1600 Final Project Milestone Report: Robotic Timer

Group: Divyam Dang, Dylan Brady, Jian Cong Loh, Sahil Bansal

1. Requirements

a. R1: The default, starting position of the magnetic ball shall be the leftmost end of the track, STARTING mode.
b. R2: The timer interval shall be initialized to 1 minute.
c. R3: If the ball is in the starting position, STARTING mode, and the start/pause button is pressed, the ball should start moving across the track at a speed such that it reaches the end of the track in the time interval.
    i. R3a: If the ball is not in the starting position and not moving, i.e. in the PAUSED mode, and the start/pause button is pressed, we enter RUNNING mode and the ball should start moving at the same speed as before.
d. R4: If the ball is moving, RUNNING mode, and the start/pause button is pressed, we enter PAUSED mode, and the ball should stop moving within 1 second.
e. R5: There shall be two LEDs that visually display the timer interval.
    i. R5a: If the timer interval is 1 minute, both LEDs shall be off.
    ii. R5b: If the timer interval is 2 minutes, LED1 shall be on and LED2 shall be off.
    iii. R5c: If the timer interval is 3 minutes, LED1 shall be off and LED2 shall be on.
    iv. R5d: If the timer interval is 4 minutes, both LEDs shall be on.
f. R6: If the increment button is pressed and the timer interval is not 4 minutes and the ball is in the starting position, STARTING mode, the timer duration should be incremented by 1 minute and the LEDs should change to indicate the current timer interval.
    i. R6a: If the increment button is pressed and the ball is not in the starting position or the timer interval is 4 minutes, there should be no change.
g. R7: If the decrement button is pressed and the timer interval is not 1 minute and the ball is in the starting position, STARTING mode, the timer duration should be decremented by 1 minute and the LEDs should change to indicate the current timer interval.

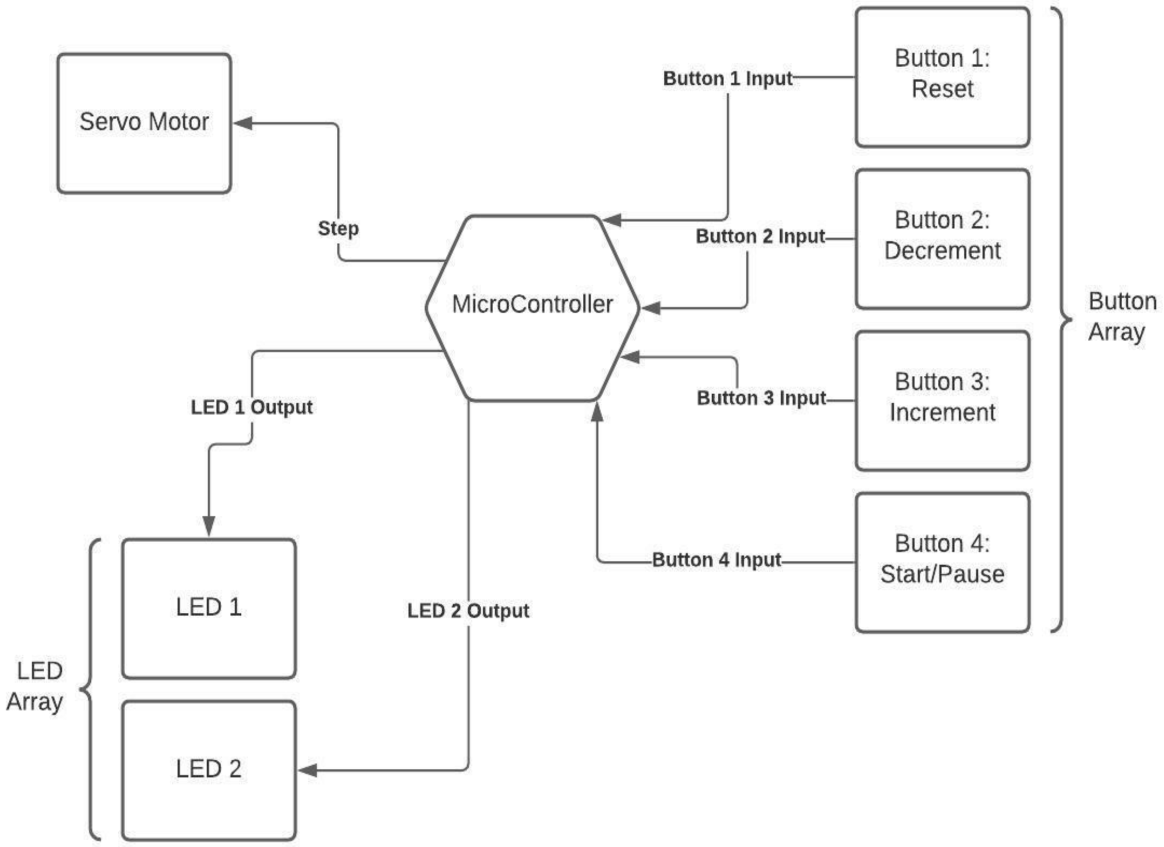
i. R7a: If the decrement button is pressed and the ball is not in the starting position or the timer interval is 1 minute, there should be no change to the system.

h. R8: If the ball is not moving, i.e. is in the PAUSED mode or the FINISHED mode, and the reset button is pressed, the ball shall travel back to the starting position.
   i. R8a: If the ball is moving, i.e. in RUNNING mode, and the reset button is pressed, there should be no change to the system.

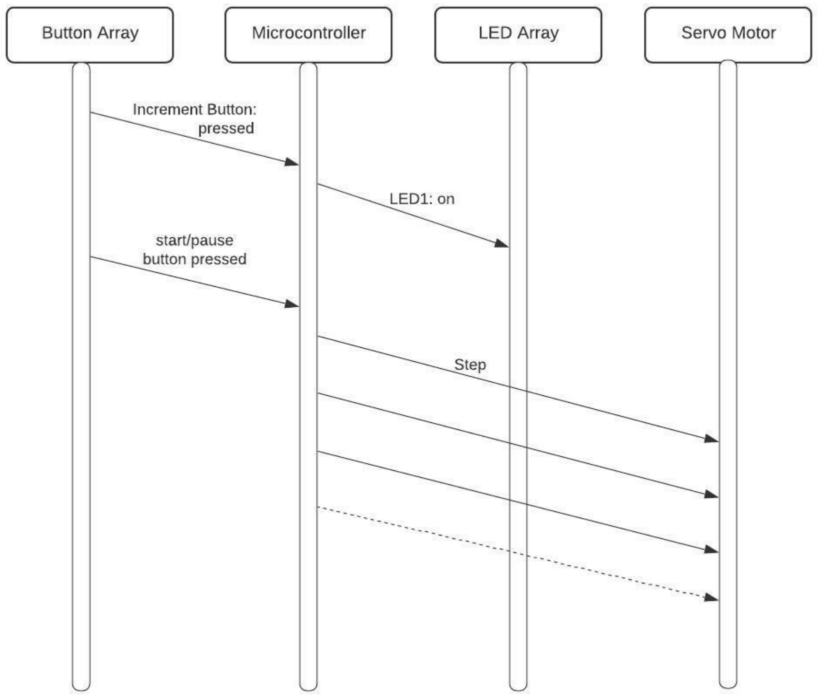
i. R9: If the ball is at the end position at the rightmost of the track, FINISHED mode, pressing any of the buttons other than the reset button shall not change the system.

j. R10: If the ball is moving, i.e. it is in the RUNNING mode and it has reached the end of the track, it shall stop moving and enter the FINISHED mode.

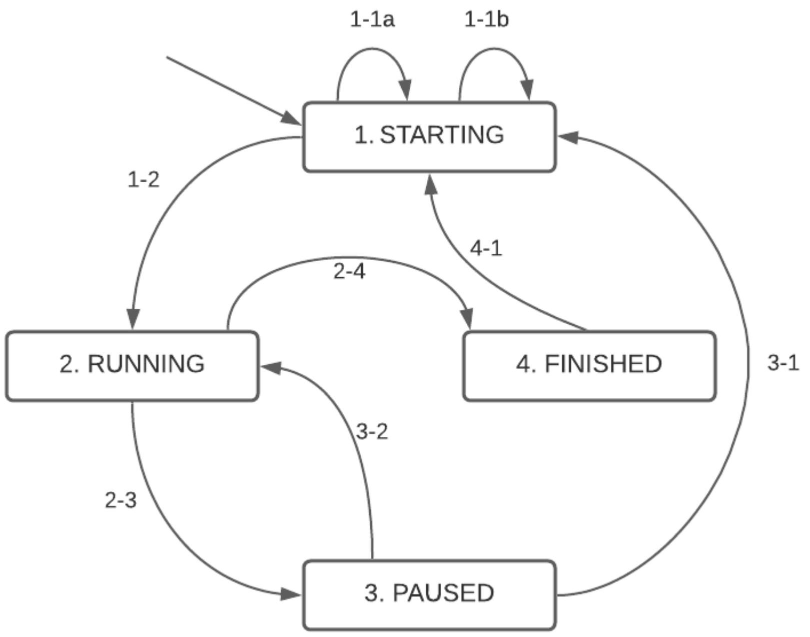
2. Architecture Diagram:



3. **Sequence Diagram**
   a. Use case: assume we are in the initialized, starting position and we want to start a 2-minute timer.



4. **Finite State Machine**



Transition	Guard	Output	Variables

INIT			timer := 1
			freq_step := track_distance/timer
			distance_travelled := 0
			inc_button := 0
			dec_button := 0
			start_pause := 0
			reset_button := 0

1-1a	inc_button = 1 $\land$ timer < 4	set_lights(timer + 1)	inc_button := 0
			timer := timer + 1
			freq_step := track_distance/timer

1-1b	dec_button = 1 $\land$ timer > 1	set_lights(timer - 1)	dec_button := 0
			timer := timer - 1
			freq_step := track_distance/timer

1-2	start_pause = 1	start_step(freq)	start_pause := 0
2-3	start_pause = 1	stop_step()	start_pause := 0
2-4	distance_travelled = track_distance	stop_step()	

3-1	reset_button = 1	reset_system()	distance_travelled := 0
			reset_button := 0

3-2	start_pause = 1	start_step(freq)	start_pause := 0

4-1	reset_button = 1	reset_system()	distance_travelled := 0
			reset_button := 0

5. **Traceability Matrix**

	R1	R2	R3	R3a	R4	R5	R5a	R5b	R5c	R5d	R6	R6a	R7	R7a	R8	R8a	R9	R10
STATE																		
1	x	x				x	x	x	x	x		x		x				
2																x		
3					x													
4																	x	x
TRANSITION																		
1-1a											x							
1-1b													x					
1-2			x															
2-3					x													
2-4																		x
3-1															x			
3-2				x														
4-1															x		x	

6. **Test Plan**
   a. As our project will take the form of a finite state machine, we have the ability to test every possible input into the system. This is very helpful, as we can be 100% sure that we are completely testing it, and we can deploy it with full confidence. In order to have full confidence, we need to create tests for each of the possible transitions between states. We won't have to test for flawed inputs because we only allow a finite number of buttons that the user can interact with the system with. As long as the initial state is the same, the possible scenarios should be limited to the ones that we have defined above in our finite state machine diagram.
   b. We will start by unit testing the button array, LED array and servo motor separately. We will use the corresponding functions that control them as shown below to ensure that they operate as the function expects. We will have to mock out the functions used by the button array so that they will print to the serial monitor so that we can check they appropriately set the variables they need to. We will be able to test all possible inputs for **`set_lights(int timer)`**,

`start_pause_handler()`, **`reset_button_handler()`**, **`inc_button_handler()`**, **`dec_button_handler()`**, **`stop_step()`** and will do so. We will test **`start_step(float freq_step)`** on only 10 different inputs because there are an infinite number of possibilities in this case and 10 different inputs will allow us to be confident that the function works as expected, given that our program will most likely only give 4 different inputs to this function.

c. For **`update_fsm`**, we will aim to have 100% branch coverage so we can test all possible transitions. Since we will all know the code it will count as white-box testing and we will aim to achieve 100% line coverage to ensure we are hitting all possible cases in all scenarios.
    i. LED array: **`set_lights(int timer)`**
    ii. Button array: **`start_pause_handler()`**, **`reset_button_handler()`**, **`inc_button_handler()`**, **`dec_button_handler()`**
    iii. Servo motor: **`start_step(int freq_step)`**, **`stop_step()`**
    iv. Microcontroller: **`update_fsm`**

d. We will then move on to integration testing. We will first test whether the button array appropriately controls the LED array. We will then test whether the button array can appropriately start and stop the servo motor. We will then test in more detail the interfaces between all these parts by testing whether the buttons give us full control of the entire timer system. We will know that the project works as intended if we are able to run a simulation of each of our sequence diagrams and the system behaves as we expect.

e. We will then move on to acceptance testing, and check whether our project works as the requirements specify. To ensure that the testing is robust, we will consider it successful if we have 3 tests per requirement that pass.

7. Reflection

a. We have met most of the things in the milestone we proposed in part 4 of the proposal. We are able to start and pause the progress of the timer, which in this case is only the motor. We are however not really able to “reset” the timer since there is no track and resetting does not make sense. As the milestone we proposed said, the timer is not accurate and does not look like a complete object yet.

8. Update on Expected Final Deliverable

a. As of now, we are unlikely to carry out the extension and produce a 2D sand drawing machine. We will instead focus our efforts on making the 1D timer work as accurately as possible, as well as develop the aesthetics and usability of the object.