



College of Engineering™

Diff-MAS2D (version 0.9) - User's Manual

A simulation platform for controlling distributed parameter systems (diffusion) with *networked* movable actuators and sensors (MAS) in 2D 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, yqchen@ieee.org
Web: <http://www.csois.usu.edu/>

10/26/2004



Technical Report No. : USU-CSOIS-TR-04-03

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

Diff-MAS2D[©] (version 0.9)

User's Manual

A simulation platform for controlling distributed parameter systems (**diff**usion)
with *networked* **m**ovable **a**ctuators and **s**ensors (MAS) in **2D** 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-MAS2D is a software package for simulation of diffusion process control using moving actuators and moving sensors. The main reason to develop this software is that no currently available software package (Matlab, Maple, Mathematica, MathCAD, Ansys, Nastran, FEMLAB) is able to solve the problem Diff-MAS2D is able to solve.

Diff-MAS2D is written completely in Matlab script and Simulink.

A. Problem formulation

Diff-MAS2D is able to solve the following problem numerically.

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

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

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

- 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 $u(x, y, t)$ 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 $0 \leq x \leq 1$ and $0 \leq y \leq 1$.

Diff-MAS2D 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-MAS2D. 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-mas2d.zip` and the installation is done!

All necessary files to run Diff-MAS2D are in the top-level directory. In the subdirectory “demo”, there are four example files showing how to write user-supplied file using a simple example. In the subdirectory “manual”, there is a Diff-MAS2D user’s manual, which is what you are reading right now.

D. Bug report

Please email to jsliang@ieee.org or yqchen@ece.usu.edu if you believe you have found a bug.

II. USING Diff-MAS2D

Diff-MAS2D 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-MAS2D will abort and corrections should be made following the error messages.
- 5) Diff-MAS2D 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. 40 is a good candidate.
- `N`: discretization level of $0 \leq y \leq 1$. The index of the grid will be from 0 to N . The bigger this number is, the more accurate and slower the simulation will be. 40 is a good candidate.
- `NA`: number of actuators.
- `NS`: number of sensors.
- `ND`: number of disturbances.
- `PA0`: a two-column matrix for initial actuator positions. The number of rows should be equal to `NA`. For the i th row, `PA0(i,1)` and `PA0(i,2)` define the x and y coordinates, respectively, of the i th actuator.
- `VA0`: a two-column matrix for initial actuator velocity. The number of rows should be equal to `NA`. For the i th row, `VA0(i,1)` and `VA0(i,2)` define velocity in x and y directions, respectively, of the i -th actuator.
- `PS0`: a two-column matrix for initial sensor positions. The number of rows should be equal to `NS`. For the i th row, `PS0(i,1)` and `PS0(i,2)` define the x and y coordinates, respectively, of the i -th sensor.
- `VS0`: a two-column matrix for initial sensor velocity. The number of rows should be equal to `NS`. For the i -th row, `VS0(i,1)` and `VS0(i,2)` define velocity in x and y directions, respectively, of the i -th sensor.
- `BDD`: a five-column matrix for the Dirichlet boundary conditions. Each row defines a segment of Dirichlet boundary condition. For example, `BDD=[0,0.1,0,0.5,2]` means there is only one segment of Dirichlet boundary condition with $C=2$ from $[0,0.1]$ to $[0,0.5]$.
- `BDN`: a six-column matrix for the Neumann boundary conditions. Each row defines a segment of Neumann boundary condition. For example, `BDN=[0,1,0.7,1,0.2,0.3]` means there is only one segment of Neumann boundary condition with $C_1 = 0.2$ and $C_2 = 0.3$ from $[0,1]$ to $[0.7,1]$. Each point on the boundary **MUST** be defined as either belong to `BDD` or to `BDN`. Although currently `Diff-MAS2D` does not check this, the user can check it from the plot describing the initial positions and boundary conditions. Dirichlet boundaries are plotted in black and Neumann boundaries are plotted in red. If any segment of boundary is not plotted, the user should press “n” to abort and modify `initialization.m`.
- `DT0`: a three-column matrix describing the disturbance. Each row describes the initial position and disturbing rate (strength) of a disturbance. For example, `DT0=[0.5,0.6,10]` means there is only one disturbance with the initial position $[0.5,0.6]$ and disturbing rate of 10.
- `u_0`: Initial condition, i.e., $u(x,y,0)$, excluding boundary points. Defined on the grids. So the initial condition $u(x,y,0) = 0$ can be specified as

`u_0=zeros(M-1,N-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 modelled as two double integrators:
%           $\ddot{x}=f_x(t)$,
%           $\ddot{y}=f_y(t)$.
% $f_x(t)$ and $f_y(t)$ are defined in this file.

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

% actuator position matrix. [apos(i,1),apos(i,2)] is the
% current position of the i-th actuator
apos = reshape(in(3:2*NA+2), NA, 2);

% actuator velocity matrix. [avel(i,1),avel(i,2)] is the
% current velocity of the i-th actuator
avel = reshape(in(2*NA+3:4*NA+2), NA, 2);

% current sensor information.
% [sinfo(i,1),sinfo(i,2)] is the current position of the i-th sensor
% [sinfo(i,3),sinfo(i,4)] is the current velocity of the i-th sensor
% sinfo(i,5) is the sensed data of the i-th sensor
sinfo = reshape(in(4*NA+3:4*NA+5*NS+2), NS, 5);

% 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 [fx_1,fx_2,...,fx_NA,fy_1,fy_2,...,fy_NA]'.      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In Diff-MAS2D , the actuator is modelled as two double integrators as follows

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

$$\ddot{y} = f_y(t). \tag{5}$$

The user defines $f_x(t)$ and $f_y(t)$ 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

```
out = [fx1, fx2, fy1, fy2]';
```

- sctrl.m

```
function out = sctrl(in)
% sctrl: sensor position control. Output the acceleration
% of each sensor.
% The sensor is modelled as two double integrators:
%           $\ddot{x}=f_x(t)$,
%           $\ddot{y}=f_y(t)$.
% $f_x(t)$ and $f_y(t)$ are defined in this file.

in = in(:);

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

% sensor position matrix. [spos(i,1),spos(i,2)] is the current
% position of the i-th sensor
spos = reshape(in(3:2*NS+2), NS, 2);

% sensor velocity matrix. [svel(i,1),svel(i,2)] is the current
% velocity of the i-th sensor
svel = reshape(in(2*NS+3:4*NS+2), NS, 2);

% current sensor data. sdata(i) is the current sensed data of
% the i-th sensor
sdata = in(4*NS+3:5*NS+2);

% actuator information.
% [ainfo(i,1),ainfo(i,2)] is the current position of the i-th actuator.
% [ainfo(i,3),ainfo(i,4)] is the current velocity of the i-th actuator.
ainfo = reshape(in(5*NS+3:5*NS+4*NA+2), NA, 4);

% 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 [fx_1,fx_2,...,fx_NA,fy_1,fy_2,...,fy_NA]' %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In Diff-MAS2D , the sensor is modelled as two double integrators as follows

$$\ddot{x} = f_x(t), \tag{6}$$

$$\ddot{y} = f_y(t). \tag{7}$$

The user defines $f_x(t)$ and $f_y(t)$ 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 = [fx1, fx2, fy1, fy2]';
```

- 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),sinfo(i,2)] is the current position of the i-th sensor
% [sinfo(i,3),sinfo(i,4)] is the current velocity of the i-th sensor
% sinfo(i,5) is the sensed data of the i-th sensor
sinfo = reshape(in(2:5*NS+1), NS, 5);

% number of actuators
NA = in(5*NS+2);

% actuator information.
% [ainfo(i,1),ainfo(i,2)] is the current position of the i-th actuator.
% [ainfo(i,3),ainfo(i,4)] is the current velocity of the i-th actuator.
ainfo = in(5*NS+3:5*NS+4*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. $f_c(x, y, t)$ 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 $u(x, y, t)$ to zero). There will be five actuators, five 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 $(0.5, 0.25)$ with fixed disturbance rate of 15. The disturbance will move back and forth in sinusoidal between $(0.2, 0.25)$ and $(0.8, 0.25)$.

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

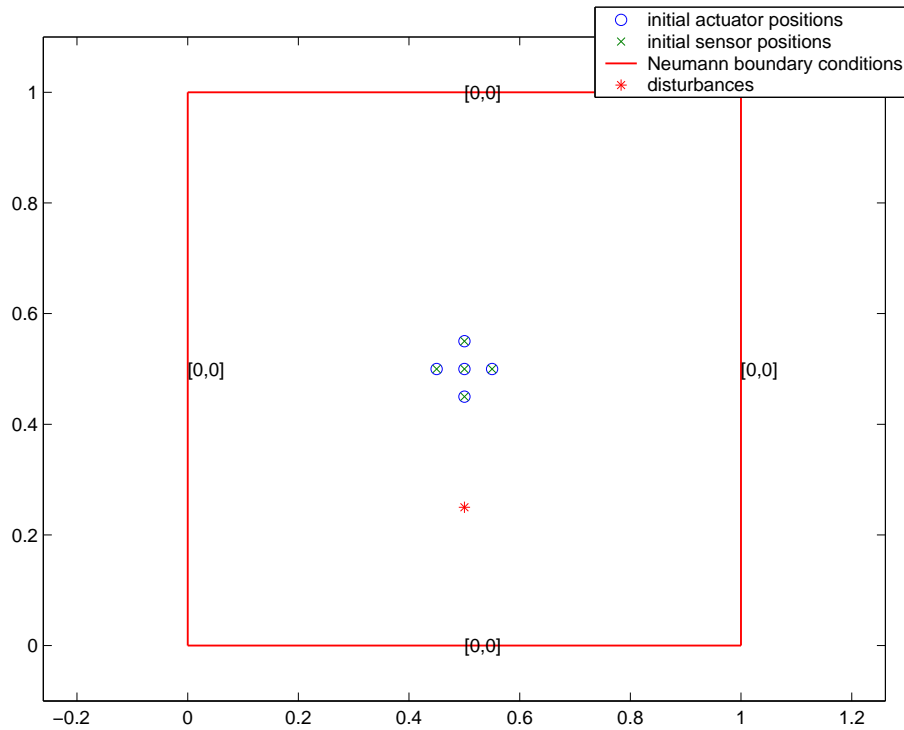


Fig. 1. Initial layout and boundary conditions

```
ugrad = 0.5*[sdata(3)-sdata(2),sdata(5)-sdata(4)];
```

Since the current gradient direction is usually different from the current velocity direction of sensors/actuators, control force can not be simply applied to the sensors/actuators in the gradient direction (try this to see what happens). 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 $k1$ and $k2$, as shown in `sctrl.m` and `actrl.m`

```
sout = k1*ugrad-k2*svel(1,:);
```

The algorithm to control the diffusion process is a simple proportional controller, i.e., $f(x, y, t)$ from actuators is simply proportional to the error between $u(x, y, t)$ 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 output is visualized in a form of a movie clip. Please point your web browser to

http://www.csois.usu.edu/people/yqchen/mas-net/diff-mas2d_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-mas2d_demo_init.jpg