Simulation of Motion:

Stephen P Smith
POB 841
Parksley VA 23421

Part 1: An Improved Lunar Lander Algorithm

About the Author

Stephen P Smith's pet project as an amateur is a PASCAL compiler for a personal computer. Professionally, he leads the Computer Sciences Corporation support team attached to the range safety office at NASA Wallops Flight Center, where he and his team of analysts develop analytical methods and construct digital simulations of flight paths, flow fields and structural responses of rockets and aircraft. The BASIC programs which are part of this article and the remaining parts to come in several installments were developed and run on a Tektronix 4051, which uses a 6800 microprocessor and includes a BASIC interpreter.

One of the most delightful applications for personal computers is games, not just playing them, but creating them. If you are like most enthusiasts, you will have begun with random number games like blackjack, but sooner or later you will want to work with games involving moving objects. To describe that motion using a microcomputer you will need to use a form of simulation. The simulation could involve detailed mathematical models solved with elegant numerical techniques.

More likely, the novice will begin by following the pattern of the simple lunar lander games which have appeared often in BYTE (see "Kim Goes to the Moon," by Butterfield in April 1977 BYTE, or "Controlling Small DC Motors with Analog Signals" by Dwyer, Critchfield and Sweer in September 1977 BYTE). The truly advanced simulations are best left to professionals with mainframe computer power, but the home user can progress well beyond the simple lunar lander game. By picking up the basic physics and simple numerical methods presented in this article and the following ones, you will learn to simulate a wide variety of motion. Whether you use these simulations to create games, like the real time LEM simulator presented here, or to develop new applications for your personal computer system, you will acquire some valuable additions to your applications software toolbox.

For any application involving motion, your simulation will be required to predict the speed and position of an object at some time in the future. The predictions can be made using a microcomputer if you first limit the type of motions considered at any point in the program. In the lunar lander game, for example, the excursion module (LEM) is only allowed to move up and down. The simulation is said to have one degree of freedom. Other degrees are possible, but the separation into different degrees of freedom is an important first step.

Let's see how a one degree of freedom simulation is performed. Thanks to Sir Isaac Newton and his apple (that was a fruit, not a computer), we know that an object will continue to move in any degree of freedom without changing speed until a force acts on it. To predict how the LEM will move, we need only to examine the forces which might be present and determine how they effect the up and down motion.

Because the moon has no atmosphere to involve us in aerodynamics, only two forces need be considered, gravity and thrust. Gravity makes the LEM fall faster. Thrust

Generic Unit	Metric		
Length	1 meter	=	3.2808 feet
Velocity	1 meter per second	=	3.2808 feet per second
		=	2.2369 miles per hour
Acceleration	1 meter per second per second	=	3.2808 feet per scond per second
Mass	1 kilogram	=	2.2046 pounds (mass)
		=	0.0685 slugs (mass)
Force	1 newton	=	0.2248 pounds (force)

Table 1: This article was written using the metric system of units. As the front runners in an exciting new technical hobby, we should be more ready than most to accept the coming metric conversion in this country, but if you haven't been converted yet, the above table will be useful.

Vertical Motion

$F = ma$

$F = Kg(m/s^2) = N$

F = engine thrust (N)
m = mass of lander, adjusting for fuel consumption

$a = \left(\frac{F}{m} - \underbrace{g_s}_{\text{Surface Gravitational constant}}\right)\underbrace{t}_{\text{Time (sec)}}$ - Sign indicates direction (- = down)

Altitude = Previous alt + $\dot{a}s$
where s = stepsize or screen update period.

$a = \left(\frac{10{,}000 N}{1000 Kg}\right) - 1.62 m/s^2 = 8.38 m/s$

Test - speed -100m/s, speed 0.1s in future is: @ 10,000m
✓ $-100 m/s + (.1s \cdot 8.38 m/s) = -99.162 m/s$

Altitude(m) = $10{,}000 m + (.1s \cdot -99.162 m/s) =$
$= 9990.08 M$

Thrust 0-100% of rated engine

makes it fall more slowly. The exact effect of each can be calculated with only a few operations.

Gravity is the simpler of the two. It has exactly the same effect on every object. During each second of a lunar landing near the moon's surface, the moon's gravity will make a LEM fall 1.62 meters per second faster. (Those of you who wish to land on more exotic heavenly bodies are referred to table 2.) In most simulations, speed and position are considered positive if they are directed upward, in this case away from the lunar surface. To simulate 1 second of fall through lunar gravity we must subtract 1.62 meters per second from the present speed. If the LEM is moving at -100 meters per second now (100 m/sec downward), 1 second later it will be moving at -101.62 meters per second.

In many games, the effect of thrust is also simulated by a constant change in speed. Often it is given in multiples of gravity called "g"s. One "g" of thrust adds 1.62 meters per second to the speed, just as gravity subtracts that amount. Two "g"s add twice that, and so on. This assumption reduces the complexity of the simulation, but it fails to demonstrate the way in which forces actually cause changes in speed.

Unlike gravity, forces such as thrust do not have the same effect on every object. They have a larger effect on light objects than they have on heavier ones. It is important to consider this fact in accurate simulations, because weights can change. The LEM becomes lighter as it burns fuel to create thrust. A given value of thrust will have a larger effect toward the end of the flight than it will at the beginning.

Weight is not really the correct term to use when calculating that effect. We should talk instead of mass. The difference is subtle, but important. Mass is a basic property of matter. Weight is the result of gravity pulling on the mass. A man on the moon weighs only 1/5 as much as he does on earth, but his mass is the same. This is true because the moon's gravity pulls only 1/5 as strongly on his mass. The effect of a force is determined by the mass of an object, not by its weight. A given thrust will have the same effect on a LEM whether the LEM is landing on the moon, on earth, or is floating "weightless" in space.

In the metric system, the unit of mass is the kilogram. The unit of force is the newton. These units are very convenient for calculating the effect of a force on the motion of an object. The force (in newtons) divided by the mass (in kilograms) is exactly equal to the rate of change in speed ("acceleration" in meters per second per second). No additional constants are needed as they are when units of feet and pounds are used. For example, let our LEM have a mass of 1000 kg and let its engine produce a thrust of 10,000 newtons. To simulate 1 second of thrust, a program would add 10 meters per second to the speed (10000/1000) to account for 1 second's worth of acceleration.

Remember though that during the same second 1.62 meters per second must be subtracted to simulate the effect of gravitational acceleration. The actual change in speed will be 10.-1.62=8.38 meters per second. In two seconds, the change will be twice that or 16.76 meters per second. In half a second, the change will be one half as much and so on. While this may seem obvious, it illustrates an important point. The change that each force makes in the speed in 1 second may be determined separately. The separate effects are added up and then multiplied by the length of time we are simulating to find the actual value the simulation program will add to the speed.

Now that we can predict speed, let's apply the same technique to predict the

Heavenly Body	Surface Gravity (m/sec^2)	Heavenly Body	Surface Gravity (m/sec^2)
Moon	1.62	Asteroids	
Earth	9.80	Ceres	0.85
Mercury	3.95	Pallas	0.54
Venus	8.72	Juno	0.21
Mars	3.84	Vesta	0.43
Jupiter	23.16	Jupiter's moons	
Saturn	8.77	Ganymede	3.43
Uranus	9.46	Io	2.26
Neptune	13.66	Europa	1.98
Pluto	4.89	Callisto	3.20

Note that the gravitational accelerations shown in this table are surface accelerations, valid during the final stages of a landing when a spacecraft is relatively near the heavenly body. A more complicated simulation is required if movement far away from the heavenly body is contemplated.

Table 2: Players who grow adept at lunar landings may wish to try landing on some other heavenly bodies. The above table of accelerations due to gravity is provided for them.

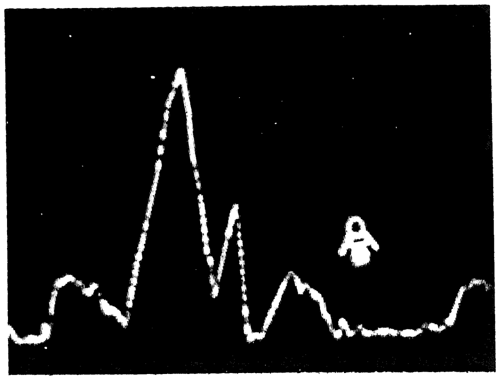

Photo 1: A scene from the "lunar lander" program which is the Digital Equipment Corportation's graphics equipment demonstration program. This simulation is a real time model of a lunar landing in which a light pen is used to input control information and displays track the landing. The object of the game is to land near (but not on) the only MacDonalds' hamburger stand on the moon. This simulation, like the one discussed in the article, has two degrees of freedom; superficially it differs from the program of this article largely in its incorporation of real time graphic display light pen control inputs and a model of the lunar terrain.

Horizontal Motion

Thrust + = left
− = right

Same Data — no horizontal thrust
t_0 : $a_v = -100$ t_1 : $a_v = -99.16$
alt = 10,000 alt = 9990.08
Δt = 0.1sec

Downrange distance to target (left of target)
S = 100m @ −10m/s
Δt = 0.1s
N_t = 5000N thrust
$a_h = -10 + (0.1 \cdot \overset{\nearrow 5000/1000}{5}) = -9.5$ m/s
$Alt_h = 100 + (0.1 \cdot -9.5$ m/s$) = 99.05$m

Thrust −100 − 100% of rated engine thrust

position. We have shown that if the LEM is moving downward at 100 meters per second now, (speed=−100) then in 2 seconds the speed will be −100.+2.x(THRUST/MASS −1.62). Similarly, if the LEM is 10000 meters above the moon now, in 2 seconds it will be 10000.+2.x(speed) meters up. Just as we multiply the forces by time and add the product to the speed, we multiply the speed by time, and add the product to the position.

What we have just done is to predict the speed and position at a "step" of 2 seconds into the future. In the jargon of simulation, 2 seconds is the step size. The step size can take any value you choose. Returning to the 1000 kg LEM, let the step size be 0.1 seconds. For a present speed of −100 meters per second, the speed predicted for 0.1 seconds in the future is −100.+0.1x(10000./1000.−1.62)=−99.16 meters per second. If the position now is 10000 meters, then the position predicted for 0.1 seconds in the future is 10000.+0.1x(−99.16)=9990.08 meters above the moon.

Using these values of speed and position we can find new values for the forces and mass. We can then step the simulation into the future once again. The process can continue indefinitely, but usually one or more variables is tested for an end condition at each step. The test might be on position (Are you still above the moon?), on mass (Is there fuel remaining?), or on some other variable. Should any of the tests fail, the program will branch and end the simulation.

Adding a New Degree of Freedom

You now know the basic procedure for simulating motion in one degree of freedom. The LEM simulation has been in one degree because we have only predicted the up and down movements. These are called vertical motions. Suppose that we also predict the way the LEM moves horizontally, in other words, from side to side. The pilot must not only reach the surface of the moon successfully, but also land close to his target. While the pilot's task has become more complicated, our simulation fortunately has not. Just as we are able to calculate the effects of each force separately, we are able to make calculations for speed and position separately in each degree of freedom.

To make those calculations for the second degree of freedom, first determine what forces are acting. Gravity, by definition, acts only up and down. It does not enter into the horizontal calculations. So far, thrust has also been limited to vertical action, but we can easily add a second thrust acting to the side. Positive horizontal thrust should cause the LEM to move left, while negative thrust moves it right.

Since there are no other forces to consider, the change in horizontal velocity (in meters per second) will be exactly equal to the horizontal thrust (in newtons) divided by the mass (in kilograms). This is, of course, the same equation used in the first or vertical degree of freedom. Similarly, the same equations used to calculate vertical speed and position will be used to calculate horizontal speed and position.

→ Return to the example used earlier, but also consider the horizontal motion. Let the LEM start 100 meters to the left of its target moving at 10 meters per second to the right. Generally motion to the left will be considered positive and to the right negative, so the horizontal speed is −10 meters per second. We found that during a step of 0.1 seconds the vertical speed changed from −100 to −99.16, and the position changed from 10000 to 9990.08. Quite apart from those calculations, we may set a horizontal thrust, say 5000 newtons, and find that during the same step the horizontal speed will become −10+0.1x (5000/1000) or −9.5 meters per second. The horizontal position will become 100.+0.1x(−9.5)=99.05 meters. After making these calculations, the simulation

Continued on page 216

Continued from page 22

Listing 1: Most of the lunar landing games I have seen are not flexible enough to run a two degree of freedom real time simulation as described in this article, so I have included this listing. At each initialization, this program finds a random set of starting conditions (speed, position, mass, etc) that is consistent with a safe landing. It then keeps track of speed, position and fuel consumption, printing them as required, and indicating when the surface has been reached. The following adjustments will have to be made by each user:

1. *Function USR(X) must be provided to return the current desired thrust settings, 0 to 100% in the vertical direction, and −100 to +100% in the horizontal direction. These inputs are best achieved by analog to digital conversion from joysticks or slide pots.*
2. *The step size and print interval must be adjusted for your system clock and peripheral speed in order to simulate real time operation accurately.*
3. *Function RND(1.) is assumed by the program to return values between 0. and 1. Alterations may be necessary to suit your version of BASIC.*
4. *The comments printed by the program have deliberately been kept short. A better game could be fashioned by adding instructions, comments on performance, low fuel warning, etc. In other words, customize the simulation to suit your own tastes.*

```
010  REM LUNAR LANDING SIMULATION
020  REM SET FUEL SAFETY FACTOR
025  REM ADJUST TO CONTROL DIFFICULTY
030  LET S=1.3
040  REM SET STEP SIZE AND PRINT INTERVAL
050  LET D=0.01
060  LET K=1.0
070  REM SET GRAVITY ACCELLERATION
080  LET G=1.62
090  RANDOMIZE
100  REM SET NEW STARTING CONDITIONS
110  LET M=1024.+1024.*RND(1.)
115  PRINT "LEM MASS =",M
120  LET F=G*M*(4.+4.*RND(1.))
130  PRINT "MAX THRUST =",F
140  LET A=1.333*F/M-G
150  LET V=F/M*64.*RND(1.)
160  LET U=0.
170  LET Y=V**2./(2.*A)*(1.+RND(1.))
180  LET X=V
182  REM V IS VERTICAL SPEED
184  REM U IS HORIZONTAL SPEED
186  REM Y IS VERTICAL POSITION
188  REM X IS HORIZONTAL POSITION
190  REM HALF OF MASS IF FUEL
192  REM M-P IS FUEL REMAINING
200  LET P=M/2.
210  REM FIND FUEL BURN RATE, I
220  LET I=(2.*Y+V**2./G)/(1.+A/G)
230  LET I=P/(SQR(I/A)*F*S)
240  PRINT "ALTITUDE, SPEED, FUEL, RANGE"
250  REM BEGIN DECENT CALCULATIONS
260  PRINT Y, V, M-P, X
270  LET T=0.
280  IF M=P THEN 360
285  REM GET VERTICAL THRUST
290  LET A=USR(1.)*F/100.
300  REM GET HORIZONTAL THRUST
305  LET B=USR(2.)*F/100.
310  LET M=M-(A+B)*I*D
320  IF M>P THEN 360
330  PRINT "FUEL EXHAUSTED"
340  LET A=0.
342  LET B=0.
350  LET M=P
358  REM PREDICT NEW U, V, X, Y
360  LET V=V+D*(G-A/M)
370  LET U=U+D*B/M
380  LET Y=Y-V*D
390  LET X=X-U*D
395  REM TEST FOR END CONDITIONS
400  IF Y<4. THEN 440
410  T=T+D
420  IF T>K THEN 260
430  GOTO 290
440  PRINT "MODULE HAS LANDED"
450  PRINT "SPEED =",V
460  PRINT "RANGE =",X
470  IF V<8. THEN 500
480  PRINT "BETTER LUCK NEXT TIME"
490  GOTO 110
500  IF X<128. THEN 530
510  PRINT "ITS A LONG WALK TO BASE"
520  GOTO 110
530  PRINT "CONGRATULATIONS, GOOD LANDING"
540  GOTO 110
550  END
```

program should check for end conditions and, if none are found, begin another step.

The speed of a computer makes it possible to find results quickly for times far into the future, even if the step size is quite small. This is fortunate, because as our simulation stands now, an error is introduced at each step that becomes worse as the step size becomes larger. The error occurs because in a real LEM the mass and speed are changing all the time, but in our simulation they can be changed only between steps. A variety of numerical methods have been developed to cope with this problem. In our example, simply using the average of the beginning and ending values in each step would be quite effective. Actually, if the step size is small, say 0.01 seconds, even this is not necessary. It is true that by using the average values the program could be made to run faster, but it would also need to store several extra variables, require more lines of code, and use more memory. Obviously, there are trade-offs to be made among speed, accuracy, complexit and size. In each simulation, the programme must decide which combination is best.

For our games application, the combina tion is not critical. A high degree of accurac is not required, and the program is shor enough that memory requirements shoul not be a problem. The selection of speed however, presents an opportunity tha is unique to the user of a dedicated system With a little trial and error, it should b possible to find the step size which cause your system to take exactly 1 second t calculate and display the speed and positio

Circle 131 on inquiry card.

BUSINESS APPLICATION SOFTWARE

MINI WORD PROCESSING (MWP) enables the user to prepare letters, text and mailing labels or envelopes. When used for correspondence processing, MWP allows each entry in the name/address file to be described by a number of group codes and document codes. For example, an entry could be group coded by date and inquiry type. Phrases, paragraphs or pages can be specified as document codes to produce an individualized letter for each name/address. MWP provides in-line editing and page or phrase insertions during text generation. The letter and text output modules provide text insert or replacement, margin control and page numbering.

DISK SORT supports fixed or variable length sequential files of any size and will sort or merge on any number of keys anywhere in the record. An interactive generator allows the user to define a customized sort/merge program for each task. Multiple sort/merge tasks can run unattended with user defined job-stream links (eg. sort 12 files, merge them with another sorted file and link to your report program). Memory and disk space are managed by the system to minimize processing time.

UNIVERSAL DATA ENTRY (UDE) system interacts with the operator to generate custom key-to-disk modules. User defined displays provide "fill in the blanks" simplicity. Validation procedures such as check digits, value tables, range tests, batch totals and record counts improve data quality. Selectable field duplicate or increment eliminate repetitive entries. UDE supports fixed or variable length disk files. Specialized UDE modules can be generated for any application that requires keyed input.

The above systems are extremely easy to use and include carefully created prompts and error recovery, well written user manuals with varied examples and extensively documented programs with detailed remarks. Each of these systems is priced at $195. The programs are supplied on diskette and run under MITS Disk Extended BASIC. See your computer dealer or contact us.

THE SOFTWARE STORE
706 Chippewa Square
Marquette, MI 49855

Master Charge • 906/228-7622 • VISA

1 second into the future. One machine might do 100 steps of 0.01 seconds and then print the speed, etc. Another might do 64 steps of 0.015625 seconds before displaying new results. Still another, with slow peripherals, might output speed and position only once every 2 or 3 seconds. In any case, as long as the simulated data appears at the same time that real data would, your system will be said to be running a real time simulation. A real time lunar lander game gives you exactly the same time to react as would be given a real excursion module pilot.

To help you implement this idea on your own system, a BASIC language program has been included with this article as listing 1. It should be easy to follow, but a few points are worth explaining. At each step the program will need to obtain the thrust settings. This is done through a function called USR. Because systems differ widely, the content of USR is left to you. Some systems will be able to use a register to hold the thrust; others will access memory location; and some may have to query an input port. Also left to the user is the manner in which the thrust settings are updated. Obviously, they cannot be entered at the keyboard for each 0.01 second step. The keyboard could be used via an interrupt routine however. Ideally, you could implement Thomas Buschbach's joystick interface (March 1977 BYTE, page 88) to allow continuous control of thrust in both degrees of freedom. What began as a simple game will now have become a real time lunar landing simulator requiring quick thinking and a good bit of practice to master.

If you use this idea then my article will have succeeded in its purpose of introducing some of the basic concepts of simulation. Techniques like separating the problem into degrees of freedom, determining the effect of each force separately, and stepping the simulation into the future are all fundamental to any prediction of motion. The differences between this lunar lander game and the complex simulations used in the space program lie in the way forces are determined and in the numerical methods used to calculate speed and position. In future articles, other applications for simulation on microcomputers will be discussed as a means for demonstrating some of those advanced techniques. For now, try applying the ideas presented here to create a game of your own.■

6800 REFERENCE GUIDE

FOR PROFESSIONAL AND HOBBYIST

The **TOOL** you have been waiting for. A 6800 Reference Data Guide (similar to the IBM green card) containing just about every piece of pertinent information that you are currently spending valuable time searching for. The guide saves **TIME, EFFORT** and **WORK SPACE** when programming, debugging and utilizing the 6800 system.

Included are **INSTRUCTIONS, SPECIAL OPERATIONS, HEXADECIMAL/DECIMAL CONVERSION TABLES**, code translation tables for **MACHINE CODES/MNEMONICS/ASCII/HOLLERITH/BINARY, 6800 MPU, 6820 PIA, 6850 ACIA** and almost everything else that will increase your productivity.

TRY IT, YOU'LL LOVE IT.

Send $4.95 plus 35¢ shipping and handling to:

MICRO AIDS, P.O. Box 1672, St. Louis, Missouri 63011

Circle 84 on inquiry card.

Part 2:

Simulation of Motion An Automobile Suspension

Stephen P Smith
POB 841
Parksley VA 23421

Have you ever taken your system out to a club meeting or demonstration, only to find that something is ruining your car's handling? Was it because of the heavy power supply in the back seat? Would heavy duty shock absorbers help? You can answer these questions using your personal computer and the simulation techniques found here.

Last month *[page 18]*, I introduced some basic ideas used in simulating motion. A games application was used as an example. This month I'll expand on that base, explain some additional ways that forces can act, and demonstrate a more accurate technique for computing speeds and positions. The example I'll use will be a simulation of an automobile suspension and its response to a varying road surface. Automobile enthusiasts will be able to see how different springs and shock absorbers would affect the way a car rides. More important, all computer users will acquire some additional tools to use in their own simulations and gain insights into new applications for their personal systems.

First, let's review the basic points made in the last article. When beginning a simulation, you will first divide the motion being simulated into degrees of freedom. In other words, you will decide which motions you want to simulate, up and down, side to side, etc. From then on, calculations will be made separately for each degree of freedom. Next you will decide which forces are acting in each direction and determine how much each force would change the speed of some object in 1 second. If you use the metric system of units, the change, or acceleration (in meters per second per second), will be exactly equal to the force (in newtons) divided by the mass of the object (in kilograms). You will now be ready to predict the speed and position of the object at a step of D seconds into the future. Add up the effects of the individual forces. Multiply the total by D (the step size) and add the product to the present speed. This is the speed of the object at a time D seconds into the future. Now multiply the speed by D and add that product to the present position. This is the position the object will take in D seconds. The simulation program will now calculate new values for the forces and mass and step the simulation forward once more. The process will continue until an end condition is reached.

In the lunar lander game simulation, two degrees of freedom were considered, up and down, and side to side. The up and down or vertical motion was affected by gravity and thrust. The side to side or horizontal motion was affected only by thrust. Both of these forces were determined independent of the speed and position of the lander. Gravity provided a constant change in speed, and thrust was controlled by the user. In this article we will explore variable forces which are not determined by the user, but directly by the speed and position we are simulating.

As mentioned earlier, the example we'll use is an automobile suspension, the parts which connect the wheel to the body. The most important of these parts are the spring and the shock absorber. We will assume that there are other parts which keep the wheel from moving back and forth, but only the wheel's up and down motion will be considered (see figure 1). Of course, the entire car can also move along the road. We will consider that as a second degree of freedom. Let's examine separately the forces that contribute to vertical and horizontal motion.

Motion down the road results when the car's motor, through the wheels, pushes the car forward. Air resistance and rolling friction try to slow it down. To simplify the simulation, we will assume that these forces balance each other exactly. This means that the speed along the road will not change. If the speed starts at some value other than zero, the horizontal position will change. As we will see later, the

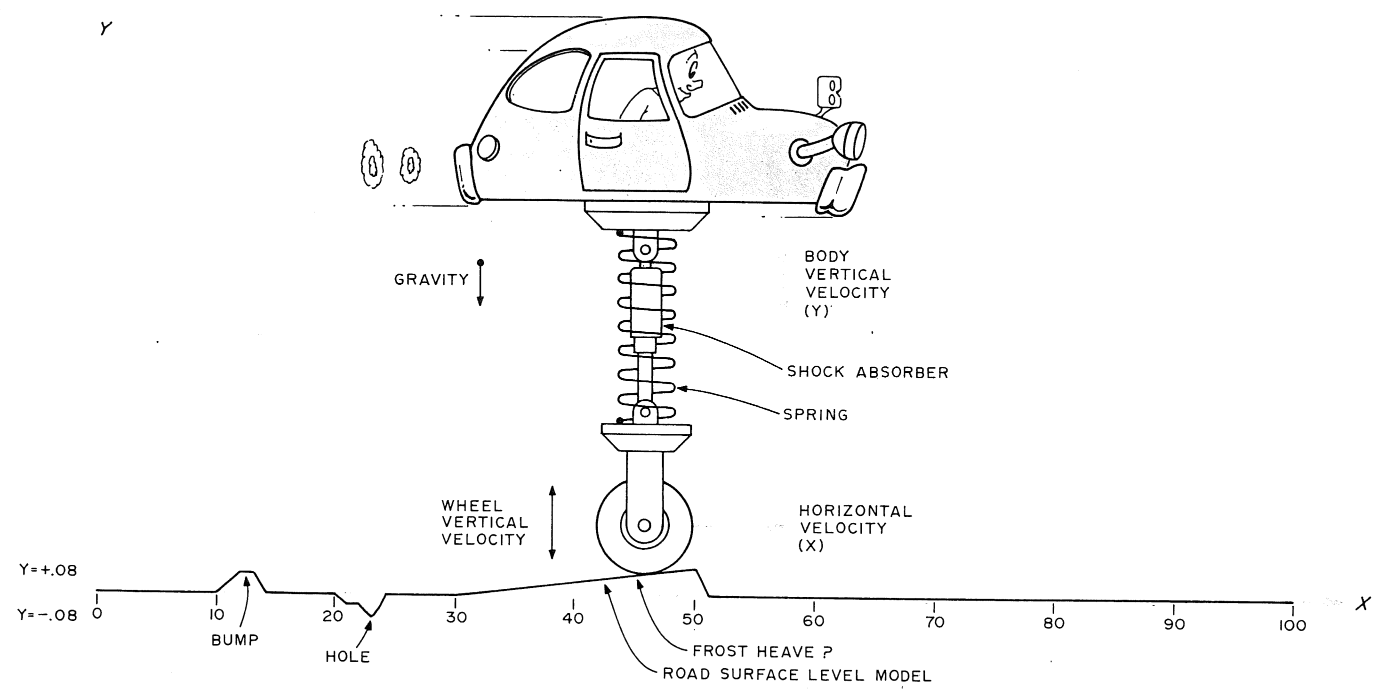

Figure 1: A conceptual model of the "automobile" (unicycle, rather) which is modelled in the sample program of listing 1 as discussed in this article. The wheel in this model tracks the road surface exactly, and has its own vertical velocity due to the horizontal velocity interacting with bumps in the road. The actions of the wheel in turn couple through the spring and shock absorber suspension to the "body" of the automobile. The purpose of this simple model is to calculate the vertical position of the car body at any given point down the road, given the effects of gravity, shock absorber, spring and excitement provided by the bumps and holes. The table in the figure is taken from lines 605 and 606 of the BASIC program of listing 1, and is used to plot the road surface. A better dynamic model of a car would have many more degrees of freedom than this simple model.

simulation program must keep track of the position along the road, because it will determine how the wheel, and in turn the body, moves up and down.

In the vertical degree of freedom we will need to consider gravity. You will remember from the last article that to simulate gravity a program subtracts a constant value from the speed for each unit step. (Speed and position are considered positive if they are directed upward.) On the earth the gravitational acceleration constant is 9.8 meters per second per second, so for each second of simulated time, velocity changes by 9.8 meters per second. Since the car obviously does not continue to move downward, there must be other forces balancing gravity. These are produced by the spring and shock absorber, and are determined by the vertical speed and position of the body.

Let's examine the spring first. At its normal length (often called the free length) a spring produces no force at all. If it is compressed, in other words forced to become shorter, it will push back on whatever is compressing it. The shorter the spring is forced to become, the harder it will push back. This is an example of a force that depends upon position. In the automobile example, as gravity pulls the body down, the spring is compressed. The spring begins to push upward on the body, and at some point the two forces balance each other. The body will eventually come to rest there.

Knowing a little information about the spring we can compute that point. Springs produce forces which are equal to the distance they are compressed times a constant. The metric units for the constant are newtons per meter. Sample values are shown in table 1, column a. Suppose that gravity exerts a force of 5000 newtons on the car body; then a spring with a constant of 100000 newtons per meter would have to be compressed .05 meters (100000.x.05=5000.) to balance the pull of gravity. At this point the system would be in equilibrium.

What about the shock absorber? It was designed to produce a force that depends not on how far it is compressed, but on how fast it is being compressed. The faster you try to move it, the harder it resists being moved. Like the spring, a constant is used to calculate the force, this time multiplying the speed. The metric units for this constant are newtons per meter per second and some representative values are shown in table 1, column b. At equilibrium there is no motion, so the shock absorber produces no force. If you were to push down the car

Vehicle	(a) Spring	(b) Damping
Full size car (LTD, etc)	3200	1450
Intermediate (Torino, Cutlass, etc)	3000	1200
Compact (Nova, Aspen, etc)	2800	1000
Subcompact (Vega, Pinto, etc)	2600	700
Add 20% for heavy duty suspension.		
Subtract 20% for front wheel.		

Table 1: Representative spring and damping constants for automobiles. The units are metric: the spring constant is quoted as newtons per meter of compression; the dampling constant is expressed as newtons per meter per second.

body at a speed of 2 meters per second, a shock with a constant of 50 would resist that motion with a force of 100 newtons (50 x 2). When you let up on the body, the spring would exert a greater force than gravity and the body would move upward. The shock absorber would also resist that motion. This action is called damping. The damping in an automobile suspension must be carefully chosen so that the body returns quickly to equilibrium, but does not continue to bounce back and forth for very long afterward.

Armed with your present knowledge of simulation you should be ready to make just such a choice using a trial and error approach. Calculate the forces on the body, and then use them to find the speed and position one step into the future. That speed and position will be used to calculate new values for the forces, which in turn will be used to step the simulation forward once more. Repeating the process continuously, you will simulate the motion of the car body. Try different values for the spring and damping constants until the desired output is achieved for a given set of inputs.

The inputs, you'll remember, are going to be determined by the simulated position in the horizontal degree of freedom. At each position along the road the input routine will determine the height of the road surface above or below normal. If we assume that the wheel does not leave the road this will also give us the up and down motion of the wheel. The data can be stored in a table in memory. By entering different values for the horizontal speed at the start of the simulation, we can also vary how fast the car will pass over our model road. At each step the program will enter the table to find the road height which corresponds to the current horizontal position.

This method will work as long as there is an entry in the table for every horizontal position we will find. That could be a very big table, especially if the step size is small. To eliminate the need for large tables, we can use a technique called interpolation. Very simply, interpolation is done like this. When the program enters the table, but doesn't find an entry exactly equal to the current horizontal, it finds the next smaller entry and the next larger entry. An interpolation formula is then used to figure out where the present position falls between the two table entries, and to calculate the road surface which lies at the same point between the corresponding table entries of road height. For example, suppose a program entered table 2 to find the road surface corresponding to a horizontal position of 11. It would find entries at 10 and 12 with corresponding road heights of 0.0 and 0.08. Because 11 lies halfway between 10 and 12, the interpolation formula will find a corresponding road surface that lies half way between 0.0 and 0.08 or 0.04. There are other interpolation formulas that use three, four, or more of the table points, but this method using two is generally accurate enough with a reasonably detailed table. To simplify your implementation of interpolation, I have included a BASIC function in the program of listing 1 which uses the 2 point method. Users can simply place their own tables in the data statement and use the function in their programs, or they can follow through the equations and implement them directly.

In our automotive simulation, the interpolated table data will give us the vertical road and wheel position. The difference between this and the vertical position of the body will be the amount the spring is compressed. We can quickly calculate the resulting force. If the simulation program retains the wheel's position from the previous step, it can also calculate the wheel's vertical speed. Reversing the equation used to find a new position, the speed is equal to the difference in the two positions divided by the step size. If the wheel moved from .08 meters to .04 meters in a step of 0.01 seconds, its speed would be (.04−.08)/.01 = −4.0 meters per second. The difference between this speed and the speed of the body is used to calculate the force produced by the shock absorber. All these calculations are included in the BASIC program of listing 1. Readers who want more detail on the equations will find them there as well-commented program statements.

Also in that program is a new method for computing speeds and positions. The equations used in the lunar landing game worked fairly well when the forces did not depend upon the speed and position. In this simulation they do, and even small errors can snowball if not corrected. To do this, we will use a powerful numerical technique, one which uses the results from three previous steps to help predict the next, and

Table 2: A sample road surface table. This table is used to draw the surface curve shown in figure 1.

Horizontal Position	Road Surface
0	0.0
10	0.0
12	0.08
13	0.08
14	0.0
20	0.0
21	−.04
22	−.04
23	−.08
24	0.0
30	0.0
50	0.1
51	0.0
100	0.0

which then goes back and corrects the step when the predicted results are available. It is called, logically enough, a predictor-corrector method. Rather than attempt to explain it here, I'll provide a BASIC programming example which you can adapt to your own simulations. Readers with a good background in math may wish to reference a book on numerical methods for more details. In either case you will have acquired a tool which will be very useful in future simulations.

Looking back over the two articles you should begin to see some ideas for your own simulations. They could involve forces which are constant, user controlled, or which depend directly on the motion you are simulating. Inputs can come from your keyboard, from an analog device such as a joystick, or from tables interpolated by your program. The outputs might tell you how well you are playing a game, or which of several configurations is best for a design you are contemplating. In the next article I'll continue to expand on the types of forces considered. In particular, I'll show how you can handle forces which act in more than one degree of freedom and suggest some ways to handle rotary motion.■

Automobile Suspension Simulator

Listing 1: This program was written to help interested readers follow the mathematics of the accompanying article. Particular attention should be paid to the interpolation subroutine and to the equations for the predictor-corrector method of predicting future positions and velocities. The program was not intended to be efficient; readers will surely be able to shorten it once the method is understood. The following table defines the variable names I've used.

K1	*spring constant*
K2	*damping constant*
M	*mass supported by the spring*
V	*horizontal speed of the entire car*
D	*time step size*
T	*elapsed time in the simulation*
P, S, A	*predicted values for vertical position, speed, and total effect of forces*
P1, S1, A1	*present values of vertical position, speed, and total effect of forces*
S2, A2	*speed and effect of forces one step past*
S3, A3	*speed and effect of forces two steps past*
S4, A4	*speed and effect of forces three steps past*
F1	*change in speed due to spring*
F2	*change in speed due to damping (shock absorber)*
I1	*current vertical position of the wheel*
I2	*current vertical speed of the wheel*
X	*current position of car along the road*
Y	*road height at position X*
X1, Y1, X2, Y2	*table entries for positions immediately greater than and immediately less than the current value of X*

```
100 REM SET PROGRAM CONSTANTS
110 K1=-100000
120 K2=-2000
130 M=500
140 V=10
150 D=0.05
151 REM SET INITIAL SPEED AND POSITION
160 T=0
170 P1=9.8*M/K1
171 S1=0
172 REM FIND INITIAL ROAD SURFACE
175 I1=0
176 X1=0
177 Y1=0
178 X=0
182 GOSUB 600
183 I2=(Y-I1)/D
184 I1=Y
185 REM CALCULATE INITIAL FORCES
186 F1=(P1-I1)*K1/M
187 F2=(S1-I2)*K2/M
188 A1=F1+F2-9.8
189 REM SET PAST DATA EQUAL TO INITIAL DATA
190 S2=S1
200 S3=S1
210 S4=S1
230 A2=A1
240 A3=A1
250 A4=A1

259 REM      BEGIN SIMULATION
260 REM PREDICT SPEED AND POSITION
270 S=S1+D/24*(55*A1-59*A2+37*A3-9*A4)
280 P=P1+D/24*(55*S1-59*S2+37*S3-9*S4)
281 X=X+V*D
282 REM FIND NEW ROAD SURFACE
283 GOSUB 600
284 I2=(Y-I1)/D
285 I1=Y
290 REM PREDICT NEW FORCES
291 F1=(P-I1)*K1/M
292 F2=(S-I2)*K2/M
300 A=F1+F2-9.8
310 REM CORRECT SPEED AND POSITION
320 S=S1+D/24*(9*A+19*A1-5*A2+A3)
330 P=P1+D/24*(9*S+19*S1-5*S2+S3)
340 REM CORRECT FORCES AND UPDATE SAVED DATA
341 F1=(P-I1)*K1/M
342 F2=(P-I2)*K2/M
350 A4=A3
360 A3=A2
370 A2=A1
380 A1=F1+F2-9.8
390 S4=S3
400 S3=S2
410 S2=S1
420 S1=S
430 P1=P
440 T=T+D
450 PRINT T,S1,P1
460 IF X<100 THEN 270
470 END

600 REM INTERPOLATE TABLE TO FIND
601 REM  VALUE OF Y CORRESPONDING
602 REM  TO GIVEN VALUE OF X
605 DATA 0,0,10,0,12,0.08,13,0.08,14,0,20,0,21,-0.04
606 DATA 22,-0.04,23,-0.08,24,0,30,0,50,0.1,51,0,100,0
611 REM TABLE FORMAT IS X(1),Y(1),X(2),Y(2), ...
620 IF X<X1 THEN 670
630 X2=X1
640 Y2=Y1
650 READ X1,Y1
660 GO TO 620
670 Y=Y2+(Y1-Y2)*(X-X2)/(X1-X2)
680 RETURN
```

I expect it will occur to many of you that graphic rather than printed output will make this program much clearer. The waveform produced by a plot of the data would give you a much better feel for the motion of the car body. For the example in this listing, try plotting position from −0.1 meter to +0.1 meter versus time from 0 to 100 seconds.

One final note: to avoid losing data, it is important that the interval between points of the table in the interpolation subroutine is larger than the distance the car moves in one step. In other words, if you want to model a road that changes rapidly, you will have to reduce the step size (D) to a value less than the minimum of (X(n) − X(n+1))/V.

Simulation of Motion Part 3:

Model Rockets and Other Flying Objects

Stephen P Smith
POB 841
Parksley VA 23421

Since becoming involved in personal computing, I've only met a few real applications oriented users. Most microcomputer owners are either hardware oriented, eg: hams or other electronics hobbyists, or they are software hackers. Both groups tend to love their machines for their own sake and not necessarily because they are useful. The users I've met who are more interested in the answers they get than in how they got them have all been running financial programs. Despite this thin showing, I believe that the next large group to "discover" personal computing is going to be applications oriented. They will be the business people and hobbyists who need more computing power than is available in a pocket calculator, but who can't justify access to a large computer.

Among this group will be the model rocket and aircraft builders. Those people delight in creating miniature NASAs, but they have always lacked one important resource, computing power. The government and the aerospace industry invest a great deal of effort in simulating flights long before any hardware is put together. Few hobbyists could do this until now. In this article, I'll show how a microcomputer can be used to simulate the flights of model or amateur rockets and aircraft. The simulations can be used as aids in design and "mission" planning. I also hope to demonstrate the desirability of personal computing as an adjunct to other pastimes.

As in my previous articles, these simulations are intended to serve as examples. The techniques involved can be applied in almost any real world application. The lunar lander game, for example, served to illustrate how simulations are separated into degrees of freedom, and how speed and position are predicted for steps into the future. Those concepts were also applied when we simulated the motion of an automobile suspension system. That simulation illustrated how forces could depend directly on the motion being simulated. It also served to introduce some powerful numerical techniques. All of these concepts will apply to the flight simulations. In addition, I'll show how to calculate the effects of a force which acts in two degrees of freedom at the same time. I'll also introduce angular motion and demonstrate the way in which a simulation keeps track of how fast a body is turning and where it is pointing. While developing a flight simulation in detail, I'll try to point out specific areas where these new techniques can be applied to the earlier applications.

Let's begin, in fact, by outlining a game

Body	C_d
flat plate (1 square meter)	0.7
sphere (0.1 meter diameter)	0.003
airplane body (2 square meters)	0.08
wing or fin edgewise (1 square meter)	0.012
model rocket (2 cm diameter)	0.0001
automobile (2 square meters)	0.6
motorcycle and rider (2/3 square meters)	0.25

Table 1: Drag coefficients for various bodies. These coefficients include the body area and air density term ($1/2 \times 1.192\ kg/m^3$). They are intended to be used in an equation of the form:

$$DRAG = SPEED^2 \times C_d$$

If larger or smaller bodies are used, a simple ratio of areas will convert the coefficient.

simulation you can program. In the lunar lander game, aerodynamic forces were neglected, but these forces must be considered for atmospheric flight. They also present a good example of forces which act in more than one degree of freedom. We will investigate them through the use of a simple game I'll call EVEL. The object of the game is to select the ramp angle and speed with which to leap a motorcycle over a given number of cards and land successfully on the downward ramp. The motion will be in two degrees of freedom, vertical and horizontal. The forces will be gravity and aerodynamic drag.

We have seen in the previous articles how gravity affects speed, and most people have an intuitive understanding of drag. Drag is the force you feel when you hold your hand out the window of a moving car. It is the resistance of air to a body moving through it and it acts directly opposite the motion. Drag is calculated in much the same way as the force created by an automobile shock absorber. In that example the force was equal to the speed multiplied by a damping coefficient. To calculate drag, we will multiply the speed squared by a constant called the drag coefficient (symbolically C_d). Drag coefficients for some common bodies are given in table 1. C_d takes into account the size, shape and surface texture of the body. In our simulations, it will also include a factor for the density of the air (1.19 kg per cubic meter). More detailed simulations will take into account the changes in air density and temperature which occur at higher altitudes and adjust the aerodynamic forces accordingly. To avoid this complication, we will restrict our simulations to altitudes within a few thousand meters of sea level. The formula we will use for calculating drag is DRAG = SPEED^2 x C_d.

If the only motion is upward, the drag acts only in the vertical degree of freedom. There are also cases in which it acts only horizontally; but in general, there will be motion in both directions, and the drag, which acts directly opposite the motion, will be felt in both degrees of freedom. Because of its dependence on the square of the speed, we cannot calculate separate vertical and horizontal drags. We must calculate one force and apportion it between the two degrees of freedom.

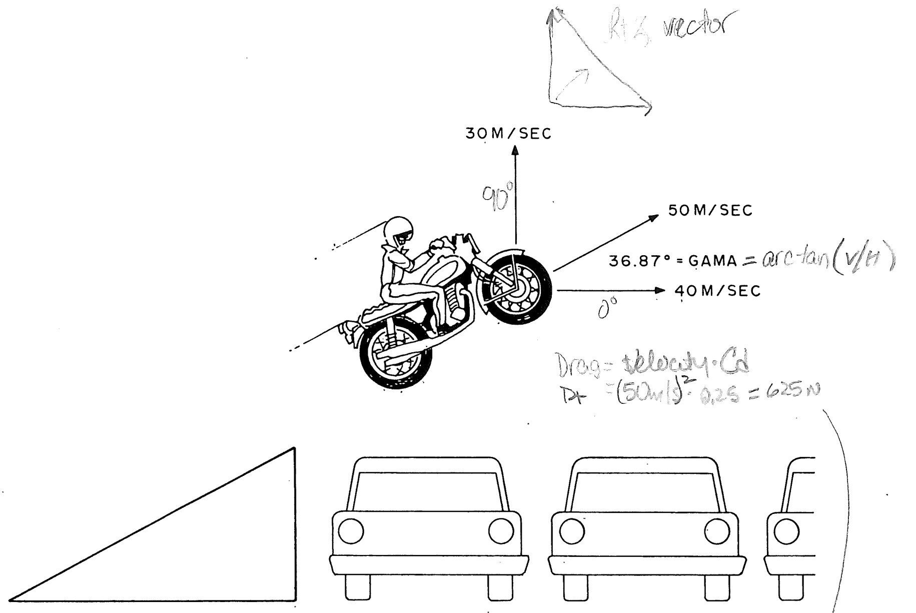

Figure 1: The total speed and flight elevation angle can be calculated from the horizontal and vertical speed components.

Apportioning Drag

$D_v = D_t \cdot SIN(GAMA) = 625 \sin(36.87) = 375$

$D_h = D_t \cdot COS(GAMA) = 625 \cos(36.87) = 500$

$D_t = \sqrt{D_v^2 + D_h^2}$ Since motion is represented by rt. triangle.

Assume mass = 150 kg $\Delta t = 0.1$ sec

$a_h = \frac{500}{150}(0.1) = 0.333$ m/s $50 - .333 = 49.667$

$a_v = 30 - \left(\left(\frac{375}{150} + 9.8\right) \cdot 0.1\right) = 28.77$ m/s

Vertical acceleration

g (gravitational constant)

Figure 1 shows a typical case. Here a daredevil motorcyclist has just left his takeoff ramp. Suppose we know from a previous simulation step that his vertical speed is 30 meters per second (m/sec) and his horizontal speed is 40 m/sec. To calculate drag, we must first find the total speed. Our fortunate selection of degrees of freedom now becomes apparent, because the vertical and horizontal velocities can be seen to form two sides of a right triangle. We can compute the third side, or hypotenuse, by applying the theorem of Pythagoras ($C^2 = A^2 + B^2$). The total velocity will be equal to the square root of the sum of the squares of the speeds in each degree of freedom. In this case, the daredevil is moving at $\sqrt{30^2 + 40^2} = 50$ m/sec. Using C_d from table 1, we calculate a drag of 0.25 x 50^2 = 625 newtons, acting at some angle between horizontal and vertical. This angle is called the flight elevation (symbolically GAMA in some computer programs). It can be found using a little trigonometry. If we let horizontal be 0 degrees, and let vertical be 90 degrees, then GAMA is equal to the arc tangent of the vertical divided by horizontal velocity. In this case, GAMA = arc tan (30/40) = 36.87 degrees. Knowing the angle, it is easy to apportion the drag. The forces which result are called components of drag. The vertical component (symbolically D_v is given by D_v = DRAG x SIN (GAMA). The horizontal component, D_h, is given by D_h = DRAG x COS (GAMA). In

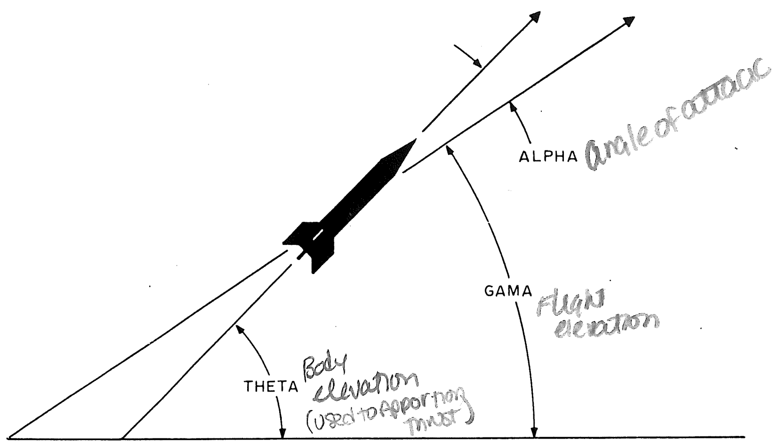

Figure 2: The direction of flight is called the flight elevation (GAMA). The direction the rocket is pointing is called the body elevation (THETA). The difference between them is called the angle of attack (ALPHA= THETA -GAMA).

the current example, $D_V = 625 \times \text{SIN}(36.87) = 375$ newtons, and $D_h = 625 \times \text{COS}(36.87) = 500$ newtons. You can check these calculations by noting that $\sqrt{375^2 + 500^2} = 625$. Readers familiar with trigonometry will be able to confirm that

$$\text{DRAG} = \sqrt{D_V^2 + D_h^2}$$

in every case. This is the same formula we used to find the total speed, so it should not be surprising to find that the vertical and horizontal speeds are also referred to as components.

The effect that the components of drag have on the components of speed depends, of course, on the mass. If our daredevil is fairly small, and rides a light motorcycle, the total mass in flight might be 150 kg. During a step of 0.1 seconds, the horizontal speed will decrease by 500/150x0.1 = 0.333 m/sec. A similar change occurs in the vertical speed, but there we must also include gravity. Remember from the previous simulations that each second, gravity subtracts 9.8 m/sec from the vertical speed. In 0.1 seconds it will change from 30 m/sec to 30-(375/150 + 9.8)x0.1 = 28.77 m/sec. Knowing the new speeds, you can compute the new position, the new drag and the new flight elevation. The simulation can be stepped forward again and again until the daredevil returns to earth.

Part 1 of this series, "An Improved Lunar Lander Algorithm," began on page 18 of November 1977 BYTE, while part 2, "An Automobile Suspension," appeared on page 112 of December 1977.

Because the drag components depend on the square of the speed, it will probably be necessary to use the predictor-corrector formulas from my previous article to obtain realistic results. The initial conditions of total speed and ramp angle must be chosen. For the first simulation step set GAMA equal to the ramp angle and let the vertical speed component equal SPEEDxSIN (GAMA) and the horizontal component equal SPEEDxCOS (GAMA). Figure that a car is about 6 meters long, a bus about 15 meters, and the Snake River Canyon is 1451 meters wide. Good luck.

In many respects, a rocket or aircraft in flight is much like our daredevil. It will be moving horizontally and vertically and will be acted upon by gravity and drag. There are some other forces to be considered, however. For example, early in a rocket's flight, its engine will be producing thrust. Unless the rocket is pointing directly upward or directly parallel to the ground, we will also have to apportion this force between horizontal and vertical directions.

One way to do this is to assume that the rocket always points in exactly the direction it is moving. The flight elevation angle, GAMA, can then be used to apportion thrust just as it was used for drag. At each simulation step, the program can interpolate a table to find the value of thrust corresponding to the current time. It then computes the components and applies them to predict new speeds and position. In this manner we can build a two degree of freedom rocket trajectory simulation. This simulation might also be used for a game. Set up a duel between two artillery battalions, or better still, program a real time simulation like the lunar lander game of my first article and try to hit a moving target.

For most flight vehicles, the tendency to point in the direction of movement is a desirable characteristic. Unfortunately, this is not generally the case. Vehicle imperfections, the effect of wind, and just the fact that it takes a finite amount of time to turn the vehicle, all affect the pointing of a rocket. In order to indicate where the rocket is pointing, we define another angle, the body elevation angle (symbolically THETA). THETA tells us where the vehicle is pointed and is used to apportion thrust. The difference between THETA and GAMA is called the angle of attack (symbolically ALPHA). It is illustrated in figure 2.

Unlike GAMA, there are no components which may be used to compute THETA. We must keep track of it with a third degree of freedom. Just as altitude is calculated as the position in the vertical degree of freedom, THETA may be calculated as the position in this third, or angular, degree of freedom.

Angular motion is handled in exactly the same way as the linear motions we have already been simulating. Just as position has its angular equivalent, so do speed, force and mass. It is as easy to calculate the speed with which a body turns as it is to find the speed with which it falls.

First we must calculate the angular equivalent of a force. These are called moments (also called "torque"), and have metric units of newton meters. As the units imply, each moment has a force as part of its definition, but it also depends on how much leverage the force has. For example, suppose you apply a force of 10 newtons to the end of a 15 cm (0.15 meter) wrench. A moment of 10 x 0.15 = 1.5 newton meters will be transferred to the bolt. If a 25 cm wrench were used, a moment of 10 x 0.25 = 2.5 newton meters would result, and the bolt would be proportionately tighter. In rocket flights a moment results as an aerodynamic effect whenever THETA is not equal to GAMA. It has its own coefficient, C_m. For small angles of attack, this coefficient is multiplied first by the square of the speed (linear speed, not angular) and then by the angle of attack (ALPHA=THETA-GAMA). The aerodynamic moment is therefore caused because the rocket is trying to point itself in the direction it is moving.

The next element to be considered in the angular equation is the equivalent of mass. This is called moment of inertia, and the leverage concept applies to it also. The moment of inertia (usually shown symbolically as an upper case I) depends on the mass of the body, and also how widely spaced the mass is. For example, a 50 kg set of bar bells would have a much larger moment of inertia than a single 50 kg weight. The units of I are kilogram meters. Like mass, moment of inertia is a property of matter. In our rocket simulation, both may change as fuel is burned, but their values can be determined at any given time. It is general practice to construct tables of mass and moment of inertia which parallel the thrust table we have already introduced. In each simulation step, our program will interpolate the table to find the current value of moment of inertia and use it to find the effect of the moments. The value of the metric units will now become apparent.

Just as the force in newtons was divided by the mass in kilograms to find exactly the

(NM)
For small alphas, torque =
(Cm(speed^2)) • Alpha

I (moment of inertia) = KgM
= property of matter, function of
distribution of mass

Change in angular speed =
$\frac{T}{I}\ \frac{NM}{KgM} = N/Kg$

Speed in °R/S (radians/sec)
Position in radians
(2π radians = 360°)

```
100 REM PITCH PLANE TRAJECTOTY SIMULATION
110 REM INITIALIZE VARIABLES
120 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
130 READ U,W,V,U1,W1,V2,X1,Z1,Q1,F1,M1,I1,K
140 REM INITIALIZE TIME(T),STEP SIZE(H), PRINT INTERVAL(K1)
150 H=0.01
160 LET T=0
170 LET T1=T
171 LET K1=0.1/H
180 REM SET INITIAL ALTITUDE (Z1>0)
190 LET Z1=1
200 REM SET DRAG(D) AND MOMENT(P) COEFFICIENTS
210 LET D=-1.0E-4
220 LET P=-3.0E-5
230 REM SET LAUNCHER LENGTH(R) AND ELEVATION(L)
240 L=90
250 PRINT "LAUNCH ELEVATION =";L;" DEGREES"
260 LET L=L*0.01745
270 LET L1=L
272 LET G=L
280 LET R=Z1+1
300 REM SET WIND SPEED
310 N=5
320 PRINT "WIND SPEED =";N;" METERS/SECOND"
330 LET U1=N
340 REM END INITIALIZATION, BEGIN TRAJECTORY
350 PRINT "TIME                    ALTITUDE      RANGE        SPEED"
360 PRINT "(SEC)                   (METERS)     (METERS)     (M/SEC)"
370 REM GET THRUST(F), MASS(M), & MOMENT OF INERTIA (I)"
380 GOSUB 950
390 LET T=T+H
400 LET K=K+1
410 REM PREDICTOR FORMULAS
415 REM B,W,Z,ARE VERTICAL ACCELERATION,SPEED & POSITION
420 LET B1=(SIN(G)*D*V2+SIN(L)*F)/M-9.8
430 LET W=W1+H*B1
440 LET Z=Z1+H*W1
460 REM A,U,X ARE HORIZONTAL ACCELERATION, SPEED & POSITION
470 LET A1=(COS(G)*D*V2+COS(L)*F)/M
480 LET U=U1+H*A1
490 LET X=X1+H*(U1-N)
500 LET V2=(U+N)↑2+W↑2
510 LET V=SQR(V2)
511 REM NO ANGULAR MOTION ON LAUNCHER
512 IF Z<R THEN 590
513 LET G=ATN(W/(U+N))
514 IF U+N>0 THEN 530
515 LET G=G+3.14159
520 REM NEGLCT ANGULAR MOTION AFTER BURNOUT
530 IF F=0 THEN 610
535 REM C,Q,L ARE ANGULAR ACCELERATION, SPEED & POSITION
540 LET C1=P*(L-G)*V2/I
550 LET Q=Q1+H*C1
560 LET L=L1+H*Q1
590 GOSUB 950
600 REM CORRECTOR FORMULAS
610 LET B=(SIN(G)*D*V2+SIN(L)*F)/M-9.8
620 LET W=W1+H/2*(B+B1)
630 LET Z=Z1+H/2*(W+W1)
640 LET W1=W
660 Z1=Z
680 LET A=(COS(G)*D*V2+COS(L)*L)/M
690 LET U=U1+H/2*(A+A1)
700 LET X=X1+H/2*(U+U1)
710 LET U1=U
720 LET X1=X
730 LET V2=(U+N)↑2+W↑2
740 LET V=SQR(V2)
750 REM NO ANGULAR MOTION ON LAUNCHER
751 IF Z<R THEN 880
752 G=ATN(W/(U+N))
753 IF U+N>0 THEN 770
754 G=G+3.14159
760 REM NEGLECT ANGULAR MOTION AFTER BURNOUT
770 IF F=0 THEN 880
780 LET C=P*(L-G)*V2/I
790 LET Q=Q1+H/2*(C+C1)
800 LET L=L1+H/2*(Q+Q1)
810 LET Q1=Q
820 LET L1=L
880 IF K<K1 THEN 390
890 PRINT T,Z,X,V
900 LET K=0
920 IF Z>0 THEN 390
930 REM STOP WHEN ALTITUDE RETURNS TO ZERO
940 STOP
950 REM INTERPOLATE TIME TABLE FOR F,M,& I
960 DAT 0,5.6,0.0458,1.1E-4,0.1,13.3,0.0446,1.1E-4,0.2,5.6,0.0434,1.1E-4
970 DATA 1.6,5.6,0.0337,1.0E-4,1.7,5.6,0.0333,1.0E-4,100,0,0.0333,1.0E-4
980 IF T<T1 THEN 1050
990 LET T2=T1
1000 LET F2=F1
1010 LET M2=M1
1020 LET I2=I1
1030 READ T1,F1,M1,I1
1040 GO TO 980
1050 LET H4=(T-T2)/(T1-T2)
1060 LET F=F2+(F1-F2)*H4
1070 LET M=M2+(M1-M2)*H4
1080 LET I=I2+(I1-I2)*H4
1090 RETURN
1100 END
```

A Model Rocket Simulator

Listing 1: This BASIC program illustrates the use of an angular degree of freedom to apportion a force between two linear degrees. It simulates the flight of a model rocket that might be built from widely available kits. During the early part of the flight (approximately 1 meter), the rocket rides on a guiding rod. The rod prevents the body from turning, so angular motion must be suppressed in the simulation. After leaving the launcher, the vertical, horizontal and angular pitching motions are all considered. A short time later, however, the fuel will be exhausted and the thrust will drop to zero. Because the only reason for computing THETA was to apportion the thrust, angular motion may be neglected from this point until the vertical position becomes zero (rocket returns to earth) and the simulation ends.

This type of simulation is particularly useful for determining the effect of wind on a rocket. The tendency a rocket has to point in the direction it is moving also causes it to turn into the wind. The highest altitude may not, therefore, be achieved with a vertical launch. By running a series of simulations on his/her personal computer, a model rocket builder can determine in advance the ideal launch angle for various steady state wind speeds. The plots shown in figure 3 illustrate this type of study. Note that in each case the horizontal degree of freedom is aligned with the wind.

Because the forces and moments in this simulation depend on the motion, the predictor-corrector method has been applied. To conserve space, however, it is not the fourth order method used in the automotive simulation of last month's article, but a shorter second order version. Improved accuracy would result if you were to implement the longer formulas. You may also want to customize the output. For example, a plot of angle of attack versus time might be interesting. To print out ALPHA in degrees, use (L−G) x 57.295 to convert from units of radians. Special output exactly at apogee (maximum altitude), impact, or motor burnout might also be helpful in evaluating performance. The code is well-commented, so feel free to dig in and adapt this program to your own needs.

change in speed each second, so the moment in newton meters can be divided by the moment of inertia in kilogram meters to find exactly the change in angular speed. In angular degrees of freedom the units of speed are radians per second and the position will be computed in radians. There are 2π radians in a full circle, so 1 radian equals 57.296 degrees. Because most BASIC interpreters perform trignometric calculations in radians, these units are to be preferred. Conversion to degrees is easily made for input and output.

Once the effect of each moment is known, it is multiplied by the time step size and added to the speed in exactly the same technique used for linear motion. Similarly the angular speed is multiplied by step size to update the angular position. The predictor-corrector method may be applied by saving moments, and speeds from previous steps. I have included a BASIC program with this article which involves both the predictor-corrector formula and angular motion in a three degree of freedom (pitch plane) trajectory simulation. Noting the similarity between the angular and linear equations used there should make this technique easier to understand.

The concept of angular motion is easily transferred to other applications, but to make best use of it you really need some familiarity with the forces and moments which may appear. For example, an automotive enthusiast will already understand how the road surface induces motion in the body of a car. We saw last month how to simulate that motion for one wheel. If you include two wheels, the front and back on one side, a second degree of freedom in body motion is introduced. Most people would recognize that, but only someone familiar with automobiles might realize immediately that the two ends of the body will not move up and down entirely independently. The second degree of freedom must be an angular one. It will measure the pitching motion of the car while the original degree measures its overall up and down motion. The forces created by each set of suspension parts will contribute to the linear motion, and will be multiplied by their leverage (perhaps their distance from the driver's seat) to determine their affect on the angular motion. At each step, the angular motion will be combined with the linear motion to compute new forces, moments, etc, and the procedure will begin again. Just how those angular motions are combined with the linear ones, and how the leverage of a force is determined, will be the subjects of the fourth and last article in this series.■

Figure 3: The program of listing 1 can be used to determine the best launcher elevation for any given wind.

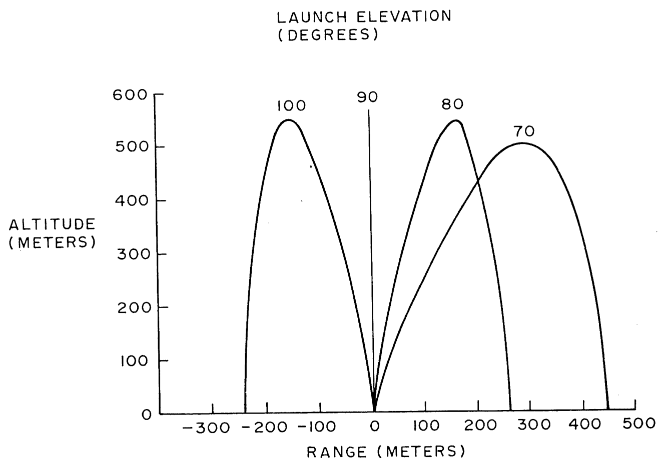

Figure 3a: Trajectories with no wind.

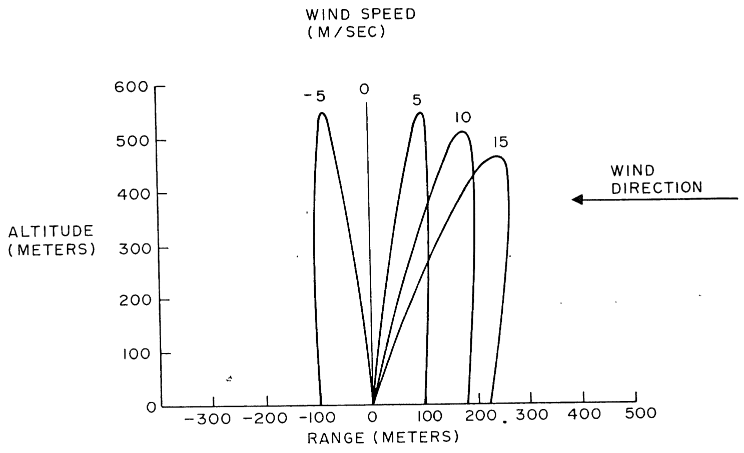

Figure 3b: 90° launch with wind.

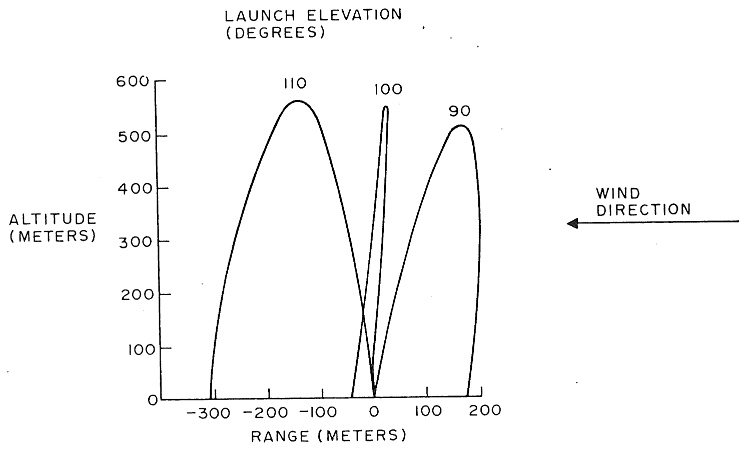

Figure 3c: Trajectories with 10 m/sec wind.

Simulation of Motion

Part 4: Extended Objects, Applications for Boating

Stephen P Smith
POB 841
Parksley VA 23421

Have you ever wondered why the shapes of boat hulls differ so widely? Boating enthusiasts know that certain designs will be best in lakes and rivers, and certain others in open seas. Some boats are much roomier than others; some are safer in rough water; but what penalties in stability and riding comfort might you pay for the extra room or seaworthiness? The motion of a boat depends on its response to the variety of waves it encounters. These motions can be simulated on your personal computer. You can determine how a given design will respond to any sea condition. The basic equations for stepping speed and position into the future will still apply, as they were discussed in the earlier articles of this series; but you'll also need some new techniques. As you implement this simulation, you'll discover that forces in a linear degree of freedom can also produce moments and their resulting motion in an angular degree of freedom. In this article, I'll show how that interaction is handled. I'll also introduce the concept of distributed forces, and a numerical technique to handle them. Although developed for a boating application, these new ways of calculating forces should find use in updating many of our previous simulations.

We have already seen quite a variety of ways to calculate forces. Gravity, a force present in every simulation in the last three articles, simply made a constant change in the vertical speed at each step. Thrust, used in rocket and aircraft simulations, came either from a user input or from a table interpolation. Forces in an automobile suspension were found to depend directly on the vertical position (spring force) and the vertical speed (damping force). Aerodynamic forces were computed by multiplying a coefficient (ie: constant) by the sum of the squares of the speeds in each linear degree of freedom. While these examples cover most of the situations you are likely to encounter in simple models, any new simulation might present some unique requirement.

For example, in all the calculations, the forces have had one thing in common. They acted at a single point. We call such forces discrete. In reality, some are not discrete, but act at many points on a body simultaneously. These are referred to as distributed forces. Aerodynamic drag is a typical distributed force. Although we used the drag coefficient to calculate a single force, the retarding action of the air acts all over the body. A coefficient is just one tool used to convert distributed forces into discrete ones. Not all distributed forces can be converted using coefficients, so I'll introduce a more general technique using the boating simulation as an example.

The principle forces on a boat are gravity and buoyancy, the floating power of the hull. Because buoyancy is an upward push provided by the water, it is not difficult to see that it is a distributed force covering the entire area of the boat below the water line. Converting this distributed force to a discrete one will allow us to simulate the vertical motion of the boat.

Perhaps more important to the boat designer or buyer will be the angular, rolling

Wind Speed	Rivers	Lakes	Inlets	Bays	Open Sea
			Typical Bodies of Water		
2 m/sec					
period (sec)	0.6	1.0	1.5	2.0	3.5
length (m)	0.56	1.5	2.3	3.1	20.0
height (m)	0.02	0.06	0.12	0.15	0.5
5 m/sec					
period (sec)	0.8	1.2	2.0	2.4	4.5
length (m)	0.1	2.25	5.0	9.0	30.0
height (m)	0.05	0.08	0.2	0.25	0.75
10 m/sec					
period (sec)	1.25	2.0	3.0	4.25	7.0
length (m)	2.4	6.25	14.0	28.0	80.0
height (m)	0.08	0.15	0.35	0.7	2.0
20 m/sec					
period (sec)	2.5	4.0	6.0	8.5	14.0
length (m)	10.0	25.0	56.0	110.0	300.0
height (m)	0.25	0.65	1.4	2.8	7.5

Table 1: Characteristics of waves. The height, period and length of waves all vary, but for certain conditions, average values have been established. The wave length and period are affected by the depth of the water. The height depends on the wind speed, how long it has been blowing, and the width of the body of water. Readers who want to model real sea conditions should find a good oceanography text, but the above summary should prove adequate for casual use.

and pitching motion of the boat. Angular motion was introduced in a rocket flight simulation (see January 1977 BYTE, page 144). In that case, it was entirely independent of the linear motion. At the end of the same article I suggested that the motion of an automobile body should also be simulated using an angular degree of freedom, but that the angular and linear motions could no longer be considered separately. This is also true in the boating example. The moments used to compute the angular motions will be calculated directly from the linear motions. Because the forces in the automotive example are discrete, we'll develop the technique to handle combined angular and linear motion using the distributed forces of the boat example. In that way, one simulation will serve to demonstrate both of the new concepts. I'll leave the development of a two or four wheel automobile suspension simulation to interested readers.

The motion of a boat is similar in many ways to that of the automobile body. When it is launched, a boat settles into the water in response to gravity. As the hull displaces more water, the buoyant force becomes larger, until at some point, it balances gravity and the boat stops sinking. This point is called equilibrium and is analogous to the equilibrium of an automobile suspension. Unless there is a disturbance, the boat will remain at equilibrium. In the automotive example, disturbances came in the form of a rising or falling road. With boats, we encounter a rising and falling sea, in other words, waves.

Sea waves occur in a variety of shapes. Their length (distance peak to peak), their height (distance peak to trough), and their period (time to rise and fall), all vary apparently at random. In fact, these parameters have fairly well defined relationships. Readers with an interest in modeling sea states should refer to a good marine science text. For this simulation, we'll represent waves with a sine function, and use the data in table 1 to compute their size.

Dealing just with forces for a moment, let's see how a small object is affected by wave motions. Figure 1 shows a bottle, floating in a body of water. We know from our previous simulations that every second, gravity subtracts 9.8 meters per second from the bottle's vertical speed. If the bottle is to remain stationary, the effect of buoyancy (force divided by mass) must be equal and opposite (ie: 9.8 meters per second per second upward). The mass of the bottle should be known. Let it be 0.1 kilograms. The buoyant force is equal to the weight of the water displaced by the bottle. Remember that weight is a force, the effect of gravity acting on a mass. The weight of the water, in newtons, is equal to its mass, in kilograms, times the effect of gravity, 9.8 meters per second per second. Each 1000 cubic centimeters (cc) of water has a mass of 1 kilogram, and thus a weight of 9.8 newtons. Knowing this, and the mass of the bottle, we can calculate the amount of water that the bottle displaces. In other words, we can find the volume, V, of the bottle below the water line at equilibrium. Force divided by mass must equal 9.8 to balance gravity, so the equation 9.8 = 9.8/

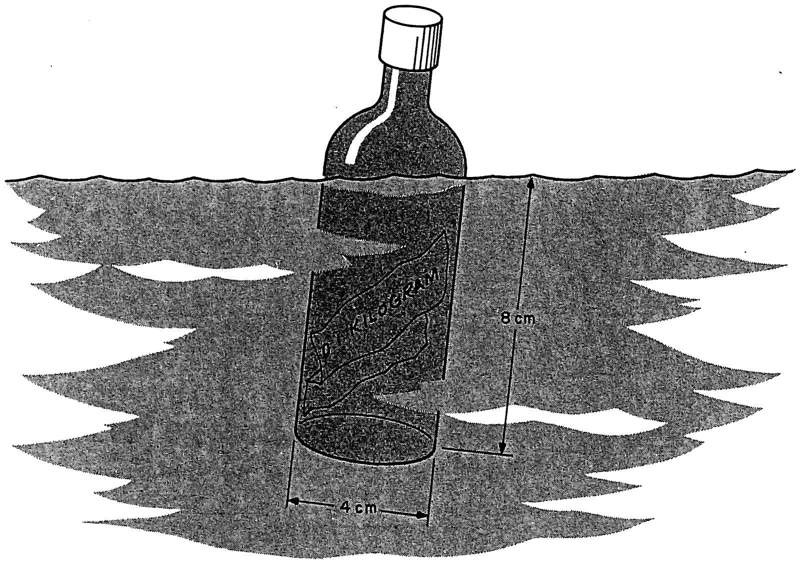

Figure 1: A bottle sinks until it displaces an amount of water equal to its own weight.

1000 * V / 0.1 can be solved for V to find that 100 cc of the bottle is under water. If the bottle is 4 centimeters in diameter, we will find (from the formula for the volume of a cylinder) about 8 centimeters of its length must be below the surface.

Now suppose that the surface of the water rises suddenly. More than 8 cm of the bottle will be underwater, and the buoyant force will exceed gravity. The vertical speed of the bottle will increase and it will rise with the water. When the bottle reaches equilibrium again it will still have a positive vertical speed, so it will pass through that point and continue to rise. Now, however, it is gravity which is the larger force, and the vertical speed will be reduced until the bottle begins to descend. Eventually, these motions will disappear (due to the drag force applied by the water) and the bottle will come to rest at equilibrium. This happens so quickly that the bottle appears to be moving up and down exactly with the waves.

For larger objects, boats for example, the actual motion may be more apparent. We could treat a boat exactly like the bottle and simulate its up and down motion. As I suggested earlier, however, it is the angular rolling and pitching motion of the boat that is of real interest. To simulate these motions,

OUTSTANDING TERMINAL EQUIPMENT VALUE

IBM 2741 — compatible or standard ASCII (with Break and Control code features). RS232-C interface.

RECONDITIONED SELECTRIC TERMINALS

CONTACT KEN PAYNE

Dal-Data Inc. · 1111 W. Mockingbird Lane Suite 1400 · Dallas, Texas 75247 · 214-630-9711

IBM Correspondence or BCD Code $795.00
- IBM 2741 Compatible
- RS232C Interface

ASC II Code $995.00
- 300 Baud 200 Character Print Buffer
- RS232C Interface
- Break and Control Code features

30 DAY WARRANTY NATIONAL SERVICE AVAILABLE

Circle 16 on inquiry card

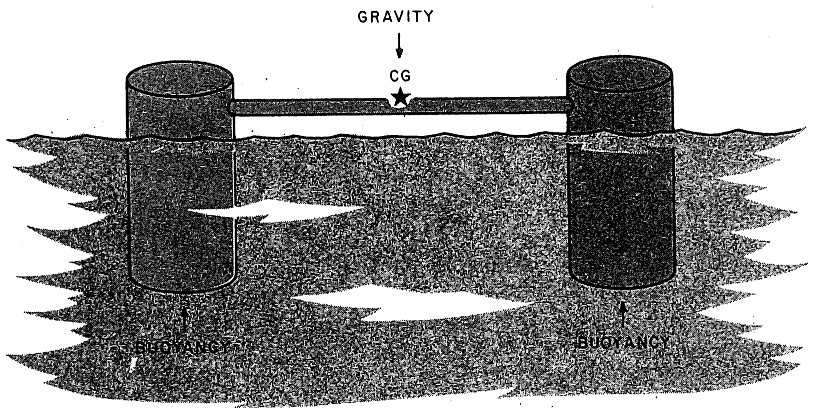

Figure 2: The distribution of the buoyant force determines the angular motion about the center of gravity.

we will need to know not just the total buoyant force, but also how it is distributed over the hull. The device shown in figure 2 will illustrate a general technique for finding the distribution.

You can think of this device as two bottles joined together with a stick or as the two hulls of a catamaran. Just as we calculated the buoyant force on the bottle in figure 1, we can calculate separate forces on each of these two bottles. The sum of the forces can be used to compute the vertical motion of one point on the device. This point is called the center of gravity (CG). The location of the center of gravity is critical. If you were to place the stick on a knife edge and find the point at which it balanced, this would be the center of gravity. It is the point on a body where the effect of gravity appears to be concentrated. Because the mass of an object is distributed throughout its volume, weight is a distributed force. By locating the center of gravity, however, we have a tool that transforms it into a discrete one.

We could also define a center of buoyancy, the point at which the total buoyant force appears to act. Unfortunately, the location of this point can move significantly as the boat rises and sinks in the water. The center of gravity is also subject to some movement, such as when a passenger moves from the back to the front of the boat. Unlike the movement of the center of buoyancy, however, changes in the location of the center of gravity are not tied directly to the results of our simulation, the linear and angular motion of the boat. For this simulation, we will treat the center of gravity as stationary, and try to avoid dealing with the moving center of buoyancy.

Since we cannot deal with buoyancy as simply as we do gravity, we will have to deal with the individual parts of a body more directly to find a general method of handling the distribution. In the case of the "catamaran" in figure 2, this is fairly easy. First, we assume that the bottles are small when compared to the length of the stick. Next, we assume that the center of buoyancy of each bottle is at its center, no matter how it sits in the water. Now, as far as our simulation is concerned, the entire buoyant force on the bottle acts at a point which is at a known distance from the center of gravity. This makes no difference to the vertical degree of freedom, but it is the key which allows us to simulate the angular motion.

Remember from the last article that a moment is the product of a force times a distance. In the current example, each bottle creates a moment equal to the buoyant force times the distance of the bottle from the center of gravity. Note that we define distances to the right as positive, and to the left as negative. Thus an upward force on the righthand bottle creates a positive (counterclockwise) moment. An upward force on the lefthand bottle creates a negative moment. In each simulation step, the moments are summed and then divided by the moment of inertia to find the change in angular speed each second. With this value, we can step the angular degree of freedom into the future, and return to compute new forces and moments.

Now we must determine how the combined angular and linear motion can be used to compute the new buoyant force. The force is proportional to the volume of the bottle below the waterline. For a single bottle, it was computed from the position in the vertical degree of freedom, and the location of the water surface. With the two bottle device, the vertical degree of freedom tells us only the position of the center of gravity. We must use the angular degree to find the relative position of other points on the device. If there is a positive angular position, the device will be turned counterclockwise around the center of gravity. Consequently, the lefthand bottle will be lower than the center of gravity and the right one will be higher. The exact difference is calculated by multiplying the sine of the angular position by the distance of the bottle from the center of gravity. Again, note that points to the left have a negative distance from the center of gravity. Positive angular positions move them down.

Let's illustrate this with an example. Suppose the vertical position of the center of gravity is 0.01 meters, and the angular position is 2 degrees (0.035 radians). A bottle 1.2 meters to the left would be at

0.01 + SIN (0.035) * (−1.2) = −0.32 meters

in other words, about 3 centimeters below its equilibrium position. A bottle 1.2 meters to the right would be

0.01 + SIN (0.035) * 1.2 = 0.052 meters high.

The vertical position of any other point can be found similarly.

Having found the positions of the bottles, we must now find the positions of the water surface at each bottle. These will come from a sine function modified by a representative wave height, period and length. The argument of the function will be the sum of the current time divided by the period, and the bottle location (distance from the center of gravity) divided by the wave length, all multiplied by two π (to convert to radians). Once evaluated, the function is multiplied by one half the wave height (amplitude) to arrive at a final surface position. Using this scheme the surface varies with both time and location in a good approximation of sea waves.

The data in table 1 can be used to continue the example we began above. Let's place our two bottle catamaran in an inlet with 5 meter per second winds. We have determined that the water surface is given by the following formula.

```
S = HEIGHT/2 * SIN (6.28318*(TIME/
        PERIOD + LOCATION/LENGTH) )
```

At TIME = 1.8 seconds, we would find that the water surface at the left bottle is 0.2/2* SIN(6.28318*(1.8/2+(1.2/5))=0.084 meters, just below the equilibrium position. At the right bottle, the surface is at 0.2/2*SIN (6.28318*(1.8/2+1.2/5))=0.077 meters. If we subtract the positions of the bottles from these values and add the 8 centimeter length of the bottles underwater at equilibrium, we will have calculated the length of each bottle below the surface at TIME = 1.8 seconds. For the left bottle this will be (−0.084) − (−0.032) + 0.08 = 0.28 meters. For the right bottle this will be 0.077 − 0.052 + 0.08 = 0.105 meters. If the bottles are 4 centimeters in diameter, then the left one displaces 0.04**2 * 3.14159/4 * 0.028 = 3.45 * 10 −5 cubic meters and has a buoyant force of 9800 * 3.45 * 10 −5 = 0.345 newtons. The moment it produces is 0.345 * (−1.2) = −0.414 newton meters. Similarly, the right bottle displaces 9.67 * 10 −5 cubic meters and produces a force of 0.948 newtons and a moment of 1.14 newton meters. The sum of the forces, 1.293 newtons, is used to update the vertical degree of freedom. The sum of the moments, 0.534 newton meters, is used to update the angular degree of freedom. Now we can compute new positions, forces, moments, etc, and begin the cycle again.

Simulating the motion of a two bottle catamaran may not be very useful, but the technique is easily extended to real boats. Instead of thinking of the hull of your boat as a continuous surface, think of it as a collection of "bottles." Figure 3 shows a boat hull that has been divided in this

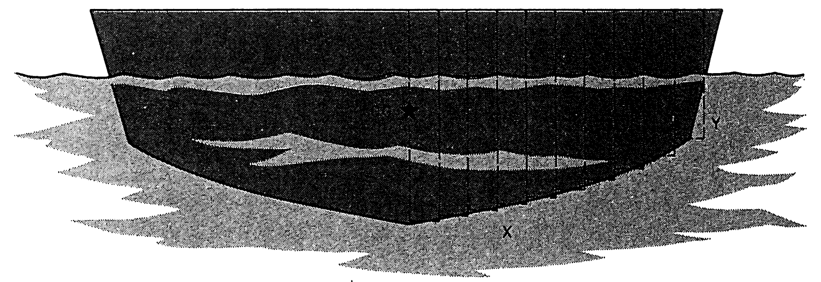

Figure 3: The continuous hull of a boat can be divided into a series of discrete segments or "bottles." X is the distance from the center of gravity to the center of the bottle. Y is the length of the bottle below the water line at equilibrium. Note that the symmetry about the CG enables us to describe the hull while only segmenting half of it.

X(m)	Y(m)
±0.05	0.50
±0.15	0.49
±0.25	0.47
±0.35	0.45
±0.45	0.42
±0.55	0.39
±0.65	0.36
±0.75	0.31
±0.85	0.37
±0.95	0.22

PAiA 8700 Computer/Controller

The Answer For... Student Hobbyist Manufacturer

8700 Processor: 6503 MPU. Wear free "Active Keyboard". Micro-Diagnostic® Extensive documentation. Fully Socketed.

Piebug Monitor: Relative address calculator. Pointer High-low. User Subroutines. Back-step key.

Cassette Interface: Load & Dump by file #. Tape motion control. Positive indication of operation.

Applications systems from $90 (10 unit quantity)

Development systems from $149 (single unit)

TELL ME MORE I want to see for myself that the 8700 is The Answer.

( ) Please send documentation and price lists. $10 enclosed.

( ) I don't need documentation please send price lists.

( ) Please send FREE CATALOG.

name: ______

address: ______

city: ______ state: ______ zip: ______

PAiA ELECTRONICS DEPT. 2-B • 1020 W. Wilshire Blvd. • Oklahoma City, OK 73116 (405) 843-9626

Circle 91 on inquiry card.

manner. The hull can now be described by a table of Xs and Ys. The Xs represent the distances from the center of gravity to the center of each bottle. The Ys represent the lengths of each bottle below the waterline at equilibrium. Now, instead of making a series of calculations for one or two bottles, we make them for many. Just as before, the sum of the forces influences the vertical motion of the center of gravity, and the sum of the moments influences the angular motion around the center of gravity.

We now have an effective method for dealing with distributed forces. We simply divide the area over which the force acts into small segments. Within each segment, we neglect the distribution and calculate a discrete force and moment. Finally, we sum the force and moments to find the effects on the linear and angular speeds.

I have included a BASIC program with this article to illustrate the technique as applied to our boating example. With the data supplied, it computes the rolling motion of the hull cross section pictured in figure 3. Boating enthusiasts will be able to insert some other hull cross sections (deep vee, trihull, etc) in the data statements and compare the response to the sample sea states. If lateral (side to side) sections are used, the program will simulate rolling motion. If longitudinal (fore and aft) sections are used, the program will compute pitching motions. Interested readers should be able to extend the program to include three dimensional boat models and simulate both angular motions simultaneously.

With the inclusion of techniques for handling distributed forces and combined angular and linear motion, your collection of software tools for simulating motion is fairly complete. When using these tools on a personal computer, you should try above all to limit the scope of your simulations. Determine which motions really interest you and neglect or restrict the others. Divide the simulation into degrees of freedom, preferably three or less if your program is to execute with reasonable speed. Compute each force and moment individually, then apportion and sum them within the degrees of freedom. Finally, step the velocity and the position into the future. Use a small step size in your early runs, 0.01 seconds or less. Increase it to save run time only as long as your results do not change significantly. Following this procedure, and using the BASIC programming examples I have provided as a guide, you should be able to find some interesting new applications for your personal computer.

Listing 1: This program simulates the vertical and angular motion of a boat in response to sea waves. Because it involves a lengthy summation, it is inherently slow. I have, therefore, used only the second order predictor corrector formulas, and have employed a large step size. Readers who want more accuracy and who can afford to wait for results should implement the fourth order equations presented in my previous article on automotive applications (December 1977 BYTE, page 112). They should also increase the number of "bottles" used to describe the hull, and decrease the step size.

It should also be noted that the program does not simulate the viscous damping action of the water. As a result, if you are unfortunate enough to specify a resonant frequency of the hull as the wave period, the boat will appear to leap out of the water. While this result is obviously erroneous, it will highlight a design to be avoided.

```
100 REM SHIP MOTION SIMULATION
110 REM DESCIBE HULL CROSS SECTION
111 REM X IS DISTANCE FROM CG TO CENTER OF BOTTLE
112 REM Y IS LENGTH OF BOTTLE BELOW WATER AT EQUILIBRIUM
120 DIM X(10),Y(10)
140 DATA 0.05,0.5,0.15,0.49,0.25,0.47,0.35,0.45,0.45,0.42
150 DATA 0.55,0.39,0.65,0.36,0.75,0.31,0.85,0.27,0.95,0.22
160 FOR J=1 TO 10
170 READ X(J),Y(J)
180 NEXT J
190 REM SET BUOYANCY FACTOR; "BOTTLE AREA"*DENSITY*9.8
200 B=0.01*1.03*9.8
209 REM COMPUTE MASS(M) & MOMENT OF INERTIA OF CROSS SECTION
210 M=0
211 I=0
212 FOR J=1 TO 10
214 M1=B/9.8*Y(J)
216 M=M+M1*2
218 I=I+M1*X(J)*2
220 NEXT J
230 REM SET SEA STATE: HEIGHT(H),LENGTH(L),PERIOD(P)
240 H=0.2
250 L=5
260 P=2
270 REM INITIALIZE INTEGRATION VARIABLES
280 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0
290 READ Z,Z1,V,V1,A,A1,R,R1,Q,Q1,C,C1,T
300 REM INITIALIZE STEP SIZE AND PRINT INTERVAL
310 D=0.1
315 K=0
320 K1=0.1/D
321 PRINT "TIME(SEC)    VERTICAL POSITION(M)  ANGULAR POSITION(DEG)"
330 REM SUM FORCES AND MOMENTS ON THE "BOTTLES"
340 GOSUB 600
345 REM PREDICT VERTICAL MOTION
346 REM A,V,Z ARE ACCELERATION,SPEED, AND POSITION
350 A1=F/M-9.8
360 V=V1+D*A1
370 Z=Z1+D*V1
375 REM PREDICT ANGULAR MOTION
376 REM C,Q,P ARE ACCELERATION,SPEED, AND POSITION
380 C1=G/I
390 Q=Q1+D+C1
400 R=R1+D*Q1
410 REM SUM NEW FORCES AND MOMENTS FOR CORRECTOR FORMULAS
420 K=K+1
430 T=T+D
440 GOSUB 600
445 REM CORRECT VERTICAL MOTION
450 A=F/M-9.8
460 V=V1+D/2*(A+A1)
470 Z=Z1+D/2*(V+V1)
475 REM CORRECT ANGULAR MOTION
480 C=G/I
490 Q=Q1+D/2*(C+C1)
500 R=R1+D/2*(Q+Q1)
508 REM PREPARE FOR NEXT STEP
510 V1=V
520 Z1=Z
530 Q1=Q
540 R1=R
550 IF K<K1 THEN 340
560 PRINT T,Z,R*57.296
570 IF T<10 THEN 340
571 STOP
600 REM CALCULATE AND SUM FORCES AND MOMENTS ON "BOTTLES"
630 F=0
640 G=0
660 FOR J=1 TO 10
665 REM POSITIVE HALF OF HULL IS GIVEN
666 REM W IS VERTICAL POSITION OF WATER SURFACE AT BOTTLE J
667 REM W1 IS LENGTH OF BOTTLE BELOW WATER SURFACE
668 W=H/2*SIN(6.28318*(T/P+X(J)/L))
669 W1=Y(J)-Z-SIN(R)*X(J)+W
```

Listing 1, continued:

```
670 IF W1>0 THEN 672
671 W1=0
672 F1=B*W1
673 G1=X(J)*F1
675 REM MIRROR IMAGE GIVES NEGATIVE HALF
678 W=H/2*SIN(6.28318*(T/P-X(J)/L))
679 W1=Y(J)-Z+SIN(R)*X(J)+W
680 IF W1>0 THEN 682
681 W1=0
682 F2=B*W1
700 G2=-X(J)*F2
710 F=F+F1+F2
720 G=G+G1+G2
730 NEXT J
740 RETURN
750 END

TIME(SEC)  VERTICAL POSITION(M)  ANGULAR POSITION(DEG)
 0.1               0.00147774275297  0.579343661886
 0.2               0.00834856069988  2.1484430172
 0.3               0.0240789139748   4.23893821533
 0.4               0.0488239037315   6.19033468235
 0.5               0.0789839977127   7.30318099994
 0.6               0.10789765357     7.01051374777
 0.7               0.12746469314 5   5.02510841752
 0.8               0.130129340759    1.41142156817
 0.9               0.110974303806    -3.40279261698
 1                 0.0695560412445   -8.66151908825
 1.1               0.0105695497413   -13.4108553239
 1.2               -0.0569528427092  -16.6916991803
 1.3               -0.120883364889   -17.7238637849
 1.4               -0.169478381978   -16.0583471104
 1.5               -0.193013619278   -11.6821229305
 1.6               -0.18645821109    -5.06270420258
 1.7               -0.150554633294   2.89082493065
 1.8               -0.0917208520104  10.9686349622
 1.9               -0.0206887363075  17.8818895739
 2                 0.0503241234783   22.5634234899 ■
```

Circle 89 on inquiry card.

*Pat's Pending

OAE'S new **PP-2708/16** PROM Programmer is the *only* programmer with all these features:

- Converts a PROM memory socket to a table top programmer: No complex interfacing to wire—just plug it into a 2708 memory socket*
- A short subroutine sends data over the address lines to program the PROM
- Programs 2 PROMS for less than the cost of a personality module. (2708s and TMS 2716s)
- Connect 2 or more in parallel — super for production programming
- Complete with DC to DC switching invertor and 10 turn cermet trimmers (for precision pulse width and amplitude alignment)
- All packaged in a handsome aluminum case

PP-2708/16 . . A & T $295.
KIT $245.
PP-2716 (Programs Intel's 2716). . . . . A & T $295.
KIT $245.

OAE

Oliver Advanced Engineering, Inc.
676 West Wilson Avenue
Glendale, Calif. 91203
(213) 240-0080

THE COMPUTER AGE FOR THE SMALL BUSINESS HAS ARRIVED!

The Computer Mart is offering the first complete line of off-the-shelf professional business software!

APPLICATIONS INCLUDE:

- General Ledger—Daily Journal, Balance Sheet, Income Statement, G/L Report.
- Accounts Receivable—Daily Journal, Aged Receivables, Sales Journal, Statements, Customer Listing, Cash Receipts.
- Accounts Payable—Daily Journal, Open Payable, Vendor Checks, Check Register, Cash Disbursements.
- Payroll—Payroll Register, Quarter/Year-to-Date, Checks, 941's, W-2's.
- Solar Energy Analysis—analysis of feasibility of solar energy.
- Mail-Label—add, delete and change functions, full sorting and letter writing capability.

COMING SOON!

- Project Control Accounting—provides logistic control and financial accounting of projects undertaken by the organization.
- Educational Performance Evaluation—a system to store and analyze student performance for any course which utilizes tests, quizzes and homework for grades.
- Medical Billing & Scheduling—clinical accounts receivable package.
- Point of Sale Inventory Control—full inventory control from point of sale.
- Micro Information Retrieval—sophisticated system for record access and manipulation.
- Time Input Processor—stand-alone system for processing time cards.
- Word Processor—full function, CRT display, editing.

We will not be undersold by any company with nationally advertised prices on any of the following hardware:

IMSAI	SOLID STATE MUSIC
DIGITAL GROUP	SD SALES
CROMEMCO	HEURISTICS
PROCESSOR TECH	DATA GENERAL
NORTH STAR	NATIONAL MULTIPLEX
VECTOR	TELETYPE
TDL	DECWRITERS
LEAR SIEGLER	GRI
ICOM	ALPHA MICRO
SWTP	MICRO-TERM
SEALS	INTEGRAL DATA
IMS	

Call for the lowest quote!

(TDL Software interfaced to North Star Disk.)

RICK INATOME — Computer Mart

1800 W. 14 Mile - Royal Oak, Michigan 48073 - (313) 576-0900

OPENING SOON

Dearborn Hgts., Michigan

420 Park Ave. W. Chatham, Ontario (519) 354-2840

Windsor, Ontario Canada

Circle 28 on inquiry card.