SAMPLE PROGRAMS:

Two Examples are shown in this section, the first example is easier to start without sensor and map section. The second example is more complete which includes more features of MBSim.

First Sample Program:

Main Code: Vehicle.cc

```
//  Test program Vehicle.hh
//  Version 1.000

#include <Graphics_View.hh>
#include <Environment.hh>
#include <Mechanism.hh>
#include <unistd.h>


main() {

/////////////////////////////////////////////////
// Build View object, and .view file

   Graphics_View view1;  // FG allow you to take control of view windo
   view1.ReadView("auto.view");


/////////////////////////////////////////////////
// Build Mechanism Object, and .md file
// .md file(need a .gd file to make view nicer)
// if use tricycle, a build-in wheel.gd is needed

   Mechanism auto1;
   auto1.ReadDescription(view1, "auto.md");


//////////////////////////////////////////////////////
// Build Position Motion Requirement, and .pos is needed// move the object upon
// position
// MovePosition() is integration of ReadPosition()

//   auto1.MovePosition(view1, "auto.pos"); //

/*  ifstream inputfile("auto.pos",ios::in);

   if(inputfile == NULL) {
      cerr << "\nError: Position file, "
           << ", could not be found.\n";
```

```
      exit(1);
      }

   while( !auto1.ReadPosition(inputfile) ) {
      sleep(1);      // let the process pause for a second
      view1.Regenerate_Screen();
      cout << "Time: " << auto1.Time() << "\n";
      }

   inputfile.close();
*/

////////////////////////////////////////////////////////////
// Build Velocity Motion Requirement, and .vel is needed
// move the object upon Velocity
// MoveVelocity() is integration of ReadVelocity()

//   auto1.MoveVelocity(view1, "auto.vel");

   ifstream inputfile("auto.vel",ios::in);

   if(inputfile == NULL) {
      cerr << "\nError: Velocity file, "
           << ", could not be found.\n";
      exit(1);
      }

   while( !auto1.ReadVelocity(inputfile) ) {
      sleep(1);
      auto1.IntegrateVelocity();
      view1.Regenerate_Screen();
      cout << "Time: " <<auto1.Time() << "\n";
      }

   inputfile.close();

   while(1)
      view1.Regenerate_Screen();

   cout << "End of Program\n";
}
```

---

auto.view file

```
700 700

1220 840 410
0 0 0
```

```
–350 330 700
0 0 0

//size of window
//where is eye
//where try to look
```

---

auto.md

```
Auto
T 0 0 0 0 0.2 200
base.gd

100 1 1 1
1 0 0
0 1 0
0 0 1

R 0 100 100 0
turret.gd

50 1 1 1
1 0 0
0 1 0
0 0 1


// Mechanism name
// Type, x,y,z, theta(rad), streering(rad), length
// graph file .gd
// sensor file (if none, leave space), #sensor(none,then space)
// mass(doesn't matter), center of mass(respcet to link origin)
// inertia matrix (3x3)
```

---

auto.gd

```
RP 300 200 100 10
1 0 0
0 1 0
0 0 1
50 0 50

C 50 50 11 4
```

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

50 0 125

C 10 200 10 5

0 0 1
0 1 0
–1 0 0

200 0 125


// x,y,z, color
// rotation matrix
// translation
```

Wheel.gd

```
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
```

---

auto.pos

```
0.0  0 0 0 0
0.1  100 0 0 .2
0.2  200 0 0 .3
0.3  300 0 0 .4
0.4  350 0 0 .5
0.5  350 0 0 .4
```

```
0.6  350 0 0 .2
0.7  350 0 0 0
0.8  350 0 0 –.2
0.9  350 0 0 –.4
1.0  400 0 0 –.3
1.1  420 10 0 –.2
1.2  450 20 0 0
```

---

auto.vel

```
0.5 50 –.2 1
1.0   10 .1 2
2.0   20  .1 2
3.0   10  .1 2
4.0   50   .1 2
5.0   –100  .1 2
6.0   20  –.1 0
6.1   40 –.2 0
6.3   30 –.2 0
6.7   20 –.1 –3
6.8   10 0 –5
7.0   10 0 –5
8.0   70 0  0
9.0   80 0 0
10.0  90 0 0
10.2  100 0 0
10.3  200 0 1
10.4  200 0 2
10.5  100 0 3
11    100 –0.1 0
12    100 –0.1 0
13    100 0 0
14    100 0 0
```

Second Sample Program:

Following is an example program main() that pulls together all the pieces described above. It tells which script files are to be read and builds a tricycle (a simple board and three wheels: see Figure 11) with a torque plots for the front wheel and a force plot for the tricycle base. At each time increment, range data from the first four sensors is returned.

===

Example: source code

```
// Tricycle – Ultrasonic sensors
//         with force torque plots

#include <Graphics_View.hh>
#include <Environment.hh>
#include <Mechanism.hh>
#include <unistd.h>
#include <Sensor.hh>
#include <Map.hh>

main() {
//Declare Graphics View, give it an arbitrary name, e.g., view_one
  Graphics_View view_one;
  view_one.ReadView("Tricycle.view");

//Declare Environment, give it an arbitrary name, e.g., environment
  Environment environment;
  environment.ReadDescription(view_one, "Tricycle.env");

//Start reading in the mechanism, named tricycle
  Mechanism tricycle;
  tricycle.ReadDescription(view_one, "Tricycle.md");

//Read configuration (optional initial conditions or use .md positions)
//   tricycle.ReadConfiguration(view_one, "Tricycle.cfg");

  Plot forceplot("Force");   // See example above
  forceplot.SetLimits("force.plot");

  Plot torqueplot("Torque");  // Torque required to turn front wheel
  torqueplot.SetLimits("torque.plot");

// Create a Map
  Map DataMap;
  DataMap.ReadMap("SWAMI.mp");

//Begin motion, read velocity file

  ifstream inputvelfile("Tricycle.vel",ios::in);

  int sensor_on = 1;                    // turn sensors on
  float time;
  double force, torque;
```

```
  Vector3 zeros;
  Iterator<Joint_Value> forces_iter;  // intelligent pointer for force/torque plots

  while( !inputvelfile.eof() ) {

    tricycle.ReadVelocity(inputvelfile);  // Read next line of velocity file
    tricycle.IntegrateVelocity();          // Get joint position data from given velocities

    Sensor_Data *Range;
    if ( sensor_on ) {                                        // Activate sensors
        Range = tricycle.ActivateSensorNum(6);
        cout << "Range from sensor number 6:   " << Range−>Range() << "\n";
                                    // used to retrieve individual sensor range readings
        }

DataMap.UpdateMap();   // activate all sensors listed in SWAMI.mp and put
DataMap.ClearMap(1);    // clear map and fill each cell with value of 1

    view_one.Regenerate_Screen();    // Update and redraw screen

//Calculate current inverse dynamics of tricycle and put in forces_iter
    forces_itert = tricycle.InverseDynamics(zeros, zeros);

// Draw force and torque plots
    forces_iter.Head();    // Go to head of torque_list
    force = forces_iter()−>JointValue();
    forces_iter++;
    torque =forces_iter()−>JointValue();

    time = tricycle.Time();      // Time stamp torque data
    cout << "Time: " << time << "\n";
    forceplot.Draw(time, force);   //Sends data to plot and draws
    torqutplot.Draw(time, torque);

    }

  inputvelfile.close();   // Stop reading velocity file and close it.

  cout << "End of Program\n";
}
==============================================================
```

CONTINUING DEVELOPMENTS

The two main areas of continuing development are involved with Map and path planning. Currently, Map can retain the Sensor readings from the previous time step. By translating, fewer computations are performed, the Map stays consistently oriented, and discretization round off errors will not occur to the moved data. From the discretized Map data, useful information needs to be extracted for use in path planning and obstacle avoidance. Some methods being considered for extracting Map data are potential field, edge detection, and histogrammatic methods. The connection between these methods which further interpret the Map data and the path planning and obstacle avoidance schemes needs further work.

From the other side, path planning along a nominal path for nonholonomic vehicles is underway. A common approach with nonholonomic vehicles is to compose the path of straight lines and circles. This works fine if the vehicle comes to a stop at the interface of each path segment. For example, a tricycle going around a circle will have a constant non−zero steering angle. On a straight line, the tricycle will have a steering angle of zero. At the circle to line interface the steering wheel must instantaneously change from one angle to zero. A real vehicle cannot achieve this since it would require an infinite amount of torque by the steering actuator. In the absence of a closed loop controller since we are initially assuming no disturbances and that we can follow the path, the open loop path planning must transition between the circle and line segments. Since the nonholonomic vehicles have a minimum turning radius greater than zero when the vehicle speed is nonzero, it is desired to use a curve that can be specified for a known curvature. The curvature is the inverse of the radius of the curve at that instant. The transistion curve choosen was quarter cosine wave since the curvature at the peaks can be readily matched to the circle and the inflection point curvature at a quarter of the wave's period matches the straight line. There are some difficulties with the cosine segment if the arc around the circle is too small.

A potential method for determining the joint variables of nonholonomic vehicles uses the curvature and rates of curvature of the path segments and the vehicles kinematic equations. Curvature relations of the Knife, Tricycle, and Dual Wheel links have been initially developed but not fully tested. The Knife link has been initially simulated and tested with mostly success. All of these issues are currently under development and look to be promising.

REFERENCES:

[1] Borenstein, Johann and Koren, Yoram, Histogramic In−Motion Mapping for Mobile Robot Obstacle Avoidance, *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, August 1991, pp. 535−539.

[2] Cameron, Jonathan, Modeling and Motion Planning for Nonholonomic Systems, Doctoral Thesis, Georgia Institute of Technology, December 1993.

[3] Craig, John, Introduction to Robotics: Mechanics and Control, 2nd Ed., Addison−Wesley Publishing, 1989.

[4] Schmitt, Hogan, Cameron, and Book, Integrated Modeling, Simulation, and Animation of Rigid Arms and Vehicles through Object−Oriented Programming, *Proceedings of the IMACS Symposium on Mathematical Modelling*, vol. 3, February 1994, pp. 566−569.

[5] Hogan, J.E. and Book, W.J. "Simulation, Animation, and Control of Multi−Body Systems", "Spectrum '94: Nuclear and Hazards Waste Management, International Topical Meeting" Atlanta, GA. August 14−18, 1994, vol. 2, pp1068−1071.