Capture, Code, Compare: Integrating computational modeling with video analysis

W. Brian Lane, Ph.D.
@LetsCodePhysics

Capture, Code, Compare (CCC)

The Need for Sensemaking

CCC Structure

Example Activities

Successes & Challenges

The Need for Sensemaking

My students didn’t think their computer models actually represented the real world.

- “This is interesting, but it’s **just a model**.”
- “This code doesn’t **take everything into account**.”

If I ask these students **why** they think this, they don’t have a good reason!

- Do they think an intro class can’t be very accurate?
- Is this general distrust of science?
- Or do they simply **never compare** models and measurements?

Students need to be taught to **compare** models and the real world.

Capture, Code, Compare (CCC) Structure

I revamped my **intro mechanics** course to follow a three-stage process each week:

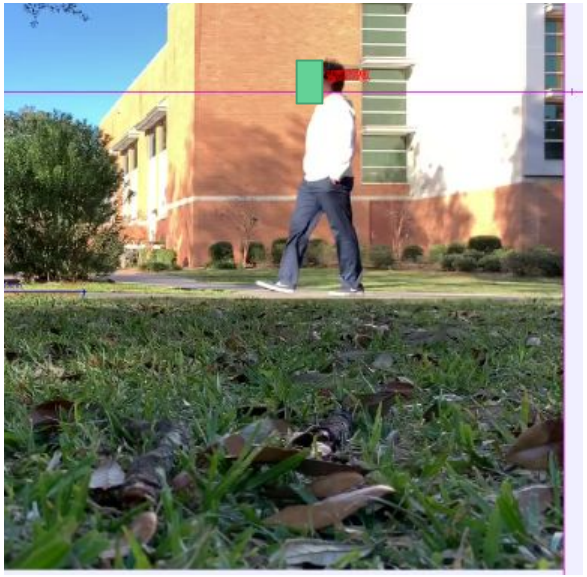
1. Students **capture** an experiment/activity/event in a video and study the motion with video analysis (Tracker).
2. Students develop/modify a **code** that reproduces the motion in a computer animation (VPython on Trinket).
3. Students **compare** the results (motion, graphs, important values).

Assessed using Letters Home.

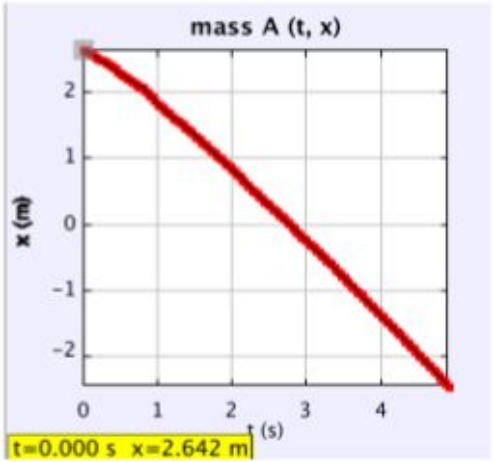
Easy to adapt to online learning!

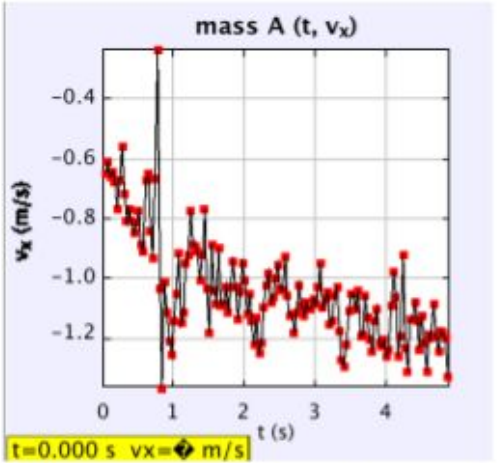
Let's look at some examples...

Example: Constant-Velocity Walk



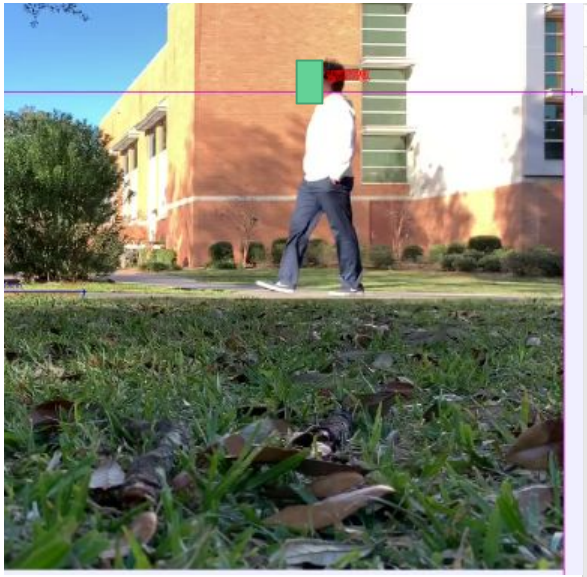
graph is sloping downwards because I was walking from right to left, which is technically the negative direction).





The graph on the right displays my velocity which looks very scattered but the increments on the graph are very tiny so my velocity is shown at about 1 meter per second but in the negative direction (meaning I was walking to the left).

Example: Constant-Velocity Walk



main.py

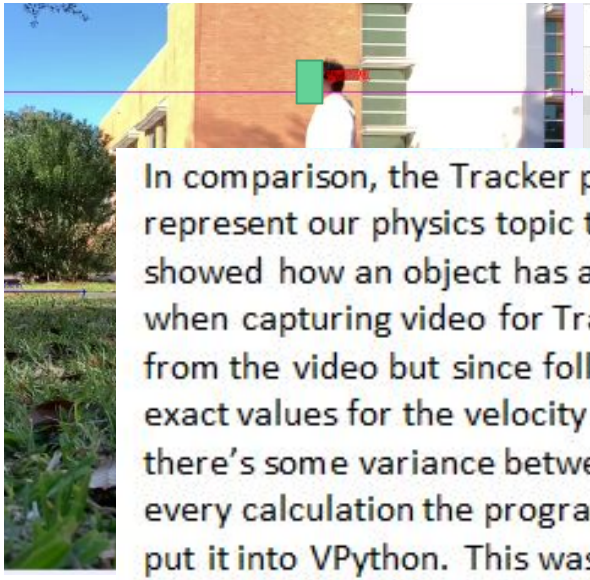
```
GlowScript 2.7 VPython

sean = sphere( pos=vector(0,0,0), radius=0.25, color=color.green,
sean.pos.x = 0 # scalar
velocity = 1.191 # scalar
dt = 0.1 # scalar
dx = velocity*dt # Step size. scalar
time = 0
# Make a loop.
while (sean.pos.x <= 5):
  rate(10) # Number of frames/loops per second. function
  sean.pos.x = sean.pos.x + dx # Dot indicates an attribute. scalar
  time = time + dt
print("End of program.")
print("total time =",time)
```

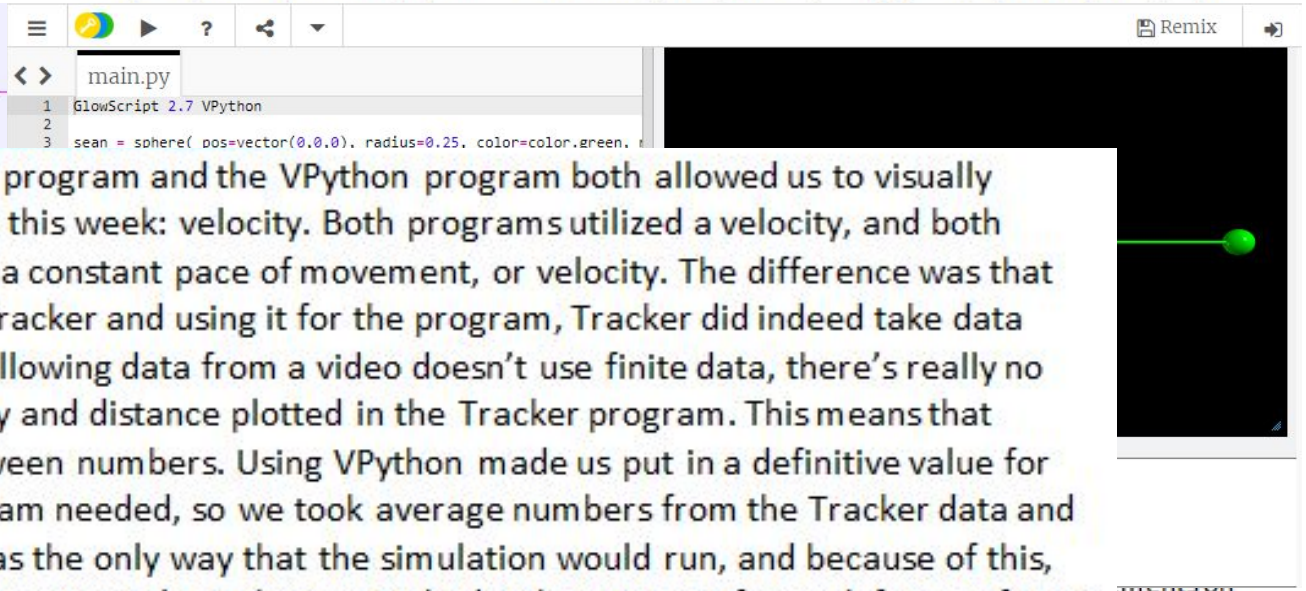
Remix

```
End of program.
total time = 4.2
```

the graph are very tiny so my velocity is shown at about 1 meter per second but in the negative direction (meaning I was walking to the left).

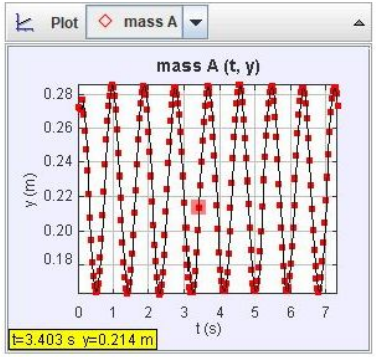
Example: Constant-Velocity Walk



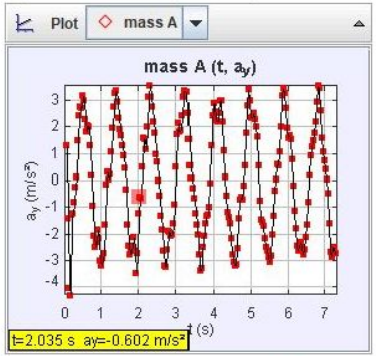


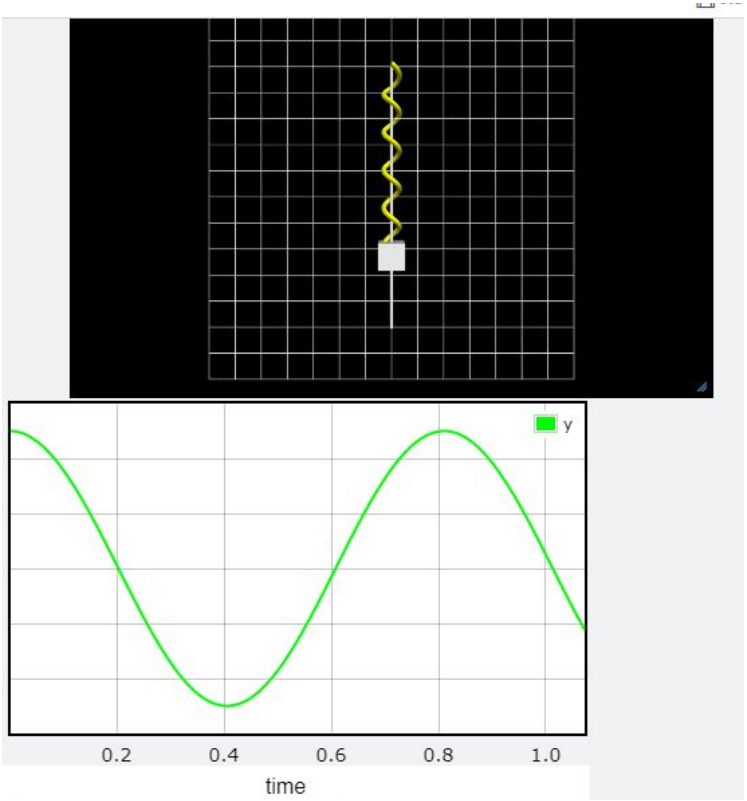
In comparison, the Tracker program and the VPython program both allowed us to visually represent our physics topic this week: velocity. Both programs utilized a velocity, and both showed how an object has a constant pace of movement, or velocity. The difference was that when capturing video for Tracker and using it for the program, Tracker did indeed take data from the video but since following data from a video doesn't use finite data, there's really no exact values for the velocity and distance plotted in the Tracker program. This means that there's some variance between numbers. Using VPython made us put in a definitive value for every calculation the program needed, so we took average numbers from the Tracker data and put it into VPython. This was the only way that the simulation would run, and because of this, VPython displayed a true constant velocity because it had to be constant for each frame of time in the simulation. Overall, VPython was constant in showing velocity because it used the same average velocity throughout the simulation but Tracker showed a large amount of variance in average velocity.

Example: Spring Oscillations

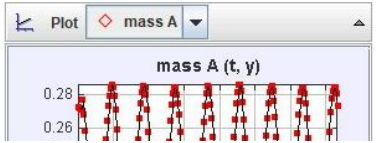


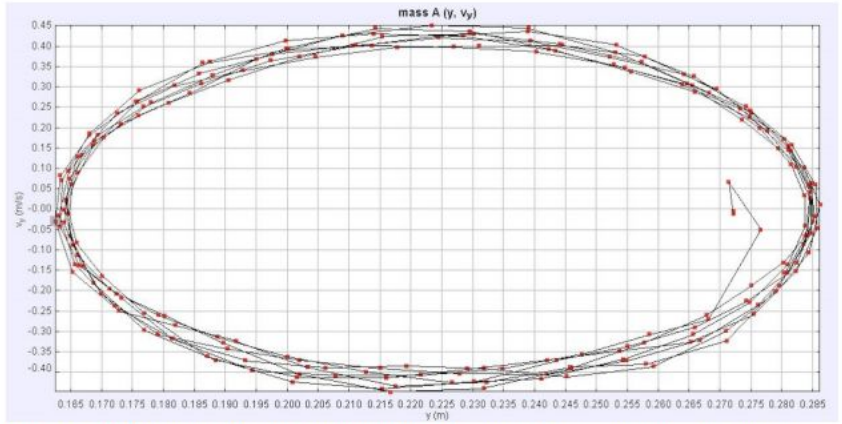
The vertical axis on the position graph represents the position of the object. The slope of a position graph represents the velocity of the object. So the value of the slope at a particular time represents the velocity of the object at that instant.

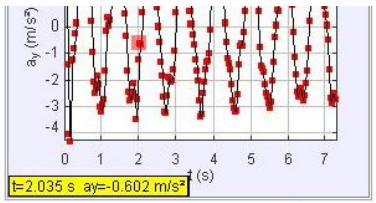


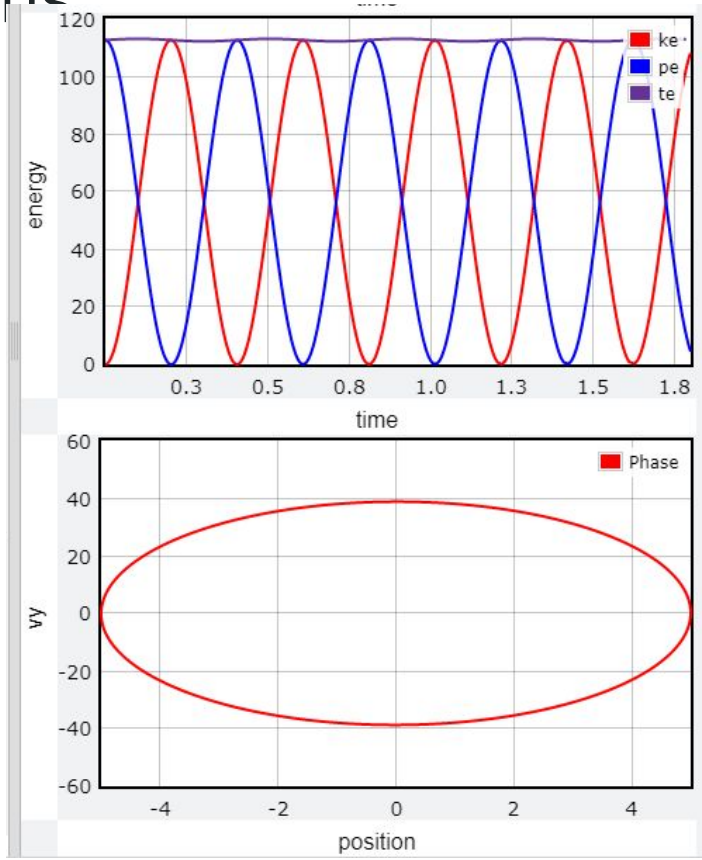


Example: Spring Oscillations

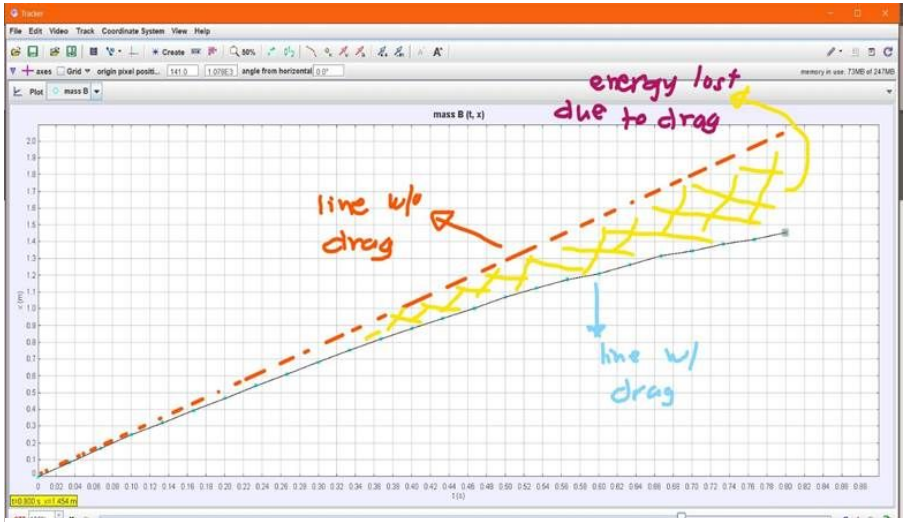


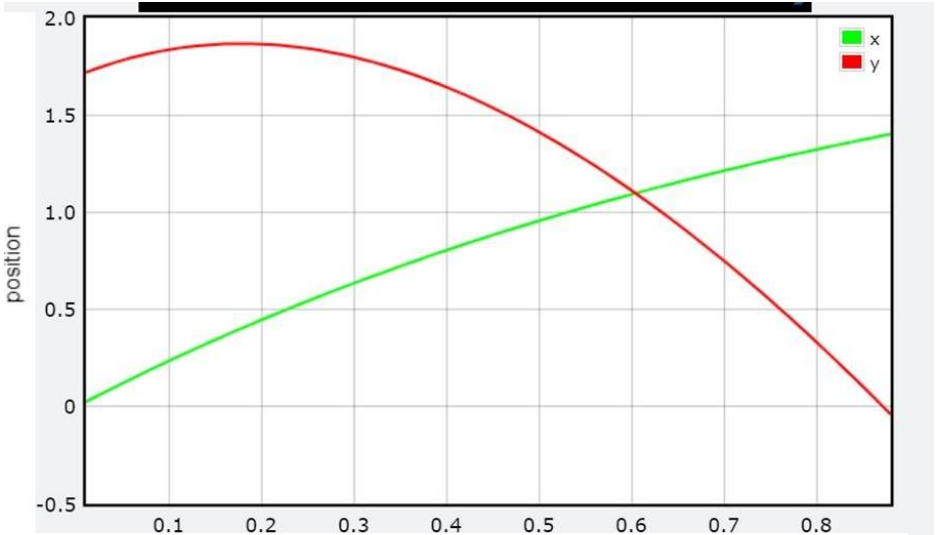






Example: Projectile Motion (WITH DRAG!)





With Tracker's data, we can use them to make another model in terms of code and animation, in order to test the consistency of our result. In the creation of this model, we used the VPython programming language which was coded in Trinket, an online programming software/website. In the code we need to input Tracker's data such initial velocities of the trajectory $x = 2.5\ m/s$ and $y = 2\ m/s$, initial position of the ball $x = 0\ meter$ and $y = 1.7\ meters$, mass of the ball $m = 0.0032\ kg$, and the drag coefficient or the rate of drag which was around ≈ 0.0035. Here's the link of the code if want to check it out: https://trinket.io/glowscript/1f91dc2b08

Successes & Challenges - THANK YOU!

Successes

- Most students (eventually) learn to compare.
- Most students learn concepts throughout activity.
- Open-ended, a lot more enjoyable, skills transferable between weeks.
- Easily adaptable to online.
- Different enough from high school physics to equalize the challenge level.

Challenges

- Some students still focus on surface features.
- Learning not always demonstrated in write-up ➔ Conceptual or writing shortcoming?
- How to extend this to intro E&M?

SCAN ME