Ryan Cao

Date: January 18th, 2017

Arrival time: 1:30 PM

Departure time: 3:45 PM

Today, I got back on it and completed my update() function for the body, which basically took all the values from the dictionaries that were storing the positions and orientations of every single body part, and reassigned them to the body parts so that the changes would be displayed during the animation. I came across one main problem, which was that for some reason, the dimensions of some body parts would be set to 1 inch after I was finished updating them, and so things like the torso would be 1-inch-wide, and the upper arm 1-inch-long, and so I went back and reassigned all the dimensions of the body parts, which fixed the problem but was redundant considering that the size of the body parts should never change. I got to work on creating motion methods to test out my body. First, using sine and cosine functions on the upper arms, I was able to get them to rotate in a circle, and by setting the orientations of the lower arms and hands the same as those of the upper arms, to make it seem as though the person was wind milling their arms. This was important for two reasons, firstly to prove that the dictionaries were working and that all the data transfer was being done correctly (if something had gone wrong here, the body would not have moved at all or something else visually odd would have happened), and secondly to make sure that the body parts were indeed all “attached”, that is, when the upper arm moved, that the lower arm and wrist would follow. I created a second motion function, this time to make the entire torso rotate back and forth, not too differently from someone swaying forward and backward, to make sure that everything else was indeed attached. Having done that, I went online to find more research papers on the actual motions (which I have now seen to be incredibly complicated) of an actual swimming stroke, butterfly, in particular. Unfortunately, I was unable to locate any actual equations, and thus attempted to derive my own equations from what I knew about the butterfly stroke and a couple of animations I was able to find. Although I am a little bit stuck here, seeing how complicated the math is behind the equations (I will elaborate on this in the next journal), I will ask Dr. Feng for help and hopefully we can get this geometric human figure to begin swimming soon!

Date: January 20th, 2016

Arrival time: 1:40 PM

Departure time: 4:30PM

I was again somewhat stuck today on the motion derivations today. I emailed Dr. Feng so that I would be able to meet with him in person and see what direction I must take next, but in the meantime I attempted to work through some functions to move the various body parts. For example, in the butterfly stroke, the main movement of the upper arm is around in a circle, going forward and backward, up and down (movement on the y-z plane only; x-component is relatively constant). Thus, for a continuous angle theta, the motion function I derived looked something like (0, -cos(theta), -sin(theta)) in order to allow the upper arm to rotate from (0, -1, 0) [hanging straight down] and back/up, to (0, 0, -1), then forward and up, to (0, 1, 0), then forward and down, to (0, 0, 1), and finally back down again to (0, -1, 0). Because butterfly also uses a dolphin kick, where the torso and legs essentially rock back and forth (oversimplified, of course), I made a function where the entire body would rock from standing up to leaning forward to leaning backward and back again. This one was similar to the one causing the upper arms to move in circular motion, but with the angle theta decreasing after a specified degree (pi / 3, or 60 degrees) so that the body would not rock too far, and increasing again after reaching (-pi / 3, or -60 degrees). However, with the lower arms, I am quite stuck. They essentially follow the orientation of the upper arms for the majority of the stroke, with the entire arm being relatively straight, but after a certain point, which I estimated to be around 1.25 \* pi (225 degrees), they make a complex maneuver, stroking in, then pushing out, and stroking in again, and finally pushing out very quickly. This was a problem, because in the current model, theta changes constantly, and I am not able to account for a sudden acceleration. I am thinking of transferring everything to be in terms of a time variable, and basing theta as a function of this time variable, so that theta’s rate of change can vary with time (the angular velocity/velocity in general can be faster during that last component of the stroke).

Date: January 24th, 2017

Arrival time: 1:35 PM

Departure time: 3:45 PM

Today, I was able to meet with Dr. Feng. After showing him an online side-view video of a model swimming butterfly, he let me know that rather than attempting to come up with my own explicitly defined function, such as sin(x) or cos(x), I was going to have to take video of either a real-life or animated model, plot certain points (defining markers at key joints, such as the wrist, knees, ankles, hips, etc.), and take “snapshots” of the locations of these points, so that I had a discrete position vs. time dataset, for which I could then fit a polynomial (ax^n + bx^n-1 … + yx^1 + zx^0) and so then this “function derivation”, rather than me coming up with a function, would simply be the computer determining (in a method not unlike linear regression, creating a best fit line through a set of data points, but with a polynomial function instead of a linear function) the position function for me. However, for now, he asked me to manually look at the video, frame by frame, and try to manually (just via observation) view the movements of the various body parts and plot them by hand. Finding this extremely difficult, I opened up my OpenCV skills from earlier this year and got to work. I found a function that would allow the computer to pick up certain “edges”, and wrote a program to play the video in grayscale frame by frame (so that I could advance frames with the press of a button instead of trying to scroll through the video via a trackpad), as well as draw circles where the computer thought there might be edges on the video display. However, although the function was pretty good, I found it unable to pick up on all the areas I needed it to, and will continue working to find a way to mark points, possibly with the mouse.

Date: January 26th, 2016

Arrival time: 1:35 PM

Departure time: 3:50PM

Unfortunately, the computer vision algorithms were not sensitive enough (and the video quality too poor) to catch all the points I wanted, and so I found a different method, which would allow me to track mouse movements and specifically pinpoint the x/y coordinates of where the mouse was clicking. Because the program defined (0, 0) as the upper left hand corner, with x increasing to the right and y increasing to the bottom (oh no. I just realized… I handled the y’s backwards. I will have to go back and write another program converting those points correctly), I did a transformation making (0, 0) the center of the screen and altered all the coordinates accordingly. Next, I altered my program so that the video would be played frame-by-frame, where for each frame I would click on the point I wanted marked, and press “c” on my keyboard to advance to the next frame. The program would record the position of my mouse clicks, and at the end of all the frames, I was able to directly copy and paste all of these coordinates into a .txt file, which I formatted slightly (adding titles and such to delimitate between the coordinates of different parts). I ran the program nine separate times, each time marking the position coordinates of the fingertips, wrist, elbow, shoulder, top of head, hip, knees, ankles, and toes for every frame. Afterwards, following Dr. Feng’s instructions, I began writing another program that would take input from the .txt file and calculate the near-instantaneous velocity at each position (by taking the differences in positions and dividing by the time interval (3/71 seconds) as derived from the total time of the video (3 seconds) divided by the number of frames [71]). I will finish this program next time, as well as correcting the minor error in my coordinate transformations (all I need to do is invert all the y-coordinates, multiplying them by -1), and hopefully be able to calculate the velocity and acceleration between all frames.

Date: January 30th, 2017

Arrival time: 1:35 PM

Departure time: 3:45 PM

Today I continued working on the program that would allow me to discretize the velocity and acceleration of the body parts as presented in the video. Again, I discovered that, because of the nature of this project, I was forced to type a bunch of code repeatedly and create lots of structures, which bothers me significantly, as I feel like there should be a better way to do this. In any case, after typing all of that out and successfully reading the values from the .txt file, Chris (from my lab) came over to talk to me about some parts of my project, and although he went through a couple of pretty high-end points, the most important of them was the idea of skipping frames to analyze (such as analyzing every other frame, every third frame, etc.) in the video analysis portion of the project, to conserve computing power while still keeping a reasonable level of accuracy. Since the video was already only 3 seconds and 72 frames, however, and since I had already taken coordinate readings of all the necessary joints for all frames, I decided to keep my current points; finally, I realized that in order to have a more precise time estimate (for better calculation of velocity and acceleration), I would have to know the exact duration of the video; I was able to find a shortcut, control + I, which told me the properties of the video (25 FPS [frames per second] playback), and multiplying that by the number of frames, I was able to find the exact duration of the video.

Date: February 1st, 2017

Arrival time: 1:50 PM

Departure time: 3:45 PM

Today I was finally able to finish my program! I made two 2-dimensional ArrayLists and converted my original 9 position ArrayLists into a single 2D ArrayList as well, organizing the indices of each as such:

(Imagine a large grid, each row of which is a list of items):

[i1, j1] [i1, j2] [i1, j3]…

[i2, j1] [i2, j2] [i2, j3]…

…

Where i is the row number/index and j is the column number/index. In the 2D grid (I will refer to the ArrayLists as grids, to make it less confusing, because functionally they can be thought of as so), each row, representing a list of position coordinates, corresponded to the positions of various body parts. By standardizing the order of the lists, (e.g. 1, head; 2, fingertips; 3, wrists; 4, elbows; 5, shoulders, etc.), I could have one large 2D grid rather than a bunch of separate 1D lists. I also created Velocity and Acceleration classes, with x and y components in each, and calculated finite-frame velocities and accelerations by taking the distances between two corresponding points (e.g. the positions of the fingertips at frames 1 and 2, adjacent frames) and dividing by the total time it took for them to “move” there (0.04 seconds per frame, as calculated last time). Having finished the motion things, I will show Dr. Feng as soon as possible and get to work learning how to analyze the super complex forces on the various body parts.

Date: February 7th, 2017

Arrival time: 1:35 PM

Departure time: 3:50 PM

Today, I began to consider the force calculations that would have to take place for objects moving through water. I started with the simplest consideration – a small box being dropped into a container of water. I quickly drew up a 3-D cubical container of water and placed the box just above the water, then got to work planning out how to calculate the forces on the box. I started with gravity, a constant -9.8 meters per second in the y-direction, then considered the drag forces on the box, consisting of pressure drag (drag caused by perpendicular movement against a fluid) and frictional drag (drag caused by parallel movement through a fluid, with the particles of the fluid brushing past the object’s surface creating such drag). With both of them consisting of the same general equation, Fd = Cd \* rho \* v2 \* A, with Fd being the drag force, Cd being the coefficient of drag, rho being the viscosity of the fluid, v2 being the parallel/perpendicular component of the object’s velocity relative to the fluid, for frictional and pressure drag, respectively, and A being the exposed surface area (parallel for frictional, perpendicular for pressure). Being unable to truly determine units and/or have precise measurements, I guesstimated the values for Cd and rho, but for v2 used the respective x/y/z component of the instantaneous velocity vector, and for A used the lengths/widths/heights of the correct faces multiplied together to determine the exposed surface area. Although I ran into quite a number of problems along the way (the first of which was my own silly venture into attempting to use calculus to calculate the instantaneous change in velocity – acceleration – and ended up with a meaningless formula), I was eventually able to successfully simulate the box’s interaction with still water, and verified by results by printing out the velocity of the box after awhile, and, seeing it constant (having reached terminal velocity in the y-direction as a result of gravity and the drag force cancelling out), had my thoughts confirmed in that way.

Date: February 9th, 2017

Arrival time: 1:35 PM

Departure time: 3:50 PM

Today I attempted to tackle the dynamics and kinematics of the simplest rotational scenario, which unfortunately turned out not to be very simple at all. I considered a rod, essentially a thin, elongated rectangular prism (I used length 10, width 2, and height 2 as my model), and attempted to calculate the forces caused by the fluid on it if it did not actually move, but just spun in place with a one-time initial angular velocity (kind of like how a pencil in water might behave if it were floating just beneath the surface right after someone spins it, with nothing else but the water interacting with it). Although the angular acceleration of the rod was constant throughout, the forces on each component of the rod were not, because the linear velocity on each component of the rod was not uniform (things further out move faster; things closer to the center move slower), and thus the drag force was not consistent throughout. Thinking this through, I decided to break the rod down into many small components, with each component being small enough that they could be seen to have a constant linear velocity (and thus uniform drag force) throughout. Additionally, since the drag force always opposes the direction of motion, I had to figure out what the direction of the velocity vector was, so that I could make the drag force’s direction equal to -1 \* direction of velocity vector; after a lot of spatial visualization with my pencil, I realized that I could take the cross product of the axis around which the rod was rotating (a vector) and the axis of the actual object (another vector, perpendicular to the first), and the resultant vector would be the directional vector of the linear velocity. With all this finished, I used an integral to break down the parts of the rod into infinitely small parts and to add the force components of each. I then added the previous few steps into code, trying to see if the forces would be correct, and although it was hard to tell based on the animation, it is likely that they did alright. Next time, I will be tackling the torque exerted on the rod, so that I can calculate the angular deceleration and finally make the thing spin/slow down.

Date: February 13th, 2017

Arrival time: 1:35 PM

Departure time: 3:50 PM

Today I was able to meet with Dr. Feng and update him on my progress; in discussing with him, however, I got quite lost on what my project was supposed to be and what component I was currently working on. Having done the whole video analysis program, I was asked to calculate the forces acting on each body part using the acceleration vectors I had calculated multiplied by the mass. However, this didn’t quite make sense to me because the points for which I had recorded the position and acceleration were the joints, the end points of body parts, and because the various parts are rotating, with each segment moving at a different linear velocity through the water (angular velocity is the same for rotation but the further out you move from the center of rotation, the faster the linear velocity is), and since the drag force depends on the square of the velocity, I thought that the drag force would be different for every single point on the body portion which was rotating, and thus it didn’t make any sense to just use a single F = ma equation for the entire body part. Additionally, I was asked to calculate the net torque on each part, which also didn’t make a bunch of sense, as I had no idea what the points of rotation might be for something such as the forearm, which is attached to the upper arm but also the shoulders, and thus depending on the definition of the axis of rotation, the torque calculated would be radically different. Also, none of this was accounting for translational plus rotational velocity, which would be combined to produce the total linear velocity and thus movement and thus drag force and torques. Hopefully I will be able to get this clarified soon.

Date: February 15th, 2017

Arrival time: 1:30 PM

Departure time: 3:45 PM

Today I went and talked to Dr. Feng for a little bit and asked him a question, but one of his other students came and he had him answer the question. Then I left and went to my dad’s office, so that he could help me understand the math going on behind the force calculations. First, as I mentioned before, we went over the (purely) translational and rotational drag forces, with the drag force formula Fd = 1/2 \* rho \* Cd \* v2 \* A accounting for frictional and pressure drags. The linear formula was quite basic, as the velocity vector was just decomposed into its “x” and “y” components, that is, the components parallel and perpendicular to the axes of the rod. The reason this is important is because any component of the velocity parallel to the axis will cause frictional drag, and thus that component can be used in the calculation of that portion of the drag force, whereas any component of the velocity perpendicular to the axis will cause pressure drag, and thus that component can be used to calculate that portion of the drag force. Additionally, to calculate the total force on a rotating body, I went through the steps to integrate the drag force as based on the linear velocity (radius \* angular velocity), and was able to find an explicit formula (one not involving an integral) to calculate the total force exerted on the body part spinning around. Next week I will be delving further into the math sections of everything.

Date: February 22nd, 2017

Arrival time: 1:40 PM

Departure time: 3:50 PM

I will begin my discussion of the dynamics of a simple rectangular rod moving about in the water. There are a lot of symbols used in the math here, as well as vectors and integrals. Since MS Word is a little bit limited in the types of things that can be easily expressed, I will be using the following notations: tv for the unit vector pointing in the tau direction (the direction of the axis of the block), nv for the unit vector pointing in the n direction (the direction of the other axis of the block), v0 for the translational velocity of the block, Vt for the velocity in the tau direction, Vn for the velocity in the n direction, Cd for the drag coefficient, rho for the water density, omega for the angular velocity, xi for a specific distance from the center where xi \* tv is a vector pointing to the section of choice from the block, theta for the current angle of the axis (from the horizontal, counterclockwise), L for the length of the rod, Ft for the force in the tau direction, Fn for the force in the n direction, and if there are any that I missed, I will discuss them when I get there. I am also going to assume that the only significant drag forces on the rod are those of pressure and frictional drags along the length of the rod (and thus the width of the rod is negligible for force calculations here). One last note: for vectors, I will be using \* to represent the dot product and x to represent the cross product. For scalars/scalars to vectors, \* will be simple multiplication.

First is the purely rotational motion: if the center of the block has a relative velocity of 0 to the water, meaning that only the rest of the block is spinning in place, the linear velocity at any point xi distance from the center may be calculated by v(xi) = xi \* omega \* nv (r \* theta \* directional vector in a sense), and thus the force on that single point is F = ½ \* Cd \* rho \* v(xi)2 \* d(xi), where d(xi) is a small segment of the length of the rod (corresponding to the cross-sectional portion of the “area” component of the drag force formula).

Date: February 13th, 2017

Arrival time: 1:30 PM

Departure time: 4:30 PM

Now we consider translational plus rotational motion. Firstly, because the rod is going to be moving in odd directions in the water (and not just parallel/perpendicularly to its surfaces and orientation), we must decompose all of its velocity vectors into ones parallel and perpendicular to its surfaces (and thus the tau vector, its axis direction), so that the force calculations are correct (imagine throwing a ball at a 45 degree angle; your throw will have both a vertical and horizontal component, only the vertical of which is affected by gravity; in this sense, we care about these components [which do add up to the entire vector] rather than the vector as a whole, because like in the analogy, the ball’s horizontal component of velocity will not be affected by gravity and we want to know about this). To do this, I projected the velocity vector onto the n and tau directions:

Vt = (Vo \* tv) \* tv (the first component is the dot product of the two, but since the tau vector is a unit vector, will give the correct length of the projection, which is then multiplied to the tau vector for the direction).

Vn = (Vo \* nv) \* nv (same logic as the previous statement).

Thus, the total velocity may be expressed as:

V(total) = Vt + Vn

V(total) = (Vo \* tv) \* tv + [(Vo \* nv) + omega \* xi] \* nv

(Note that the omega \* xi component of the Vn vector comes from the rotational velocity, which is always going to be in the direction of nv. The other component is the linear component of Vn, shown above.)

So, according to the drag equation, the total drag force in the tau direction may be expressed as: Ft = ½ \* Cd \* rho \* delta(xi) \* (Vt)2 for a single slice of the rod xi, and the total is the integral from (-L/2 to L/2) of that equation – note that an integral is just an infinite sum of infinitely small slices, and the reason we are going from –L/2 to L/2 is because the rotation point is in the middle of the rod, and thus the slices start from one end of the rod, half a length away, to the other end, also half an end away. After integration, we find the explicit formula L \* ½ Cd \* rho \* (Vt)2, which makes sense, as the force in the tau direction is affected only by linear velocity.

This appears to be quite enough math for today. The hard part has not yet been considered, but will be detailed in next week’s log.

Date: February 28th, 2017

Arrival time: 1:35 PM

Departure time: 3:50 PM

Let us now consider what happens when the rod is moving in water both with a linear velocity and a rotational velocity (spinning and moving along). Since we found last time that rotation only affects the velocity around the n direction (not tau), the velocity vector can be decomposed as such, keeping in mind that the linear velocity of a particle/slice of rod at xi distance from the center of the rod is omega \* xi:

Vtotal = (v0 \* tv) \* tv + [(v0 \* nv) + omega \* xi)] \* nv.

The first component is simply the projection of the overall linear velocity vector in the tau direction, whereas the second component is that projection in the n direction, plus the linear velocity in the n direction caused by the rod’s rotation.

Let’s first calculate the force exerted in the tau direction: (drag force)

On a single point/slice, the force is, as from the drag equation,

Ftau = ½ \* Cd \* rho \* delta(xi) \* (Vtau)2. (Delta(xi) is the length of a small segment of rod, which we are using for A, area, in the original equation)

Thus, if we integrate over the entire length of rod,

Ftau (total) = Integral from –L/2 to L/2 of ½ \* Cd \* rho \* (Vtau)2 d(xi), which gives us the explicit formula ½ \* L \* Cd \* rho \* (Vtau)2.

I apologize for the notation. It’s exceedingly hard to tell what’s going on based on what I just typed, especially without being able to see the integral sign and limits of integration.

Now for the force exerted in the n direction:

Substituting Vn + omega \* xi for v in the original equation, we have:

Fn = ½ \* Cd \* rho \* delta(xi) \* (Vn + omega \* xi)2

There is one thing we have to take into account, however. Since the rod is rotating, that means that it is possible that some parts of the rod are going one way, and some parts are going the other way (imagine a pencil lying on the table. Spin it counterclockwise slowly without otherwise moving it, and you will see how the right half of the pencil moves up and to the left while the left half moves down and to the right. These different linear velocities will produce different drag forces, and thus the direction matters a lot). Since the direction of the force is always opposite to that of the velocity, we want to know where the “pivot” of rotation is, that is, the point on the rod where the linear velocity is zero, if any (this is likely the point where you were pinching your fingers, spinning the pencil – your fingers, and that part of the pencil, likely did not actually move during rotation), because we know that underneath that, the rod will be moving one way and over that, the rod will be moving a different way. Thus, we must set the linear n component of velocity equal to 0: omega \* xi + Vn = 0, and solving for xi, the point of rotation, we have xi = -Vn / omega.

This tells us a few things. If xi is less than –L/2, the entire rod is moving in the same direction in terms of n, and if xi is greater than L/2, it’s the same thing, but in the opposite direction. If xi is between the two, then the rod is truly “spinning”, with some parts moving in one direction and some moving the other way.

I understand that I am taking quite a lot of time to explain this, but I have also been doing further maths and implementing the entire thing into VPython, so that if I input any initial velocity and angular velocity values for a rod submerged in water (I can also add gravity if needed), the rod will behave exactly as if it were a real rod submerged in that way. I will finish explaining the formulae for these three situations that I just detailed above next time, and will also get to how I am programming the motion/animation/calculations.

Date: March 2nd, 2017

I talked to Dr. Feng, and he said it was ok for me to continue working on my own; I am currently at BPA state.

Date: March 6th, 2017

Arrival time: 1:40 PM

Departure time: 5:00 PM

So to continue the rotation discussions, two integrals are needed for movements where the rotation movement is such that the point of zero linear velocity is somewhere on the rod i.e. the “overall” rotation point is on the rod, and thus part of the rod is moving by spinning one way and the other part is moving by spinning the other way. Assuming the rod is moving in a counterclockwise direction, the part below the rotation point is generating a net positive force in the n direction, whereas the part above the rotation point is generating a net negative force in the n direction. Thus, the two integrals must be subtracted from each other –

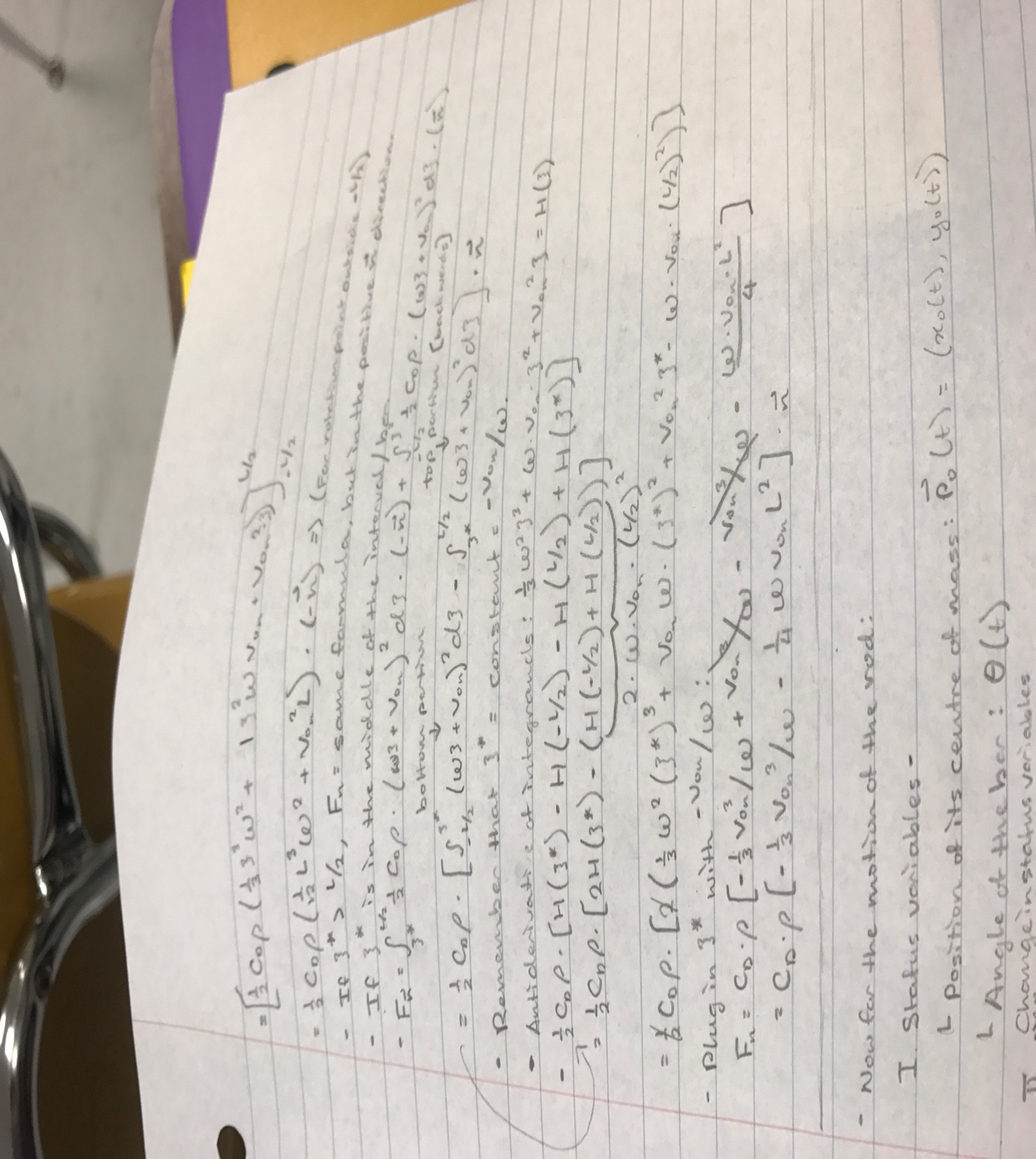
Integral from –L/2 to xi-star of ½ \* Cd \* rho \* xi \* (Von + omega \* xi) \* d(xi) – Integral from xi-star to L/2 of ½ \* Cd \* rho \* xi \* (Von + omega \* xi) \* d(xi)

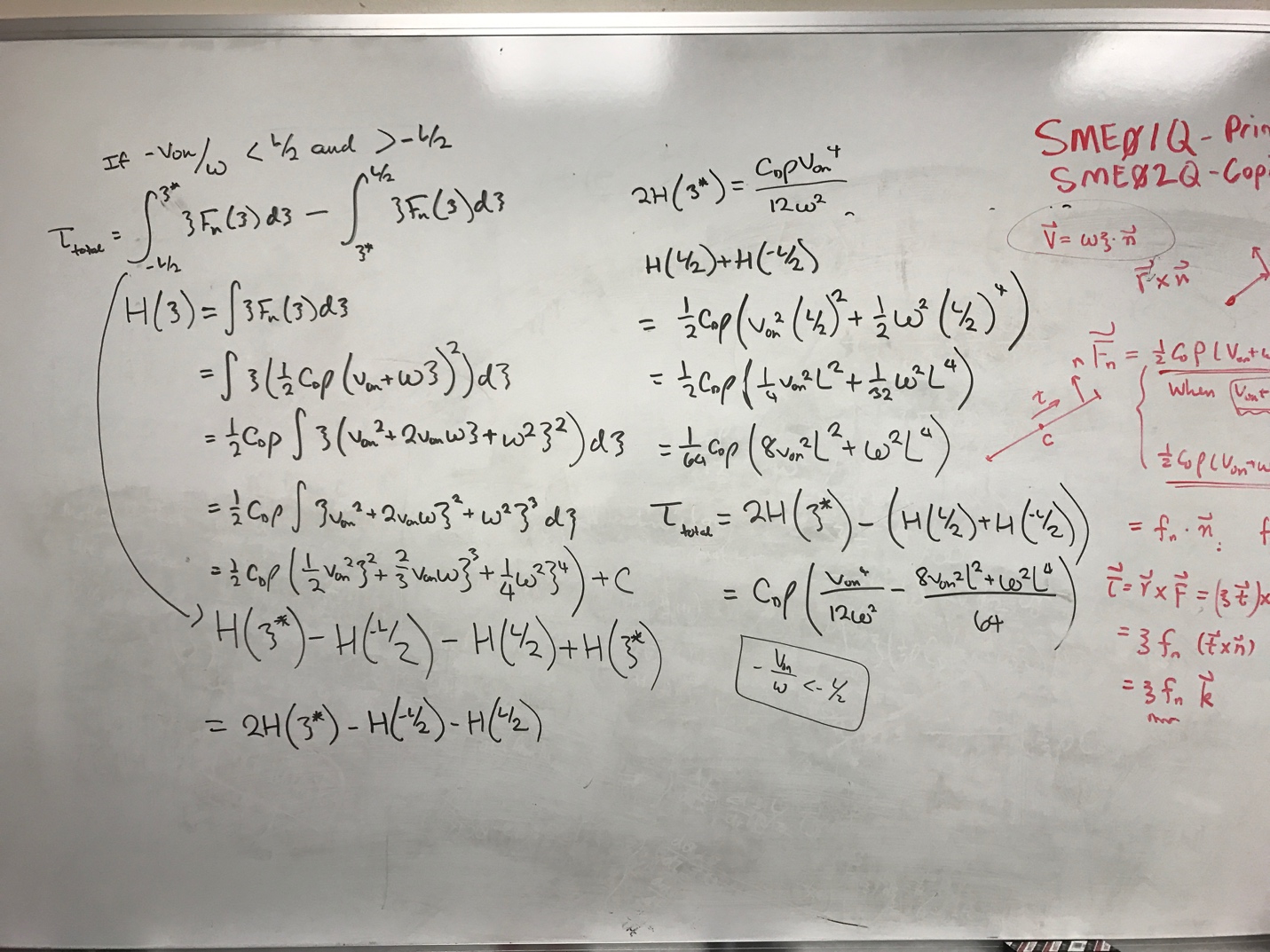
Where xi-star is the critical point, where the linear velocity is 0. Since the linear velocity is given by (Von + omega \* xi), we have then Von + omega \* xi = 0, and thus xi = -Von / omega. (We will substitute this in later)

Expanding the integrands (since they are the same for both scenarios), we have:

½ \* Cd \* rho \* (xi^3 \* omega ^ 2 + 2 \* xi \* omega \* Von + Von ^ 2) d(xi).

Since the symbols are too confusing on Microsoft Word, I am putting pictures of the rest of the net force and torque calculations below:





The final torque formula given above is on the second column above on the whiteboard, labeled Ttotal, and the final force formula given above is denoted by Fn, just above the line on the paper.

Date: March 8th, 2017

Arrival time: 1:40 PM

Departure time: 3:45 PM

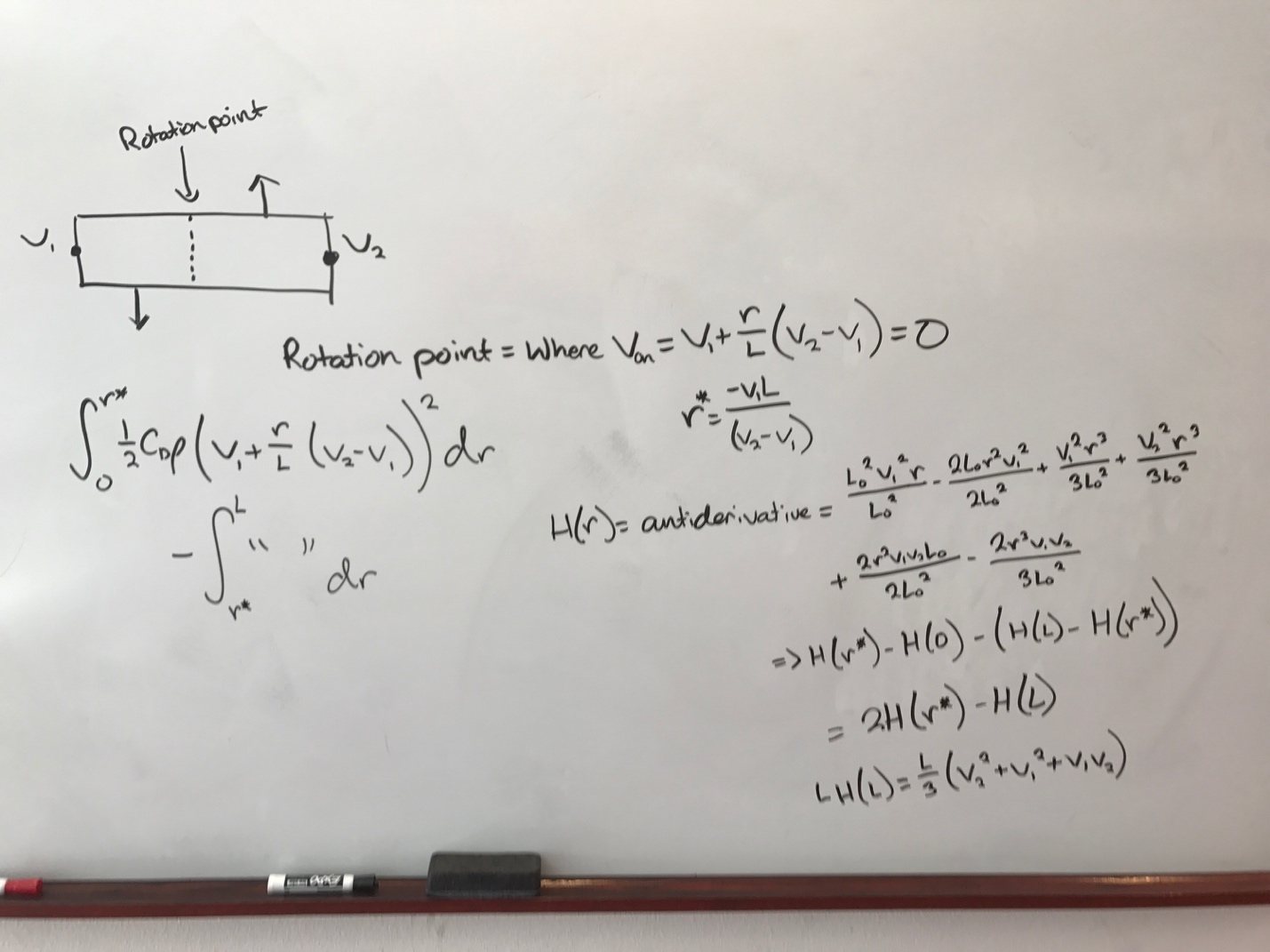
Today I hardcore debugged my program that I was working on to simulate the rod spinning and moving around in water. At first, there were many issues, including that I had become so caught up in the formulae that I had actually forgotten to put in some of the constants, such as Cd and rho into the methods. However, even after fixing that, I noticed that some of the time, the rod, rather than slowing down in the water, would spin faster and faster until its speed was too large for the computer to handle. And so I figured that I must’ve reversed the sign on the torque calculation/force calculations, since that the force was making the rod spin/move even faster would explain this dramatic acceleration. Additionally, I hadn’t considered that torque, just like the forces on the rod, was very much dependent on the rotation point, and since I had only derived the formula for a rotation point outside the rod, I had to derive the formula for “rotation” about a point on the rod. After putting that formula in, everything seemed to work ok, but in the animation it looked as though a little after the rod started slowing down, it “jumped” forward and I wasn’t sure what was going on. So to make sure the program wasn’t doing anything silly, I changed the function I was using to animate the rod to have a graphing option, and then plotted the linear and angular velocities, all of which seemed to decrease very nicely. The last thing I had to do, then, was to make sure that the program worked for any initial velocity, and not just positive ones (oops). However, as I entered a negative omega (initial angular velocity), the rod again started spinning out of control and I was not sure how to change the signs to fix it. Hopefully next time I will be able to work on that, and figure out the problem with the signs.

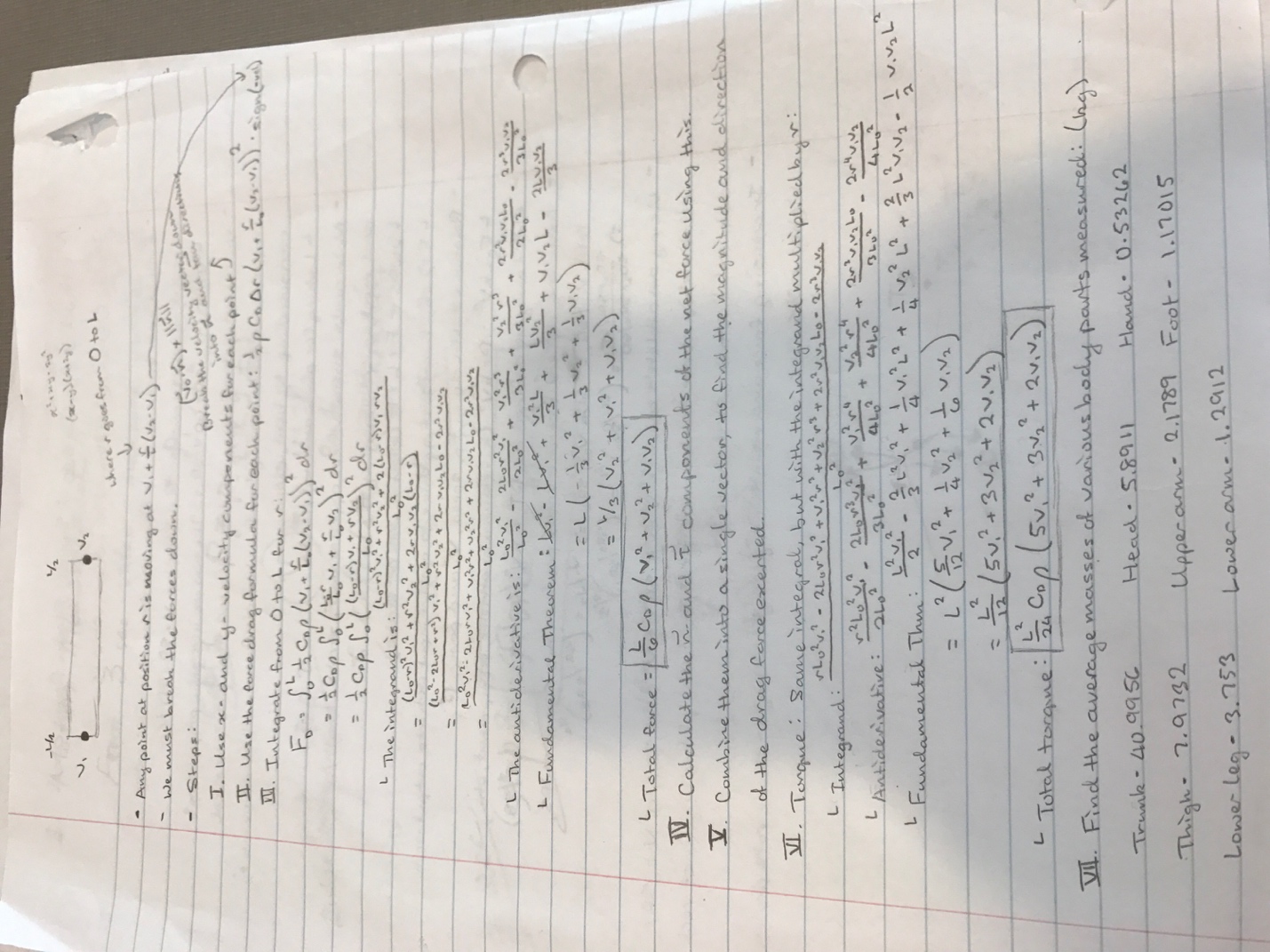
Date: March 21st, 2017

Arrival time: 1:50 PM

Departure time: 3:50 PM

With the spinning rod problem finished, today I considered again analyzing the video of the swimmer I had gotten online. The previous time, I had gotten extremely stuck on this, as I only had the positions of the joints over time and didn’t see how I was supposed to model the forces on entire body parts. However, I then realized that I could use an interpolation strategy – consider a rod with one end moving at velocity v1 and the other end moving at velocity v2. If we break this rod down into many tiny segments, with the position of each segment denoted by r, where r is the distance from one end and goes from 0 to L, with L being the length of the rod. In this scenario, the velocity of any point on the rod may be modeled by V = v1 + r/L \* (v2 – v1). (Think about it; when r = 0, that is, at one endpoint, the entire second term r/L \* (v2 – v1) is zero, and thus V = v1, which is what we expect, and at the other endpoint, r = L and thus r/L = 1, and the equation reduces to V = v1 + v2 – v1, which gives us V = v2, which is also what we expect.) Using this function for the velocity function, I again integrated over the entire rod and found an explicit formula for the force on the entire rod. Then, I integrated over the entire rod and found an explicit formula for the torque over the entire rod, with the reference point centered at one end. Finally, I looked up the body mass percent compositions for each body part, and multiplied by the average mass of an adult human to find the average mass of each part, and recorded them. (Calculations for force and torque shown below)



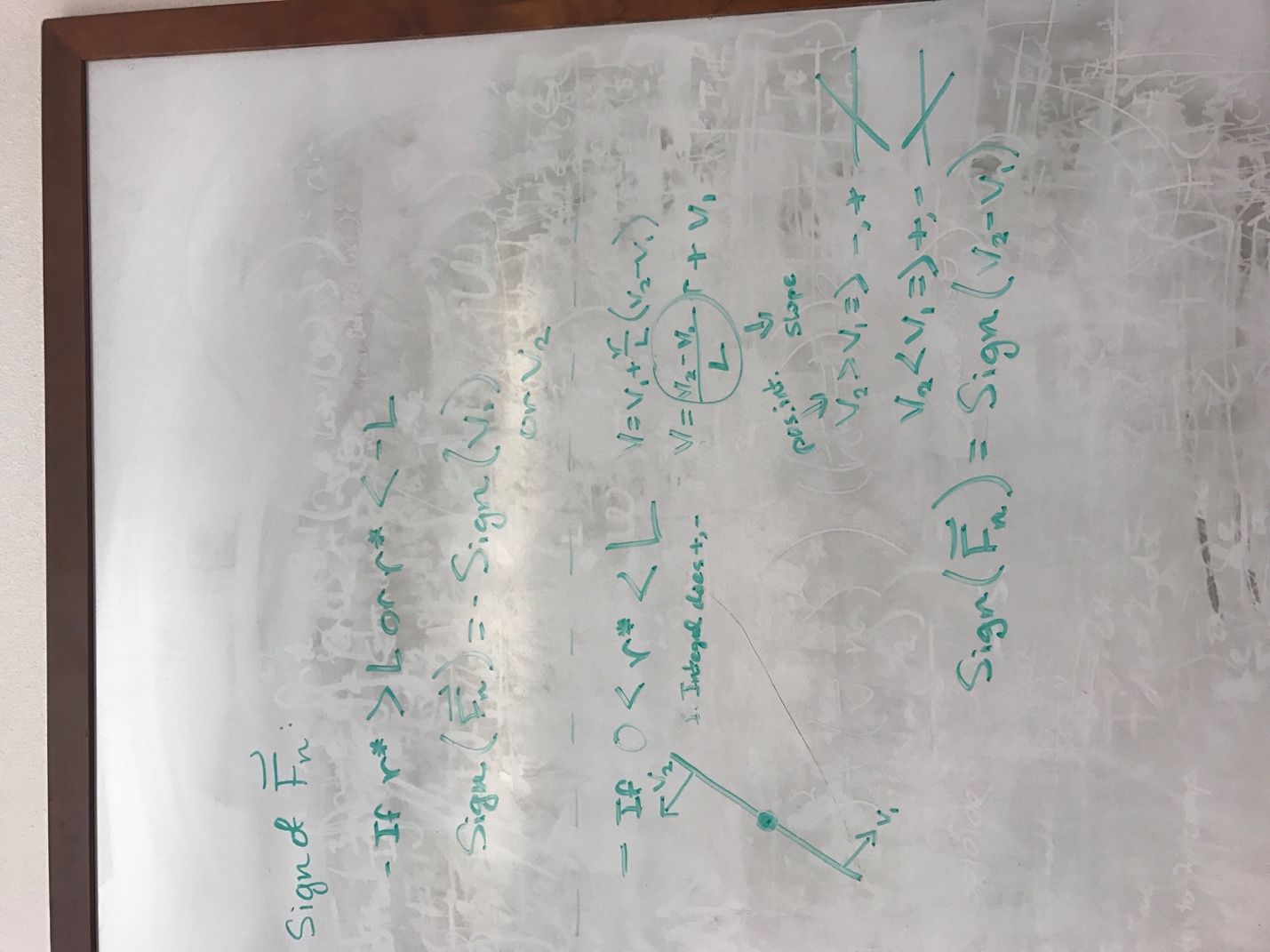


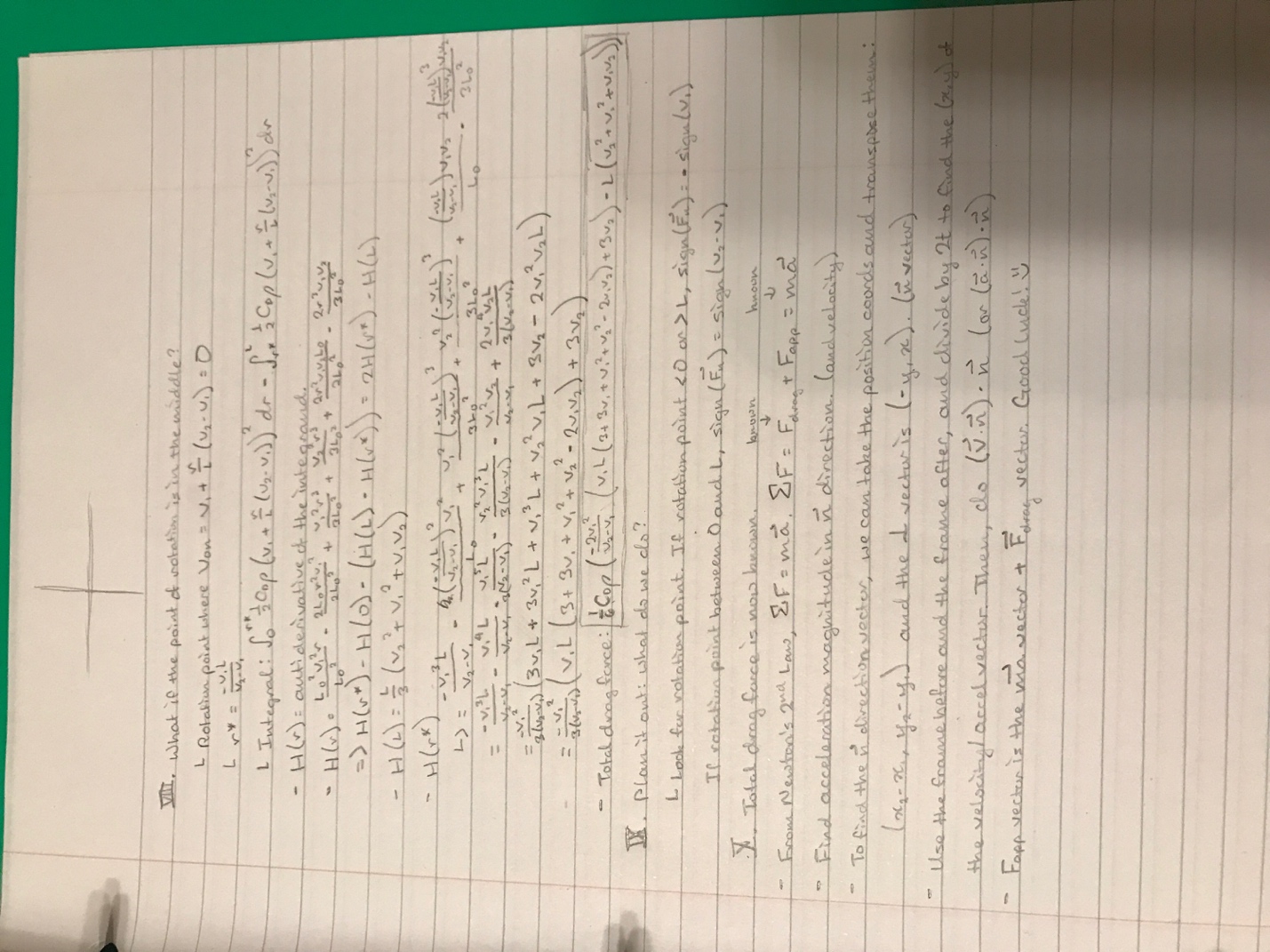
Date: March 23rd, 2017

Arrival time: 1:40 PM

Departure time: 3:45 PM

Today I completed the steps needed in planning out everything on paper, and began considering doing actual calculations via code. In planning, there were three main steps I needed to take: firstly, just as with the spinning rod problem, I had to determine how to find an explicit formula for the net force on the body part when that part was spinning with a rotation point somewhere along it (rather than totally above or below it). By setting the velocity formula I got last time V = v1 + r/L \* (v2 – v1) equal to 0 and then solving for r, I was able to derive where (point r along the rod) the rotation point was, and by breaking down the formula into two integrals (again, same as last time, but with different formulae), one going from 0 to r\* (our special r value where velocity = 0), and one going from r\* to L, and reducing everything, I found an explicit formula for the total force. The next thing I had to do was consider the sign of the force (positive/negative). While this seems trivial, it turns out that it was actually a lot more complicated to think through than I originally thought. However, by considering the velocity function, I finally found that for points of rotation outside of the rod, the sign of the drag force was –sign(v1) or –sign(v2), and for points of rotation inside the rod, the sign of the drag force was sign(v2 – v1).





Date: March 27th, 2017

Arrival time: 1:40 PM

Departure time: 3:50 PM

Today I continued implementing all of the things I had done into code. I first created a Vector class, to replace my “position”, “velocity”, and “acceleration” classes, so that Vector objects (essentially items) could replace everything. For the Vector class I added a dot product, which was needed for projecting my currently-measured velocities onto the n-direction (direction perpendicular as determined by the right-hand rule to the current orientation). Next, I added lists within the program storing the Vector quantities of n-velocities, and the magnitudes of the forces in the n-direction, as well as those Vectors too (3 lists in all). I created loops which allowed me to use the formula, V dot n \* n, where V is the original velocity vector and n is the unit vector in the n-direction, to find the n-directional component of velocity. I stored all of those into the list, and then, following the formulae I derived from last week closely and creating a function for even using the sign of a number/expression, I made a function for calculating the total n-directional force magnitude based on the velocities of the endpoints of body parts. For example, the total force calculated for the hand would be based on the velocity of the fingertip and the wrist, as well as the length of the hand. When I had finished all of this, I went ahead and attempted to display the calculated results for the n-velocity and total force, but everything turned out as undefined! Although this seems very strange, I will get to work debugging next time and see if I can figure it out.

Date: March 29th, 2017

Arrival time: 1:50 PM

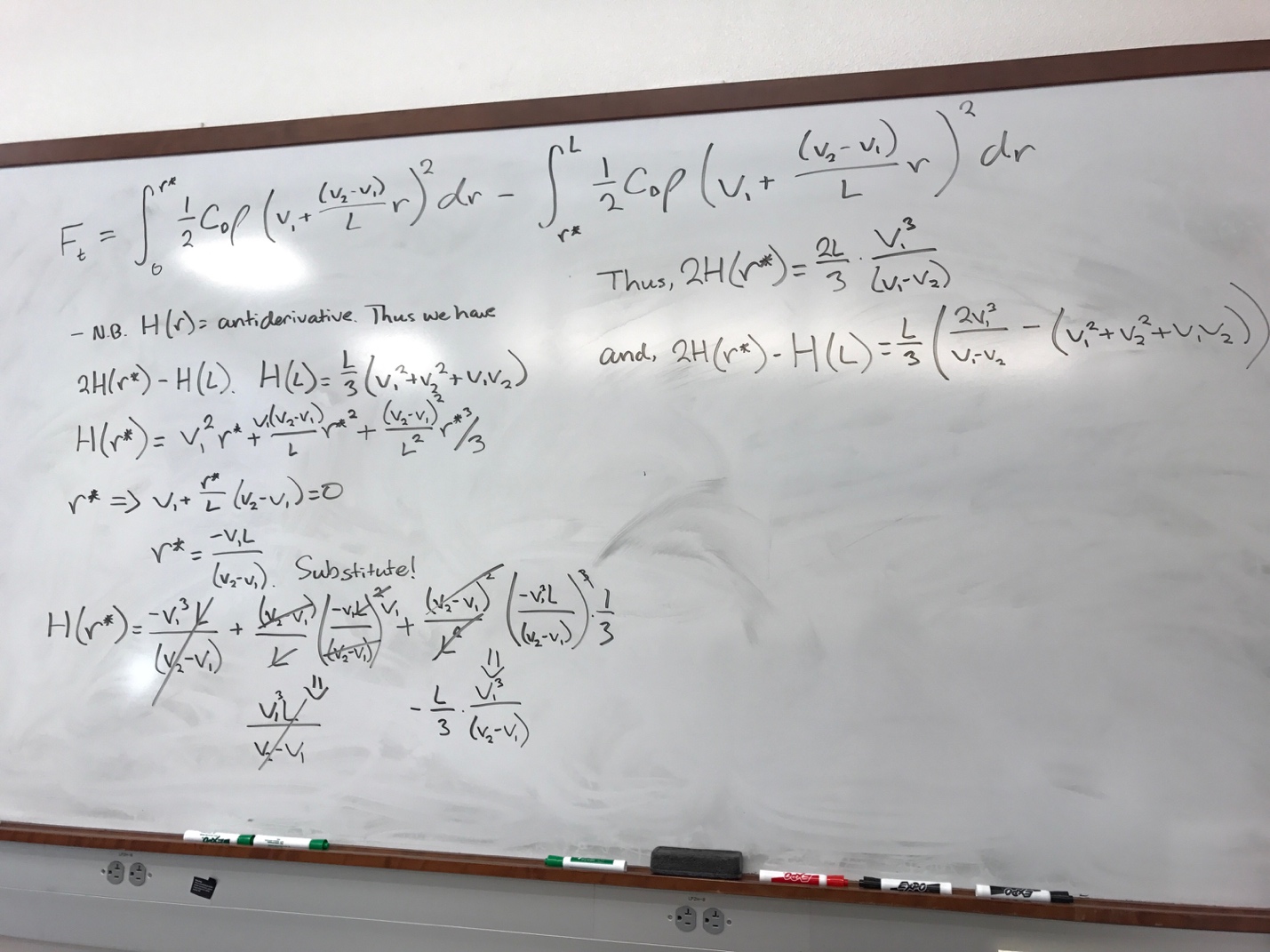
Departure time: 3:50 PM

Today I went in to speak with Dr. Feng about my project. I discussed with him the steps of my plan, which were to first use math and determine explicit formulae for calculations of forces and torques, then analyze the video’s motion to be applied to my own model. I also brought up to him the issue of needing relatively accurate drag coefficients for each body part, and he addressed this by telling me about a new flume, essentially a wind tunnel with water rather than air as its medium, that I could stick various parts into to determine drag coefficient. After this, I went to begin debugging my program. I first printed out the last elements of the lists right after I added them, to make sure I was adding the proper things, but the lists appeared to all be filled with the same element after I was done! After numerous debug statements and asking one of the master’s students working in Dr. Feng’s lab, I realized that the problem was my definition of the new Vector class – I had inadvertently made it static, meaning that there could only be one instance of the class, (one object created from it reserved in memory), and by “creating new” objects of the type, I was just referencing the original one created and thus changing that one, altering all of them. I fixed the class declaration, and was able to get the code to run, but the values for the forces I was getting were extremely nonsensical, and so I am going to have to check my math on these again, as well as possibly re-do my method of drawing the values from the data.

Date: April 4th, 2017

Arrival time: 1:40 PM

Departure time: 3:50 PM

Since the values I had calculated from last week were very very absurd, I attributed this to two things, the first being that my actual method of sampling the positions of body parts at different times was inaccurate, with the entire video being only 300 pixels by 300 pixels, and the second being that the formula I was using to calculate the net forces on the body parts when the rotation point was in the middle was incorrect. Addressing the latter first, I re-worked my algebra in a much simpler way, as shown below, and was able to find a different (hopefully correct formula) for the total force calculation: 

After this, I input the new formula into my program. To remedy the second problem, I went online and downloaded a program which allowed me to resize my video (after a long process of attempting to resize it using iMovie, then reading about mp4 files being inconsistent in some aspect and not being supported [in general] by iMovie, then finally giving up and downloading an entirely new 3rd party software), and began recollecting all the points, so that my measurements would be more precise/accurate. Next time, I will finish collecting the points and rewrite certain elements of the program so that I can calculate the forces the same, as well as come up with a new conversion from pixels to meters (the old conversion was 1 pixel = 1 cm).

Date: April 6th, 2017

Arrival time: 1:45 PM

Departure time: 3:50 PM

Today I met up with Dr. Feng, and we discussed how I was using the video analysis to determine the motion of/calculate forces on the swimming model within the video, and then applying that to the actual 3-D model I was building to simulate motion and forces. However, as I went back and resumed collecting data from the joint points, I realized that I really needed to know just how accurate the force formula I had calculated last time actually was, and instead of going through the whole process to collect data and use the formula on the data, I decided to create a separate program first, a spinning/moving rod simulation very similar to the one I had completed, using the endpoints of the rod rather than its center for position, and attempting to calculate the total force and thus predict its movement. Through this, however, I realized that while I knew the net force, this did not mean that I knew how the ends of the rod (especially without torque) were going to be affected, and I decided that the best way to validate the formula was simply to come up with a few simple examples and see if the results from the formula were logical.

Date: April 10th, 2017

Arrival time: 1:35 PM

Departure time: 3:50 PM

Since I was still getting crazy values for my data, even after using some data points from the enlarged, more precise/accurate video, I decided to take a look at what was going on. After going through the process and reconverting from the bigger screen’s pixels to centimeters (for my velocity/acceleration/force calculations), I realized that my units were off – I was measuring certain things in meters, and certain things in centimeters, and because of the hundred-fold discrepancy, the orders of magnitude of many of my values were extremely off, resulting in crazy calculations. Thus, I fixed all the conversions, and re-printed all the values, which this time gave me much more reasonable numbers. I also realized that the way I was defining the n-direction (the direction perpendicular to the axis of the body part I was looking at) was incorrect – I was using the vector from the predefined origin (0, 0) to one end of the body part and taking the vector perpendicular to that one instead of subtracting the two vectors of the end points of the body parts and taking the vector perpendicular to that one. I fixed that for my velocity calculations, and now must implement it in my force calculation. I am really close to being finished with this section, and will hurry to apply it to the model I have already constructed.

Date: April 12th, 2017

Arrival/Departure time: N/A

At the Translational Research Conference today, one of the projects, done over stem cells and the ECM (extracellular matrix), really intrigued me – rather than studying the genetic predisposition of certain stem cells to predict their differentiation behavior, the researcher instead looked at the composition of the surrounding extracellular fluid, and specifically that produced by adipocytes and osteocytes/osteoblasts, to see how well (or not) the specific ECM was able to conduct cellular growth and promote stem cells specializing into the desired cell type.

Date: April 17th, 2017

Arrival time: 1:40 PM

Departure time: 3:40 PM

Having gotten the hang of how to calculate and store the forces, I decided that today would be a good idea to clean up my program. Taking everything I had done and thinking about only what was necessary, I came up with a plan that looked something like this:

1. Read in position data for body joints (e.g. neck, wrist, elbow, shoulder, ankle, etc.) and store those in lists.

2. Calculate discrete velocities for body parts and store those in lists.

     - Read in position data from previously created lists

     - Use position data from frame i - 1 and i + 1, taking the x and y differences

     - Divide differences by the time it takes for two frames to pass

     - Store calculated velocities in lists

3. Calculate discrete accelerations for body parts and store those in lists.

     - Steps are exactly the same as those for velocity, except using velocities instead of position data

4, Calculate drag forces for body parts and store those in lists.

     - Read in position data from the two endpoints of the body part (for example, the forearm would be wrist + elbow)

     - Read in velocity data from the two endpoints of the body part

     - Using the position data, calculate the vector perpendicular to the axis of the body part

     - Project the velocity vectors onto the perpendicular vector (decompose the vectors)

     - Plug the magnitude of the velocities and the length of the body part into the formula for calculating the magnitude of the force

     - Store the magnitude of the force data into a list

     - Multiply the perpendicular vector by the magnitude of the drag force, and store this vector into a list.

I implemented this, and began to consider the calculation of the applied internal forces (e.g. that of the torso on the upper arm, that of the thigh on the lower leg, etc.), which would be the quantity we are ultimately trying to study here.

Date: April 19th, 2017

Arrival time: 1:35 PM

Departure time: 3:45 PM

Today, I considered equations for the internal forces. Drawing free body diagrams for each of the body parts, I realized that the human body is connected physically in ways described by Newton’s Third Law – for example, the force exerted by the arm on the torso is equal and opposite to that of the torso on the arm, and because I had gathered data for the drag force on each body component and the accelerations of the body components, I would be able to solve equations describing the dynamics of the body:

For example, let’s take the upper arm. There are three main forces acting on the upper arm – that of the body on the upper arm, that of the forearm on the upper arm, and that of the water on the upper arm. To sum this all up in an equation, we have:

F(net) = m\*a = F(lower arm on upper arm) – F(upper arm on body) – F(drag), where F() represents a force.

Thus, working down the arm, I derived a system of equations which looked like:

Upper arm: F(lower on upper) - F(upper on torso) - Fd(on upper) = ma

Lower arm: F(hand on lower) - F(lower on upper) - Fd(on lower) = ma

Hand: - F(hand on lower) - Fd(on hand) = ma

Thus, since I already know m (the mass) for each body part, a (acceleration) for each body part, and Fd (drag force) for each body part, I can solve the system from the bottom up, finding F(hand on lower), and by reverse substituting, find all the forces and thus gather the crucial internal force data.

Date: April 25th, 2017

Arrival time: 1:35 PM

Departure time: 4:10 PM

Today I went to talk to Dr. Feng about the next steps in my project. Having finished the plans for my force calculation program, I realized that one huge shortcoming was that my drag coefficient values for all my force calculations, regardless of the body part involved (or where it was, for that matter; the drag coefficients of objects are significantly different in air than in water), were a standard, made-up value of 0.1. Thus to deal with this, Dr. Feng put me in contact with a modelling team, and talked to me about the options for using a special flume channel (imagine a wind tunnel, but much smaller, like an extremely straight stream, in which one can hold a solid object and observe the flow around the object, as well as calculate the drag force) to measure the drag coefficients on body parts – to 3D print the object, to use already created models (but we would have to disassemble them, in a rather gruesome way), or to create our own models physically from scratch. Because my computer was still giving me difficulty, I grabbed a USB keyboard and mouse and continued to fix bugs in my program, and contacted the modelling team to meet with them before next SRD class so I have a plan.

Date: April 27th, 2017

Arrival time/departure time: N/A

Today, I hopped on the plane to visit Stanford University. California is amazing! The weather is impressively nice ☺

Date: May 2nd, 2017

Arrival time: 1:35 PM

Departure time: 4:05 PM

Today I met with the senior design team from UTSA who are working under Dr. Feng, and discussed ideas about how to model the body parts I needed for the flume channel. There were several options, one of which was to build the entire model from scratch, utterly unfeasible and comparable to reinventing the wheel, a second of which was to model the body parts using clay first by hand, then scanning those in and editing them via the software, not entirely unfeasible, but severely hampered by my skills as an artist, and a third of which was to simply grab a model online, divide it into the parts I needed for 3-D printing, and use those. We also discussed how to calculate the drag coefficient once we had the models; the only ideas the team had were to insert multiple pressure sensors on the body part, and to get an idea of the forces on various parts of the object and thus calculate the drag coefficient of the overall object. When I went back to the lab, Santiago, the graduate student I talked about (probably at the beginning of the year), gave me advice on how to create the models; he suggested that I go online and grab an STL file (a 3D-printable file) of a human body mesh, and use a program, MeshMixer, to cut up the model into the body pieces that I needed.

Date: May 4th, 2017

Arrival time: 1:35 PM

Departure time: 4:00 PM

Today I reinstalled my hard drive from the backup (the Apple people had told me that my laptop may have been this way because of the software installed on it, so I tried wiping it clean after backing it up), downloaded MeshMixer, found a free STL file of a human body, and used the plane slice tool to divide the body into the components that I needed. However, I realized that I didn’t know what size to print, or whether I would need to make special edits to the pieces for pressure sensor attachment or anything else, and emailed Dr. Feng and decided to hold off on actually printing the pieces. Additionally, I worked out some bugs in my program, and fixed the calculations (after much diagraming and deliberation) so that the formulas that I was feeding the data into were using the correct velocity, acceleration, and force frames (I’ll actually put a diagram here):

P5

P4

P3

P2

P1

A1

V3

V2

V1

(So Microsoft Word is terrible and the diagram ruined my regular typing document, so now there is this really wonderful textbox that I created so I can keep typing, ahaha.) So as the diagram shows, the acceleration and velocity for specific frames are calculated using the previous and next frames, but within the list, the indexes are off because of the calculations. Thus I had to adjust for the frame position within the calculations.

Date: May 8th, 2017

Arrival time: 1:50 PM

Departure time: 4:40 PM

After a ton of debugging (including fixing the density of water, which I had set to 1 g/mL rather than 1000 kg/m3), I saw that my data was still extremely rough and didn’t show a very good smooth swimming trend; thus, I went to a rather classic mathematical technique of smoothing the data, which was to take a weighted average of the current point and the points around it, using 1 \* previous point + 2 \* current point + 1 \* next point and dividing that sum by 4, so that the values of the points would be more affected by those around themselves; after a lot of work, making the data smoothing function into a method and transferring tons of data to text files and graphing them in VPython, I saw that the data smoothing was very necessary, and fixed my original problem of being a little too precise by using every frame rather than taking points after several frames (which resulted in jagged data). After learning how to manage the scales, titles, and keys of the graphs, I screenshotted them and added them to my poster, drawing conclusions based primarily off of the drag and applied force data. I was finally able to finish my poster today, and am waiting on feedback; I am also continuing preparations for the actual drag coefficient experiment, and will hopefully wrap that up by next week.

Date: May 10th, 2017

Arrival/Departure time: N/A

Today, I took the AP macroeconomics test. Although I was incredibly nervous because of the calculus BC exam just the previous day, I feel rather solid about that exam, but am definitely glad it’s over ☺

Date: May 16th, 2016

Arrival time: 1:40 PM

Departure time: 3:45 PM

Today I was able to discuss with Dr. Feng what he found out about the flume channel last week; he let me know that it was 90 centimeters across, and wanted me to scale the figure accordingly. I researched the dimensions of a typical Olympic pool lane, and, using that as the standard and in combination with measurements I had taken earlier for my VPython model, was able to scale down the size of an average human [insert any body part here] to be tested in the flume channel. With the scale factor as 9 : 25, I set out to 3-D print the model; but first had to export the files as .obj files, and since my computer was not connected to the 3-D printer, I had to send the files to Santiago to print.

Date: May 18th, 2016

Arrival time: 1:30 PM

Departure time: 5:00 PM

When Santiago came to lab today, I handed him the flash drive so the files could be printed; however, when opening up the first file, the human hand, we found that the scaling would not be good – for some reason, the x/y/z local axes around the object were not properly aligned (for example, the length of the hand, from the wrist to the tip of the middle finger, which should have been the y-axis, actually formed the diagonal of the rectangular axis box). After a ton of messing around with the software, we were finally able to figure it out on Santiago’s computer, but even after doing the exact same steps on my computer, the program still failed to realign the axes. We started the printing process, which took about 2 hours, and in the end, were able to print out the human hand model.