Implementing radial basis function networks (RBFs):

MakefileConf:

Add this:

```
AMOSIIUTIL  = $(MAINDIR)/utils/rbf-framework

FILES      +=      main \
       $(AMOSIIUTIL)/ngnet

INC        += -I$(AMOSIIUTIL)
```

1. Optimized usage for Reinforcement Learning (RL):

One network is used to produce the actor's output and the critic's output (see reinforcement learning projects).

Controller:

1) Add this part in beginning of the file.cpp

```
#include "rbf-framework/ngnet.h"
NGNet* ngnet;

#define rbf_num_units 1500 //number of hidden neurons
#define rbf_num_IN 4          //number of inputs
#define rbf_num_out 2         //number of outputs (one for the critic and one for the actor)

Cell VALUE(rbf_num_units, rbf_num_IN, rbf_num_out);
```

2) Add this part in your constructor:

```
ngnet = new NGNet(rbf_num_IN, rbf_num_out);
```

3) add this in your initialization function:

```
init_funcapprox();
```

4) Add this part in your step function:

```
//to get the V value and the action value, call:
double rbf_out[2];
get_v_action_pair_from_RBF_network(xt, rbf_out); //could be with a different name the implementation for this function is down
```

```
//to update the network
//This function is optimized for Reinforcement Learning (two RBF networks one for the actor and one for the critic)
update_rbf_network(input_old, td_error, rate_valuefunction, p_error, rate_policy);
```

2) General use of RBF network library (see the sin wave example):

Controller:

1) Add this part in beginning of the file.cpp

```
#include "rbf-framework/ngnet.h"
NGNet* ngnet;

#define rbf_num_units 1500 //number of hidden neurons
#define rbf_num_IN 4          //number of inputs
#define rbf_num_out 2         //number of outputs (one for the critic and one for the actor)

Cell VALUE(rbf_num_units, rbf_num_IN, rbf_num_out);
```

2) Add this part in your constructor:

```
ngnet = new NGNet(rbf_num_IN, rbf_num_out);
```

3) add this in your initialization function:

```
init_funcapprox();
```

4) Add this part in your step function:

//get the output of the network:

```
ngnet->incsbox_output(cell, &in, &outputnet, &number_of_hidden_neuron);

//to update the network
//update the network for each sample:
//note: please pass the error with a negative value (see the learning equation of the RBF network (the implementation))
sample_error = -sample_error;
//update trace
ngnet->incsbox_trace(cell, &in[i], 0 , &number_of_hidden_neuron);
//update the network
ngnet->incsbox_update(cell, &in[i], &sample_error, learning_rate,
&number_of_hidden_neuron, wbasis, 0, 0);
```

//

Functions:

//

1) (function) get_v_action_pair_from_RBF_network:

```
double u_tmp1[2];
u_tmp1[0] = 0.0;
u_tmp1[1] = 0.0;
ngnet->incsbox_output(&VALUE, xt, u_tmp1, &nbasis);
//u_tmp1 array has the outputs
```

2) (function) update_rbf_network:

```
//call trace
 ngnet->incsbox_trace(&VALUE, input_old, lambda_v , &nbasis);
//update the actor and the critic
ngnet->incsbox_update_v_action_pairs(&VALUE, input_old, td_actor, td_error_tmp,
rate_valuefunction, rate_policy, &nbasis, wbasis, error_thresh_critic,
unit_activation_thresh_critic, update_actor);
```

3) (function) init_funcapprox:

```
//init the network:
ngnet->init_incsbox(&VALUE,4 /*4 input number*/,2 /*1 one output*/);
ngnet->reset_incsbox(&VALUE);

//find width of each input:
for(i=0;i<xdim ;i++) //for each input (xdim)
{
     basis_width[i]  = (xmax[i] - xmin[i])/(2.0*basis[i]); // variance (the measure of the
width of the distribution)
     wbasis[i] = 1.0/basis_width[i]; //width of each input
}

 //---------example two inputs----------------------//
```

```
//for each dimension of the input space, you need a for loop
for(state_index[0]=0; state_index[0]<basis[0] /*3*/; state_index[0]++) // angle left
for(state_index[1]=0; state_index[1]<basis[1] /*3*/; state_index[1]++) // angle right
{
	for(i=0;i<xdim /*2 inputs*/;i++)
	xc[i]  = (xmin[i]+basis_width[i])+state_index[i]*2.0*basis_width[i];
	ngnet->put_incsbox(&VALUE, xdim , 2, xc, wbasis, &nbasis);
}
```