HOW TO USE MBSIM

The following outline program section will present the basic steps of writing MBSim source code, compiling and execution, and some user interactive features with the output graphs.

MAIN STEPS FOR CODING AND MBSIM APPLICATION:

Main Program Structure and Coding Procedures

Name Section:
// Name.cc file (Vehicle.cc)
// Version
	It is always good to give your program a meaningful name and remember to set a version number

#Include Section:	Header files needed for the source code
MAIN PROGRAM SECTION: **{**	Start main program
View Section:	Build View Object and view window setting
Mechanism Section:	Build Mechanism object
Enviroment Section:	Build Enviroment objects
Motion Control Section:	Drive mechanism moving with position or velocity requirements
Sensors and Map Section:	Activate sensors and Get Feedback
Graphics Output Section:	Graphics output routine and end of all the program sections
}	End of main program

#Include Section:

The following header files are used in most MBSim codes:

# include <Graphics_View.hh>	– support view section
# include <Mechanism.hh>	– support mechanism section
# include <Environment.hh>	– support environment section
# include <Sensor.hh>	– support sensors section
# include <Map.hh>	– support map
# include <unistd.h>	– support basic unix c++ operation

The header files are included corresponding to the sections needed in the main program, however, it is no harm to include extra headers.

Main Program Section:

View Section:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Build View Object	Graphics_View view1;
2. Read in view script file	view1.ReadView("auto.view");

3. Write a new auto.view file in the directory as the souce code	700 700
	1220 840 410
	0 0 0
	//size of view window
	//view eye position (x,y,z)
	//where to look

Mechanism Section:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Declare a mechanism object	Mechanism auto_obj;
2. Read in view script file	auto_obj.ReadDescription(view1, "auto.md")
3. Write a new mechanism description file auto.md	Auto T 0, 0, 0, 0, 0.2, 200 auto.gd auto.sd 1 1000, 1, 1,1

```
1 0 0
0 1 0
0 0 1

//Name of .md file
//Mechanism type and parameters
//graphic description file
//sensor description file and amount
//mass, center of mass
//inertia matrix
```

4. Write a new graphic description file auto.gd

```
RP 300 200 100 10
1 0 0
0 1 0
0 0 1

50 0 50

C 50 40 11 4
1 0 0
0 1 0
0 0 1

100 0 50

//Rectangular Prism, x, y, z, color
//rotation matrix
//translation in x, y, z

//Cylinder type, r, length, sides, color
//rotation matrix
//translation in x, y, z
```

5. Write a new sensor description file auto.sd

```
Ultrasonic 1000 0.1
1 0 0
0 1 0
0 0 1

150 0 50

// sensor type, max and min range
//rotation matrix
//translation in x, y, z
```

6. Write a new wheel.gd if mechanism type is Tricycle, otherwise, ignore this part. Two flat Cylinders are wheels.

```
C  25 20 20 2
1 0 0
0 0 –1
0 1 0

0 50 10

C  25 20 20 2
1 0 0
0 0 –1
```

	0 1 0
	
	0 –50 10
	
	//Cylinder type, r, length,sides, color
	//rotation matrix
	//translation in x, y, z
	
	//Cylinder type, r, length,sides, color
	//rotation matrix
	//translation in x, y, z

Environment Section:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Write a new auto.env file	O 0 0 0 auto_env.gd //Object(O), origin, objects graphics
2. Write environment objects graphic description file auto_env.gd	RP 1000 1000 200 2 1 0 0 0 1 0 0 0 1 0 0 75 //same as other .gd file

Motion Control Section:

Position Following Control:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Move the object upon position	auto_obj.MovePosition(view1, "auto.pos");

2. Write position requirement file auto.pos for Tricycle	0.0 0 0 0 0 0.1 100 0 0 0 0.2 200 0 0 0 0.3 400 .0 0 0 //time, x, y, z, streering angle

Velocity Following Control:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Move the object upon velocity	auto_obj.MoveVelocity(view1, "auto.vel");
2. Write velocity requirement file auto.vel for Tricycle	0.0 0 0 0 0 0.1 10 .1 .2 0.2 20 0 .3 0.3 10 .2 .4 //time, link speed, body ration speed //...streering turning speed

Sensors and Map Section:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Create a Map 2. Read in *.mp file	Map DataMap; DataMap("auto.mp");

3. Write map file auto.mp	68 50
	1 100 1
	auto 1
	
	//map window size
	//1 for ping–on, cell is 0.1m x 0.1 m
	//mechanism name, sensor id list

4. Turn on sensors 5. Declear an array to store data	int sensor_on = 1; Sensor_Data *Range;

6. Activate the sensors	`if(sensor_on){ Range= auto_obj.ActivateSensorNum(1); }`
7.Activate all sensors in auto.mp	`DataMap.UpdateMap();`
8.Clear map and reset each cell to 1	`DataMap.ClearMap(1);`
9. (Optional) Histogram declare	`Histogram Di("Sensor Data Histogram", DataMap.Get_Ncols(), DataMap.Get_Nrows(),5);`
10. Histogram data drawing	`Di.Draw( DataMap.Get_Map() );`

Graphics Output Section:

BASIC STEPS AND DESCRIPTION	CODING EXAMPLES
1. Refresh the screen and redisplay	`view1.Regenerate_Screen();`
2. End of all the scetions and the main program.	`cout << "End of Program\n";`

For addtional information about a particular section or file, please refer to **Detailed MBSim References Section.**

Compiling and Execution of the Program:

We used a Makefile (in directory as the source code) to make executable code. The main reasons are: simple, easy to use and modify, and more manageable. The Following is a sample Makefile for Vehicle.cc .

```
#Makefile
#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
#                    COMPILER DEFAULTS

# –g = debugger info
# –O = optimize
CFLAGS = –O

DIR = /usr/people/mbsim/WorkSpace/mbsimLibrary
INC = –I$(DIR)
LIB = $(DIR)/mbsim.a

MORELIBS = –lm –lgl_s –lg++ –lXm_s –lXt_s –lX11_s

.SUFFIXES:
.SUFFIXES: .o .cc .hh .h .c .diff .log

#OBJECTS = .o filr name list

#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
#                      MAIN TARGETS
#
# make each .o file for a large programs


#––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––
#                          RULES

.cc.o: $(OBJECTS)
	gcc $(CFLAGS) –c $*.cc –I. $(INC)

Vehicle: Vehicle.cc $(OBJECTS) $(LIB)
	gcc –o $@ $@.cc $(CFLAGS) –I. $(INC) $(OBJECTS) $(LIB) $(MORELIBS)
```

To run executable Vehicle code, user can simply type: **Vehicle** (file name) in the same directory of the source code to run the program automatically. Program will manager all the graphics automatically also.

User Interaction:

Users can use MBSim build mouse and key stroke to manage the program outputs very nicely, especially with graphics. The following list is some nice and mostly common used mouse and key stroke:

Key Stroke:

ESC	– end of program
SPACE	– pause motion toggle
SHIFT KEY	– 1/10th scale on all motions
P	– poa/eye mode toggle
S	– print out every state
L	– line/fill mode toggle
C	– print out coordinates of eyes and poa
F	– toggle between flat shading and ground shading

Mouse:

LEFT BUTTON and MOVE HORZ	– eye mode: rotate eye about z−axis – poa mode: move poa left/right
LEFT BUTTON and MOVE VERT	– eye mode: zoom in and out – poa mode: move poa toward/away
MIDDLE BUTTON and MOVE VERT	– eye mode: rotate eye up and down – poa mode: move poa up/down z−axis