



College of Engineering™

Wave-MAS2D (version 0.8) - User's Manual

A simulation platform for controlling distributed parameter systems (Wave equation) 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 University
4160 Old Main Hill, Logan Utah 84322-4160, USA
Emails: jслиang@ieee.org, yqchen@ieee.org
Web: <http://www.csois.usu.edu/>

11/4/2004



Technical Report No. : USU-CSOIS-TR-04-05

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

Wave-MAS2D[©] (version 0.8)

User's Manual

A simulation platform for controlling distributed parameter systems (wave equation) with *networked* movable actuators and sensors (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

Wave-MAS2D is a software package for simulating control of the wave equation 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 Wave-MAS2D is able to solve.

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

A. Problem formulation

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

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

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.

Currently only the following Dirichlet boundary conditions are supported

$$u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(x, 1, t) = 0 \quad (2)$$

More boundary conditions will be supported in the future version.

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 $0 \leq y \leq 1$.

Wave-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 Wave-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 `wave-mas2d_ver_xxx.zip` and the installation is done!

All necessary files to run Wave-MAS2D are in the top-level directory. In the subdirectory "demo", there are five example Matlab m-files showing how to write user-supplied files using a simple example. In the subdirectory "manual", there is a Wave-MAS2D user's manual, which is what you are reading right now.

In the top level directory, there are totally 15 .m and .mdl files used by Wave-MAS2D, among which `initialization.m`, `actrl.m`, `sctrl.m`, `distout.m` and `controller.m` must be filled out by the user; `post_process.m` can also be modified by the user to add more post-process stuff. Any other .m files are used by Wave-MAS2D and should not be modified.

D. Bug report

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

II. USING WAVE-MAS2D

Wave-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, etc.
- 2) Fill out files `actrl.m` (actuator position control algorithm), `sctrl.m` (sensor position control algorithm), `distout.m` (pre-defined position of disturbances), and `controller.m` (the wave equation control algorithm).
- 3) In Matlab environment, run `simstart.m`.
- 4) If the user-defined parameters conflict, Wave-MAS2D will abort and corrections should be made following the error messages.
- 5) Wave-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 velocities. The number of rows should be equal to NS . For the i th row, $VS0(i, 1)$ and $VS0(i, 2)$ define the initial velocity in x and y directions, respectively, of the i -th sensor.
- $PD0$: a two-column matrix describing the initial disturbance positions. The number of rows should be equal to ND . Each row shows the initial position of the i -th disturbance. For example, $PD0 = [0.5, 0.6]$ means there is only one disturbance with the initial position $[0.5, 0.6]$.
- BDD : a five-column matrix for the Dirichlet boundary conditions. Currently BDD is fixed and should not be modified by the user.
- 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
$$u_0 = \text{zeros}(M-1, N-1)$$
- $NOISE_SU_EXIST$: a logic variable defining whether noise exists for displacement sensor. Valid value is 0 or 1.
- $MEAN_NOISE_SU$: mean value of displacement sensor noise, if $NOISE_SU_EXIST$ is equal to 1. Can be any number if $NOISE_SU_EXIST$ is equal to 0.
- VAR_NOISE_SU : Variance of displacement sensor noise, if $NOISE_SU_EXIST$ is equal to 1. Can be any number if $NOISE_SU_EXIST$ is equal to 0.
- $NOISE_SV_EXIST$: a logic variable defining whether noise exists for velocity sensor. Valid value is 0 or 1.
- $MEAN_NOISE_SV$: mean value of velocity sensor noise, if $NOISE_SV_EXIST$ is equal to 1. Can be any number if $NOISE_SV_EXIST$ is equal to 0.
- VAR_NOISE_SV : Variance of velocity sensor noise, if $NOISE_SV_EXIST$ is equal to 1. Can be any number if $NOISE_SV_EXIST$ 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`

```

% actrl: actuator position control. Output the acceleration of
%     each actuator.
%     The actuator is modeled 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 ith 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 ith 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 ith sensor
% [sinfo(i,3),sinfo(i,4)] is the current velocity of the ith sensor
% sinfo(i,5) is the measured displacement from the ith sensor
% sinfo(i,6) is the measured velocity from the ith sensor
sinfo = reshape(in(4*NA+3:4*NA+6*NS+2), NS, 6);

% 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 Wave-MAS2D, the actuator is modeled as two double integrators as follows

$$\ddot{x} = f_x(t), \tag{3}$$

$$\ddot{y} = f_y(t). \tag{4}$$

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 modeled 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 ith 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 ith sensor
svel = reshape(in(2*NS+3:4*NS+2), NS, 2);

% current sensor data.
% sdata(i,1) is the current measured displacement from the ith sensor
% sdata(i,2) is the current measured velocity from the ith sensor
sdata = reshape(in(4*NS+3:6*NS+2),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(6*NS+3:6*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_NS,fy_1,fy_2,...,fy_NS]' %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In Wave-MAS2D, the sensor is modeled as two double integrators as follows

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

$$\ddot{y} = f_y(t). \tag{6}$$

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]';
```

- distout.m

```
function out = distout(in)
%out = distout(in)    return positions and disturbing forces at current time

ND = in(1); %number of disturbances
PD0 = reshape(in(2:end-1), ND, 2); % initial disturbance position matrix

% current time
t = in(end);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code below this line should be written by the user to specify %
% the disturbance positions and disturbing forces at time t.      %
% The output is a vector with length 3*ND.                        %
% out=[x_1,x_2,...,x_ND,y_1,y_2,...,y_ND,fd_1,fd_2,...,fd_ND]' %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

distout.m output the pre-defined disturbance position and disturbing force. The output is a vector with length of $3 * ND$.

- controller.m

```
% controller: control algorithm of each actuator for control of
% the wave equation
```

```
in = in(:);
```

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

```
% current sensor information.
% [sinfo(i,1),sinfo(i,2)] is the current position of the ith sensor
% [sinfo(i,3),sinfo(i,4)] is the current velocity of the ith sensor
% sinfo(i,5) is the measured displacement from the ith sensor
% sinfo(i,6) is the measured velocity from the ith sensor
sinfo = reshape(in(2:6*NS+1), NS, 6);
```

```
% number of actuators
NA = in(6*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(6*NS+3:6*NS+4*NA+2), NA, 4);
```

```
% current time
t = in(end);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% code below this line is to be written by the user to control the %
% wave equation. The final Output should be a vector in the format of %
% out = [output_1,output_2,...,output_NA] %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The wave equation 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 to suppress the vibration caused by the disturbance. There will be eighty-one actuators, eighty-one sensors, and one disturbance. The actuators and sensors are collocated, i.e., each sensor is bound to an actuator. Throughout the simulation, the actuators and sensors are fixed.

The initial position of the disturbance is at $(0.5, 0.25)$ and position is fixed throughout the simulation. The disturbing force is 0.5. The disturbance exists in the first five seconds of simulation and then disappears (similar to an impulse disturbance).

The algorithm to suppress the vibration is simply for each actuator to generate control force proportional to the measured velocity but in the opposite direction.

The initial setup is shown in Fig. 1.

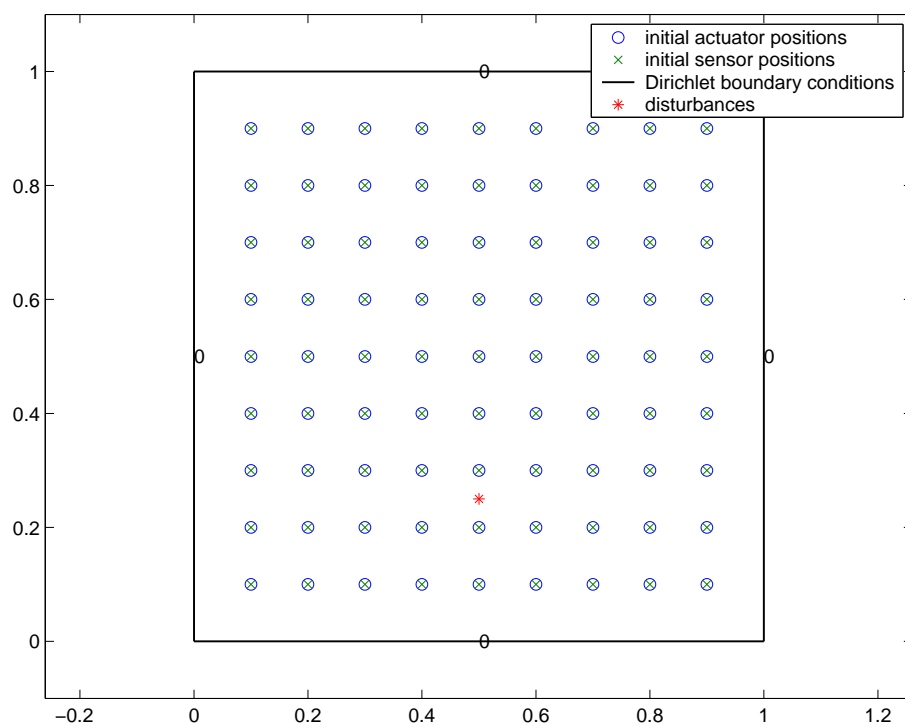


Fig. 1. Initial layout and boundary conditions

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 controlled 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/wave-mas2d_demo_controlled.gif

The un-controlled 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/wave-mas2d_demo_uncontrolled.gif

The initial setup/configuration is shown in the figure

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