Assignment 3:

Goal: To understand state transition diagrams and how they can help us build complex algorithms by visually representing the algorithm. Build timing systems on the arduino UNO that will allow us to precisely calculate derivatives, integrals, and declare control actions at discrete points in time that have a constant duration between executions.

Pre-requisites: [TinkerCAD](https://www.tinkercad.com/) account

Hardware used:

* Arduino UNO
* Quadrature Encoder (<https://www.dynapar.com/technology/encoder_basics/quadrature_encoder/>)
* DC Motor (<https://en.wikipedia.org/wiki/DC_motor>)
* H-Bridge (L293D or LN298), (<https://www.elprocus.com/h-bridge-motor-control-circuit-using-l293d-ic/>) (<https://cdn-shop.adafruit.com/datasheets/l293d.pdf>)

Software Concepts:

* State Transistion Diagrams (aka State Machine) (<https://www.youtube.com/watch?v=v8KXa5uRavg>)
  + This video is very informative!

State Transition Diagrams:

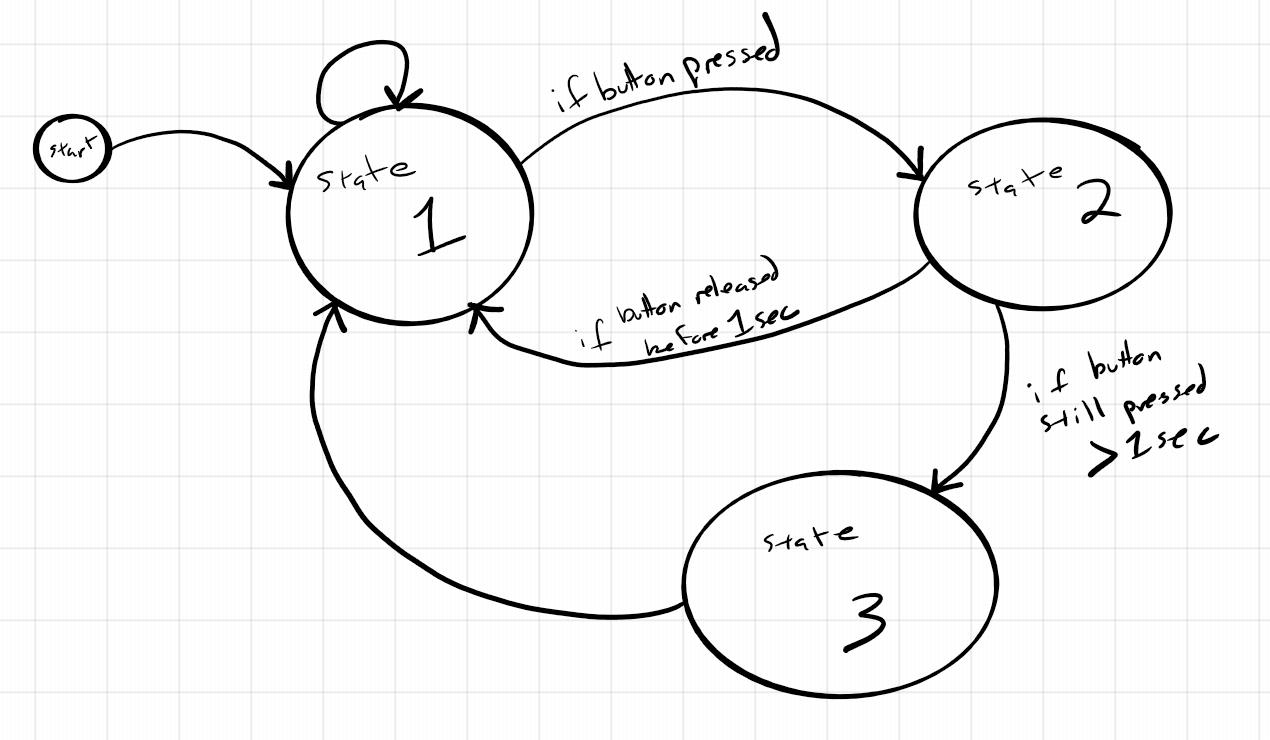
A state transition diagram is a way to represent the flow of a logical system. Typically it is a very useful diagram to help decide algorithmic flow, and to also easily replicate the algorithm into usable code for a microcontroller or regular script. There are a few important rules to follow though while creating these logical systems. The most important is that any operation that happens does not “hang” the processor! This is extrememly important because we want the logical system to check it’s conditions as fast as possible and as many times per second as possible.

Let’s build the State Transisition Diagram for a State Machine that is trying to detect a long button press. First let’s layout what we think the system will do as a numbered list, then look at its diagram.

1. Detect button press
2. Continue checking if button is pressed
3. If button is pressed for >1 second print “long button press detected”
4. If button is released before 1 second print “long press NOT detected”
5. Reset related variables and go back to 1.

In a State transition diagram there are 3 important shapes. The first is the start/end termination node. This node represents either the start or the termination of the state machine. We can think of the start node as plugging the power into the arduino and running the void setup() function. Next we have the state node which is just a circle with the state’s name/number in it. This node tells us which state we are currently in and has arrows connected to it that let us know which states lead to this state, and which state this state can lead to! Lastly, we have the arrows the represent the transition between states. Typically an arrow or state transition condition will have an “if” statement tied to it that will check that a transition condition is met before jumping into the new state. (NOTE: I do somewhat break this rule by placing a Serial.print() command inside a state. This is done for simplicity! I highly recommend NOT putting print statements inside states when time critical processes are running from these state machines, but for these teaching purposes I let it slide!)

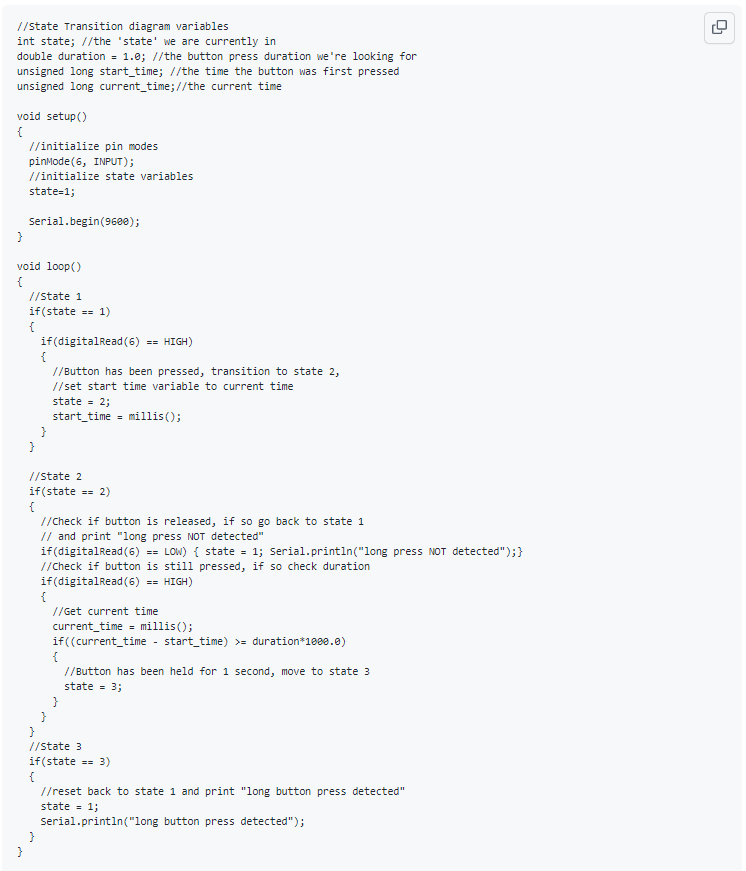
Here is an example diagram of the “long button press detect” state machine:



A couple of important notes: If the arrow going between states (or from a state to itself) has no condition, that means immediately after we are in that state, we go to the state the no condition arrow points to. Of course we make sure any variable updates happen that are necessary before we switch states, but that is all that happens. In this case we are just detecting a button press by reading the voltage on a digital pin. Okay, so let’s turn this diagram into some usable code. First we’ll need some variables to allow the system to function:

1. A “state” variable, we will use an integer to represent the states
2. A “duration” variable that we will use to reference if the button has been pressed long enough.
3. A “start\_time” variable that will hold the time the beginning of the button press has been detected.
4. A “current\_time” variable that will hold the current time so we can compare the difference between state\_time and current\_time to the duration variable

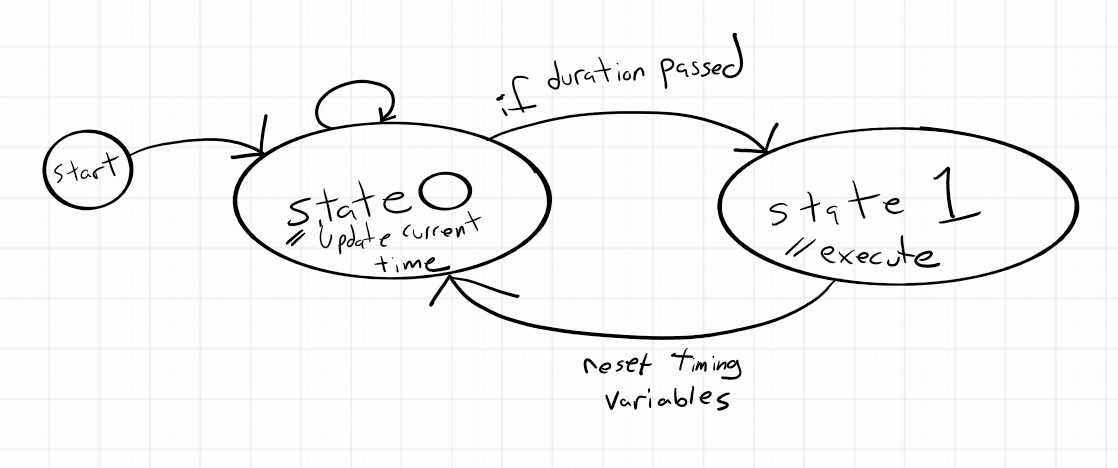
Hopefully you watched the state transition diagram video linked above. In that video a switch:case structure is used to define the state machine. Since you’ve seen that, we will instead create this with some nested ‘if’ statements just to see another way to build the same type of system in code!



There is one main issue with this state machine! If you continue holding the button after the first long button press detected, it will detect another button press!! Even though we never re-pressed the button! This is because we aren’t looking for a release of the button before we reset the system! Think of a way you can solve this issue by either adding a state, or changing the transition condition for one of the states! This will be added in the TODOs at the bottom.

Single Timing System:

A very useful state machine is what I call the “Single Timing System”. I used this to solve the Approximate Angular Velocity bonus from the Controls – 2 assignment. In that assignment it doesn’t look like I’m using a state machine because I have no ‘state’ variable. But the logical process does fit into that of a state machine. It just so happens that the machine only has 2 states, so I do not need a state variable as I can just check the state transition condition. But for this example I will build the single timing system with a ‘state’ variable so it is easy to see the correlation.



The code for this example is in the TinkerCAD example Controls - 3

Async Multiple Timing Systems:

Similar to the Single Timing System, a multiple timing system is essentially just multiple Single Timing System state machines that run by themselves. If we’re clever and we prevent any processor hanging inside any states, we can create a pseudo “parallel processing” inside the arduino even though the arduino does not support parallel processing! Take a look at the code example in TinkerCAD for the multiple timing system example.

**Setup actions:**

1. Open up [“TinkerCAD Controls - 3”](https://www.tinkercad.com/things/edD3F0CavKq-fantastic-trug-snicket/editel?sharecode=FMhOmGFpJ1ucDOIBXlD5als3Xm2-rJilfFr4O7n8mJo) tinkerCAD circuit
   1. Duplicate this file by going to your TinkerCAD home page and clicking the gear on the thumbnail for this circuit.
   2. **PLEASE do NOT edit the original document! Either duplicate it or make your own circuit and copy the code over!**

**TODO:**

Fix the bug in the state transition diagram example of a long button press:

* Redraw the state transition diagram for the long button press but now with your fix included
* Edit the example code such that your changes in the state transition diagram are reflected
* Re-write the state transition nested if statements as a switch:case structure

Draw the State transition diagram for the multiple timing system:

* Draw the state transition diagram for the multiple timing system. Remember each system is isolated from the others! (other than the “start” node)
* Think of 2 examples where using a multiple timing system would allow you to solve a problem requiring a sort of parallel processing that using delay() or while() loops would prevent proper execution

Finish the angular velocity bonus from Controls – 2:

* Using the single timing system come, calculate the approximate angular velocity using the single timing system to execute when the calculations happen