# RealTimeSensing Project

## Simulation of ESP32 Microcontroller and Sensors

There are two tasks. One is loopTask and the other one is readSensorsTask.

The readSensorsTask reads each of the sensors (temperature, humidity, distance from ultrasound sensor, and photoresistor illuminance) once. An array is used to store the data for each sensor. Even though only a single value is read and processed for each sensor in each iteration, an array is used for the average calculation. Then, the loopTask is notified to process and send data. The readSensorsTask blocks until loopTask notifies it to continue after it processes and sends the sensor data values read. Notification is used because the reading and processing/sending should not overlap in execution, otherwise there will be a race condition since the same arrays are modified by the readSensorsTask and accessed by the loopTask. The readSensorsTask clears the array of sensor data values if full. Then, it blocks until 1 second after the most recent start of the task iteration. This allows the task to read sensors every 1 second.

The loopTask has to process and send the data read from each of the sensors in a sequence. It connects to the MQTT server. For each sensor, the original unfiltered value, the low pass filtered value, and the custom filtered value (difference between the most recent sensor value and previous sensor value) are sent. The low pass filter uses a parameter between 0 and 1 to specify the weights of the previous low pass filter computation and current sensor data value in a weighted sum for the current low pass filter computation. The custom filter is a filter that I chose. It is just the difference between the current and previous sensor data values. The average value is calculated and sent if the array is full. For the filtering calculations, the previous value is saved. In the case where a sensor's array is cleared, the next value will be placed at the first position of the array, but the previous value will be needed for some filtering calculations so the previous values that are needed are saved.

The protocol for sending data is specifying the sensor type (Temperature: T, Humidity: H, Distance: D, Photoresistor: P) and filter type (original or unfiltered: o, low pass filter: l, custom filter: c, average filter: a). The two codes can be in any order, but the two codes must be sent at some time before sending the number of the sensor data value. The most recent sensor type and filter type are used for subsequent numbers until new codes are sent. The sensor and filter types let the receiving end know what the data value is for. When one data value per sensor and filter type combination is done being sent for all the combinations, the end of sequence code 'X' is sent.

UART Protocol is simulated with each data packet. Every letter code or number sent will be in a separate message and must be sent as a data packet, which must have a start bit (#), the data value, the even parity, and, lastly, the stop bit (&). It should follow this order. The even parity is calculated by finding the number that is added to the sum of the digits in the data value to make the sum even. The MQTT connection is made by connecting an MQTT In node to MQTT Out node. This is like connecting the UART Tx pin from the microcontroller to the

UART Rx pin of the host. The public MQTT server is used since security was not a priority in this project.

## Visualization Application

The visualization application has two processes. The main process runs the GUI main loop and updates the tkinter display of subplots. The other process is the data_processing_producer which handles checking and getting data from the data packets and sending the data to the GUI process. The GUI needs to keep running to show real time updates so it is kept in a process separate from the data receiving and processing.

The data_processing_producer process connects to the MQTT server to receive data sent by the simulation. It has a callback function that is called when a message is received. The simulated UART protocol is followed. When a message is received, the process checks for the start and stop bits in the string. Then, it calculates the expected parity bit from the data portion and checks with the parity from the message. Finally, the data portion of the message is processed. If it is one of the expected sensor types or filter types, then it is saved. If the data portion is a number and if sensor and filter types have been specified, then it is added to the list of values for the specific sensor and filter types combination in a dictionary. The least recent value will be dropped if an array is full since the maximum size of arrays need to be maintained when a new value is added. The array of sensor data for each sensor and filter type combination is the list of y-values for that specific subplot. If the data portion does not match anything, nothing is done. The real time data processing needs to continue so bad data is just ignored. When the end of the sequence is received through the 'X' data, the dictionary with the list of sensor values is sent to the GUI process through a multiprocessing queue. The data_processing_producer process just continues waiting to receive the next message (data packet) from the simulation. The multiprocessing queue has no size limit so it can be filled as messages arrive. A queue is used rather than trying to use locks/mutexes and sharing data because sending data in a queue can also act as a way to notify when to set values of the subplots, and it reduces unnecessary computation.

The GUI main loop sets up the GUI display and then updates periodically. The update function actually checks for data to be sent to it from the data_processing_producer process through the multiprocessing queue. It is a non-blocking queue so an exception is thrown if there is no data in the queue, but the update function is set to just continue execution after the exception. The GUI keeps running and should not freeze. Once the updated sensor values (organized as arrays in a dictionary indexed with sensor type and filter type keys) are received, the subplots are updated. In order to show simultaneous reading, the graphs will update together. Only the averages graph for each sensor will update in between longer time intervals to wait for a chunk of 10 values to be read by the simulation but the averages graph will update at the same time for all sensors. The x-values match the number of y-values so the x-values do not need to be saved and can just be calculated based on the list of y-values. The text display of the averages is set on each update and only shows the most recent averages. All graphs start with (0,0) since a line needs to be drawn when the first point arrives and two points are needed to draw a line so the origin is included at the beginning.

## Simulation Results

The values of the sensors can be changed in the simulation and the changes will appear on the graphs. There is a slight delay since it takes time to process, transmit, handle incoming data on the visualization application, and plot. Considering one sample per sensor and filter combination for all sensor and filter combinations at a time has less delay than updating sensors one by one. The progression is also shown more clearly than if multiple samples per sensor and filter combination for all combinations were to be updated in the subplots.