



*College of Engineering™*

**Diff-MAS1D (version 0.9.1) - User's Manual**

A simulation platform for controlling distributed parameter systems (diffusion)  
with *networked* movable actuators and sensors (MAS) in 1D domain

Jinsong Liang and YangQuan Chen

Center for Self-Organizing and Intelligent Systems (CSOIS)  
Department of Electrical and Computer Engineering  
College of Engineering, Utah State Universtiy  
4160 Old Main Hill, Logan Utah 84322-4160, USA  
Emails: [jслиang@ieee.org](mailto:jслиang@ieee.org), [yqchen@ieee.org](mailto:yqchen@ieee.org)  
Web: <http://www.csois.usu.edu/>

10/30/2004



**Technical Report No. : USU-CSOIS-TR-04-04**

Reports are available from <http://mechatronics.ece.usu.edu/reports/>

# Diff-MAS1D<sup>©</sup> (version 0.9.1)

## User's Manual

A simulation platform for controlling distributed parameter systems (**diff**usion)  
with *networked* **m**ovable **a**ctuators and **s**ensors (MAS) in **1D** domain

Center for Self-Organizing and Intelligent Systems  
Dept. of Electrical and Computer Engineering, Utah State University  
4160 Old Main Hill, Logan, Utah, 84322-4160

## I. INTRODUCTION

Diff-MAS1D is a software package for simulation of 1D diffusion process control using moving actuators and moving sensors. The main reason to develop this software is that no currently available software package (including Matlab, Maple, Mathematica, MathCAD, Ansys, Nastran, FEMLAB) is able to solve the problem Diff-MAS1D is able to solve.

Diff-MAS1D is written completely in Matlab script and Simulink.

### A. Problem formulation

Diff-MAS1D is able to solve the following problem numerically.

$$\frac{\partial u(x, t)}{\partial t} = k \frac{\partial^2 u(x, t)}{\partial x^2} + f_c(x, t) + f_d(x, t), \quad (1)$$

where  $0 \leq x \leq 1$  is the spatial domain;  $t \geq 0$  is the time domain;  $u(x, t)$  is the variable we want to control;  $k$  is a positive real constant related to system parameters;  $f_c(x, t)$  is control from the actuators and  $f_d(x, t)$  is the disturbance.

Arbitrary combination of the following two types of boundary conditions can be used as the boundary condition for each boundary ( $x = 0$ ,  $x = 1$ ).

- Dirichlet boundary condition

$$u = C \quad (2)$$

where  $C$  is a real constant.

- Neumann boundary condition

$$\frac{\partial u}{\partial n} = C_1 + C_2 u \quad (3)$$

where  $C_1$  and  $C_2$  are two real constants;  $n$  is the outward direction normal to the boundary.

We are using a number of moving sensors to measure  $u(x, t)$  and moving actuators as controllers. Control effect of each actuator is assumed to be concentrated rather than actually distributed. The error generated by this assumption can be neglected if at any time instant, the area affected by each controller is very small compared with the whole area  $0 \leq x \leq 1$ .

Diff-MAS1D is able to simulate the above problem for the following cases

- Any number of sensors and actuators.
- Sensors and actuators can be collocated or non-collocated.
- Disturbances can be movable.
- Movement of sensors and actuators can be open-loop (designed by the user as functions of time only) or closed-loop (designed by the user as functions of time, sensor data, sensor position/velocity and actuator position/velocity).
- Arbitrary control algorithms designed by the user.

### B. Platform requirement

Theoretically, any operating systems with Matlab 6.5 or higher installed should be able run Diff-MAS1D. Currently, only Fedora Core 1 Linux operating system with kernel 2.4.22 and Matlab 6.5 has been tested.

### C. Installation and software directory structure

Just unzip the compressed file `diff-mas1d_ver.xxx.zip` and the installation is done!

All necessary files to run Diff-MAS1D are in the top-level directory. In the subdirectory “demo”, there are five example files showing how to write user-supplied file using a simple example. In the subdirectory “manual”, there is a Diff-MAS1D user’s manual, which is what you are reading right now.

In the top-level directory, current there are totally 14 files, among which `initialization.m`, `actrl.m`, `sctrl.m`, `dpos.m`, and `controller.m` must be filled out by the user. The user can also modify `post_process.m` to add more post-process stuff. The other files are used by Diff-MAS1D and should not be modified by the user.

*D. Bug report*

Please email to [jsliang@ieee.org](mailto:jsliang@ieee.org) or [yqchen@ece.usu.edu](mailto:yqchen@ece.usu.edu) if you believe you have found a bug.

## II. USING Diff-MAS1D

Diff-MAS1D should be used in the following steps.

- 1) Fill out file `initialization.m`, which defines parameters necessary for the top-level simulation, such as number of sensors/actuators, initial position/velocity of sensors/actuators, boundary conditions, etc.
- 2) Fill out files `actrl.m` (actuator position control algorithm), `sctrl.m` (sensor position control algorithm), `dpos.m` (pre-defined position of disturbances), and `controller.m` (diffusion control algorithm).
- 3) In Matlab environment, run `simstart.m`.
- 4) If the user-defined parameters conflict, Diff-MAS1D will abort and corrections should be made following the error messages.
- 5) Diff-MAS1D will generate a plot describing the initial positions of sensors, actuators, and disturbances and the boundary conditions. If the description is what you mean, press “y” to continue, otherwise press “n” to abort and modify `initialization.m`.
- 6) After the simulation finishes, some plots are generated on the screen and picture files corresponding to the plots are generated on the hard disk for the user to generate `avi` or `gif` animations. The `avi` functions provided by Matlab can not be used currently due to the too-high-requirement on the memory volume.

### III. DESCRIPTION OF USER SUPPLIED FILES

- `initialization.m`

In this file, following parameters should be filled out by the user.

- `K`: the positive real constant  $k$  defined in (1).
- `M`: discretization level of  $0 \leq x \leq 1$ . The index of the grids will be from 0 to  $M$ . The bigger this number is, the more accurate and slower the simulation will be. 50 is a good candidate.
- `NA`: number of actuators.
- `NS`: number of sensors.
- `ND`: number of disturbances.
- `PA0`: a vector for initial actuator positions, the length of which should be equal to `NA`. `PA0(i,1)` defines the  $x$  coordinate of the  $i$ th actuator.
- `VA0`: a vector for initial actuator velocity, the length of which should be equal to `NA`. `VA0(i,1)` defines velocity of the  $i$ -th actuator.
- `PS0`: a vector for initial sensor positions, the length of which should be equal to `NS`. `PS0(i,1)` defines the  $x$  coordinate of the  $i$ -th sensor.
- `VS0`: a vector for initial sensor velocity, the length of which should be equal to `NS`. `VS0(i,1)` defines velocity of the  $i$ -th sensor.
- `BDD`: a two-column matrix for the Dirichlet boundary conditions. Each row defines a point of Dirichlet boundary condition. For example, `BDD=[0,2]` means there is only one Dirichlet boundary condition with  $C=2$  at  $x=0$ .
- `BDN`: a three-column matrix for the Neumann boundary conditions. Each row defines a point of Neumann boundary condition. For example, `BDN=[1,0.2,0.3]` means there is only one Neumann boundary condition with  $C_1=0.2$  and  $C_2=0.3$  at  $x=1$ .  
Both points of  $x=0$  and  $x=1$  **MUST** be defined as either belong to `BDD` or to `BDN`. `Diff-MAS1D` checks this for the user.
- `DT0`: a two-column matrix describing the disturbance. Each row describes the initial position and disturbing rate (strength) of a disturbance. For example, `DT0=[0.5,10]` means there is only one disturbance with the initial position  $x=0.5$  and disturbing rate of 10.
- `u_0`: Initial condition, i.e.,  $u(x,0)$ , excluding boundary points. Defined on the grids. So the initial condition  $u(x,0)=0$  can be specified as
 
$$u_0 = \text{zeros}(M-1, 1)$$
- `SNOISE_EXIST`: a logic variable defining whether sensor noise exists. Valid value is 0 or 1.
- `MEAN_SNOISE`: mean value of sensor noise, if `SNOISE_EXIST` is equal to 1. Can be any number if `SNOISE_EXIST` is equal to 0.
- `VAR_SNOISE`: Variance of sensor noise, if `SNOISE_EXIST` is equal to 1. Can be any number if `SNOISE_EXIST` is equal to 0.
- `SIMTIME`: total simulation time.
- `PLOTSTEP`: integer greater or equal to one. After simulation finishes, generate plots along time every `PLOTSTEP` steps. Used to reduce the number of plots.

- `actrl.m`

```
function out = actrl(in)
% actrl: actuator position control. Output the acceleration of each actuator.
% The actuator is modeled as a double integrator:
%      $\ddot{x}=f(t)$,
% $f(t)$ is defined in this file.

in = in(:);

NS = in(1); % number of sensors
NA = in(2); % number of actuators

% actuator position matrix. apos(i) is the current position
% of the ith actuator
apos = in(3:NA+2);

% actuator velocity matrix. avel(i) is the current velocity
% of the ith actuator
avel = in(NA+3:2*NA+2);

% current sensor information.
% sinfo(i,1) is the current position of the ith sensor
% sinfo(i,2) is the current velocity of the ith sensor
% sinfo(i,3) is the sensed data of the ith sensor
sinfo = reshape(in(2*NA+3:2*NA+3*NS+2), NS, 3);

% current time
t = in(end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code below this line should be written by the user to %
% achieve the desired actuator movement. The final output %
% is a vector in the format of [f_1,f_2,...,f_NA]' %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In Diff-MAS1D, the actuator is modelled as a double integrator as follows

$$\ddot{x} = f(t), \tag{4}$$

The user defines  $f(t)$  in file `actrl.m`. The first part of `actrl.m` is written by the developer and should not be modified by the user. The rest part is to be filled out by the user to define the actuator position control algorithm.

The final output is a vector. Suppose there are two actuators, then the output should be in the format of

$$\text{out} = [f_1, f_2]';$$

- sctrl.m

```
function out = sctrl(in)
% sctrl: sensor position control. Output the acceleration of each sensor.
% The sensor is modelled as a double integrator:
%      $\ddot{x}=f(t)$,
%      $f(t)$ is defined in this file.

in = in(:);

NS = in(1); % number of sensors
NA = in(2); % number of actuators

% sensor position matrix. spos(i) is the current position of the ith sensor
spos = in(3:NS+2);

% sensor velocity matrix. svel(i) is the current velocity of the ith sensor
svel = in(NS+3:2*NS+2);

% current sensor data. sdata(i) is the current sensed data of the ithe sensor
sdata = in(2*NS+3:3*NS+2);

% actuator information.
% ainfo(i,1) is the current position of the ith actuator.
% ainfo(i,2) is the current velocity of the ith actuator.
ainfo = reshape(in(3*NS+3:3*NS+2*NA+2), NA, 2);

% current time
t = in(end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code below this line should be written by the user to      %
% achieve the desired sensor movement. The final output is %
% a vector in the format of [f1,f2,...,f_NA]'                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In Diff-MAS1D, the sensor is modelled as a double integrators as follows

$$\ddot{x} = f(t), \tag{5}$$

The user defines  $f(t)$  in file sctrl.m. The first part of sctrl.m is written by the developer and should not be modified by the user. The rest part is to be filled out by the user to define the sensor position control algorithm.

The final output is a vector. Suppose there are two actuators, then the output should be in the format of

```
out = [f1, f2]';
```

- controller.m

```

function out = controller(in)
% controller: control algorithm of each actuator for diffusion process control

in = in(:);
% number of sensors
NS = in(1);

% current sensor information.
% sinfo(i,1) is the current position of the ith sensor
% sinfo(i,2) is the current velocity of the ith sensor
% sinfo(i,3) is the measured data of the ith sensor
sinfo = reshape(in(2:3*NS+1), NS, 3);

% number of actuators
NA = in(3*NS+2);

% actuator information.
% [ainfo(i,1),ainfo(i,2)] is the current position of the ith actuator.
% [ainfo(i,3),ainfo(i,4)] is the current velocity of the ith actuator.
ainfo = reshape(in(3*NS+3:3*NS+2*NA+2), NA, 2);

% current time
t = in(end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% code below this line is to be written by the user to achieve the %
% desired diffusion control. The final Output should be a vector %
% in the format of [output_1,output_2,...,output_NA] %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The diffusion process control algorithm, i.e.  $f_c(x, t)$  in (1) for each actuator is implemented in controller.m.

```

#### IV. A SIMPLE SIMULATION EXAMPLE

In this section, a demo simulation is described. All user-supplied for this simulation can be found in the “demo” directory.

This simulation shows a simple design of close-loop sensor/actuator movement and closed-loop diffusion process control. The objective of this simulation is for the actuators and sensors to try to track and catch a disturbance and remove the effect the disturbance to the environment (bring  $u(x, t)$  to zero). There will be two actuators, two sensors, and one disturbance. The actuators and sensors are collocated, i.e., each sensor is bound to an actuator and they move together.

The initial position of the disturbance is at  $x = 0.7$  with fixed disturbance rate of 10. The disturbance will move back and forth in sinusoidal between  $x = 0.2$  and  $x = 0.8$ .

The algorithm for sensors/actuators to track and catch the disturbance is simply to follow the gradient of  $u(x, t)$ . To achieve this, the distance between two sensors will be fixed at any time, as shown in Fig. 1. So the gradient at the current can be approximated as (in `sctrl.m` and `actrl.m`)

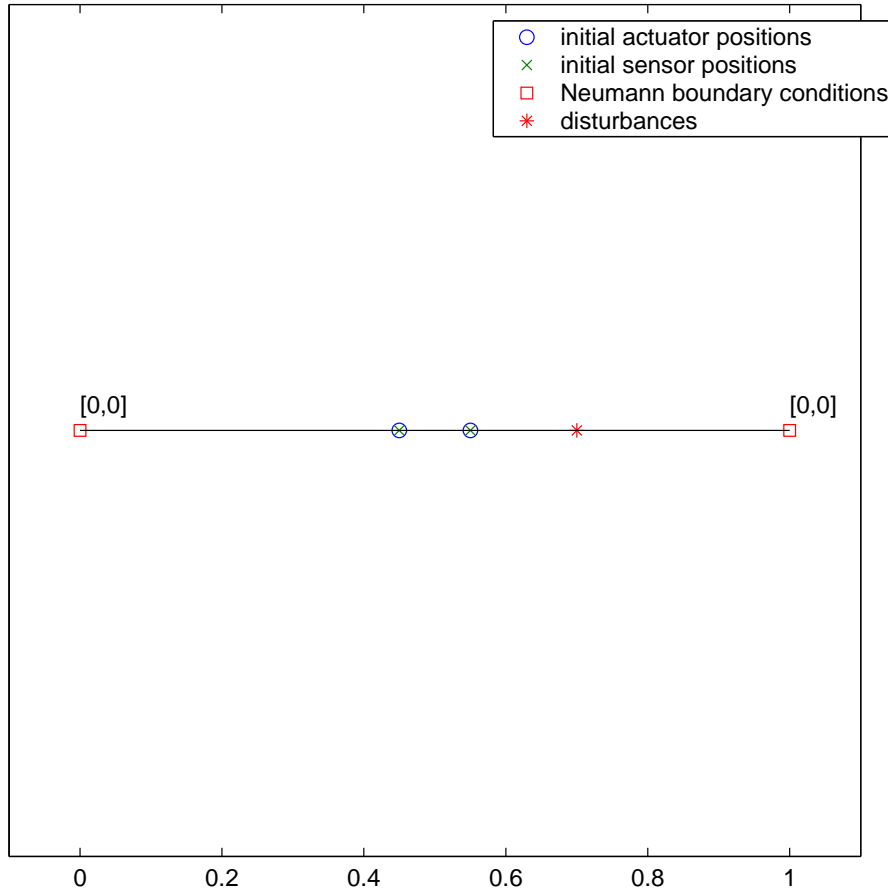


Fig. 1. Initial layout and boundary conditions

$$u_{grad} = 0.5 * [sdata(2) - sdata(1)] ;$$

Since we do not want the speed of the sensors/actuators to be too fast, the control force is composed by two components: one component points to the gradient direction to make sensors/actuators track the disturbance; another component points to the opposite direction of the current actuator/sensor velocity. The two components can be adjusted by two controller gains  $k_1$  and  $k_2$ , as shown in `sctrl.m` and `actrl.m`

$$sout = k_1 * u_{grad} - k_2 * svel(1) ;$$

The algorithm to control the diffusion process is a simple proportional controller, i.e.,  $f(x, t)$  from actuators

is simply proportional to the error between  $u(x, t)$  and the desired value (zero, actually) at the current position, as shown in `controller.m`:

```
out = -k_gain*sinfo(:,end);
```

The final control effect can be easily seen from the plots. To compare this result with the case with no control output from actuators at all, simply set the output of `controller.m` as

```
out = zeros(NA, 1);
```

The outputs are visualized in a form of a movie clip and a 3-D plot picture as shown in Fig. 2.

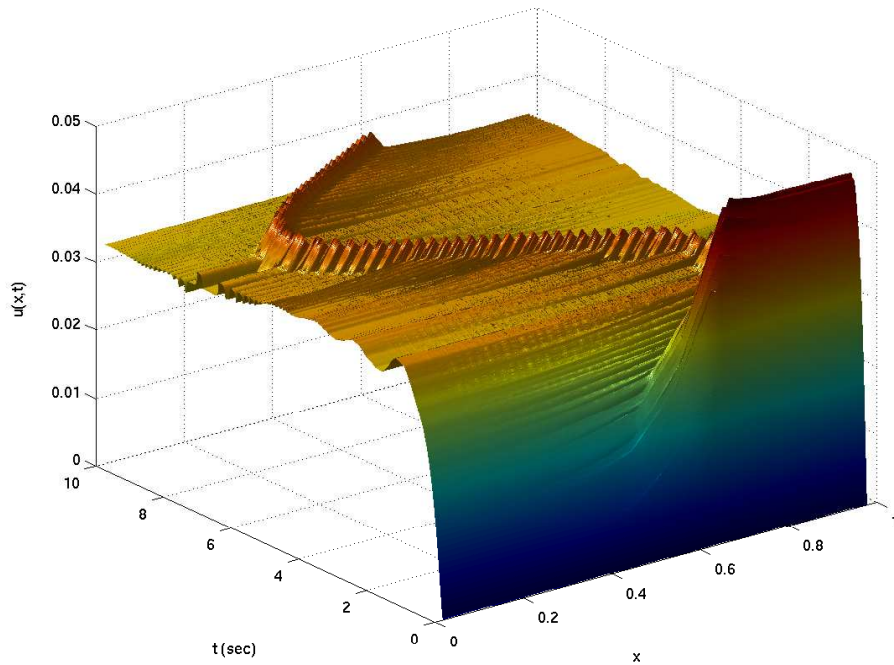


Fig. 2. 3-D plot of  $u(x, t)$

To visualize the demo output, please point your web browser to [http://www.csois.usu.edu/people/yqchen/mas-net/diff-mas1d\\_demo.gif](http://www.csois.usu.edu/people/yqchen/mas-net/diff-mas1d_demo.gif)  
The initial setup/configuration of moving source tracking and suppressing distributed control problem is shown in the figure [http://www.csois.usu.edu/people/yqchen/mas-net/diff-mas1d\\_demo\\_init.jpg](http://www.csois.usu.edu/people/yqchen/mas-net/diff-mas1d_demo_init.jpg)

## V. FUTURE WORK

- Add more terms in the PDE (1) to make it as general as possible.
- Make the constant  $k$  in the PDE (1) distributed, i.e.,  $k(x)$ .
- Solve nonlinear PDEs.