2006-810: SIMULATION-CENTERED MECHATRONICS

Michael Holden, San Francisco State University

Simulation-Centered Mechatronics

Introduction

Mechatronics is a multi-disciplinary topic that covers the intersection of electrical and mechanical design, including sensors and signal conditioning, control systems and actuator development. It is difficult to teach a well-balanced mechatronics class without favoring one discipline over the others, since most instructors are part of an electrical or mechanical engineering group. One technique for uniting the disciplines is numerical simulation of the mechatronic system [1], which allows students to focus on the system design rather than the individual components.

Mechatronics is primarily a systems design field [2], and like most complex systems, mechatronic designs can be improved through computer simulation. The simulation is used for several purposes: it ties the sensor electronics, control algorithm and actuator model together; it reinforces the mathematical concepts behind state-space models and state-space control, and it teaches the value of design by simulation, especially when the students must build the system from their numerical model as part of a lab.

The class design is novel in that the separate topics of sensors and signal conditioning, controllers and programming, and actuators are united in the simulation. The students are introduced to the individual components, their numerical models are discussed, and they are combined into a system simulation. At San Francisco State University (SFSU) the students must design and simulate a unique system as a final project, and the accompanying lab requires students to build and test the system, using their simulation as a design tool.

At SFSU the mechatronics class is comprised of a mix of electrical, computer and mechanical engineering majors. The varying backgrounds require a mix of introductory material to bring the class to a homogeneous knowledge base, and design problems that are relevant to the various disciplines represented. The prerequisites for the class are a course in classical dynamics and a linear systems class that teaches system modeling and signal conditioning, especially in the frequency domain. A feedback control course is not a prerequisite for the mechatronics class although nearly all students who take mechatronics will take control systems before they graduate, and about half the students take it before mechatronics.

The SFSU mechatronics class contains a lab, which gives hands-on experience with sensors, controllers and actuators [3]. The lab is a useful adjunct because it teaches the students practical skills, such as microcontroller applications, for their future employment; it reinforces the subject matter of the lecture; and it shows the students phenomena that are not always included in the simple linear models that are used for the numerical simulation (such as saturation, sensor noise and friction).

Course Material

The mechatronics course material for the one-semester class is broken into three disciplines: sensors, actuators, and controllers. Because of the various majors included in the class, every student will see some familiar material and some new material. The numerical simulation techniques that tie the disciplines together are unfamiliar to most of the students regardless of their major. Figure 1 shows the disciplines and how their models relate to the simulation at the core of the course.

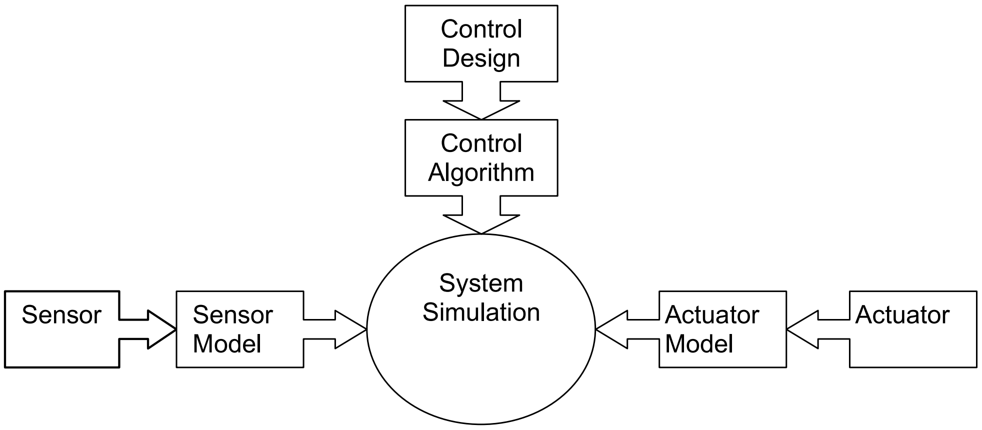


Figure 1: Mechatronics Disciplines

The sensor section of the course begins with an overview of sensors and sensor types. A large number of examples are given, to familiarize students with quantities that can be measured and typical sensor outputs. Sensors that output electrical signals are given the most attention, but converting one type of physical quantity to another that is easy to measure is also considered. Common signal conditioning, such as low-pass or anti-aliasing filters and op-amp circuits are included as well.

After the sensor overview, the sensor models can be considered. The signal conditioning circuits can be expressed as transfer functions, as can the sensors themselves. Alternately, low order transfer functions can be estimated using parameters given in sensor datasheets, such as rise time and bandwidth. Of course the simplest sensor model is the ideal sensor, and this is often a preferred method to begin the simulation. The system simulation (explained in the next section) will be in the time domain, so a review of state-space techniques is useful at this point to convert the transfer functions into time domain differential equations.

The controller section of the course focuses on both control schemes and controller hardware. Because the students are not required to have taken a control systems class, the mechatronics class focuses on simple control schemes that can be explained both physically and numerically, and control schemes that are easy to implement in a time domain model or a hardware control loop.

PID control [4] is a common control technique and can be explained both numerically and physically. Numerical derivation of closed-loop pole locations or second-order system response parameters (such as rise time and damping ratio) give design tools for students to use when creating their control algorithm. Physical explanations, such as using integrators to trim steady-state error and derivative control to increase damping on common systems helps students tune their control laws for satisfactory results.

The numerical simulation uses state-space models [4] to find the time response of the mechatronic system, as described in the next section. For this reason, full state feedback control techniques are also mentioned in the class. Although an advanced topic when observer design is included, Ackermann's formula [4] is a simple pole placement technique for state space models that students can appreciate without a strong control system background, and full-state feedback is easy to implement if the states are available in a real-time controller.

The hardware for implementing control is also considered in the mechatronics class. The class focuses on microcontrollers programmed in C but also includes PLC controllers and ladder logic, as well as PC-based controllers such as dSpace. The controller used does not change the simulation as long as the control law can be implemented in the same manner both in the simulation and the hardware.

As with the sensor portion of the lectures, the actuator lectures begin with an overview of various actuator types, their modes of operation, their strengths and weaknesses, and information needed for selection. Actuator drive circuits, such as the H-Bridge [5], are also described in order to allow the students to interface controllers to their actuators. Mechanical devices such as gears and linkages are also studied in the actuator portion of the course, so that the students can use these to design mechatronic systems.

The actuator models are designed to be simple and created either from measuring overall performance (step response, etc) or from knowledge of the components. Simple DC motor models [4,5] are used as examples of how to find a simulation model for the actuator. Once again an ideal actuator model (where the output exactly follows the command) is a good place to start the simulation, and more accurate models may be inserted later.

Once the students have an overview of the parts of the mechatronics system, they are ready to model and simulate it. The simulation is used as a design tool for the lab as well, where the students must construct the system that they simulate.

Simulation Implementation

Numerical simulation can be accomplished in many ways, with simulation packages or coded directly in a programming language. For versatility and understanding, Euler integration of the equations of motion using matlab or excel is introduced in the class first [6], then simulation packages such as simulink [7] are used.

To numerically simulate the system model using Euler integration, the equations must be in state-variable form. This requires a transformation if a transfer function model was originally formulated. The students' background in linear systems is helpful to ensure that they can accomplish this task. Several examples are solved in the lecture to help the students with the numerical method, and sample code is made available. The Euler integration is shown in both excel and matlab format. Some discussion of the time step size used is presented, although most students will find a suitable time step through trial and error. The Euler integration method starts with equations of motion for the state variables, *x*, that give the derivative of the state variable as a function of the state and the input variable, *u*:

$$\dot{\vec{x}} = f(\vec{x}, \vec{u})$$

In addition to the state and input, the state variable derivative can also be a function of any quantity available to the simulation, such as time, for example.

The Euler integration technique finds the new state of the system from the old state and the time derivative of the old state variables from the equation of motion.

$$\vec{x}_{i+1} = \vec{x}_i + \Delta t \cdot \dot{\vec{x}}_i$$

To implement the simulation in Matlab, the integration is performed in a loop from the known initial conditions to the final time. In Excel, a spreadsheet is created with the initial conditions at the top row, and successive time intervals are calculated below. Excel is useful for visually following the calculation process, but the spreadsheets get very large with small time steps.

Simulink, Matlab's graphical simulation tool, is also used in the class. This has the advantage of hiding the numerical routines and allowing the system to be built up in a graphical manner. Simulink can implement transfer functions (or state variable systems) directly, as well as a number of nonlinear elements that are commonly found in mechatronic systems.

Example

As a very simple example, a wall-following robot simulation will be derived. This wall-follower is similar to the "weasel" [8] toy robot, which uses a switch to sense the proximity of the vehicle to the wall, and two driven wheels to provide steering (using differential wheel speed) and forward motion. The control law is quite simple, and no computer is needed to implement it on the vehicle. To follow walls to the right, the control law ensures that if the switch is closed (close to the wall) the vehicle turns left, and if the switch is open (far from the wall) the vehicle turns right. This control algorithm causes the vehicle to constantly hunt back and forth as it follows the wall.

The most simple sensor and actuator models can be used for this simulation. The sensor switch is either open or closed depending on the position of the vehicle. The wheels either turn a constant speed or are stopped depending on the control input. To turn left, the left drive wheel is stopped, and the right drive wheel turns, to turn right, the right drive wheel is stopped and the left drive wheel turns.

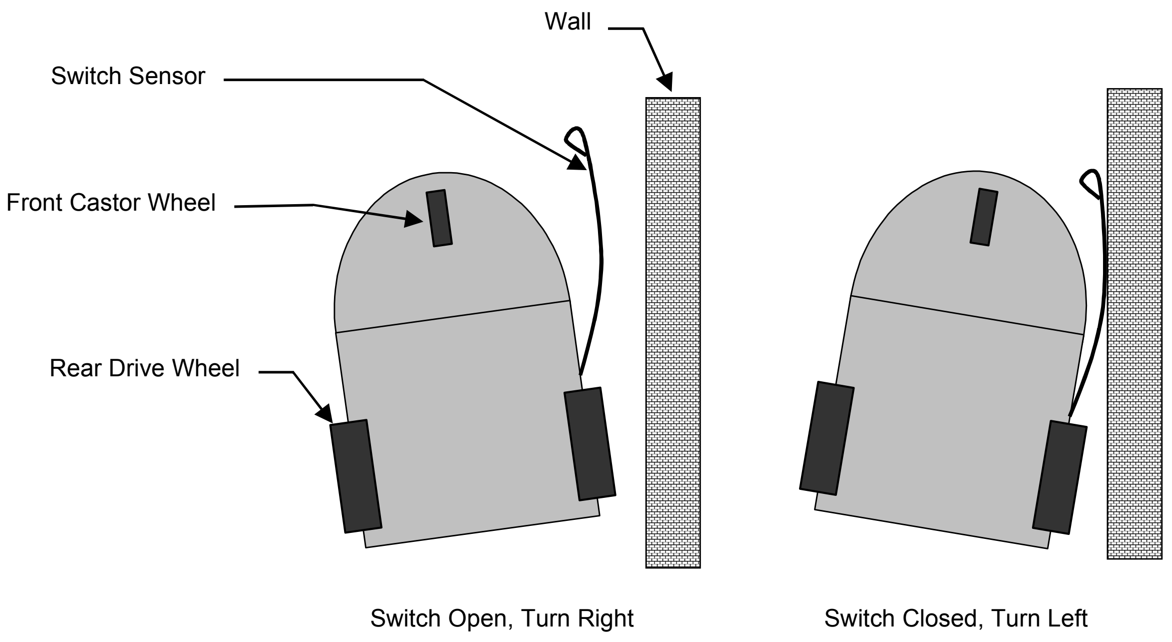


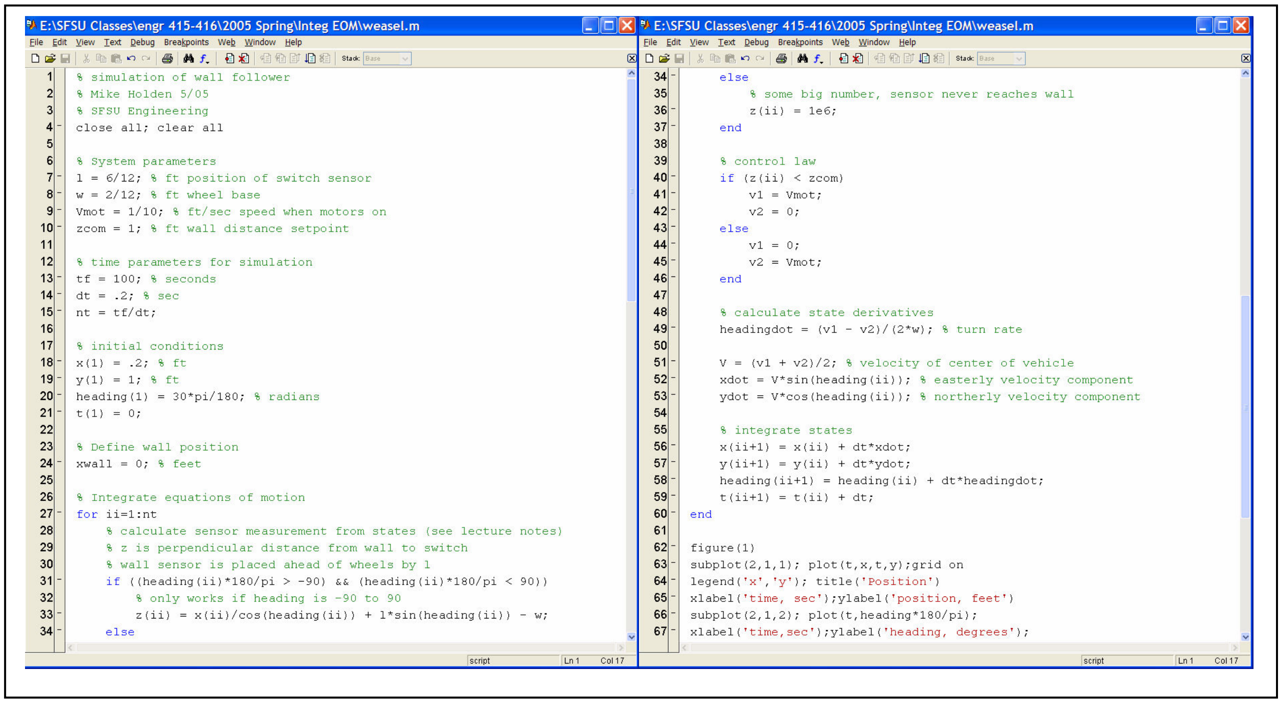
Figure 2: Wall follower control law

The equations of motion for the vehicle can be simplified greatly by using just the kinematic equations, which follows from the assumption of a constant speed from the wheels (instantaneous acceleration). The vehicle state is the position and orientation (*x,y* and *Φ*) of the point on the car between the wheels, and the equations of motion can be calculated from the kinematic equations. If the rotation of the left and right wheels creates velocities at the wheels of V_L and V_R, respectively, the state derivatives can be written as follows:

$$\dot{\Phi} = \frac{(V_L - V_R)}{2w}$$
$$\dot{x} = V \sin(\Phi)$$
$$\dot{y} = V \cos(\Phi)$$

where $V = \frac{(V_L + V_R)}{2}$ is the car's speed, and *w* is the distance between the wheels of the car.

In order to solve the equations of motion, first the switch position must be calculated based on the current state (sensor model). Then the control law is implemented (controller model), which sets the inputs to the drive wheels (actuator model). Once the drive wheel inputs are known, V_L and V_R will be calculated and the simulation can take a step. Figure 3 shows the matlab code for the simulation, while Figures 4 and 5 show the simulation time history and the path of the vehicle, respectively. Note that the vehicle quickly comes to the commanded position 1 foot from the wall, and then remains at that position as it travels along the wall. The oscillations in heading are due to the hunting method of controlling the position, where the vehicle is always turning left or right and never goes straight.


E:\SFSU Classes\engr 415-416\2005 Spring\Integ EOM\weasel.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
script Ln 1 Col 17

```
% simulation of wall follower
% Mike Holden 5/05
% SFSU Engineering
close all; clear all

% System parameters
l = 6/12; % ft position of switch sensor
w = 2/12; % ft wheel base
Vmot = 1/10; % ft/sec speed when motors on
zcom = 1; % ft wall distance setpoint

% time parameters for simulation
tf = 100; % seconds
dt = .2; % sec
nt = tf/dt;

% initial conditions
x(1) = .2; % ft
y(1) = 1; % ft
heading(1) = 30*pi/180; % radians
t(1) = 0;

% Define wall position
xwall = 0; % feet

% Integrate equations of motion
for ii=1:nt
    % calculate sensor measurement from states (see lecture notes)
    % z is perpendicular distance from wall to switch
    % wall sensor is placed ahead of wheels by l
    if ((heading(ii)*180/pi > -90) && (heading(ii)*180/pi < 90))
        % only works if heading is -90 to 90
        z(ii) = x(ii)/cos(heading(ii)) + l*sin(heading(ii)) - w;
    else
        % some big number, sensor never reaches wall
        z(ii) = 1e6;
    end

    % control law
    if (z(ii) < zcom)
        v1 = Vmot;
        v2 = 0;
    else
        v1 = 0;
        v2 = Vmot;
    end

    % calculate state derivatives
    headingdot = (v1 - v2)/(2*w); % turn rate

    V = (v1 + v2)/2; % velocity of center of vehicle
    xdot = V*sin(heading(ii)); % easterly velocity component
    ydot = V*cos(heading(ii)); % northerly velocity component

    % integrate states
    x(ii+1) = x(ii) + dt*xdot;
    y(ii+1) = y(ii) + dt*ydot;
    heading(ii+1) = heading(ii) + dt*headingdot;
    t(ii+1) = t(ii) + dt;
end

figure(1)
subplot(2,1,1); plot(t,x,t,y);grid on
legend('x','y'); title('Position')
xlabel('time, sec');ylabel('position, feet')
subplot(2,1,2); plot(t,heading*180/pi);
xlabel('time,sec');ylabel('heading, degrees');
```

Figure 3: Matlab simulation code

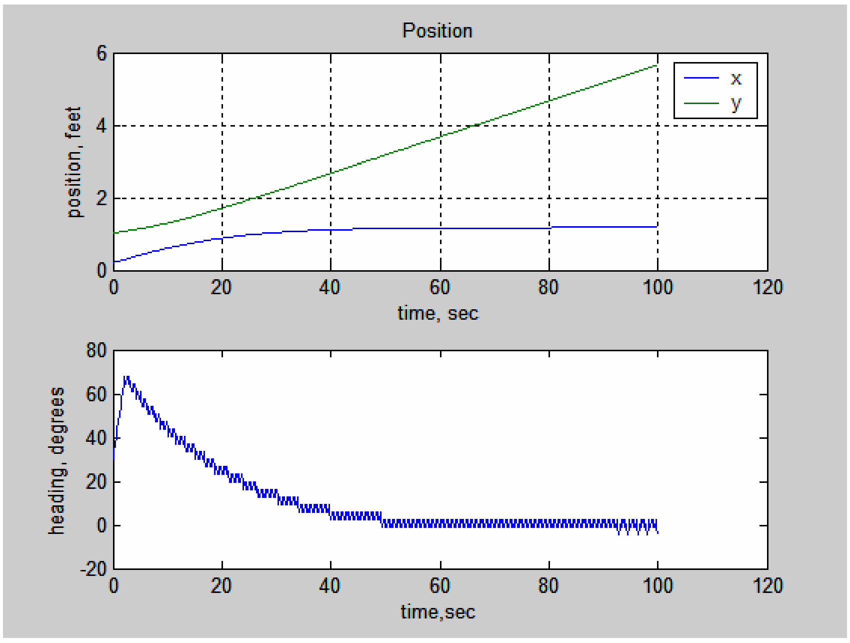


Figure 4: Simulation results

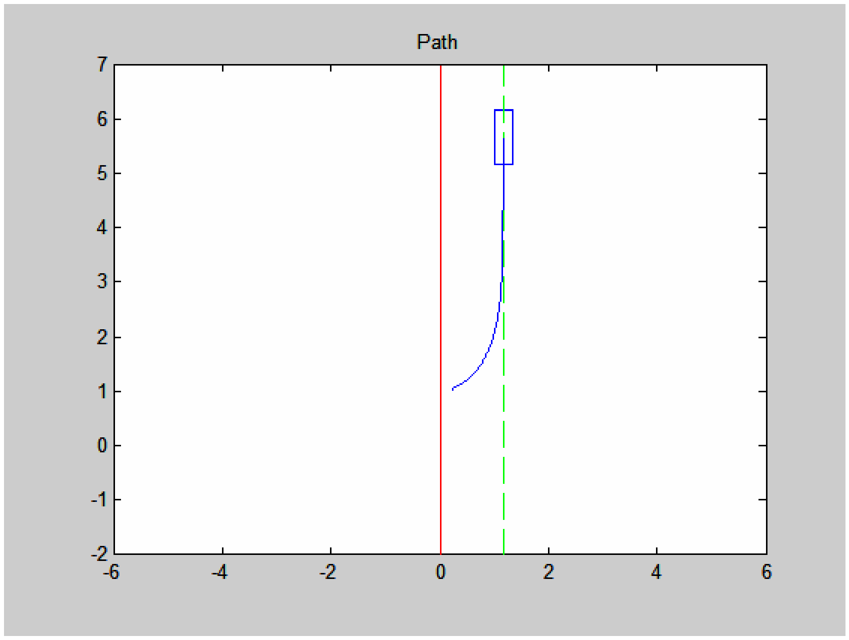


Figure 5: Simulated Path—Wall in red, command in green, path in blue.

The wall follower is an extremely simple model, and it is a good assignment for the students to add more realism, such as a switch with hysteresis, or a drive system with inertia, for example. They may also start with the given simulation and attempt to implement a more refined control law, to see if they can reduce the "hunting" behavior, for example.

The wall follower is also a good simulation example because it is possible to build the wall follower in the mechatronics lab. This combination teaches the value of the simulation as a design tool, useful for verifying control laws or investigating behavior before committing to construction. Figure 6 shows a student-built wall follower that uses an infrared sensor and an Atmel microcontroller for control, and a more sophisticated control law. The students used their simulation as a design tool before starting construction.

Figure 6: Student wall-follower project

Assessment

Students who completed the mechatronics class were asked to report their opinions on the simulation portion of the class. The results are shown in the following figures. The questions focused on the effectiveness of the simulation in helping with the students' mechatronics knowledge in general, and understanding the multidisciplinary nature of the class in particular. The survey was posed as statements that the students could agree or disagree with to varying degrees.

The first statement was "The simulation work helped me understand mechatronic systems better", which the students convincingly agreed with as Figure 7 shows. The students believed the simulation increased their understanding of mechatronic systems.

The second statement was "More class time should have been spent explaining simulation". The responses to this question, shown in Figure 8 were quite varied, almost a gaussian distribution of responses. The class time spent covering simulation topics seems about right, with some students wanting more and some wanting fewer lectures on simulation.

The third statement was "Simulation helped to tie the 3 parts of the mechatronic system (sensors, controllers, and actuators) together." Figure 9 shows the very positive responses, with no disagreement with this statement. The purpose of the simulation was to unite the subsystems of the mechatronic design, and all the students felt that the simulation achieved this goal.

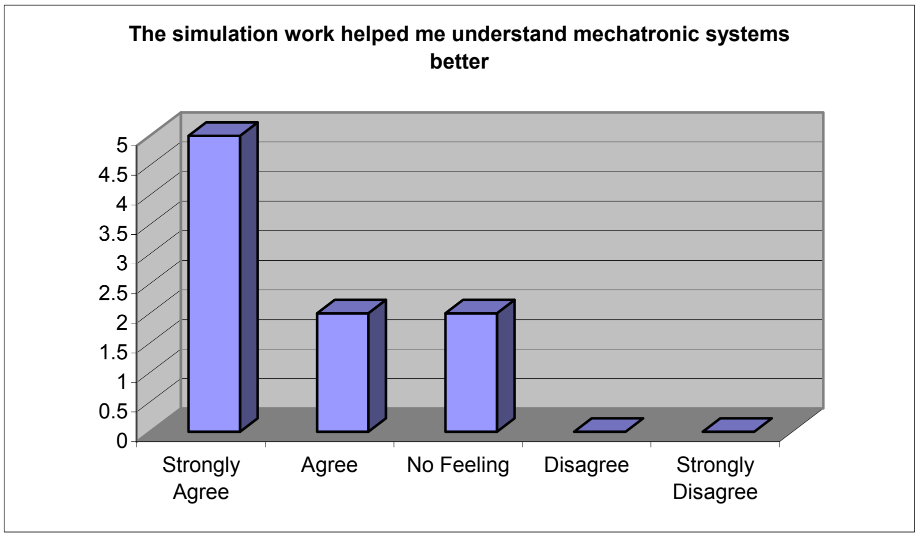


Figure 7: Survey Results—Helped Understanding

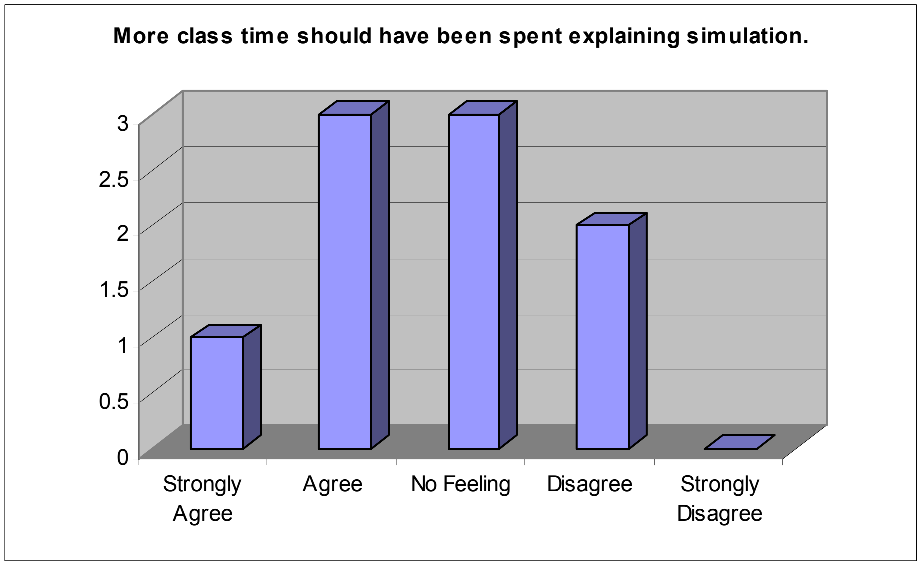


Figure 8: Survey Results—class time

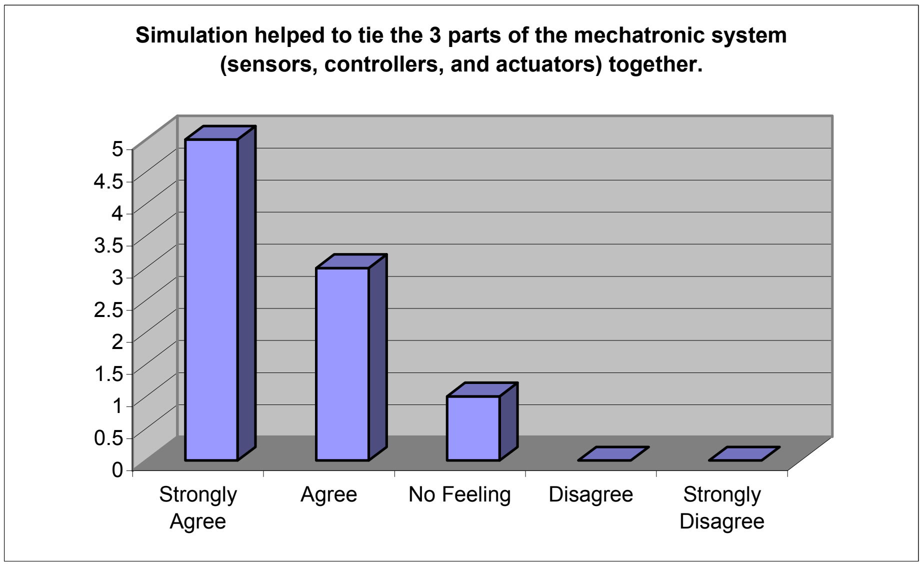


Figure 9: Survey Results—Bringing together

Conclusion

Numerical simulation is a useful design tool that can save time by letting a designer iterate within the design cycle without building the system. In the heavily multidisciplinary mechatronic field, the simulation can be used not only as a design tool, but as a teaching tool to bring the whole system synthesis into focus.

The class described splits the material into the disciplines of sensors, controllers and actuators, and the lectures cover the material in detail. The simulation portion of the lectures shows how to combine the numerical models of each component into a complete time-domain simulation.

State space equations for the system are solved using Euler integration, although software packages such as Simulink have advantages as well. The simulations for the mechatronics class can be as simple as a kinematic description of a mobile robot with ideal sensors and actuators, or as complex as the student desires, with high order component models for the sensors and actuators, and nonlinear control laws.

The simulation portion of the class was well received by the students, who felt that it helped their understanding of the way the disciplines are combined together, and who, the author hopes, will use their simulation skills as they become practicing engineers to create useful and well-designed products.

References

1. Craig, Kevin, "Is Anything Really New in Mechatronics Education", IEEE Robotics and Automation Magazine, June 2001. Online at: http://www.rpi.edu/~craigk/Coursework/RAM_Article.PDF

2. Siegwart, R, "Grasping the Interdisciplinarity of Mechatronics", IEEE Robotics and Automation Magazine, June 2001.

3. Holden, M. "Low Cost Autonomous Vehicles Using Just GPS", *ASEE Computers in Education Journal*, July-September 2005

4. Richard C. Dorf and Robert H. Bishop, Modern Control Systems, 10th edition. Pearson Prentice Hall 2005, ISBN 0-13-145733-0

5. Bolton, W., Mechatronics, 3rd edition. Addison Wesley Longman Publishing, New York, NY, 1999.

6. Erwin Kreyszig, Advanced Engineering Mathematics, 8th Edition, Wiley, 1998

7. Simulink: www.mathworks.com

8. http://www.robotikitsdirect.com/products/owi9910.html