Nathan Bain: neb553
Jacob Bloomer: jmb7722
Cheng Ma: cm44477
Sydney Shutter: ss53592
Chris Timaeus: cat3263

CS378
February 29, 2016

Assignment 1: Inverted Pendulum

To balance the pole we came up with a formula in order to determine the force applied to the cart.

$$\frac{angle}{8*0.01745} + angleDot + posDot + \frac{pos}{2}$$

For every eight degrees of the angle, the resulting force will be increased by one. This provides relatively similar values to that given by the initial code, but this has the benefit of being analog. Like the initial solution, this will try to zero the angle. Adding the *angleDot* and *posDot* has the effect of trying to zero them out as well, which means that the cart will not accelerate and the pole will not be moving. When the *angleDot* is positive, this means that the pole is moving towards the right. Adding this value to the applied force will then push the cart to the right, counteracting the movement of the pole until the pole's angular velocity is zero. By just using the first two elements the pole is balanced, but the cart continues to move to the side. Adding the *posDot* to the action seems counterintuitive, as adding a positive *posDot* to the applied force will only increase the *posDot*. In actuality, the previous two elements will change when this is applied, and the algorithm will again try to zero all of the arguments. Adding the cart's position to the force sets a small offset in the opposite direction of the origin. The pole then points towards the center, and all of the other elements then work to balance the pole, which pushes it to the center. This offset could be rather large at the sides, so the position is multiplied by a factor of a half.

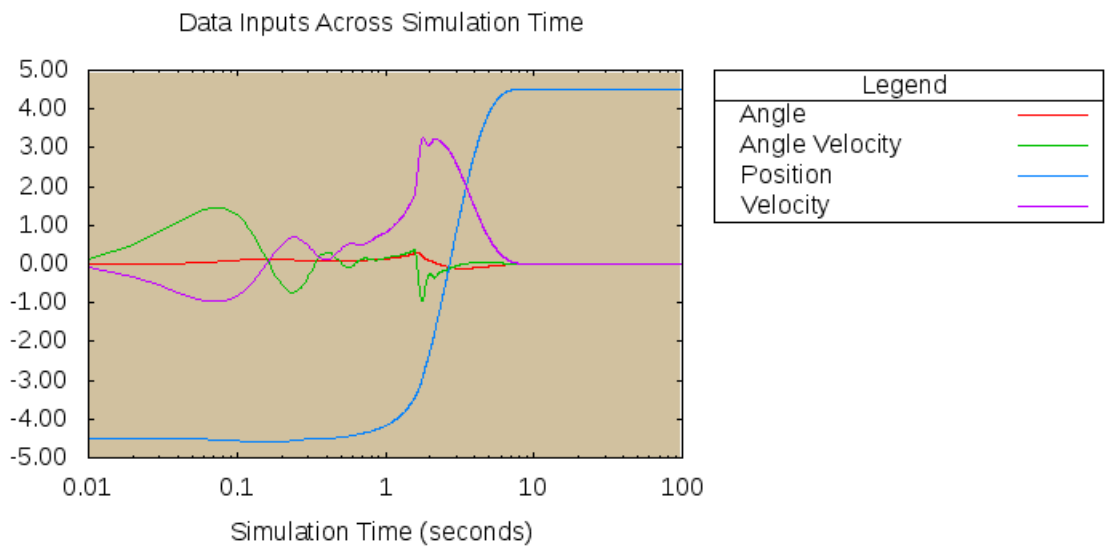
In order to move the cart to the target location the only thing that had to change was the origin that the cart was centering to.

$$\frac{angle}{8*0.01745} + angleDot + posDot + \frac{pos - dest}{2}$$

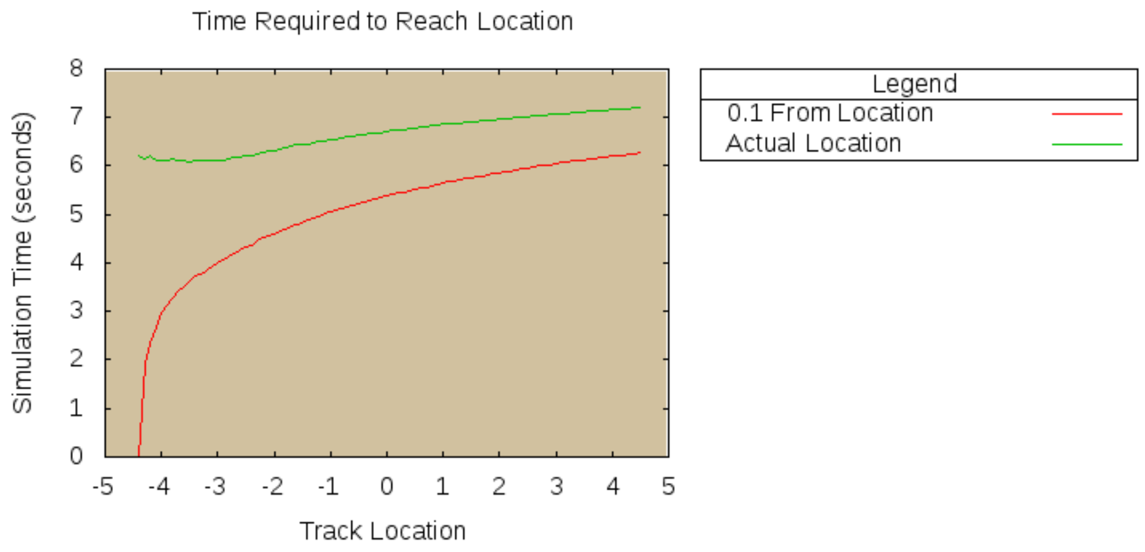
The formula works the exact same way as before except that it acts as though the origin is at *dest*. Previously, the maximum bias for the position was less than five halves, but now the bias could be almost as large as five. This swings the cart out quite far, especially closer to the edges where it could hit the bounds of the track. A check was added so that the force applied would not be directly proportional to the location until the cart was a quarter of the way to the location, but would instead be proportional to the distance to the edge of the track.

The third task required a new function for the pendulums that would follow the primary pendulum. This new function would find the closest object in the direction of the primary pendulum and the would set it's target location to a safe offset from that object. The same balancing function from the previous task is then used, but with the different target location. This allows a pendulum to be balanced while moving towards a primary cart and not hitting anything else. The traveling algorithm worked very well when it was moving towards a stationary point or a slow moving cart and would give itself ample room to decelerate. However, this algorithm would fail when the target location was moving towards the cart, as it would give itself enough force to travel to the location at that exact time. This was fixed by first checking if the two carts were moving in opposite directions. If they were, the algorithm would give cart's location another safety buffer of twice its velocity.

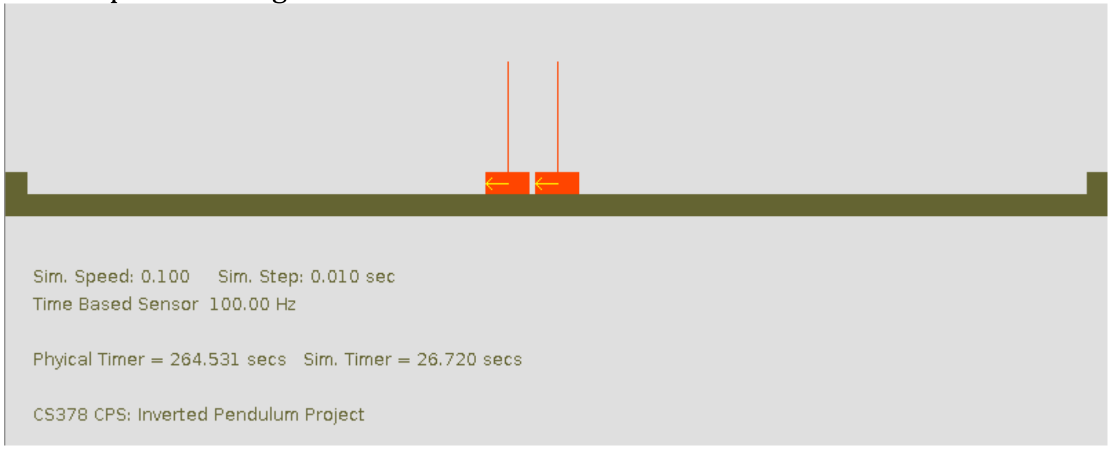
We used a multitude of different tests in order to make sure that our solution works properly. The algorithm balances the pole continuously without error, and the longer that it runs the more balanced the pole becomes. In fact, we found that after running the program for three hours straight, the pendulum was observed to be balanced with all sensing data even more balanced than earlier on. The limits of our target location algorithm are ±4.5. The cart can have any value in this range as an initial or a target location without fail. The graph below shows the sensing data as the cart travels from -4.5 to 4.5 and then balances. Most changes to the sensing data happened within the first few seconds, and reached their final values by ten seconds. The algorithm manages to keep the angle very small for the majority for the run, changing slightly with changes to the angle velocity. The angle velocity and the cart's velocity appear to be somewhat inversely related, and this is expected since the cart moving in one direction will sway the pendulum to the other side. The position is relatively smooth though, with one straight transition to the other side in the first five seconds. The chart shows a basic idea of how the travelling algorithm works. The cart moves in the direction opposite of the target destination increases the pendulum's angle slightly. With the pendulum pointed towards the target location the velocity increases to try to balance the pole.



The cart's velocity continues to increase as long as the pendulum is pointed towards the target location. When the cart is almost at the origin it begins to reverse the direction of the pole in order to slow down, and as it reaches the location all of the other sensing data reaches zero. The graph below shows how long it takes a cart located at -4.5 to reach a target destination and how long it takes to get within 0.1 of the destination. It consistently takes the cart around six to seven seconds to reach the destination, but the last one to two seconds are mostly to balance the pole.



The algorithm for the carts that follow the main cart was tested for carts moving in the same and opposite directions and with the following cart to the left or to the right of the main cart. It was tested with a target location in between the carts and to the side of the main cart. Multiple locations were tested to make sure that the carts would stop with enough distance to not hit the other cart.



Sim. Speed: 0.100   Sim. Step: 0.010 sec
Time Based Sensor 100.00 Hz

Phyical Timer = 264.531 secs   Sim. Timer = 26.720 secs

CS378 CPS: Inverted Pendulum Project

The entire solution is located within ControlServer.java. The solution retains the same basic format as the initial code. The calculate_action function takes the target destination as another parameter to the function. This value is acquired by either referencing a static target value if the pole is the primary pole, or with a call to the follow function. The follow function takes in the index of the current cart, the array of data, and a negative one or a positive one depending on whether the primary pole is to the left or to the right of the cart. The final code can be used to test all three tasks. To test the first task, the number of poles needs to be set to one and the initial position needs to be set in Physics.java. The target location should be set to zero, though the pole will balance continuously at any location. Since there is only one pole the following algorithm will never be called. The setup for the second task is similar to the first except that the target location should be set to a different value. The third task requires more changes in Physics.java. The solution requires the first value in the pole_init_pos array to be the closest to the target location. Multiple values can follow this, with a requirement being that none can be closer to the target location than the initial cart. Additional carts should also be spaced out at least 5/(number of poles) units from the closest cart, though this is largely dependent on the target location, and the spacing can oftentimes be less.