Tutorial on Neural Network Control Systems

Nikos Kekatos

19/03/2021

Requirements

MATLAB: requires license (academic, student, professional,...)

Dependencies: Breach (free and open source)

MATLAB Toolboxes:

a) Simulink
b) Deep Learning
c) Symbolic Matlab Toolbox (Breach)
d) Global Optimization Toolbox (Breach)
e) Optimization Toolbox (Breach)
f) Control System Toolbox (Tutorial)
g) Stateflow (Tutorial)

Our Code for Neural Network Control Systems: https://github.com/nikos-kekatos/NNCS_matlab

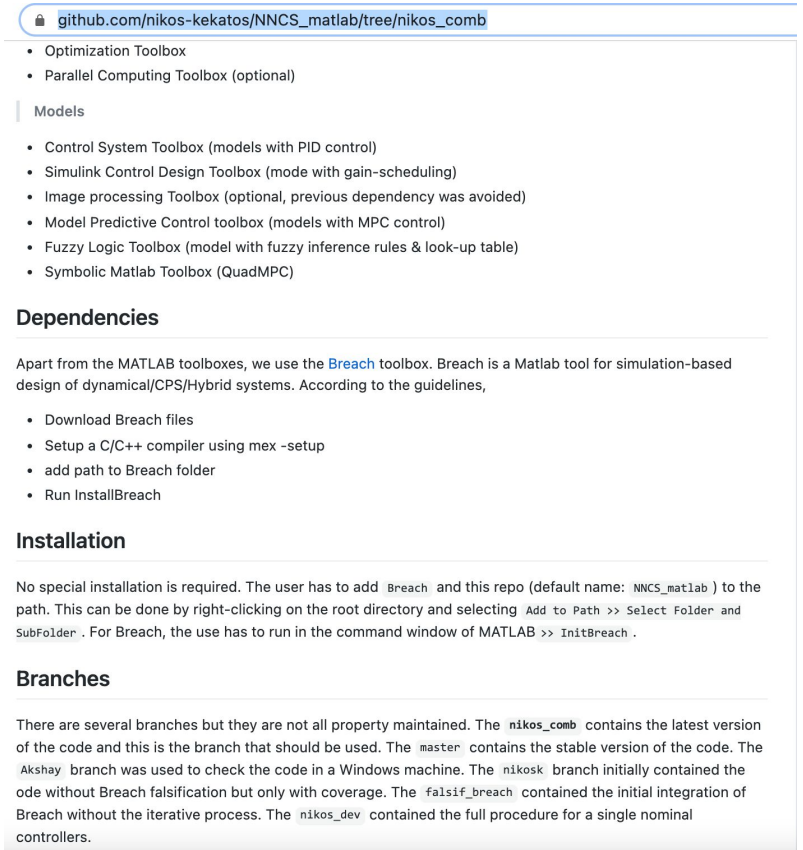
Installation

Readme :
https://github.com/nikos-kekatos/NNCS_matlab/tree/nikos_comb

No special installation, as it is MATLAB-based and it is platform independent.

Provided the existence of toolboxes, it should work without (many) manipulations.

What is your OS?



Problem Definition

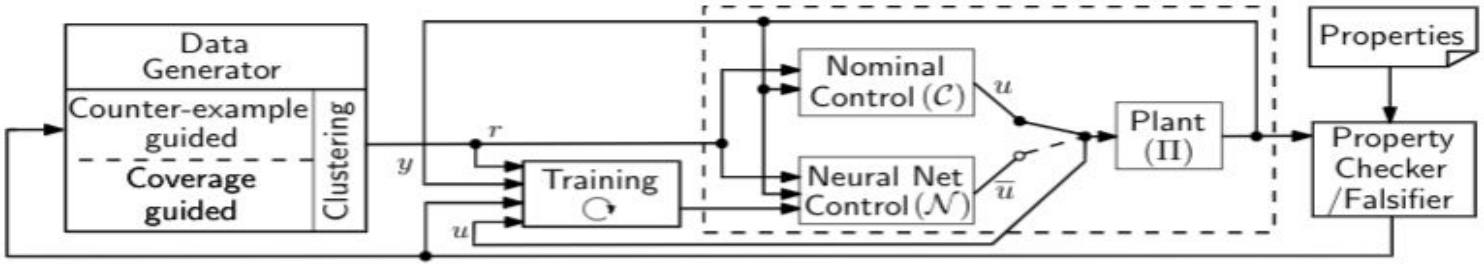
Standard (Simpler) Case

Given i) a closed-loop (controlled) system S, which is composed from a nominal controller C and a plant P (S:=C||P), and ii) a control objective/specification φ written in Signal Temporal Logic, design a neural network controller C_nn that such that the new closed-loop system S_nn:=C_nn||P satisfies the property φ.

Regarding the satisfaction of the property, we do not aim at “full” formal verification but we instead target the lighter version of semi-formal verification (using falsification).

Methodology

1. Closed-loop modeling (design or given)
2. Trace generation (finding good examples), e.g. via coverage
3. Data Selection (trimming, preprocessing, etc.)
4. Neural Network Training
    a. Architecture
    b. Training hyperparameters
5. New closed loop system (with a neural network)
6. Falsification and Retraining in a loop



What do you need to run a model?

Inputs to the tool:

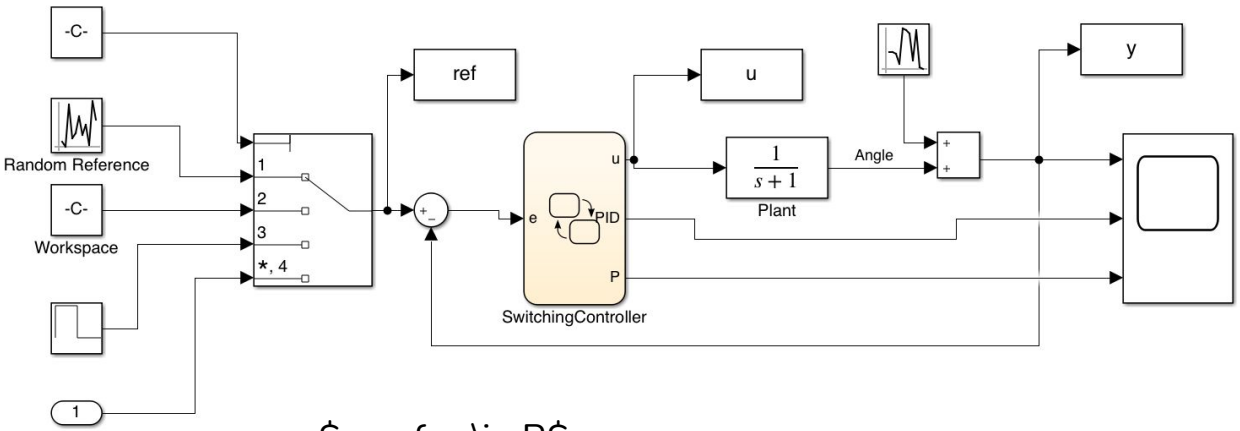
1. Nominal Closed-loop System in Simulink (not automated for MATLAB models)
2. Specification defined in Signal Temporal Logic (syntax of Breach)
3. Configuration

Then we need to run the “main” file which contains all the steps of our algorithm.
There are several “main” files for different models you can use them as references.

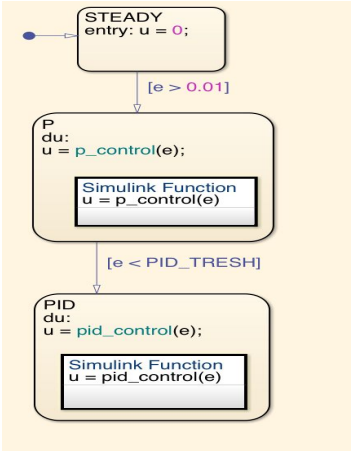
Example: Switched Controller

This example is a simple Simulink model which describes a closed-loop system.

- Plant: linear
- Controller: switched (from P -> PID when a condition is satisfied)
- Property: if there is a step reference change, then plant output y should reach and stay close to the given reference.

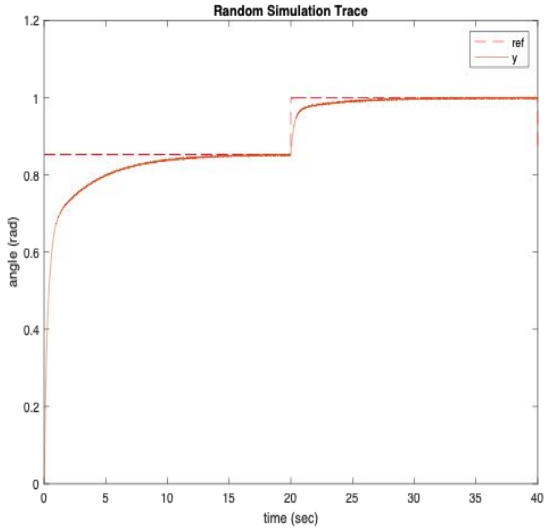


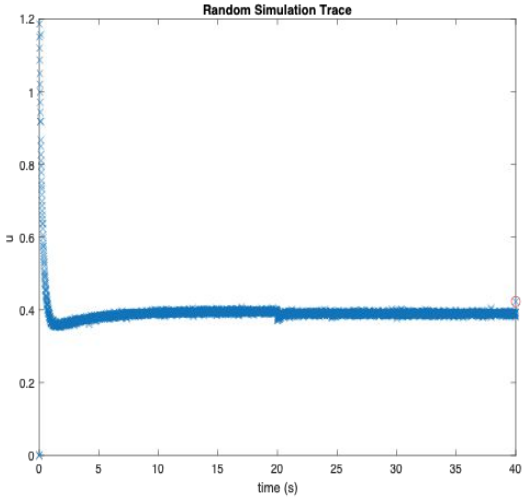
$u, ref, y \in R$



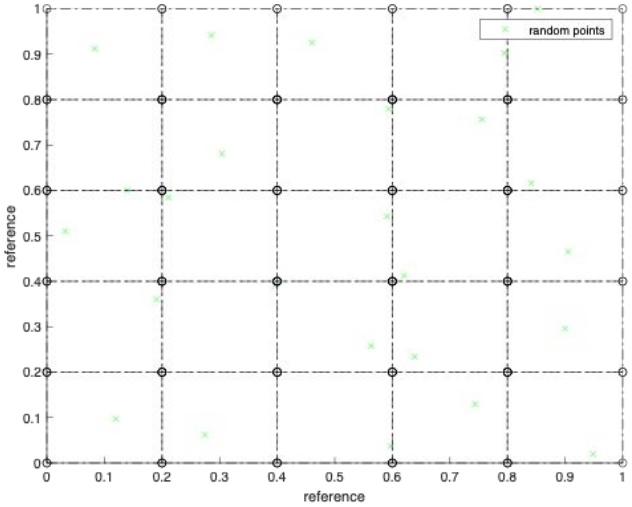
Trace Generation

Added small perturbation as noise.





Coverage: Piecewise constant reference with two pieces



Data Selection (1/2)

Data Selection can be important: Avoid duplicates, reduce size, choose area where you want to keep or delete data points.

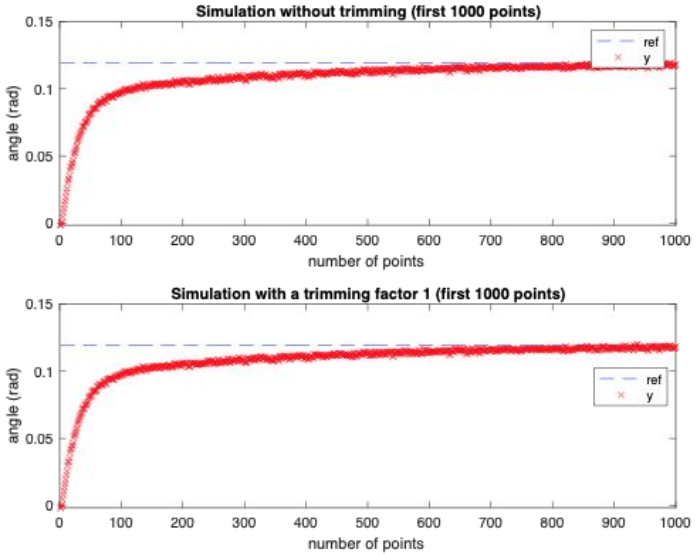
Three main options (named in the code as follows):

- Preprocessing
- Trimming
- Steady-State trimming

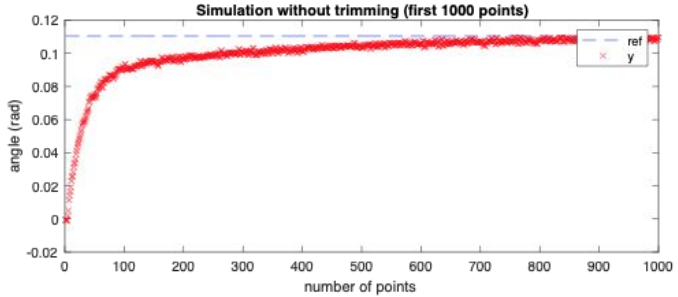
#different reference signals/simulation traces: 25

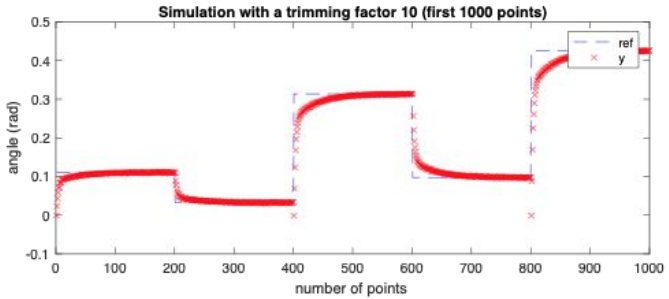
Each simulation T=40s, dt=0.01

In total: 100,000 points for ref, u, y With trimming: 62475



Data Selection (2/2)



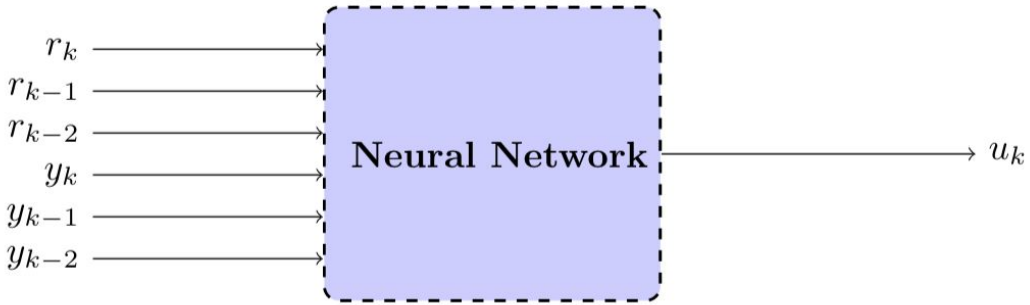


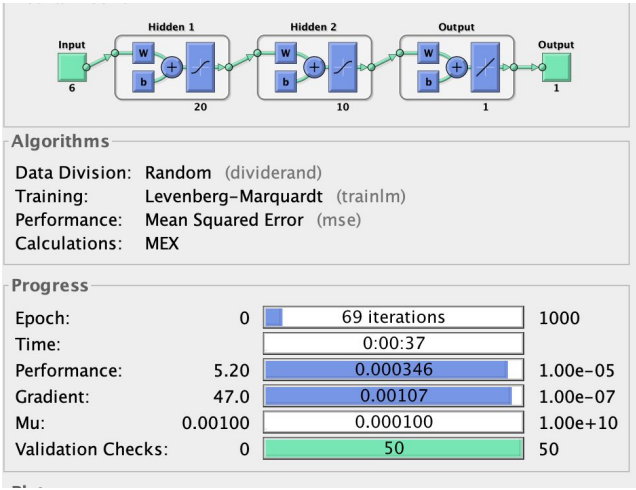
Training

== Need to choose (1) the architecture and (2) the training parameters.

- How to capture the time-dependence/evolution ? Use past values for ref, u, y.

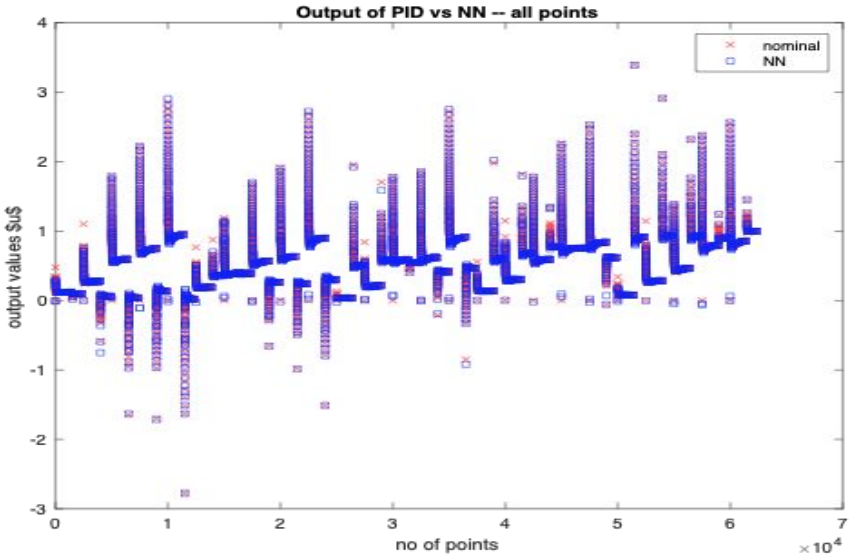
```
training_options.neurons=[20 10 ]; % 2 layers with 20 and 10 neurons each
training_options.use_error_dyn=0;      % answer: 0 or 1, use error dynamics or not
training_options.use_previous_u=0;     % answer: integer, number of previous u values
training_options.use_previous_ref=2;   % answer: integer, number of previous references
training_options.use_previous_y=2;
```



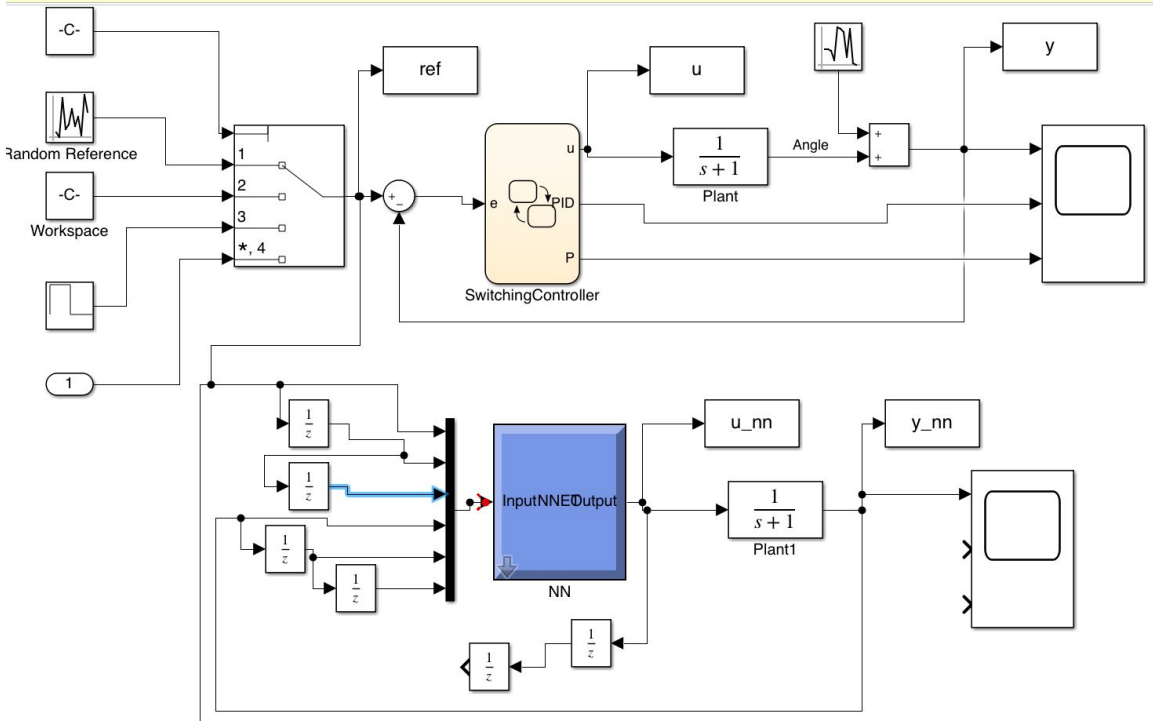


Training/Evaluation

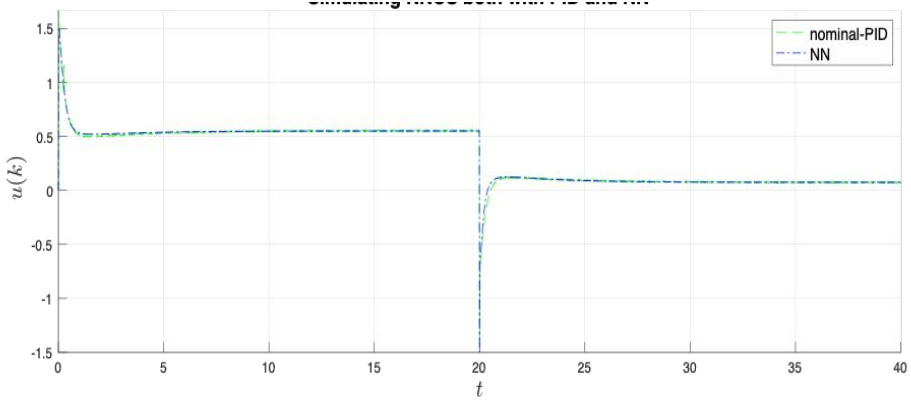
- In Open Loop (with the same inputs)
- Neural Network Inputs: r(k), r(k-1), r(k-2), y(k), y(k-1), y(k-2)
- Neural Network Output: u(k)

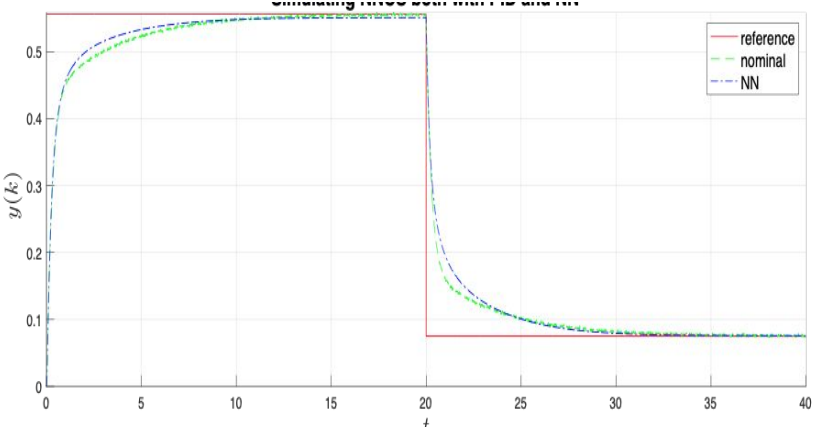


Create NNCS


-C-
Random Reference
-C-
Workspace
1
1
2
3
*, 4
ref
u
y
Angle
u
e
PID
P
SwitchingController
1
s + 1
Plant
InputNNOutput
NN
u_nn
y_nn
1
s + 1
Plant1

Simulating the new NNCS





Testing against the STL property

Out of 25 traces, all NN generated traces satisfy given STL property (tested with Breach).

param sim_time=39.9, r=0.1, dt=0.01, T=10, d=0.01, T_1=6, T_2=4,e=0.05, ov=0.05

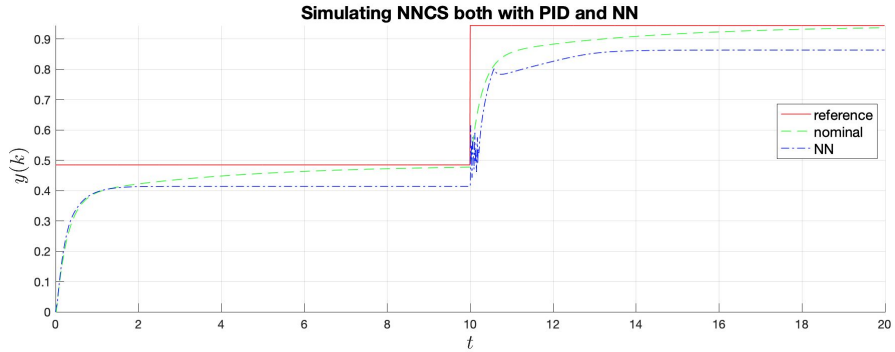
phi_1:= alw_[0,sim_time]((abs(ln1[t+dt]-ln1[t])>d or [t]==0) => (ev_[0,17] alw_[0,3] (abs(y[t]-ln1[t])<e)))

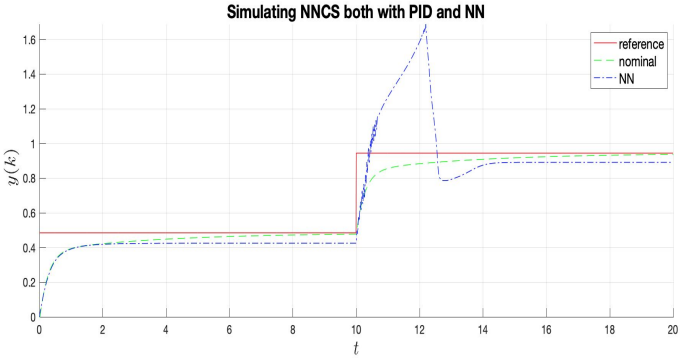
Is it enough?

Falsification with Breach

We use Breach to falsify the NNCS via sampling and optimization methods.

Fast restart (to speed up computations).

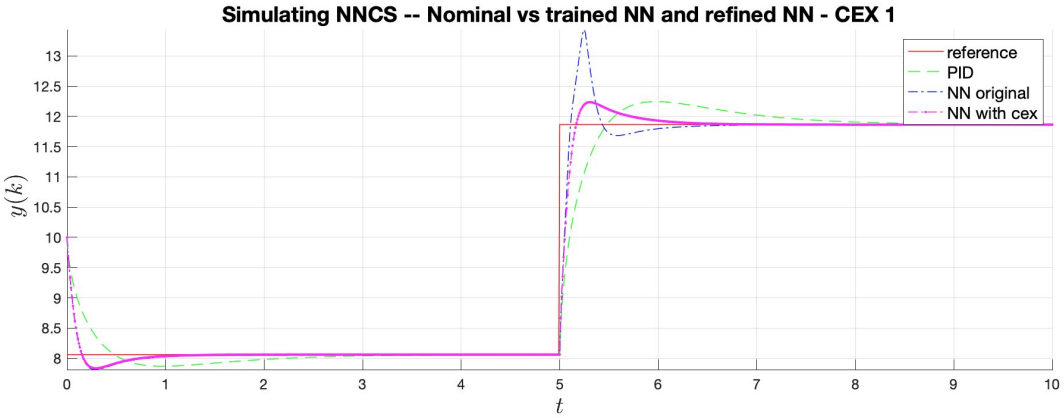




Falsification/Retraining

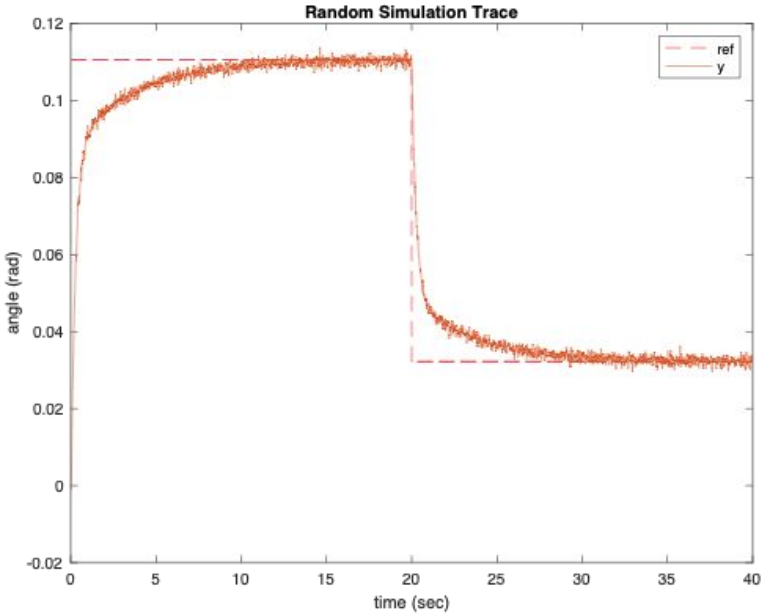
The nominal controller uses error dynamics, but we had to separate the references from the plant outputs. We chose $\mathcal{I} = \{R_k, R_{k-1}, R_{k-2}, Y_k, Y_{k-1}, Y_{k-2}\}$. We started with 25 examples and had 100,000 samples per state. We trained with [20, 10] neurons and falsification yielded 2/100 CEX. Retraining eliminated these cex. The second falsification/retraining loop eliminated 5 CEX.

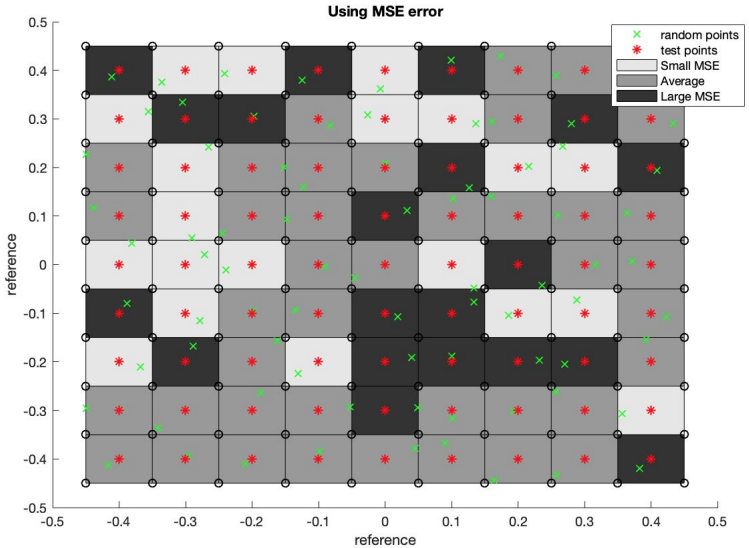
Able to design a controller that eliminates all CEX.



END

Backup 1


Random Simulation Trace
ref
y
angle (rad)
time (sec)
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
0
5
10
15
20
25
30
35
40


Using MSE error
random points
test points
Small MSE
Average
Large MSE
reference
reference
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5

Some Discussion Points (Friday, 19 March)

NNCS:

1) Control Systems & Modeling (SImlink/MAtlab)
2) Simulation & Falsification
3) Neural Network Structure & Architecture
4) Temporal Logic

CPSdebug? “MATLAB”

A: recurrent NN

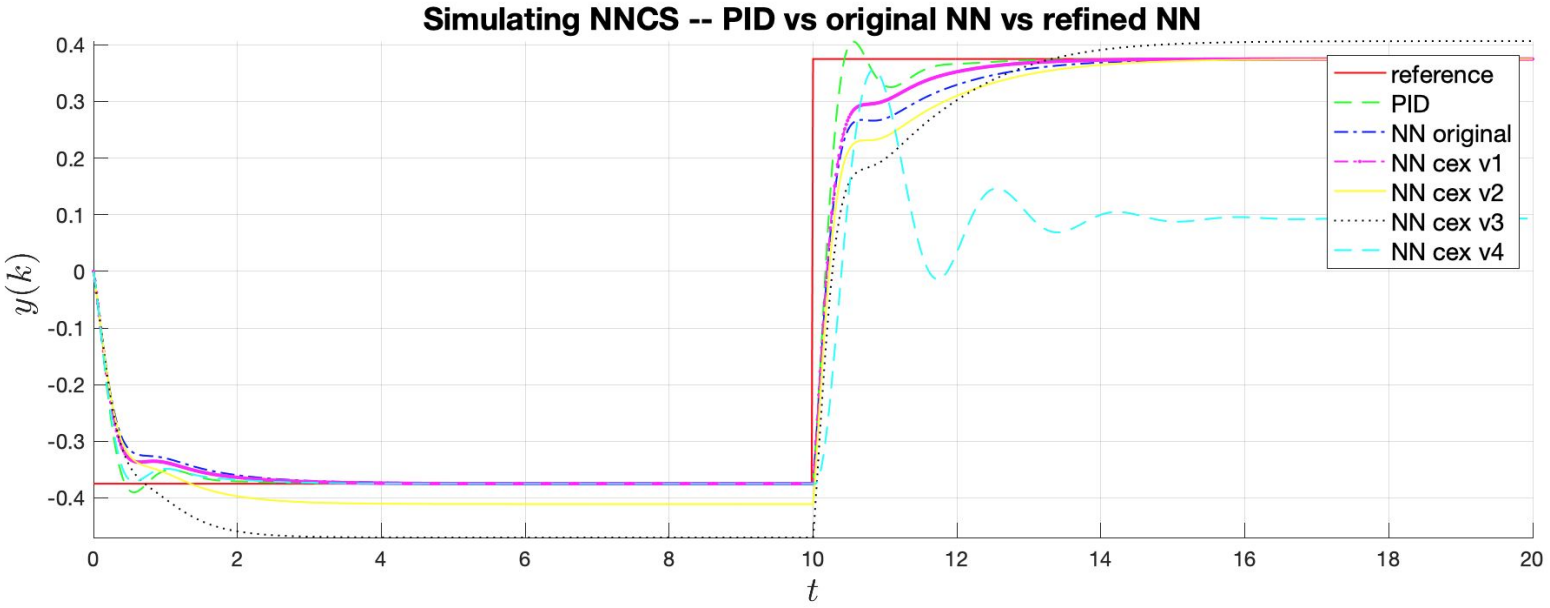
E: Minimal part of the trace (capture memory but not too much)

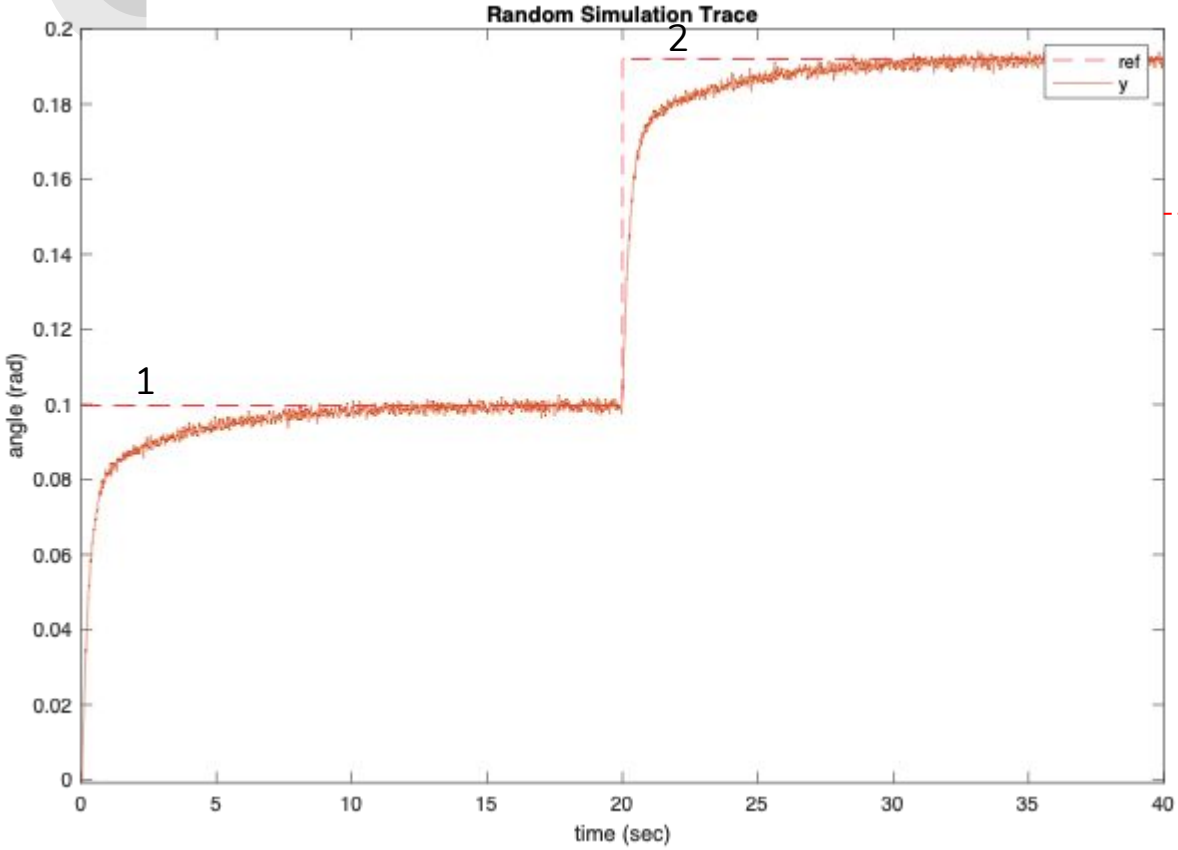
AD: temporal implicants in Breach (missing part of Until)

CPS debug -> critical time (neighborhood)

Question: Which is the last point of the trace?

Retraining algorithms




Random Simulation Trace
ref
y
angle (rad)
time (sec)
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2
0
5
10
15
20
25
30
35
40

1

2

3