Murfin ECEN3753 Final Submission – Solution Analysis

**RT Tasks**

My project tasks have the following priorities:

1. Service Physics Task

1. LED1 Out Task

2. LCD Graphics Task

3. Measure Capsense Task

4. Btn In, LED0 Out Task

My idea here is that the most important task (from the perspective of gameplay) is Servicing Physics, so it should be 1. LED1 Out Task is also priority 1, because it only happens when triggered by a fatal game violation, so I wanted it to always be prioritized when called. Next is the LCD Graphics Task, since its updates are probably the second most important task. Priority 3 is the Capsense task which is periodic at 100us, and last is the Btn In, LED0 Out task, since the Force Gain adjustments are objectively less important to the game than measuring capsense, updating the lcd or servicing physics. As far as critical deadlines go for this project, there’s not much. Since smooth physics means smoother and less buggy gameplay, keeping that periodic is my main goal – and since the LCD task is called by the physics task and directly second to it in priority, those deadlines will not be missed. Since the capsense task is the 3rd priority behind physics and LCD updates, it is possible that some measurements will be missed – I’m not too worried about this, even if it is only executed half of the time it’s supposed to (5Hz), that should be more than enough from a human reaction time standpoint.

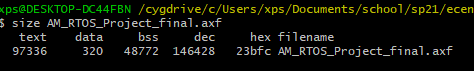
*See the .png’s in “segger\_systemview\_pics” along with the below writeup for each screenshot.*

“**overall\_scheduling.png**” – In this, you can see the periodicity of my major tasks. First, in orange, the Service Physics Task is ran on a period of 50ms. Following that task each time is the LCD Task, in red (listed as “Speed Setpoint Task” in the picture, sorry), which is clearly the most time-heavy task. In purple, you can see the measure capsense task running every 100us.

“**btn0\_task\_imperfect.png”** – Here, you can see the problem with my implementation of the btn in led0 out task. Instead of running on a dedicated hardware timer, I am using an OS delay to adjust the LED0 PWM. While this functionally works “good enough” to be able to discern changes in overall brightness, it is still a little noticeably flicker-ey. Looking at the screenshot, you can see the LED0 PWM is adjusted properly around -15ms, but at +15ms you can see it tries to run during an LCD update – where it is blocked.

“**led1\_output\_task\_at\_violation.png”** – This is an example of what happens when there is a game violation. If the pendulum has fallen (next Ball Y position calculated to bottom of screen) or the base has left the x-bounds, this is flagged in the Service Physics Task, which posts the violation type to a Msg Queue, pended on by the LED1 Out Task, which performs different LED1 routines depending on the violation type. In the screenshot, you can see LED1 Out Task runs immediately after the Service Physics Task. That is where I allowed the pendulum to fall.

**Code Size**



Flash = 97336 + 320 = 97656B = ~95KiB

RAM = 320 + 48772 = 49092B = ~48KiB

This is somewhat significant, but I don’t believe it will have a big – if any – effect on performance.

**Evaluation of My Approach to Physics Update Requirement**

Overall, I am using a modified version of Prof Haines’s “simplified” inverted pendulum calculations.

First, when entering the “Service Physics” Task, I get the previous X,Y velocity, Ball (X,Y) coordinates, Base (AKA Cart) X-Coordinate, Pendulum Theta angle, Force Gain (Multiplier) value, and Force value. These values are used for calculating the same values for the next iteration. First, I calculate the Ball Mass’s X-acceleration: (1 – cos(2(theta))), I then multiply that by the signed Force Value (as measured from the last iteration of the “Measure Capsense” Task, multiplied by Force Gain value) but only if Force is not zero. I then divide that X-Acceleration by 1000, as a scaling value found by trial-and-error. To find the Ball Mass’s Y-Acceleration, I calculate as: sin(2(theta)), then multiply that by the Force value, if it’s not zero. If the theta value is outside a couple degrees, I add the acceleration of gravity, “mg” to the value. I don’t always add mg because it was causing a strange bug where the ball would fall down through the rod when theta was close to zero and there was little or no x-acceleration. Y-Acceleration is divided by 100 to scale the speed of the game to a realistic and playable range. Next, I clamp both acceleration values to “100” so they don’t get out of control. Next I calculate X and Y velocity by simply adding the acceleration to the previous velocity, and clamp it. I then check for shifts in theta by checking to see a sign change in Y-Velocity while theta is close to zero (the task has a static Boolean variable to hold the state of negative or positive theta). Next, I calculate X and Y position by simply adding the previous position to the recently calculated velocity. Next, I calculate theta: if this is the initial state (pendulum is perfectly balanced at theta = 0), it waits for a user input on the capsense pad to simply see which direction force is applied, and then it manually sets theta to a couple degrees in the appropriate direction (this also properly initializes the thetaNeg variable, which constantly tracks if theta is negative or not.) If this is not the initial state, theta is calculated as: arccos(BallY / RodLength), with the sign adjusted depending on the previously explained thetaNeg. Finally, I get the X-Coordinate of the base by performing a sine operation on theta, and adding or subtracting (depending on the sign of theta) that value from the Ball’s X-Coordinate.

I’m honestly not super happy with how this physics turned out. It’s a little jittery and I wish I had a bit more time to tweak it, but alas it’s time to turn in the project and it’s “good enough.” The biggest challenges with this implementation were scaling the forces, speeds and time-deltas to make the game realistic and playable, and fixing all of the weird boundary cases/sign changes. The physics alone took me about 8 hours to debug the boundary cases and fix them properly, and another hour or two of trial-and-error finding all of the scaling values, scaling force of gravity, etc.

**Scaling of Variable Spaces**

As briefly mentioned in the previous section, within the Service Physics Task, there was lots of scaling values needed to make the game playable. One critical value is “Tphy”, the period with which the Service Physics task is executed. When the period is too long in duration, the game was too visibly jittery. When the period is too short, the game goes too fast, and the period of my LED0 was affected. I found a Tphy of 50ms to work nicely – the game is smooth, playable, and with LED0, while still a bit flickery, it is easy to discern the brightness changes. I also scaled the Acceleration Values down by several orders of magnitude – this was the only way I was able keep the forces from immediately spiraling out of control and to a game violation. Gravity, while realistically -9.81m/s^2 here on Earth’s surface, is not so in the game – it is -1.5. It was simple and easy to scale the gravity value to make the game easier. I adjusted this value until I was satisfied with the “falling” of the ball mass when no force was applied to the base. It should be noted that my ball mass value has remained “1” throughout my debugging of this project – since this would really just affect the gravity aspect, I instead found it prudent to just adjust the gravity itself. I also had to scale for the main boundary case: if a y-sign change is noted within 3 degrees of theta=0, the sign of theta changes. For the length of the rod, I was thinking more about the ability to play the game, and reliably show violations when I wanted to – if it was too long it would be too easy to go out of bounds and hard to keep up, whereas if it was too short it would be too hard to keep up. I settled on a length of 60 pixels.

**What I Would Do With 2 More Weeks**

Overall I’m not very satisfied, functionally, with this implementation. I do believe that this definitely meets the requirements of the project, but it’s not as elegant or pretty as it could be with some more time.

My biggest gripe is the LED0 Task, which uses an OS Delay to toggle the PWM. While this is clearly not the best solution, I chose to go with it because other students were reporting issues with using the dedicated LETIMER hardware so I considered it a risk and not worth the time. My implementation has a bit of noticeable flicker while the LCD and Physics Tasks are running, but since you can still very clearly see brightness changes when the buttons are adjusted. If I had more time to fix the PWM, my first idea would be to raise the priority of the LED0 Task *above* the LCD Task – my theory is that this would fully fix the flicker issue, while not causing any other issues since the LCD Task should be able to yield and then pick up where it left off. If that didn’t work, I would continue investigating to find a dedicated hardware timer solution to toggle the PWM.

My next big gripe is the physics. While the game is still very much playable, the boundary shifts (theta shifts) are still a bit jittery, and sometimes the pendulum lacks any movement for a noticeable fraction of a second before beginning to fall when theta is close to zero. I honestly think all this needs is a little more of my debugging.

Finally, my last remaining gripe is with the capsense measurements. I created a function using “CAPSENSE\_getPressed()” to check each of the 4 channels. My function returns either “Hard Left/Right” or “Soft Left/Right” depending on the input. Frankly, this is still a bit unreliable, with some phantom inputs happening. I’m not even really sure what I could do about this as it seems very much like a hardware issue, but it is annoying and something I would look more into in a perfect world. I’ve found that when the board stays powered on for a long time it exacerbates this issue, and if you power cycle it usually gets much better – this makes me think that this is a hardware problem.