \xi^{T,m}) \quad 0]$$

- Storing some information about the original trajectories

`[x0 , xT, Data, index] = preprocess_demos (...);`

$$\bar{\xi}^0 = \frac{1}{M} \sum_{m=1}^M \xi^{0,m}$$

Average initial points of all trajectories

$$\bar{\xi}^T = \frac{1}{M} \sum_{m=1}^M \xi^{T,m}$$

Average final points of all trajectories



# Preprocessing Data

## ➤ Merging all demonstrations

$[x_0, x_T, \text{Data}, \text{index}] = \text{preprocess\_demos}(\dots);$

$$D = \begin{bmatrix} \tilde{X}^1 & \tilde{X}^2 & \dots & \tilde{X}^M \\ \dot{X}^1 & \dot{X}^2 & \dots & \dot{X}^M \end{bmatrix}$$

## ➤ Keeping the track of where each demonstration is

- Good only for visualization and debugging purposes  
(otherwise, the order of datapoints are not important for SEDS)

$[x_0, x_T, \text{Data}, \text{index}] = \text{preprocess\_demos}(\dots);$

$$I = \begin{bmatrix} i_1 = 1 & i_2 & i_3 & \dots & i_M & i_{M+1} \end{bmatrix}$$

$$D = \begin{bmatrix} | & \tilde{X}^1 & | & \tilde{X}^2 & | & \dots & | & \tilde{X}^M & | \\ | & \dot{X}^1 & | & \dot{X}^2 & | & \dots & | & \dot{X}^M & | \end{bmatrix}$$



# Initialization

```
[ Priors_0 , Mu_0, Sigma_0] = initialize_SEDS (Data,K);
```

Data  
Collection

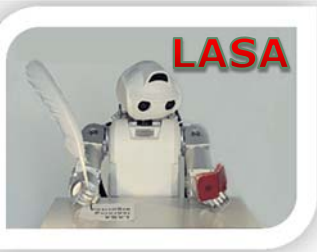
Data  
Preprocess.

SEDS:  
Initialization

SEDS:  
Training

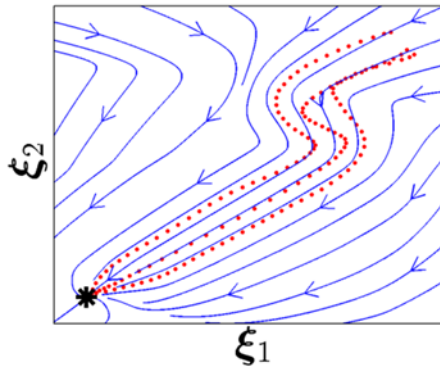
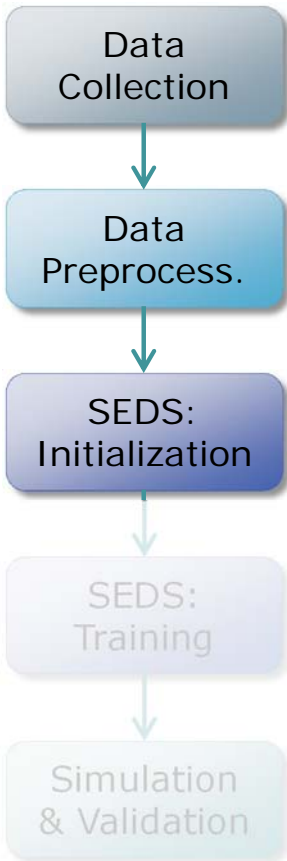
Simulation  
& Validation

- GMM is accurate, but unstable!
- SEDS can be initialized from GMM
- In GMM, only one parameter should be specified:  
 $K$  : Number of Gaussian functions
- How  $K$  should be chosen?
  - Machine learning approach: Use Bayesian Information Criterion
  - Engineering approach:  
Start with  $K=2$ , incrementally increase it until you're happy
  - Intuition + Engineering approach (inspired from DS analysis):
    - Consider one Gaussian function per curvature
    - Incrementally increase it until you're happy

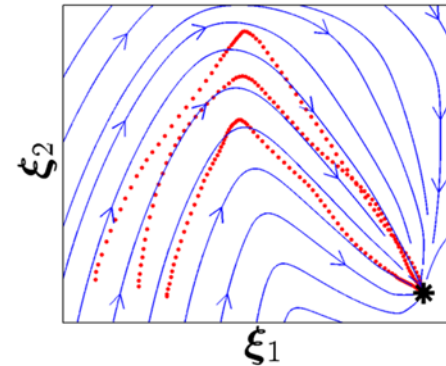


# Initialization

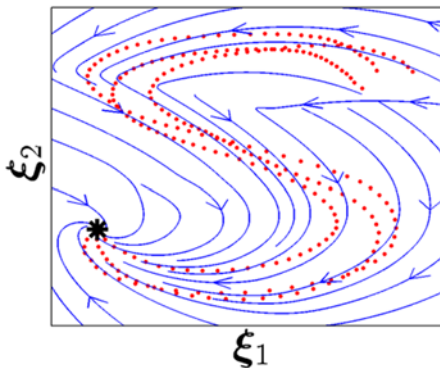
I would start with:



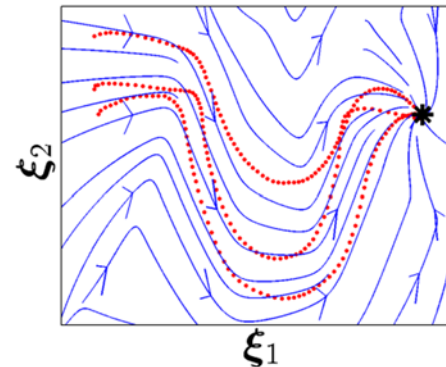
$K = 3$



$K = 2$

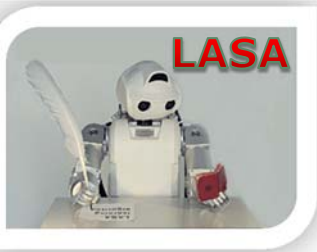


$K = 4$



$K = 5$

- Most motions can be modeled with  $K < 10$
- Big  $K \rightarrow$  Overfitting, Slower training time



# Training

```
[ Priors Mu Sigma]=SEDS_Solver( Priors_0 ,Mu_0,Sigma_0 ,Data ,options );
```

- Optimization using the MATLAB function *fmincon*
- Optimization method: *Interior Point Algorithm*
- Options (only important ones are described here):
  - `options.i_max`: Maximum number of iterations [*def.= 1000*]
  - This parameter is directly passed to *fmincon*.
  - `options.objective`: [*def.= 'mse'*]
    - MSE: Accurate, fast, and very robust
    - Likelihood: Accurate, slightly slow, but allows to compute the confidence value
    - Direction: Useful for learning hitting motions, especial care should be taken when using this model in simulation.



# Training

➤ `options.criterion`: Criterion to evaluate the negative definiteness of a matrix [*def.*= '*eigenvalue*']

➤ *eigenvalue*: Very robust

➤ *principal\_minor*: Computationally faster (per iteration)

➤ `options.tol_mat_bias` [*def.*=  $1.0e-15$ ]

➤ A bias term used for verification of stability conditions

➤ *fmincon* cannot handle strict inequality constraints:

$$\mathbf{A}^k + (\mathbf{A}^k)^T \prec 0 \quad \text{One of the SEDS stability conditions}$$

➤ Alternatively:

$$\Rightarrow \mathbf{A}^k + (\mathbf{A}^k)^T \preceq -\sigma_{bias} \mathbf{I} \quad \text{where } 0 < \sigma_{bias} \ll 1$$



# Simulation

```
[x xd]=Simulation(x0,xT,fn_handle,opt_sim);
```

Initial Point(s) ←

Target Point ←

← A handle to  
GMR function

Data  
Collection

Data  
Preprocess.

SEDS:  
Initialization

SEDS:  
Training

Simulation  
& Validation

## ➤ Simultaneous simulation of several motions:

- Create a matrix by concatenating all the initial points

$$x0 = [x0\_1, \dots, x0\_m, \dots, x0\_M]$$

← The initial point of  
the m-th motion

## ➤ Simultaneous simulation of several motions:

- Use the default description

```
fn_handle = @(x) GMR(Priors,Mu,Sigma,x,1:d,d+1:2*d);
```

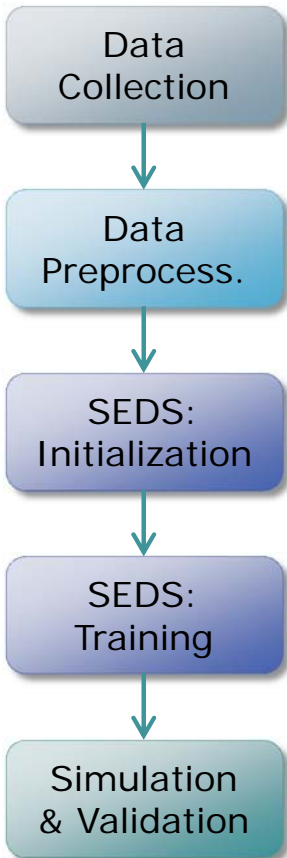
← Dimensionality of  
the DS



# Simulation

Simulation options (only important ones are described here):

- `opt_sim.i_max`: Maximum number of iterations [*def.=1000*]
- `opt_sim.dt`: Integration time step [*def.= 0.02*]
  - Not too small → Requires many iterations
  - Not too big → Causes oscillation around the target
- `opt_sim.tol`: Stopping criterion [*def.= 0.001*]
  - Stop the simulation if the distance to the target is less than the tolerance.
  - Note: stopping criterion should suit the unit of your data
    - If meter or radian →  $tol = 0.001$  is recommended
    - If millimeter →  $tol = 1.0$  is recommended





# SEDS: Final Notes

- Convention: When using a SEDS model, all computations are in the TARGET frame of reference!
- If the target is moved → The robot relative position to the target is changed

$$\tilde{\xi} = \xi - \xi^* \quad \text{Transformation to the target frame}$$

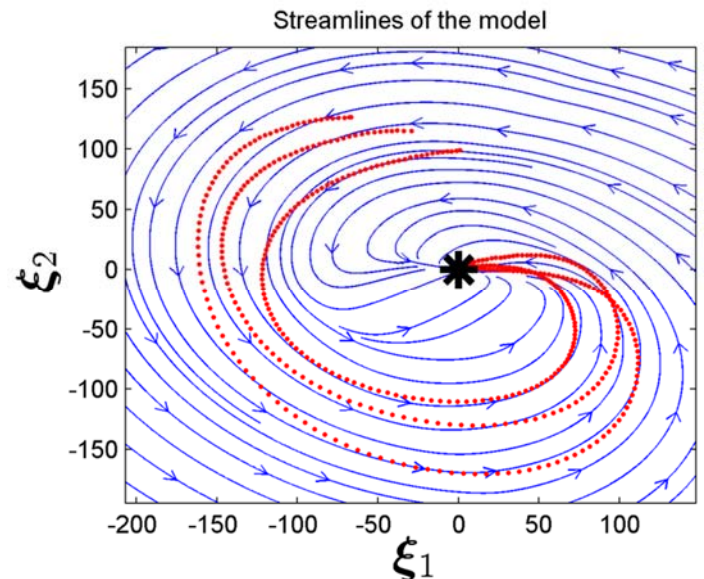
$$\dot{\tilde{\xi}} = f(\tilde{\xi});$$

$$\xi = \dot{\tilde{\xi}} \delta t + \tilde{\xi}$$

- Plotting streamlines of DS
  - Only for 2D motions

`plotStreamLines ( Priors , Mu, Sigma ,D)`

The axis range  
e.g. `D=[-200 140 -180 180]`

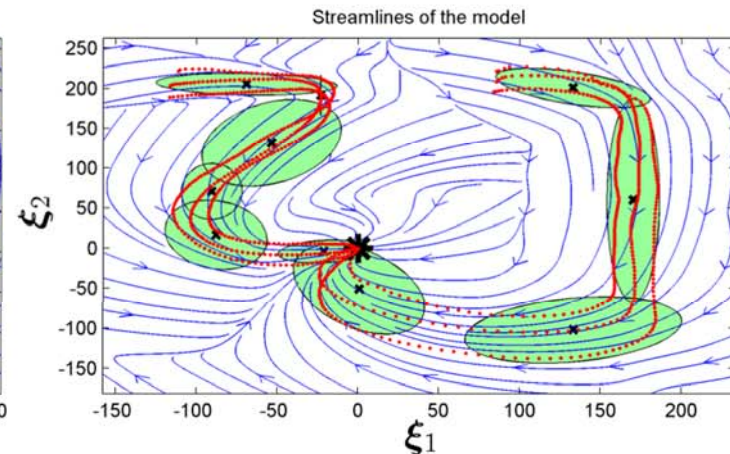
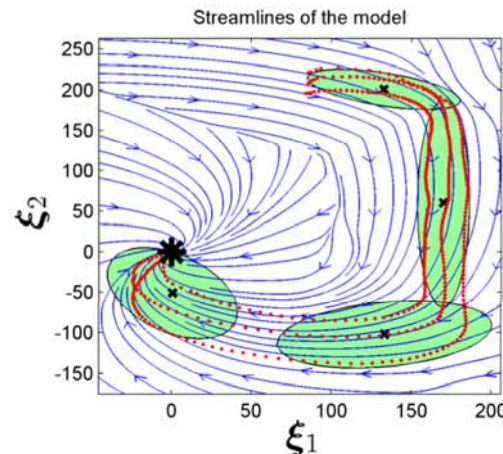
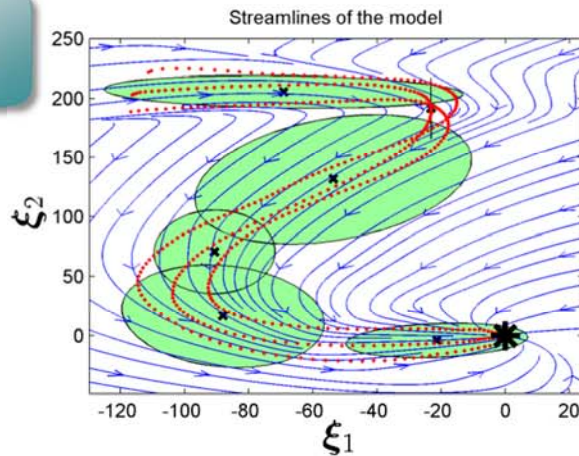




# SEDS: Final Notes

- Merging two different SEDS models:
  - Just by concatenating the parameters

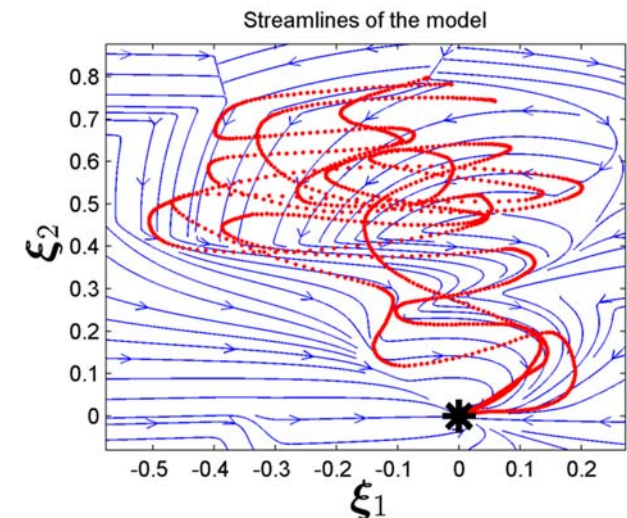
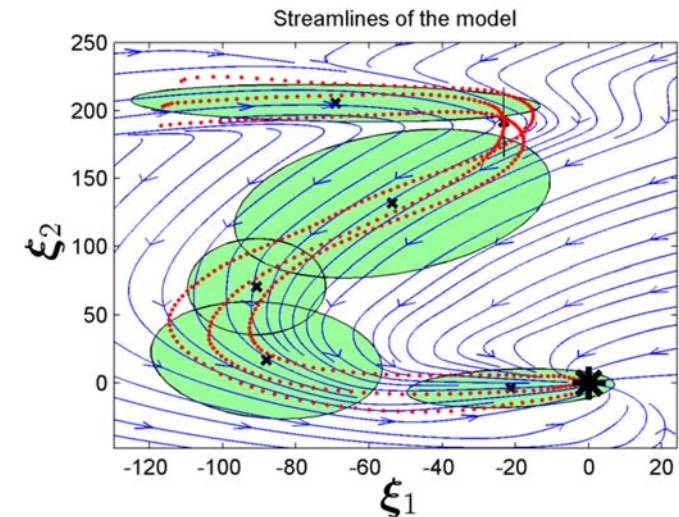
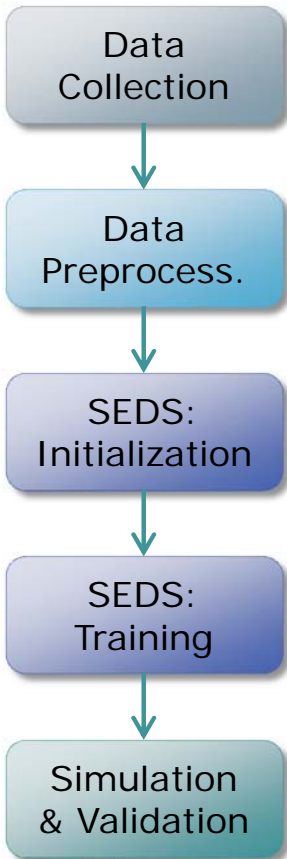
```
% First model
Priors_1 , Mu_1 , Sigma_1
% Second model
Priors_2 , Mu_2 , Sigma_2
%-----
%New Model:
Priors = [Priors_1 ; Priors_2] ; Priors = Priors / sum(Priors) ;
Mu = [Mu_1 Mu_2] ;
Sigma = Sigma_1 ;
Sigma(:, :, end + [1 : size(Sigma_2, 3)]) = Sigma_2 ;
```

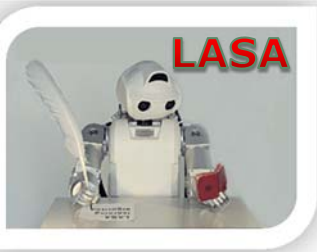




# SEDS: Final Notes

- When can I use SEDS?
- You could use SEDS when there is some consistency in your task, which should be also conveyed in your demonstrations!
- SEDS, in its usual form, considers any overlapping across demonstrations as noise, and thus makes a compromise during training in order to resolve it.





# SEDS: Final Notes

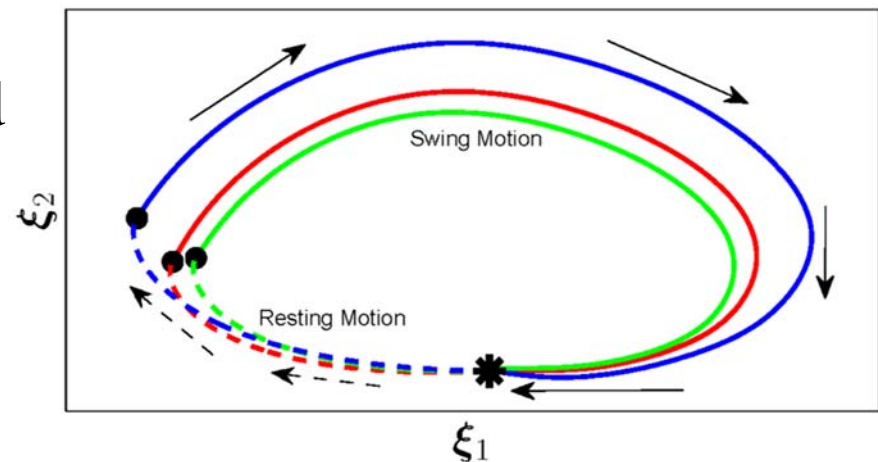
## ➤ When can I use SEDS (continued)?

- SEDS is a powerful approach for episodic tasks that requires these features altogether:

*Generalization, Reactivity, and Online adaptivity*

If your task does not require these features, then SEDS may not be very useful for you!

- SEDS is a movement primitive approach: thus, first try to think whether you could decompose your task into meaningful simple primitives. If yes, then learn each decomposed primitives separately!





# Practical Session

- **Practice #1:** Getting familiar with the SEDS library
- **Steps to follow:**
  - 1) Open the file 'demo\_SEDS\_Learning.m'. You could find the part that you could modify in lines 9-40
  - 2) Run this function a few times with its current setting. You could observe that the final model is different at each time. Could you figure out the reason?
  - 3) Change the number of Gaussian functions  $K$  and evaluate the accuracy of the learned model.
  - 4) Change the objective function from 'mse' to 'likelihood', and re-run the program.
  - 5) You could also try the steps above by loading the other available models (see line #10 in the file).



# Practical Session

- **Practice #2:** Exporting a SEDS model for use in your own simulator/robot.
- **Requirements:** SEDS ROS Package
- **Approach:**
  - Storing demonstration trajectories from your robot and save them into separate txt files.
  - Using MATLAB to train a SEDS model offline.
  - Exporting the trained model into a text file.
  - Loading the trained model from the text file, and use it in the realtime thread of your (real/simulated) robot controller.
- **Note:**
  - Currently, the SEDS ROS package only provides an estimate of the desired control policy (e.g. an estimate of the velocity) based on the current situation of the robot. The inclusion of the training phase in the ROS package is a work under progress.



# Practical Session, practice #2

- **Steps to follow in MATLAB:**
  - 1) Go to the folder 'SEDS\_Cpp\_lib', and open the file  
    'SEDS\_Export\_Example.m'
  - 2) Run this function to train a SEDS model based on 9 reaching demonstrations that were collected from the 7-DoF WAM robot. If you have your own demonstrations, change the code to load your demonstrations, and modify the parameters according to your task.
  - 3) After running the function, you should see a file with the name 'mySEDSModel.txt' in the current folder. This is the file that includes the information about the trained model. Copy this file to the folder of your robot/simulator executable file.



# Practical Session, practice #2

- **Steps to follow in your code:**

- 1) Extract SEDS and MathLib packages to your ros working space:

```
$tar -xf SEDS_ROS_Package.tar -C <your ROS workspace>
```

- 2) Use rosmake to build the SEDS package:

```
$rosmake SEDS
```

- 3) In the 'manifest.xml' file of your ros package, add the dependency on SEDS:

```
<depend package="SEDS" />
```

- 4) In the main header file of your package, include the following:

```
#include "SEDS/GMR.h"
```

```
/*The SEDS model that you will use for motion generation.
```

```
This model should be accessible in your entire code. */
```

```
GaussianMixture mySEDS;
```



# Practical Session, practice #2

- **Steps to follow in your code, continued:**
  - 5) At the initialization step of your simulator/robot, add the following lines:

```
//loading the exported model from the MATLAB
//You should change the file name according to your need
bool b_SEDSLoaded = mySEDS.loadParams("mySEDSModel.txt");
if (b_SEDSLoaded)
    std::cout << "The SEDS Model is loaded successfully" << std::endl;
else
    std::cout << "Error: Cannot find the SEDS model!!!" << std::endl;
```

**Note 1:** Check your trained model is copied in the correct folder. Otherwise, correct the path to the file.



# Practical Session, practice #2

- **Steps to follow in your code, continued :**

- 6) In the realtime loop of your robot/simulator controller, add the following code:

```
MathLib::Vector x,xd,xT; //defining the required variables
x.Resize(d); //d is the dimensionality of your model
xd.Resize(d);
```

```
/* Set the input value based on how you have defined your
SEDS model. For example, x could be the position of
the robot's end-effector. */
```

```
x = ?;
```

```
xT = ?; // Set the value of the Target (for example from the vision)
```

```
x -= xT; //Transformation into the target frame of reference
```

```
mySEDS.doRegression(x,xd); // Estimating xd at x
```

```
xd.Print("xd = "); //Printing the value of xd
```



# Practical Session, practice #2

- **Steps to follow in your code, continued :**
  - 7) If your model is trained based on the robot workspace variables (e.g. end-effector position/orientation), you need to use an inverse kinematic approach to convert  $\dot{x}_d$  to joint velocities for your robot. If your model is based on joint values, then just send the joint velocity (i.e.  $\dot{x}_d$ ) to your controller. For further intuition refer to the SEDS control architecture.
  - 8) Now you could compile, and run your program. Enjoy!



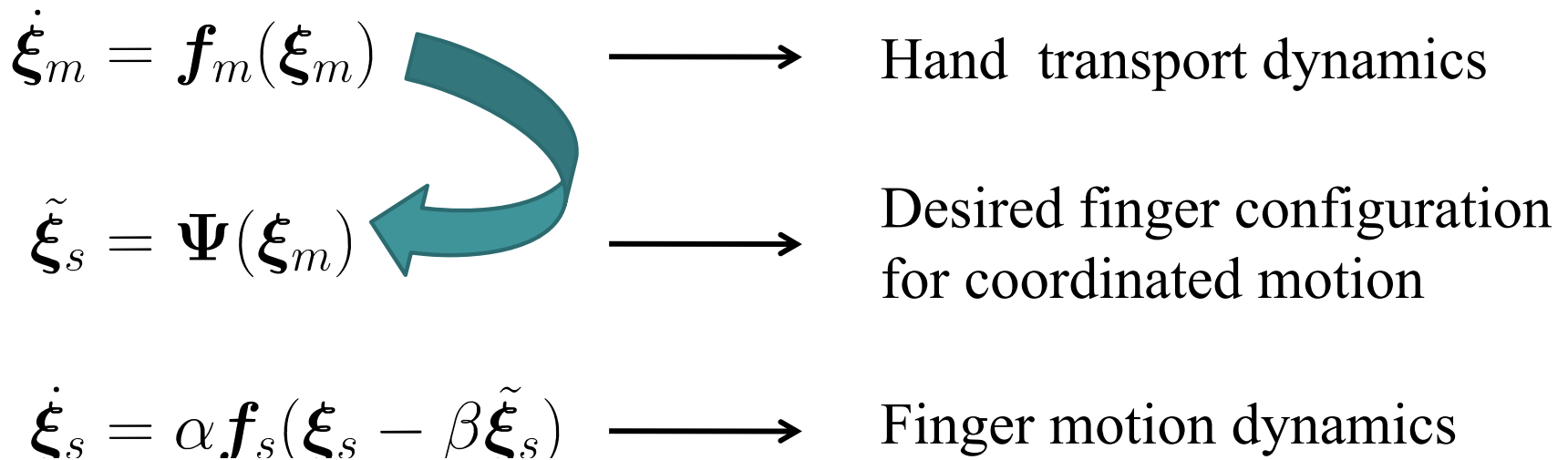
# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# Coupled Dynamical Systems\*

- Synchronize coordination between two DS
- An interesting application in human arm-hand coordination
- Individual DS for reach (master) and grasp (slave) dynamics, spatially coupled.

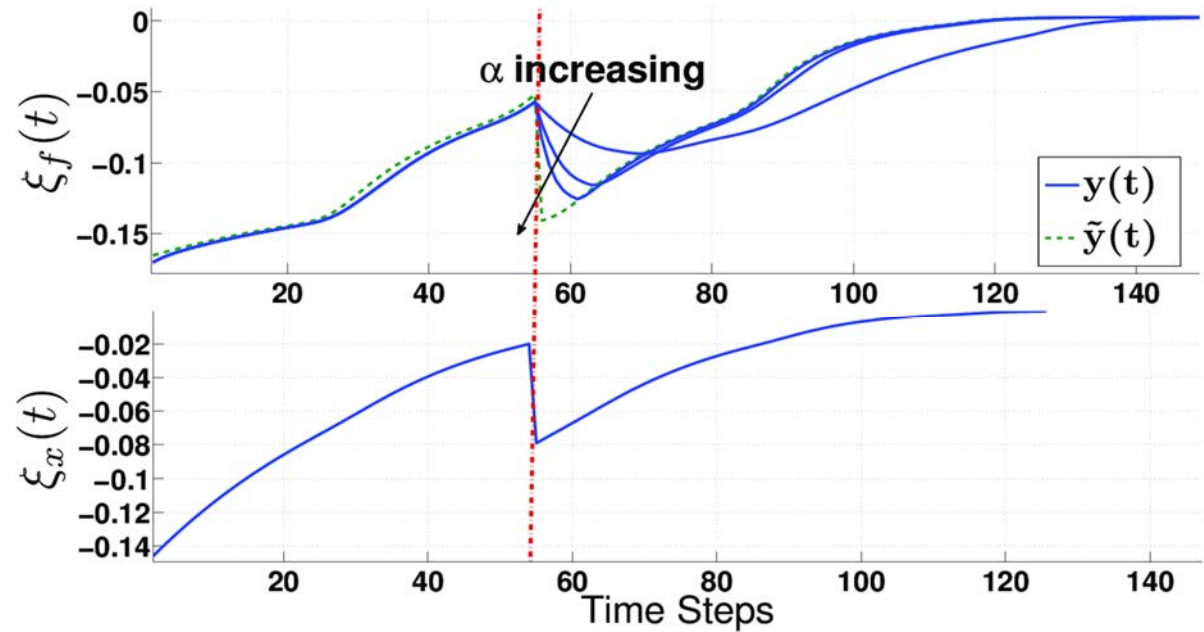


\* Shukla and Billard (2011), RSS

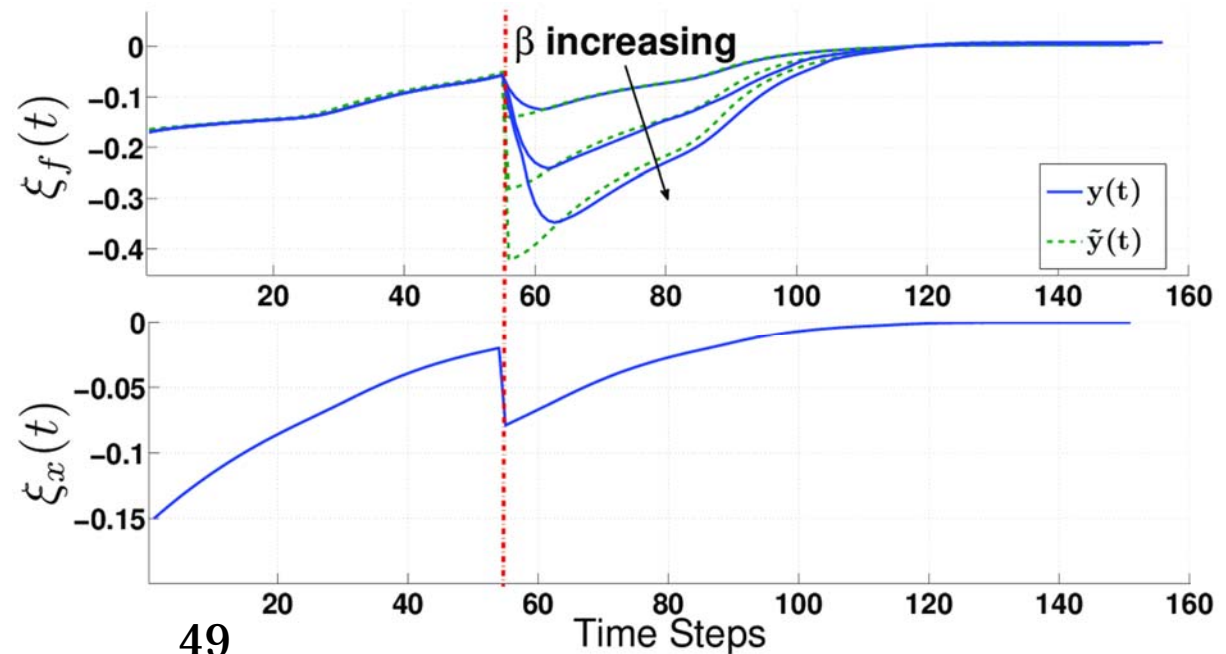


# Coupled Dynamical Systems

Alpha  $\rightarrow$  Speed of reaction under perturbation

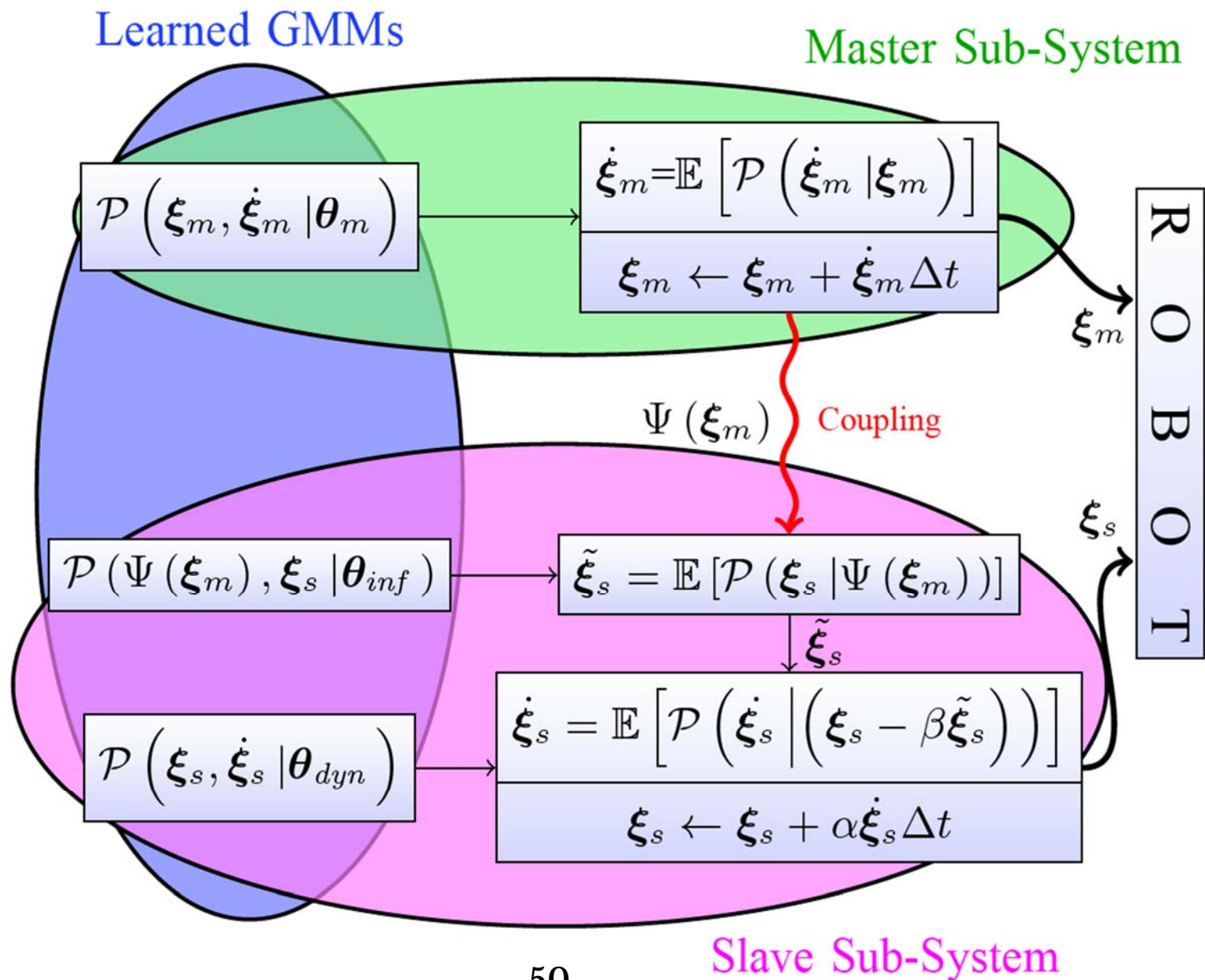


Beta  $\rightarrow$  Amplitude of reaction under perturbation





# Coupled Dynamical Systems





# Robot Experiment

- Catching a half filled bottle of water\*
  - Robot is controlled at 1000Hz.
  - Robot arm-hand coordination
  - The bottle is tracked at 240Hz.



\* Kim, Shukla, and Billard (2013), under submission



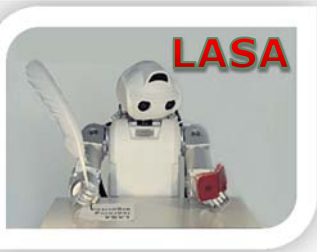
# Practical Session

- **Practice #3: CDS MATLAB Package**
- **Steps to follows:**
  - 1) Go to the folder 'Extensions/CDSv1'
  - 2) Open the file 'CDS\_example.m'.
  - 3) Run this script
  - 4) You could also follow the instructions written in the code to modify it according to your need.



# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# Hitting Motions\*

- Reaching the target with a non-zero velocity



- We formalize robot motions as a multiplication of:

1. A target field  $h(\xi) \in \mathbb{R}^d$

- Specifies the direction of motion

2. A strength factor  $v(\xi) \in \mathbb{R}^+$

- Indicates the speed of movement

- Hence the final model is:  $\dot{\xi} = f_h(\xi) = v(\xi)h(\xi)$

Hitting movements

\* Khansari-Zadeh et. al. (2012), Advanced Robotics

Kronander et. al. (2011), IROS, *winner of the JTCF best technology paper award*

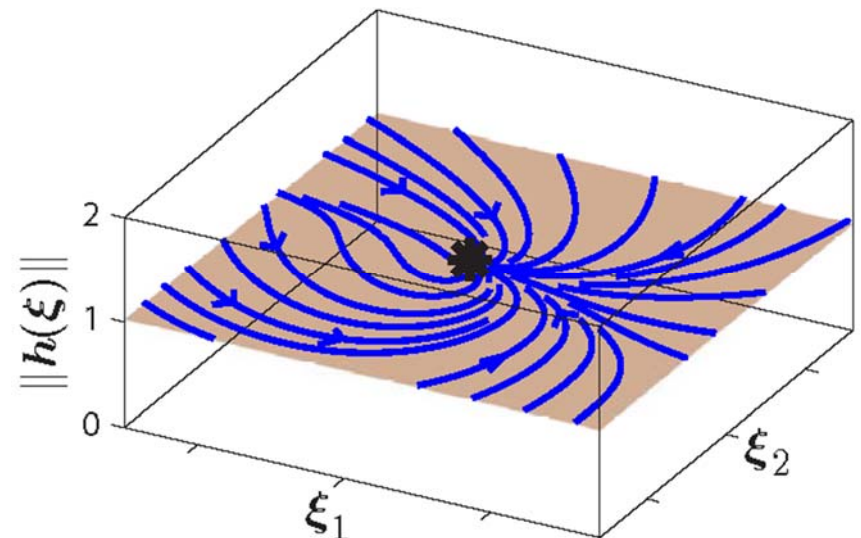
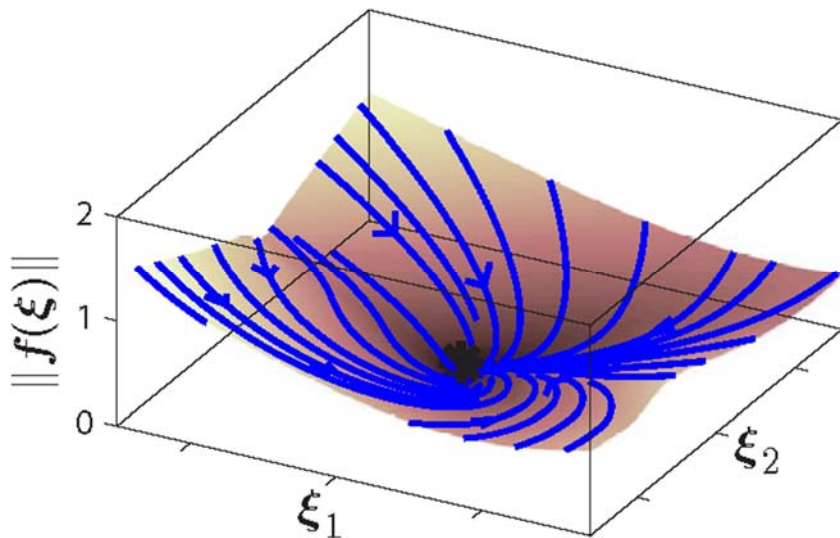


# Learning the Target Field

- We define the target field as:

$$h(\xi) = \frac{f(\xi)}{\|f(\xi)\|} \quad \forall \xi \in \mathbb{R}^d \setminus \xi^*$$

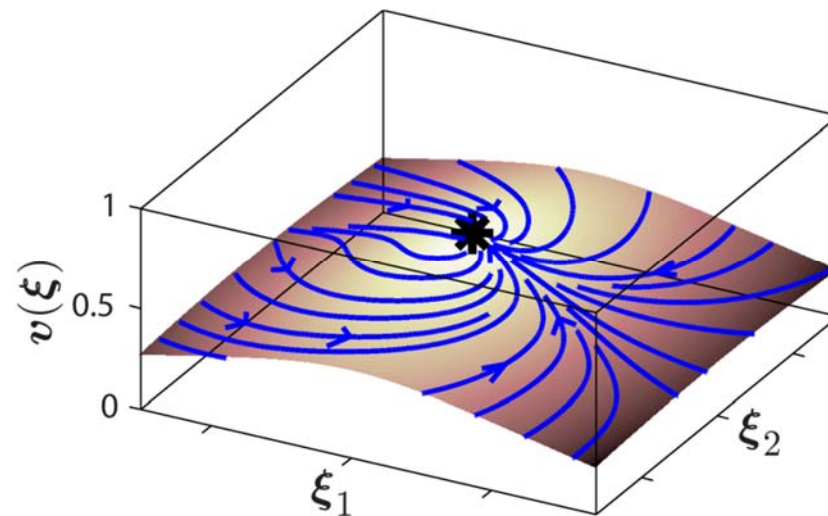
- Estimation of the target field = estimation of a globally stable DS  
 $\Rightarrow$  We could use any of the proposed learning approach





# Learning the Strength Factor

- The strength factor:
  - To form the speed profile
  - To change the hitting speed at the target
- It does not compromise the global convergence
- It can be learned from the same demonstrations using various regression techniques

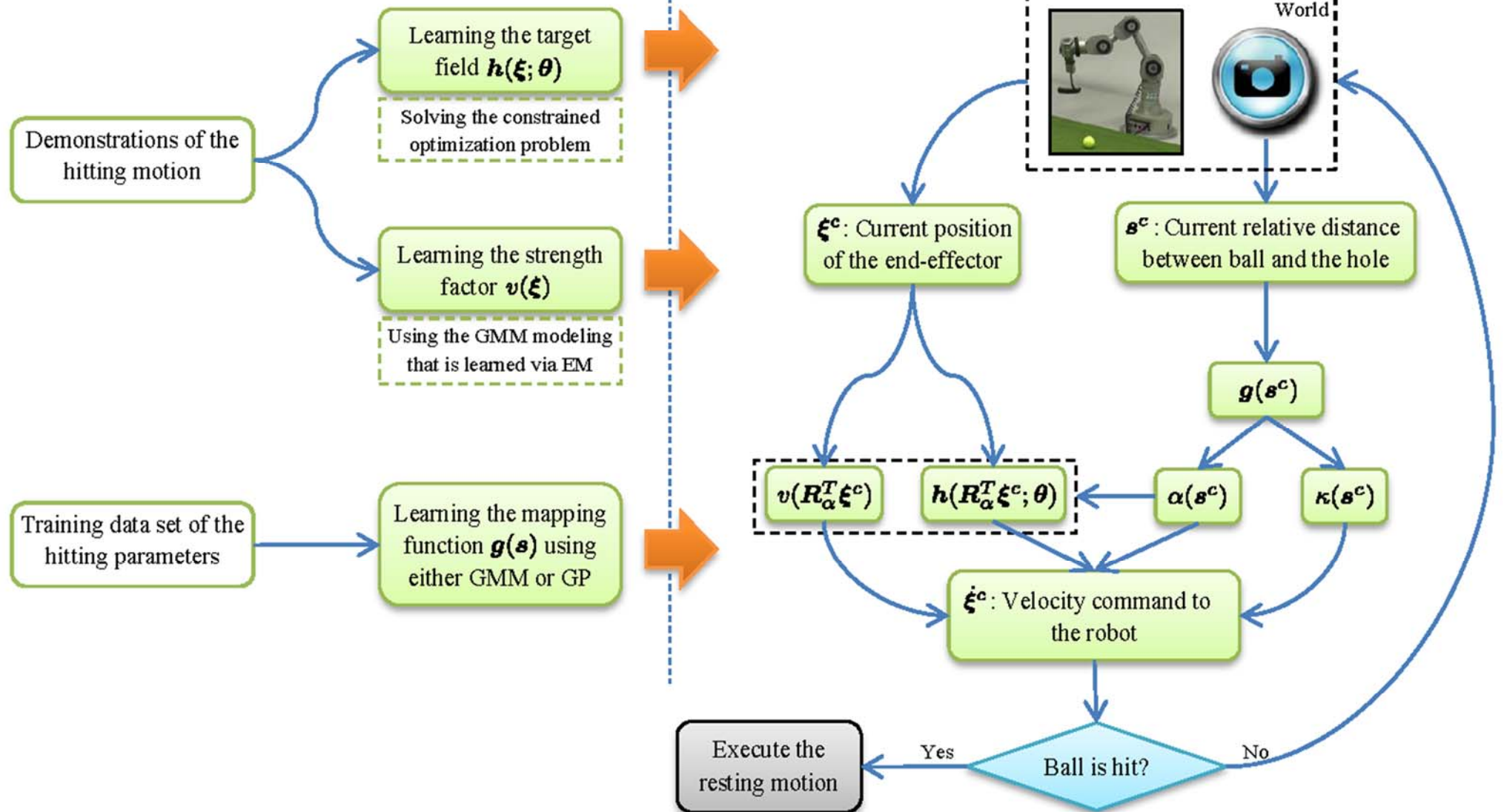




# Minigolf workflow

Training (offline)

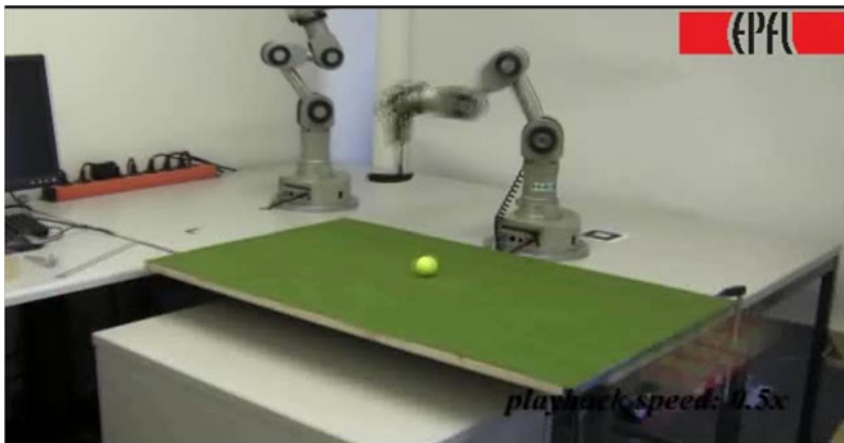
Execution (realtime)





# The Minigolf Experiment

- Learning and Generating the Swing motion in minigolf:





# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# Obstacle avoidance module\*

- Obstacle avoidance for the end-effector:

$$\dot{\xi} = \bar{M}(\tilde{\xi}) \left( f(\xi) - \dot{\xi}^o \right) + \dot{\xi}^o$$

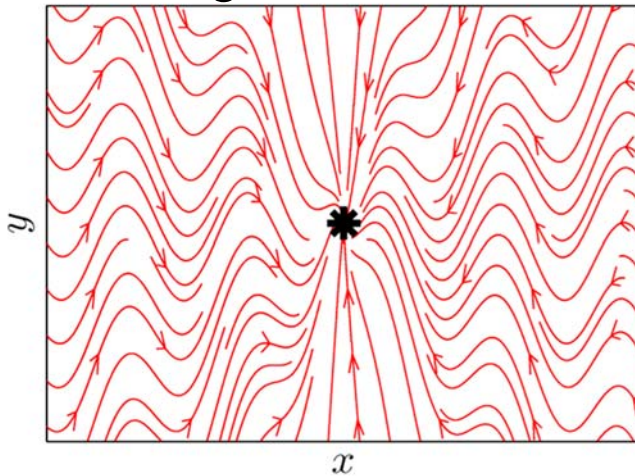
Modulation due to the presence of obstacle(s)

The total effect of all obstacles linear and angular velocity

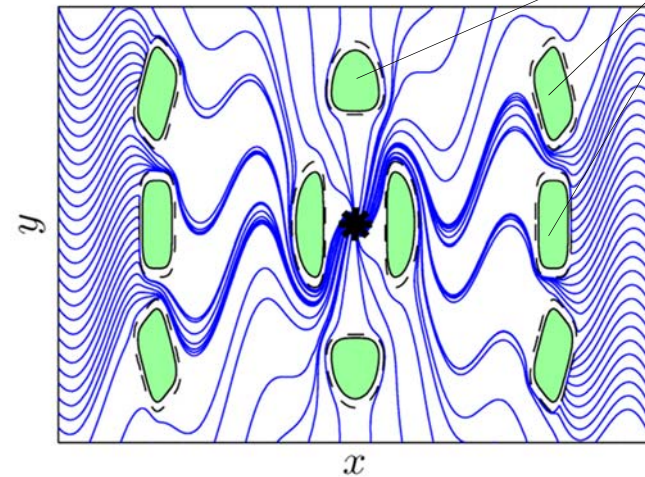
DS Model

In obstacle's frame

Original DS Model



Modulated DS



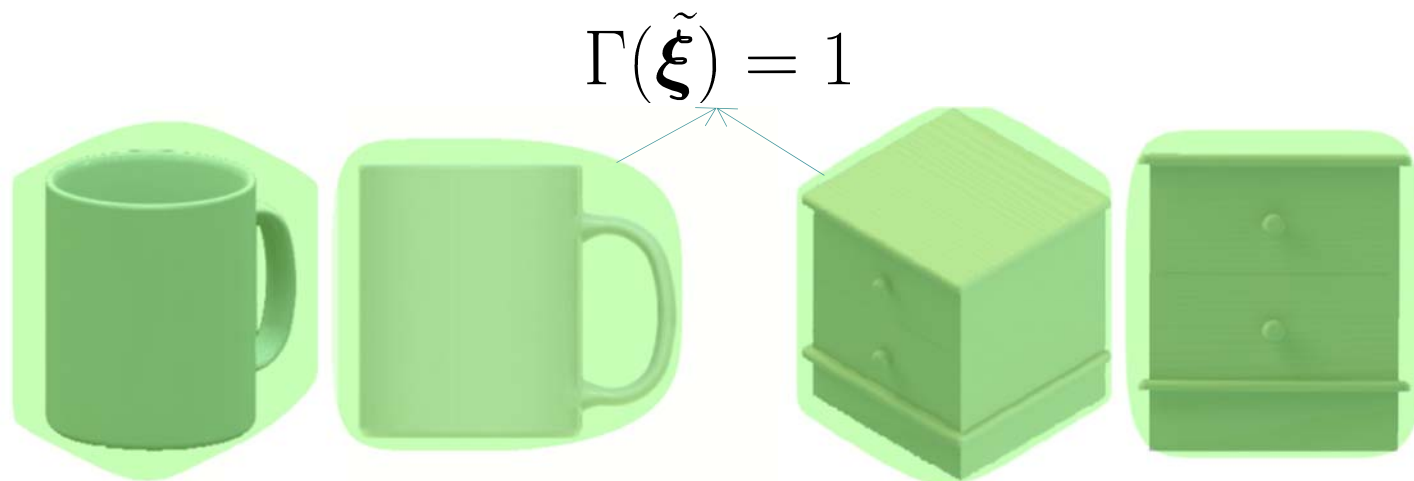
Obstacles

\* Khansari-Zadeh and Billard (2012), Autonomous Robots



# Obstacle avoidance module

- Modeling obstacles by a convex manifold  $\Gamma(\tilde{\xi})$  :



- $\Gamma(\tilde{\xi})$  has  $\mathcal{C}^1$  smoothness
- $\Gamma(\tilde{\xi})$  increases monotonically with  $\|\tilde{\xi}\|$



# Obstacle avoidance module

- The modulation matrix  $\bar{M}(\tilde{\xi})$  is given by:

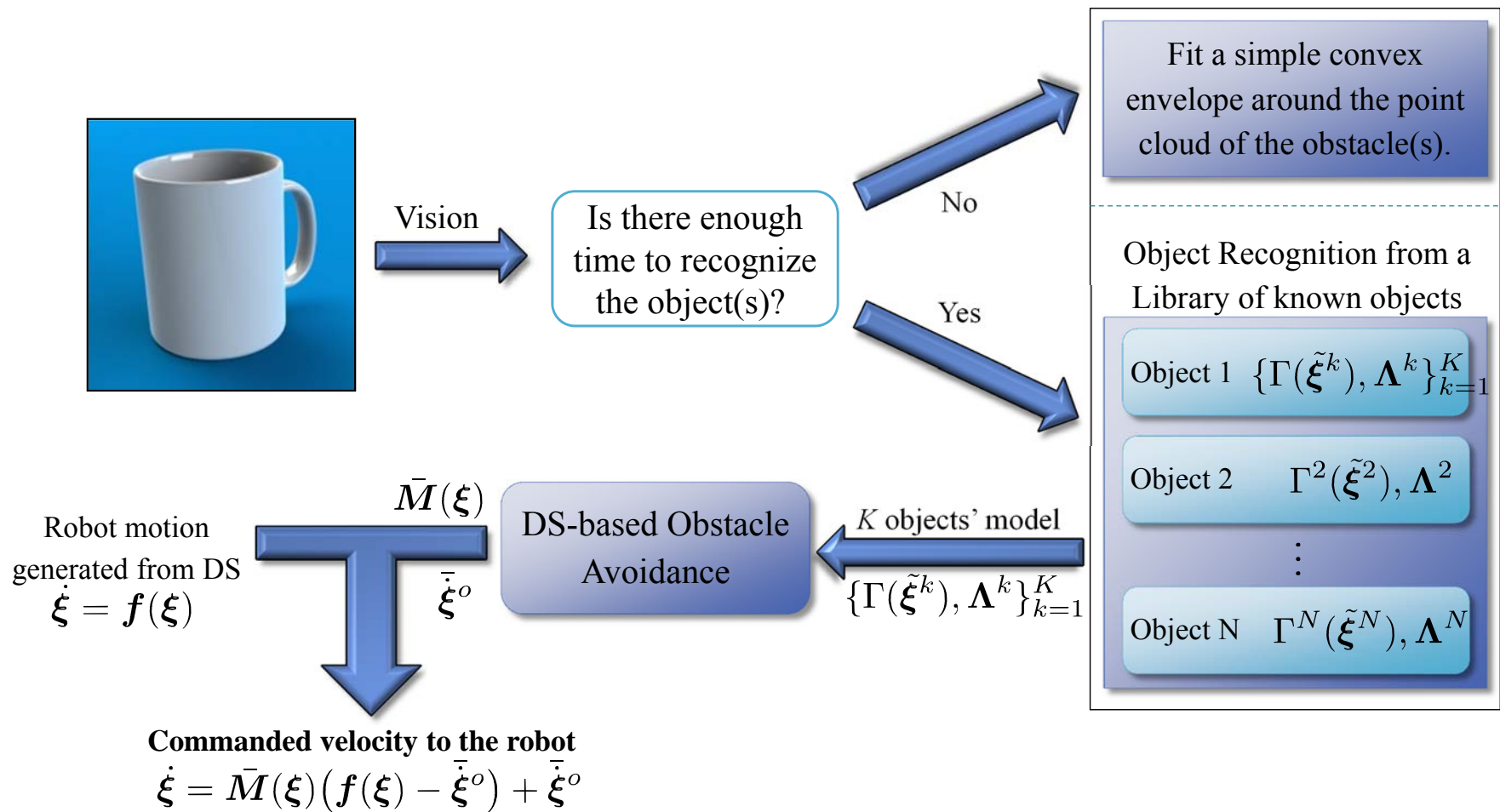
$$\bar{M}(\xi) = \prod_{k=1}^K M^k(\tilde{\xi}^k)$$

Modulation due  
to each obstacle

- Each modulation matrix is computed based on:
  - The geometry of the obstacle
  - The distance to the obstacle
  - The speed of the obstacle
  - Some user-defined properties



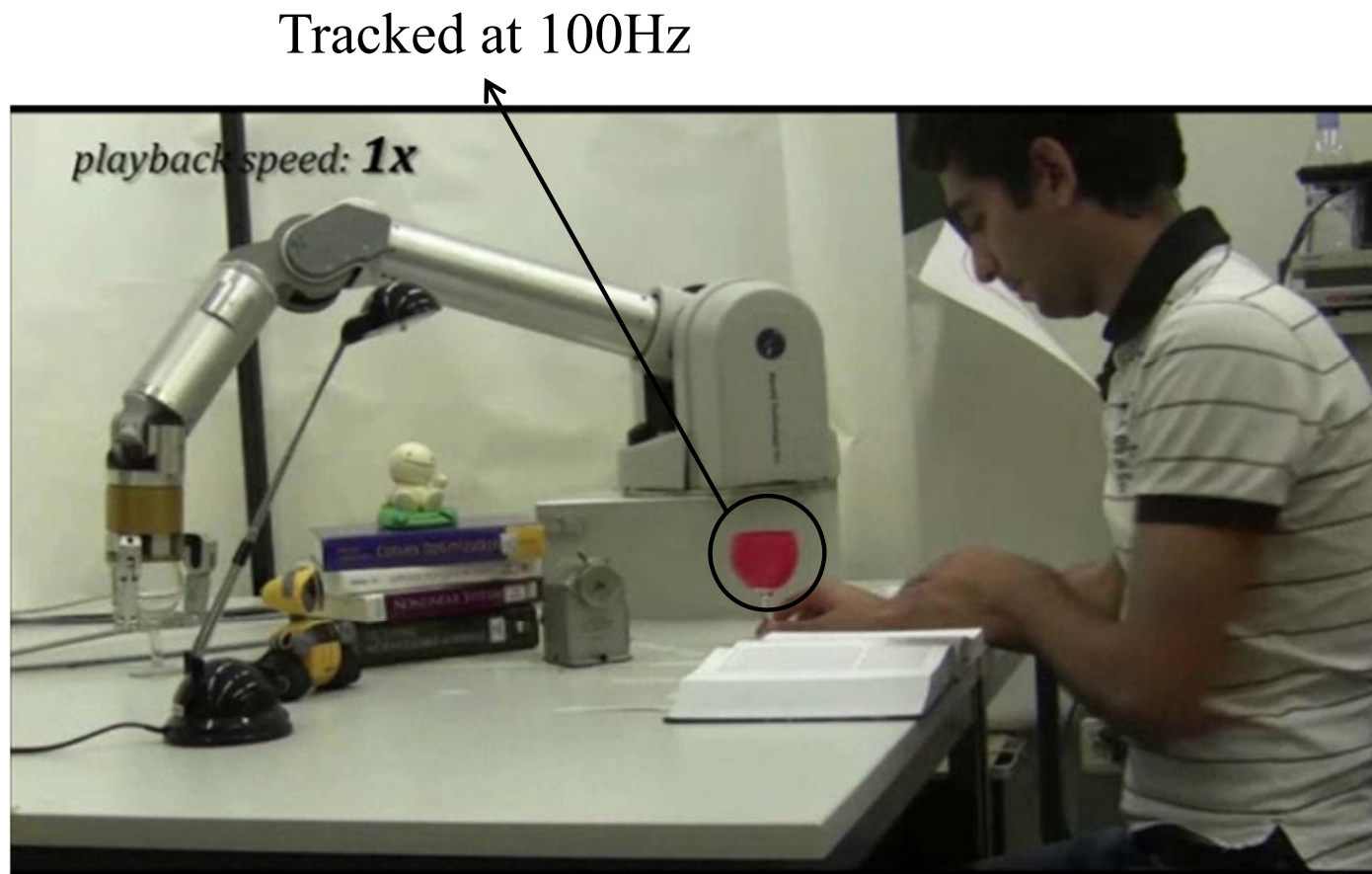
# Obstacle avoidance module





# Robot Experiments

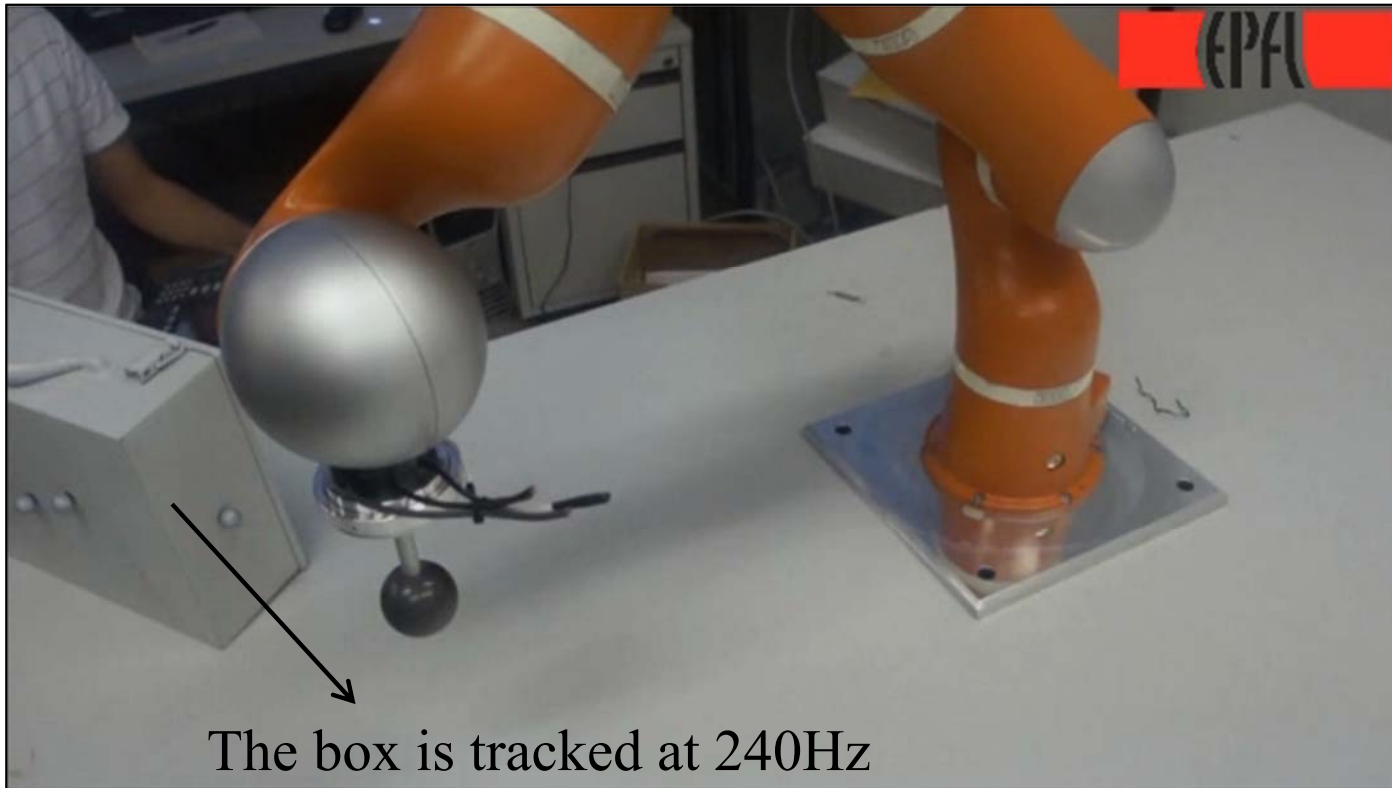
- Placing a glass in a cluttered environment
  - Robot is controlled at 500Hz.





# Robot Experiments

- Dodging a fast moving box
  - Robot is controlled at 1000Hz.
  - At each trial, the box reaches a maximum linear and angular velocities of 0.6~1.5m/s and/or 40~120 deg/s.





# Outline

- Introduction
  - State-of-the-art
  - Our approach
  - The challenge
- SEDS
  - Theory
  - Mathematical Properties
  - Experiments
- The SEDS Library
  - The MATLAB Package
  - The ROS Package
- SEDS Extensions
  - CDS
  - Hitting motions
  - Obstacle Avoidance
- Summary



# Summary: SEDS Contribution

## Robotics:

- The first imitation learning approach that ensures stability of nonlinear multi-dimensional autonomous DS.
- An all-encompassing framework to generate discrete motions with a number of interesting features:
  - Easy to program
  - Global convergence
  - Instant adaptation
  - Modular
  - Time-independent
  - Inherent robustness
  - Various applications
  - Multi-dimensional

## Machine Learning:

- Providing a statistical-based method to estimate globally stable DS from a set of demonstrations.



# Future Work & Open issues

- Modeling DS motions with kinematic variables
  - Robot dynamics is not explicitly considered
  - May introduce some inaccuracies
- Composition and superposition of DS motions
  - Is useful for generating more complex motions

# Thanks for your attention



Thanks Aude  
for her advice  
and support



Thanks Klas  
for his  
collaboration



Thanks Eric  
for the  
RobotToolKit



Thanks Martin  
for the  
vision system

Thanks to the EU Project  (FP7-ICT-248311) for supporting my research