Physics 231 – Lab 2 **Names:** _____
Measuring and Modeling 1-D Motion _____
_____

There are two parts to this lab. In part A, you will experiment with 1-D motion, using a cart and the LabPro and LoggerPro data acquisition and analysis tools. In part B, you will write a VPython simulation of the fan-cart's motion.

PART A: Measuring a Cart's Motion

(Equipment: Track, level, Motion Detector and bracket for under-track attachment, Force Probe,{motion detector and force probe should have same colored stars to be compatible with each other}, dynamics track bracket, cart, magnetic fan attachment, LabPro, Computer, scale, whiteboard)

Objectives

In this section of the lab you will learn to:

- Use a computer interface to collect and display data.
- Describe one-dimensional motion.
- Relate the impulse on an object to the change in its momentum.

Background

The motion sensor detects the position of an object the same way a bat does. It emits pulses of ultrasound (too high pitched for human ears to detect), and measures the time required for a pulse to travel to an object, be reflected by the object, and return to the sensor. Using the speed of a sound wave in air (about 330 meters per second), the computer program calculates the distance between the object and the sensor. (The ticking sound you hear is not the ultrasound, but a low-pitched sound that lets you know the sensor is working.) The sensor isn't able to measure distances less than about 20 cm.

The motion sensor does *not* measure velocity directly. The program calculates v_x by taking position measurements at two different times, and calculating $\Delta x/\Delta t$ from these measurements. Therefore, it is calculating an average velocity, but it is a very good approximation to the instantaneous velocity since the time between the measurements is very short.

I. Measuring Position and Velocity in One-Dimension

A. Setup

In this part of the lab, you will use a motion sensor to measure a cart's position and velocity. Much of this 'setup' will already be done for you, but check it over to make sure.

- Make sure that the track is level.
- Attach the motion sensor on the end of the track. Connect the cable from the motion sensor to port labeled "DIG/SONIC 1".
- At the opposite end of the track, attach the "dynamics track bracket" with the force probe mounted beneath and one of the metal hoop-springs replacing the hook at the end of the force probe. Don't bother plugging the force probe to LabPro yet.
- On the computer's desktop, you'll find a "Physics Experiments" folder, and inside that you'll find a "PHYS231" folder. Open the file "Motion.cmbl" in the directory "PHYS231". You should see an empty graph with the vertical axis labeled "Position", which should really be called "x" because only one component of the position is measured.
- Place the cart on the track. **Do not let the cart hit the motion sensor during the experiments!**

B. 1-D motion with No Net Force

In this section, you'll look at the motion of a cart with (nearly) no external forces acting on it.

- Start with the cart about 20 cm from the motion sensor.
- Press the "Collect" button and give the cart a gentle push away from the motion sensor after you hear clicking (it takes the motion sensor a moment to start working). The cart should reach the

other end of the track in 2 to 3 seconds. Be sure that you hand is not between the cart and the motion sensor because it measures the distance to the nearest object.

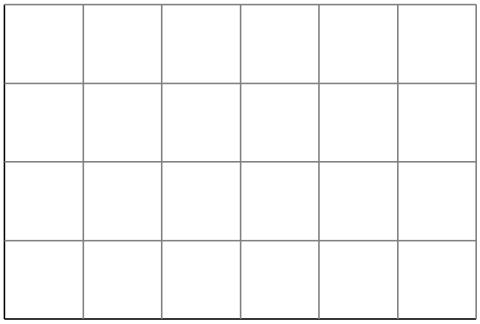
- **You want the position vs. time plot to look like a straight line with a slope; if it looks rather curved, try again with a slightly stronger push**.

- You may roll the cart too slowly or too quickly. Also, the graph of the position can sometimes look a bit rough when you know that it should be smooth. If either of those things happens, repeat the previous step.

- Copy over the graph of the *x* component of the position below, but only for the part of the motion where you are <u>not</u> touching the cart. Be sure to put scales on both axes.

1pt

x (m)



t (s)

- After clicking on the graph window, select "Examine" under the "Analyze" menu. This will allow you to get numbers from the graph by moving the cursor around. You should do this whenever you want to get data from a graph in LoggerPro.

- Take some data from the graph and use it to calculate the *x* component of the average velocity, $v_{avg,x}$, for the part of the motion on the graph above. Show all of your work.

1pt

- Now that you've done it by hand once, there's an easier way: highlight the section of the data you want to analyze (when nothing's touching the cart) by depressing the mouse button and dragging across the segment of the plot. Next, click on the linear-fit at the top of the screen, (has a diagonal line and an "R=" in it.) By this method, what is the slope of the *x* vs. *t* line, i.e., what's $v_{avg,x}$?

$v_{avg,x}$ = _____

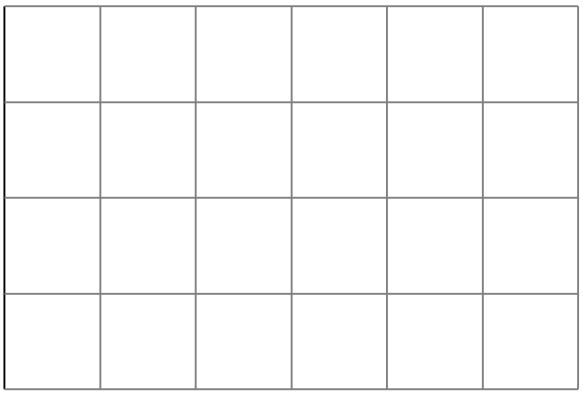
1pt

- Based on this, use a dashed line to sketch what you expect for the *x* component of the velocity on the graph below (just for the time when nothing's touching the cart.) Be sure to put scales on both axes.

Key
Line based on *x* vs. *t* plot: - - - - -
Computer's *v* vs. *t* plot: ————

2pts

v_x (m/s)



t (s)

- Change the computer graph to plot velocity vs. time rather than position vs. time. To do this, click on the label "Position" and select "Velocity" from the list that appears. The vertical axis should really be "v_x" because only one component of the velocity is measured. Add the measured v_x to the graph above, but only for the part of the motion where you are not touching the cart.

Question: In what ways did the measured graph differ from your prediction, and why?

1pt

C. 1-D Motion with Net Force

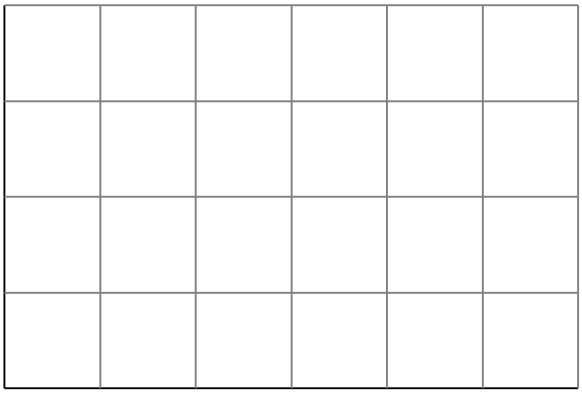
In this section, you'll look at the motion of a cart with a constant net external forces acting on it. You'll give the cart an initial push away from the motion sensor, but the attached fan will apply a force to eventually bring it back toward the sensor.

- Attach the fan to the cart.
    - Magnets are imbedded in one end of the cart and both faces of the fan attachment's base, so they'll snap together with the fan facing either way.
- Switch the graph on the computer back to "Position" instead of "Velocity".
- Start with the cart about 20 cm from the motion sensor and the fan facing as to drive the cart *towards* the sensor.

- Switch the fan to the "Low" setting (switch pushed half-way across.)
- Press the "Collect" button and give the cart a push away from the motion sensor after you hear clicking. The cart should nearly reach the far end of the track before heading back toward the sensor.
- If cart hits the hoop-spring at the far end or if the graph is particularly rough, repeat the experiment.
- Graph the *x* component of the position below, but only for the part of the motion where you are not touching the cart. Be sure to put scales on both axes.

1pt

x (m)



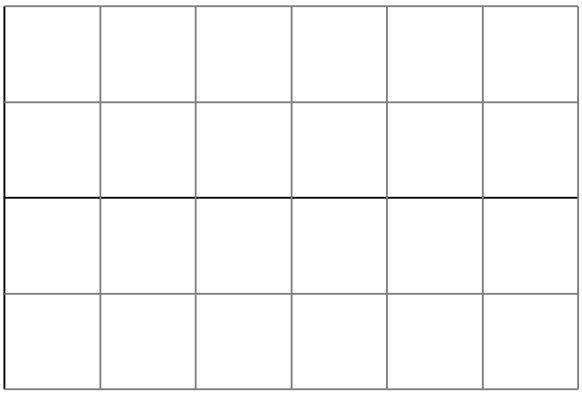
t (s)

- From the position vs. time plot, determine the velocity vs. time plot by selecting four short stretches over which to find the average slope (you can do that by hand, or you can just highlight and ask the computer to do a linear fit to each short region), and use those slope values to fill in the plot below. Represent the curve with a dashed line. Try to get the shape approximately correct. Be sure to put scales on both axes.

2pts

v_x (m/s)

0



Key
Line based on *x* vs. *t* plot: - - - - -
Computer's *v* vs. *t* plot: ———

t (s)

- On the computer graph, click on the label "Position". Select "Velocity" from the list that appears. Add the measured v_x to the graph above, but only for the part of the motion where you are not touching the cart.

Question: In what ways did the measured graph differ from your prediction, and why?

II. Impulse and Momentum Change

In this section, you'll look at both the cart's motion and the force exerted on it to see how the two are related.

A. Setup

For this part of the lab, you will use both a motion sensor and a force sensor.

- Remove the Fan from the cart.
- Connect the cord from the force probe to LabPro's port labeled "CH1".
- Place the cart on the track so that the magnets imbedded in its end face away from the force probe / toward the motion sensor.
- Open the file "MotionAndForce.cmbl" in the directory "PHYS231". You should see three graphs. The third one has a vertical axis labeled "Force", which should really be called "F_x". The force sensor is setup so that the force measured is positive when it is directed away from the motion sensor and negative when it is directed toward it.
- With nothing touching the force sensor's hook, "zero" it by clicking on the "Zero" button in LoggerPro and selecting just the force probe to zero.

B. Experiment

You will compare the impulse on the cart due to a spring and the cart's change in momentum.

- Weigh the cart. You'll need this value shortly.

1pt

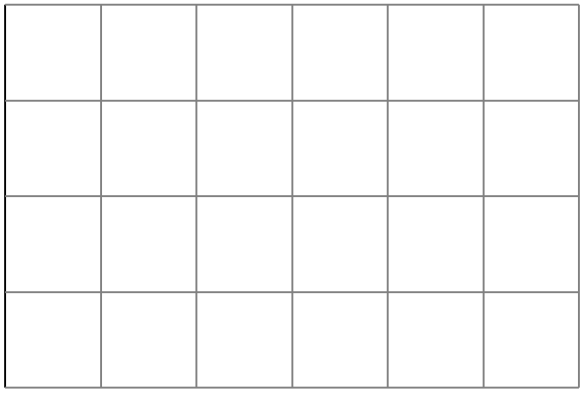
m_c = _____

- Press the "Collect" button and give the cart a push toward the force probe and hoop-spring <u>after</u> you hear clicking. If the spring completely compresses, try again with a gentler push. **Do not let the cart crash into the motion sensor on its rebound!**

- Graph v_x and F_x below, but only for the part of the motion where you are not touching the cart. Be sure to put scales on all of the axes and use the cursor to read values off the computer.

1pt

v_x (m/s)

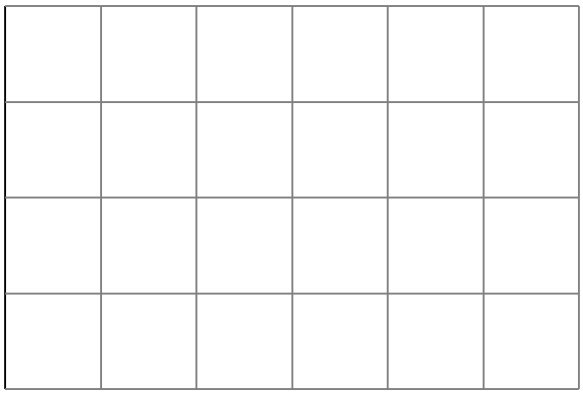


1pt

t (s)

F_x (N)

1pt



t (s)

- **Question:** Why should you analyze only the part of the motion when you are not touching the cart?

1pt

- Mark on both plots (*i* for "initial") just before the cart hit the spring and (*f* for "final") just after the cart left the spring. If you're having difficulty identifying these times, ask your instructor for help.

- Record below the times and the x-components of the velocities (including signs) at these two times – initial and final.

2pts

t_i = _____ $v_{x.i}$ = _____

t_f = _____ $v_{x.f}$ = _____

- **Question:** Considering the cart's mass and its velocities at these two times, what is the change in the cart's momentum, $\Delta\vec{p} = \vec{p}_f - \vec{p}_i$?

1pt

$$\Delta\vec{p} = \_\_\_\_\_$$

- On the Force vs. Time plot in LoggerPro, highlight the region from t_i to t_f and ask the program to integrate the area under the curve to find the Impulse exerted by the spring on the cart between those two times. The Integral function is found on the Analyze menu.

1pt

$$I_{xi\to f} = \int_{t_i}^{t_f} F_x(t)dt = \_\_\_\_\_$$

- **Question:** Theoretically, how would you expect an object' change in momentum and the applied impulse to compare?

1pt

- **Question:** How do the impulse and change of momentum in the x direction compare? (Calculate the percent difference.) If it's much greater than 10%, find and fix your mistake.

1pt

PART B: Modeling the Fan Cart's Motion with Loops

Objective: In this activity you will learn how to use loops to:

- Update the position of an object iteratively (repeatedly) to animate its motion
- Update the momentum and position of an object iteratively (repeatedly) to predict its motion

Background

The computer model will be based on the following:

Position Update Formula: $\vec{r}_f = \vec{r}_i + \vec{v}_{avg}\Delta t$

Momentum Update Formula: $\vec{p}_f = \vec{p}_i + \vec{F}_{net}\Delta t$

A computer program consists of a sequence of instructions.
The computer carries out the instructions one by one, in the order in which they appear, and stops when it reaches the end.
Each instruction must be entered exactly correctly (as if it were an instruction to your calculator).
If the computer encounters an error in an instruction (such as a typing error), it will stop running and print a red error message.

A typical program will have four sections:

(1) Setup statements
(2) Definitions of constants (if needed)
(3) Creation of objects and specification of initial conditions
(4) Calculations to predict motion or move objects (done repetitively in a "loop")

It is helpful to label these sections with comments by using lines start with the pound sign (`#`).

Overview

1. **Making Things**: First, you'll write the part of the program that:
   a. Creates 3D objects to represent the cart, the track, etc.
   b. Gives them initial positions and momenta
   c. Creates numerical values for constants we might need
2. **Moving Things**: Next, you'll use a "while loop" to tell the computer to increment the position of the objects over and over again, making them move across the screen screen. These lines tell the computer:
   a. How to calculate the net force on the objects
   b. How to calculate the new momentum of each object, using the net force and the momentum principle.
   c. How to find the new positions of the objects, using the momenta.

III. Setup Statements

- Using "IDLE for VPython" (on the desktop), create a new file and save it as "fancart.py".
- Enter the two setup statements in the IDLE editor window.

Remember that every VPython program begins with the following setup statements:

```
from __future__ import division, print_function
from visual import *
```

The second statement is typed "from space underscore underscore future underscore underscore space import space *".

You should also put you and your labmates' names in a comment at the beginning (`# John Doe,`)

IV. Defining Constants

Following the setup section of the program, you will usually define physics constants that will be used throughout the program. There is nothing to put in this section yet, but in anticipation of doing so later,

- Put a comment (`# Constants`) and a blank line to mark a place for putting constants.

V. Creating Objects and Setting Initial Conditions

A. Creating Objects

You will make to boxes to represent a track and a fan cart.

- Add a comment (`# Objects`) to mark the section where you'll define objects.
- Create a *box* object to represent the track with the following line:

```
track = box(pos=vector(0,-0.05, 0), size=(1.0, 0.1, 0.10), color=color.white)
```

The `pos` attribute specifies the position of the *center* of the box. The `size` attribute gives the dimensions of the box in the x, y, and z directions. When you first run the program, the coordinate system has the positive x direction to the right, the positive y direction pointing up, and the positive z direction coming out of the screen toward you. In this case, the "top" of the track (in the y direction) is at $y = 0$, because the center is at $y = 0.05$ and it extends $0.1/2 = 0.05$ above and below that.

- Run the program (by pressing F5) to see the track.
- Create a second *box* object to represent the fan cart. Give it a position of (0, 0.05, 0) and a size of (0.1, 0.04, 0.06). Make it some color other than white.
- Run the program to see the cart and the track. The cart should be "floating" above the track.
- Reposition the cart so that its left end is aligned with the end of the track and its bottom is just touching the track

WA

3pts

Question 1: What is the position of the cart? (log on to WebAssign and answer in Lab 2 assignment.)

B. Setting Initial Conditions

You have already set the cart's initial position, but you also need to set its initial momentum. Since the momentum (for $v << c$) is $\vec{p} = m\vec{v}$, you will need to tell the computer the cart's mass and initial velocity. There are no "built in" attributes for these quantities, but you can create them.

- On a new line type the following:

```
cart.m = 0.8
```

This creates a new scalar variable `cart.m` which now stand for the value 0.8, the mass of the cart in kilograms.

- On a new line type the following:

```
cart.p = cart.m * vector(0.5, 0, 0)
```

This creates a new vector variable `cart.p`, which is the momentum of the cart. It is assigned to the initial value of (0.8 kg)<0.5, 0, 0>m/s = <0.4, 0, 0> kg·m/s.

- Add the following line to print the momentum in the text output window:

```
print cart.p
```

- Run the program and look at the output. Afterwards, delete or comment out the print statement.

Note: We could have simply called the momentum `p` and the mass `m`, instead of `cart.p` and `cart.m`.

However, it is useful to create attributes like mass or momentum for objects because it makes it easy to distinguish the masses and momenta of different objects.

To make the cart move we will use the position update equation $\vec{r}_f = \vec{r}_i + \vec{v}\Delta t$ repeatedly in a "loop". We need to define a variable `deltat` to stand for the time step Δt, and a variable `t` to stand for the total time elapsed since the motion started. Here we will use the value $\Delta t = 0.01$ s.

- Add the following lines:

```
deltat = 0.01
t = 0.0

# Calculations
```

VI. Loops and Repeated Calculations

You're going to write a loop in the program so it can repeatedly calculate updated positions for the cart and accordingly move the block that represents the cart.

VI.I Basic Loops

But first, let's pause for a moment and get familiar with how to write a basic loop in Python. In a computer program a sequence of instructions that are to be repeated is called a loop. The kind of loop we will use in VPython starts with a "while" statement. Instructions inside the loop are indented. IDLE will indent automatically after you type a colon.

- Before writing your own loop, watch the following video to get an idea how loops are written and how they operate: **VPython Instructional Videos: 3. Beginning Loops** at http://www.youtube.com/VPythonVideos .

- Now, add the following simple loop to your program. Be sure to type a colon (:) at the end of the while statement.

```
while t < 0.2:
```

- Press the "Enter" key. Notice that the cursor is now indented on the next line. (If it's not indented, check to see if you typed the colon at the end of the "while" line. If not, go back and add the colon, then press "Enter" again.)

The "while" line tells the computer to repeatedly execute the next lines of code for as long as the given condition is true. The lines that will be repeated are the ones that are indented after the "while" statement. In this case, the loop will continue as long as the variable "`z`" is less than 10.

- On the next (indented) two lines, type:

```
print ('the time is now', t)
t = t + deltat
```

Note: In most programming languages the "=" symbol doesn't exactly mean "equals" (clearly, *t* can't "equal" *t* + deltat unless deltat is zero); rather, it represents an instruction for the computer to "assign" the value calculated on the right hand side to the variable named on the left hand side. In this case, it means that the first time through the loop, the computer adds the current value of `t`, which is 0.0, to deltat, which is 0.01, giving 0.01, and then assigns `t` this new value of 0.01. The next time through the loop, the computer again adds 0.01 to this new value of `t`, making `t` equal to 0.02, and so on.

- To show when the loop is done being executed, press *enter* and then *backspace* to un-indent the next line, then type:

```
print ('outside of loop')
```

All together, you should have

```
while t < 0.2:
   print 'the time is now', t
   t = t + deltat
print 'outside of the loop'
```

- Run the program

In the text output window, you should see a list of numbers from 0.01 to 0.2 in increments of 0.01. The first number, 0.01, is the value of `t` after the first time through the loop. Before each execution of the loop, the computer compares the current value of `z` to 0.2, and if it is less than 0.2, it executes the loop again. After the umpteenth time, the value of `t` is finally 0.2. When the computer goes back to the "while" statement for the next repetition, it finds the statement "`t<0.2`" is now false. Because the condition is false, the computer does not do any more executions of the loop, rather it skips to the first line after the loop which tells it to print "outside of loop."

WA 1pt

Question 2: What is the last value of the time printed? (answer in Lab 2 assignment in WebAssign.)

VI.II Loops and Animation

Now you've seen the basic operation of a loop and you've made use of it to update the time variable within your program. Along with updating time, we often want to update the *position* of an object, both what we calculate and what we display in the Graphics window, thus creating an animation. So we're going to add some code to update the cart's position.

- But first, watch this video to get a sense of how loops can be used to run animations: Video 4. Loops and Animation http://www.youtube.com/vpythonvideos .

A. Constant Momentum Motion

Suppose the cart is moving with constant momentum. Somebody or something gave the cart its initial momentum, but we're not concerned here with how it got that initial momentum. We'll model the cart's subsequent motion, after it acquired that initial momentum. You will use your iterative calculational "loop". Each time the program runs through this loop, it will do two things:

(1) Use the cart's current momentum (`cart.p`) to calculate the cart's new position (`cart.pos`).
(2) Increment the cumulative time `t` by `deltat`.

You know that the new position of an object after a time interval Δt is given by

$$\vec{r}_f = \vec{r}_i + \vec{v}_{avg}\Delta t \ ,$$

where $\vec{r}_f$ is the final position of the object, and $\vec{r}_i$ is its initial position. If the time interval Δt is very short, so the velocity doesn't change very much, we can use the initial or final velocity to approximate the average velocity. Since at low speed $\vec{p} \approx m\vec{v}$, or $\vec{v} \approx \vec{p}/m$, we can approximate

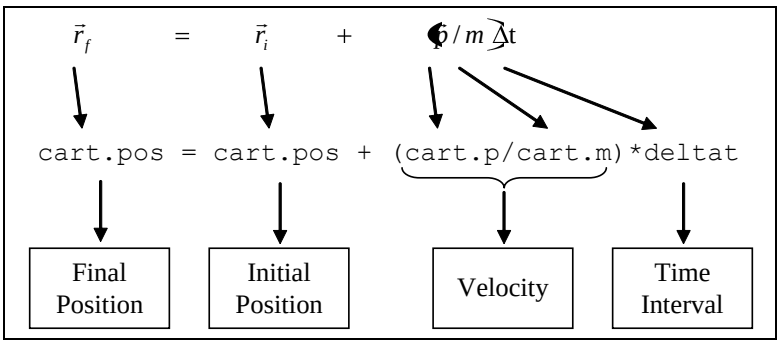
$$\vec{r}_f \approx \vec{r}_i + \vec{p}_i / m \, \Delta t \ .$$

We will use this equation to update the position of the cart in the program. First, we must write this equation in a way that VPython understands it.

- Delete or comment out the line inside your loop that prints the value of `t`.
- On the indented line right after the "while" statement (before the statement updating `t`), type the following:

```
cart.pos = cart.pos + (cart.p/cart.m)*deltat
```

Notice how this corresponds to the algebraic equation.



- Change the while statement so that the cart travels 2 meters (twice the track's length.)
- Run the program to see if it works. (You can add a "print" statement just outside the loop to check the final position of the cart.)

WA

1pt

Question 3: What is while statement after you changed it? (answer in Lab 2 assignment in WebAssign.) Remember, $\Delta x = v_{x.ave}\Delta t$.

The program is runs so rapidly that the entire motion occurs faster than we can see. We can slow down the animation by adding a "rate" statement.

- Add the following line inside your loop (indented):

```
rate(100)
```

- Run the program again.

Each time the rate command is reached, the computer pauses long enough to ensure the loop will take 1/100th of a second. Therefore, the computer will execute the loop at a *rate* of 100 times per second.

Note: The cart going beyond the edge of the track isn't a good simulation of what really happens, but it's what we told the computer to do. There are no "built-in" physical behaviors, like gravitational force, in VPython. For now, all we've done is tell the program to make the cart move in a straight line. If we wanted the cart to fall off the edge, we would have to enter statements into the program to tell the computer how to do this.

- Change your program so your cart starts at the *right end* of the track and moves toward the *left*. When you've succeeded, compare your program with that of another group.

Your running program should now have a model of a cart moving at constant velocity from right to left along the track.

B. Motion with Changing Momentum

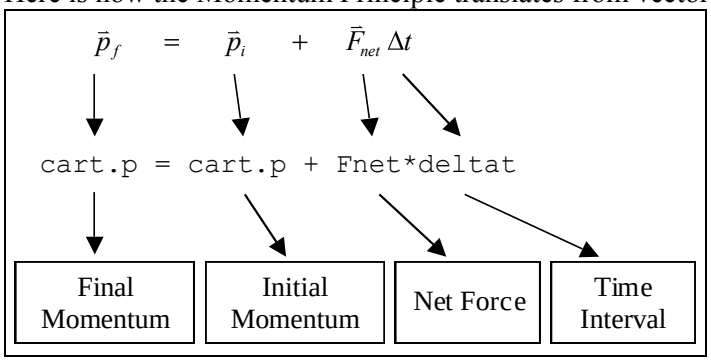
- Before writing the code to introduce a constant force, as a group, you should pause and think about what you'd expect a cart to do if a constant force is applied in the direction the cart is initially moving or in the opposite direction (recall what you saw with the real cart.)

As discussed in Chapter 2 of the textbook, an iterative prediction/model of motion can include the effects of forces that change the momentum of an object as follows:

Calculate the (vector) forces acting on the system.
Update the momentum of the system, using the Momentum Principle: $\vec{p}_f = \vec{p}_i + \vec{F}_{net}\Delta t$.
Update the position: $\vec{r}_f = \vec{r}_i + \vec{v}_{avg}\Delta t$.
Repeat.

where $\vec{F}_{net}$ is the net force. In this simple example, the force will be a constant.

Here is how the Momentum Principle translates from vector notation to VPython's syntax:



- Before the loop, create a new vector variable named `F_air` and assign a value of <0.4, 0, 0> N to it.
- Since this is the only (and constant) force we need to consider, just inside the loop add the line
  `Fnet = F_air`
  (In general, Fnet can result from multiple and changing forces, so it's good to get in the habit of assigning its value within the loop.)

- After that line calculating the force, write a line using the Momentum Principle to update the momentum.

This momentum-updating line should be above the existing line in which you use momentum to update the position. We'll discuss the subtle importance of this in next week's lab.

- Increase the time that the loop runs for 5 seconds. Run the program.

- Adjust the size of the force so that the cart just reaches the right end of the track before turning around.

WA
1pt

Question 4: How large does the force need to be? (answer in Lab 2 assignment in WebAssign.)

(you haven't yet met the analytical tools that can predict this, but you can adjust the force in your program's simulation and see what works).

VII. Additional Exercises

A. Two-Dimensional Motion

In the computer program, you can model behavior that can't easily be experimentally realized. For example, in your simulation there's no gravity (yet), so your cart is free to fly.

- Change the initial momentum so that it has a positive *y* component similar in size to the *x* component.
- Run the program, then change the initial momentum back.

B. Adding Gravity & Falling

If you include the Earth's gravitational pull down on the cart, you can "launch" the cart off the right end of its track (so it heads left, turns around, returns to the right edge, and falls off.) The gravitational force near the Earth is $\langle 0, -mg, 0 \rangle$, where g = +9.8 N/kg. For simplicity, you'll want to 'turn on' this force only after the cart goes past the end of the ramp, (more realistically, the balancing upward force of the track beneath the cart 'turns off' at this point, but it's the same effect.) While we're at it, 'turn off' the fan when the cart leaves the ramp, so the effect's like a car driving off a cliff. Here are a few steps to achieve this:

- Change the time interval to 0.005 s for better accuracy.
- Define the constant g after the setup statements.
- Replace the line that assigns Fnet its value with an "if…else…" statement just after the position is updated (inside the loop) to use the force of gravity when the cart leaves the ramp. The lines will look something like the following:

```
if cart.x < ____:
   Fnet = F_air
else:
   Fnet = vector(____)
```

Where you need to determine what goes in the blanks.

The first indented line is executed only if the condition in the "if" statement is *true*; the second indented line is executed only if the condition is *false*. You need to figure out what to put in the blanks. Be sure to account for the length of the cart.

- Run the program.

2pts

Bonus: Suppose that you want the cart to bounce when it reaches a floor at *y* = -1 m. Be sure to account for the height of the cart.

The *y* component of the momentum (refered to as `cart.p.y`) needs to be reversed.

- Add the following "if" statement just after the position is updated (fill in the missing piece):

```
if cart.y < ____:
   cart.p.y = -cart.p.y
```

- To make the animation look better add a floor so that the cart bounces off of something.
- To make the model more realistic, both components of the momentum should decrease with each bounce. Add the following line indented line to the "if" statement:

```
cart.p = 0.8 * cart.p
```

You can experiment with the factor used to reduce the cart's momentum for each bounce.

WA

11pts

Save your completed program as "fancart.py" and turn it in by uploading it through Lab 2 in WebAssign.

If you're not using your own laptop, email yourself a copy of fancart.py.

7pts **In WebAssign, answer the questions about the sample code presented there.**