Summer School on Embedded Systems and Robotics

Aptitude Test Submission

Name: Shubhranil Basak

Roll no.: IMT2023510

Q1. Assumptions: As there are multiple ways of generating square waves using mathematical functions and operations like Fourier transformations, signum function, etc. but in practical scenarios they are generated using techniques like op amp, Fourier transformation, sum of harmonics of sine wave, etc. I have taken the approach of sum of harmonics for my code. The formula is as follows:

My code is as follows:

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import matplotlib.widgets as widgets  import pandas as pd  # Square wave generator function  def generate\_square\_wave(t, frequency, amplitude, num\_harmonics, duty\_cycle):  square\_wave = np.zeros\_like(t)  for n in range(1, num\_harmonics + 1):  harmonic = (4 / (2 \* n - 1) / np.pi) \* np.sin(2 \* np.pi \* (2 \* n - 1) \* frequency \* t)  square\_wave += harmonic  square\_wave \*= amplitude    # Apply duty cycle  period = 1 / frequency  duty\_cycle\_time = (duty\_cycle / 100) \* period  square\_wave = np.where(t % period < duty\_cycle\_time, amplitude, -amplitude)    return square\_wave  # Data saver function  def save\_to\_csv(t, square\_wave, filename):  data = pd.DataFrame({  'Time': t,  'Amplitude': square\_wave  })  data.to\_csv(filename, index=False)  print(f"Waveform data saved to {filename}")  # Default parameters  default\_freq = 5.0  default\_amp = 1.0  default\_duty\_cycle = 50.0 # Default duty cycle in percentage  samples = 1000  num\_harmonics = 10000 # Number of harmonics to sum  # Time array for plotting  t = np.linspace(0, 1, samples, endpoint=False)  # Initial square wave  square\_wave = generate\_square\_wave(t, default\_freq, default\_amp, num\_harmonics, default\_duty\_cycle)  # Create plot  fig, ax = plt.subplots(figsize=(15, 10))  plt.subplots\_adjust(bottom=0.35)  plt.title('Square Wave Generator')  ax.set\_xlabel('Time (Sec)')  ax.set\_ylabel('Amplitude')  plt.grid(True)  # Plot initial square wave  line, = plt.plot(t, square\_wave, 'r', label='Generated Square Wave')  plt.legend()  # Add text boxes for frequency, amplitude, and duty cycle inputs  freq\_text\_box = widgets.TextBox(plt.axes([0.125, 0.05, 0.1, 0.05]), 'Freq', initial=str(default\_freq))  amp\_text\_box = widgets.TextBox(plt.axes([0.325, 0.05, 0.1, 0.05]), 'Amp', initial=str(default\_amp))  duty\_cycle\_text\_box = widgets.TextBox(plt.axes([0.525, 0.05, 0.1, 0.05]), 'Duty (%)', initial=str(default\_duty\_cycle))  # Add a submit button  submit\_button\_ax = plt.axes([0.725, 0.05, 0.1, 0.05])  submit\_button = widgets.Button(submit\_button\_ax, 'Save to CSV')  def update(val):  freq = float(freq\_text\_box.text)  amp = float(amp\_text\_box.text)  duty\_cycle = float(duty\_cycle\_text\_box.text)  new\_square\_wave = generate\_square\_wave(t, freq, amp, num\_harmonics, duty\_cycle)  line.set\_ydata(new\_square\_wave)    # Update y-axis limits dynamically based on waveform amplitude range  max\_abs\_amplitude = np.max(np.abs(new\_square\_wave))  ax.set\_ylim(-1.1 \* max\_abs\_amplitude, 1.1 \* max\_abs\_amplitude)    plt.draw()  def submit\_callback(event):  freq = float(freq\_text\_box.text)  amp = float(amp\_text\_box.text)  duty\_cycle = float(duty\_cycle\_text\_box.text)  save\_to\_csv(t, generate\_square\_wave(t, freq, amp, num\_harmonics, duty\_cycle), f'squarewave\_{freq}\_{amp}\_{duty\_cycle}.csv')  submit\_button.on\_clicked(submit\_callback)  freq\_text\_box.on\_submit(update)  amp\_text\_box.on\_submit(update)  duty\_cycle\_text\_box.on\_submit(update)  plt.show() |

Q2. Assumptions:

1. I assume that floating point calculations done by the calculator adhere to the IEEE 754 standards.
2. I assume that in case the final answer is a floating-point value, it is type casted to an integer at the end of all the calculations.
3. I ran the calculator at <https://circuitdigest.com/calculators/adc-analog-to-digital-converter-calculator> with different values for both Analog Voltage at input to ADC and Reference Voltage at ADC and came up with this table and ensured my code follows it as well.

|  |  |  |
| --- | --- | --- |
| Analog Voltage input to ADC | Reference Voltage at ADC | Output |
| Positive (+ve) | Positive (+ve) | Positive (+ve) |
| Positive (+ve) | Negative (-ve) | Negative (-ve) |
| Positive (+ve) | Zero (0) | Infinity |
| Negative (-ve) | Positive (+ve) | Negative (-ve) |
| Negative (-ve) | Negative (-ve) | Positive (+ve) |
| Negative (-ve) | Zero (0) | Infinity |
| Zero (0) | Positive (+ve) | Zero (0) |
| Zero (0) | Negative (-ve) | Zero (0) |
| Zero (0) | Zero (0) | NaN |

My code is as follows:

|  |
| --- |
| import tkinter as tk  def calculate():  N = entry1.get()  AIV = entry2.get()  RV = entry3.get()  try:  N = float(N)  AIV = float(AIV)  RV = float(RV)  if (RV > 0 and AIV >= 0) or (RV < 0 and AIV <= 0) :  result1 = int(((2\*\*N)\*(AIV))/(RV))  result2 = bin(result1)[2::]  elif RV < 0 or AIV < 0:  result1 = int(((2\*\*N)\*(AIV))/(RV))  result2 = '-' + bin(result1)[3::]  elif RV == 0 and AIV != 0:  result1 = "infinity"  result2 = "infinity"  elif AIV == 0 and RV == 0:  result1 = "NaN"  result2 = "NaN"  # Update the text in the output Entry widgets  output\_entry1.delete(0, tk.END)  output\_entry1.insert(0, result1)  output\_entry2.delete(0, tk.END)  output\_entry2.insert(0, result2)  except ValueError:  #Error message for invalid input  output\_entry1.delete(0, tk.END)  output\_entry1.insert(0, "Invalid input")  output\_entry2.delete(0, tk.END)  output\_entry2.insert(0, "Invalid input")  def reset():  # Reset all input fields to empty strings  entry1.delete(0, tk.END)  entry2.delete(0, tk.END)  entry3.delete(0, tk.END)  # Reset output Entry widgets to empty strings  output\_entry1.delete(0, tk.END)  output\_entry2.delete(0, tk.END)  # Main tkinter window  root = tk.Tk()  root.title("ADC Calculator")  # Main frame  frame = tk.Frame(root, padx=20, pady=20)  frame.pack()  # Input fields  tk.Label(frame, text="Number of bits in ADC:").grid(row=0, column=0, padx=10, pady=5)  entry1 = tk.Entry(frame)  entry1.grid(row=0, column=1, padx=10, pady=5)  tk.Label(frame, text="Analog Voltage input to ADC:").grid(row=1, column=0, padx=10, pady=5)  entry2 = tk.Entry(frame)  entry2.grid(row=1, column=1, padx=10, pady=5)  tk.Label(frame, text="Reference voltage to ADC:").grid(row=2, column=0, padx=10, pady=5)  entry3 = tk.Entry(frame)  entry3.grid(row=2, column=1, padx=10, pady=5)  # Output Entry widgets  tk.Label(frame, text="Numeric Digital Output:").grid(row=3, column=0, padx=10, pady=5)  output\_entry1 = tk.Entry(frame)  output\_entry1.grid(row=3, column=1, padx=10, pady=5)  tk.Label(frame, text="Binary Digital Output:").grid(row=4, column=0, padx=10, pady=5)  output\_entry2 = tk.Entry(frame)  output\_entry2.grid(row=4, column=1, padx=10, pady=5)  # Calculate and Reset buttons  calculate\_button = tk.Button(frame, text="Calculate", command=calculate)  calculate\_button.grid(row=5, column=0, padx=10, pady=10)  reset\_button = tk.Button(frame, text="Reset", command=reset)  reset\_button.grid(row=5, column=1, padx=10, pady=10)  # Tkinter mainloop  root.mainloop() |

Q3. The package name is ‘tukey’, name after American mathematician and statistician John Tukey.

The link to the PyPi package is: <https://pypi.org/project/tukey/>

The GitHub repo for the documentation can be hound here: <https://github.com/Shubhranil-Basak/tukey/>

**How to use it:**

1. Install the package using the pip command:

|  |
| --- |
| pip install tukey |

1. Open a python file and then you can use the package in the following way:

|  |
| --- |
| from tukey import tukey  # Basic arithmatic operations  print(tukey.add(1, 2)) # Output: 3  print(tukey.subtract(5, 3)) # Output: 2  print(tukey.multiply(4, 6)) # Output: 24  print(tukey.modulo(5, 2)) # Output: 1  print(tukey.divide(8, 2)) # Output: 4.0  print(tukey.divide(10, 0)) # Output: Cannot divide by zero |

Q4. In a practical setting where a resistor 𝑅 is connected across a voltage source V, the current I(R) through the resistor depends on the value of the resistor R according to Ohm's Law. Ohm's Law states that the current through a conductor between two points is directly proportional to the voltage across the two points and inversely proportional to the resistance.

Mathematically, Ohm’s law can be expressed as:

Rearranging the equation, we get:

Therefore, the current I(R) through the resistor R connected across a constant voltage V is inversely proportional to the resistance of the resistor.

But in a practical situation, temperature is also a factor that we can’t ignore. To take temperature into account, we have to go beyond the normal Ohm’s law.

The resistance (R) of a typical resistor can vary with temperature. This temperature dependency is often characterized by the temperature coefficient of the resistance (α), which describes how the resistance changes per degree Celsius (°C) of temperature change. The relationship between resistance (R), temperature (T), and the reference temperature (T0) can be expressed as:

Where:

.

.

.

Combining Ohm’s law with temperature dependence:

To incorporate temperature (T) into our analysis, we can substitute the expression for resistance (R) into Ohm’s law:

Effect of temperature on current:

If we keep the voltage(V) constant then the current through the resistor I(R) will decrease with an increase in temperature and increase with a decrease in temperature.

Q5. Some of the reasons to modulate a signal with a carrier wave are as follows:

1. **Frequency Translation**: Modulation can help to shift the frequency spectrum of the original frequency to a higher frequency range suitable for efficient transmission and reception. It is because the frequency of the original baseband signal is different from those used for communication purposes like frequency bands for Radio or microwaves.
2. **Antenna Efficiency**: The use of a carrier wave at a higher frequency facilitates efficient transmission through antennas. Antennas are more effective at radiating EM waves at higher frequencies, especially when compared to the lower frequencies typical of baseband signals. This efficiency is crucial for long-range communication and for minimizing the size and complexity of antennas.
3. **Signal integrity and noise immunity**: Modulation helps in preserving the integrity of the transmitted signal and makes it more immune to noise and interference. The carrier wave can carry the modulated signal over long distances without significant loss of information, as compared to direct transmission of baseband signals.
4. **Multiplexing and channel allocation**: Modulating signals with different carrier frequencies enables multiplexing—simultaneously transmitting multiple signals over the same communication medium.
5. **Compatibility with transmission media**: Different transmission media (like cables, optical fibers, etc.) have specific characteristics that determine their suitability for transmitting signals at particular frequencies. By modulating a signal, we can match the characteristics of the transmission medium, optimizing the signal's propagation and minimizing data loss.

Assumptions:

1. Considering there is no noise in carrier and signal wave.
2. Considering there is noise only in modulated wave
3. Considering smoothing od demodulated wave is necessary else it might have false signals because of noise.

My code for the OOK modulation and demodulation is as follows:

|  |
| --- |
| #OOK\_ani.py  import numpy as np  import matplotlib.pyplot as plt  from scipy.ndimage import binary\_dilation, binary\_erosion  from matplotlib.animation import FuncAnimation  def animate\_waveforms(x):  # Generate data  carrier\_frequency = 20  carrier\_wave = np.sin(carrier\_frequency \* x)  fig, ax = plt.subplots(5, 1, figsize=(15, 6))  plt.subplots\_adjust(hspace=0.5)  def animate(frame):  for axes in ax[1:]:  axes.clear()  pulse\_shift = frame \* 0.1  pulse\_signal = (np.abs(x - pulse\_shift) % 2.0 < 1.0).astype(float)  modulated\_wave = carrier\_wave \* pulse\_signal  np.random.seed(13213)  noise\_amplitude = 0.3  modulated\_wave += np.random.normal(loc=0, scale=noise\_amplitude, size=len(x)) \* (pulse\_signal > 0)  threshold = 0.1  demodulated\_wave = (modulated\_wave > threshold).astype(float)  demodulated\_wave += (modulated\_wave < -threshold).astype(float)  binary\_signal = (demodulated\_wave > 0).astype(int)  smoothed\_signal = binary\_erosion(binary\_dilation(binary\_signal))  ax[0].clear()  ax[0].plot(x, carrier\_wave, color='red')  ax[0].set\_title('Carrier Wave')  ax[0].set\_ylim(-1.5, 1.5)  ax[1].clear()  ax[1].plot(x, pulse\_signal, color='green')  ax[1].set\_title('Pulse Signal')  ax[1].set\_ylim(-0.5, 1.5)  ax[2].clear()  ax[2].plot(x, modulated\_wave, color='blue')  ax[2].set\_title('OOK Modulated Wave')  ax[2].set\_ylim(-1.5, 1.5)  ax[3].clear()  ax[3].plot(x, demodulated\_wave, color='orange')  ax[3].set\_title('OOK Demodulated Wave')  ax[3].set\_ylim(-0.5, 1.5)  ax[4].clear()  ax[4].plot(x, smoothed\_signal, color='green')  ax[4].set\_title('Smoothed Demodulated Wave')  ax[4].set\_ylim(-0.5, 1.5)  for axes in ax:  axes.tick\_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)  ani = FuncAnimation(fig, animate, frames=100, interval=50)  plt.show() |

|  |
| --- |
| #OOk\_static.py  import numpy as np  import matplotlib.pyplot as plt  from scipy.ndimage import binary\_dilation, binary\_erosion  def generate\_waves(carrier\_frequency, x, pulse\_amplitude=1, pulse\_width=2.0):  # Generate modulated and demodulated waves  carrier\_wave = np.sin(carrier\_frequency \* x)  pulse\_signal = (np.abs(x) % pulse\_width < pulse\_width/2).astype(float) \* pulse\_amplitude  modulated\_wave = carrier\_wave \* pulse\_signal  # Adding noise  np.random.seed(13213)  noise\_amplitude = 0.3  modulated\_wave += np.random.normal(loc=0, scale=noise\_amplitude, size=len(x)) \* (pulse\_signal > 0)  # Demodulation  threshold = 0.1  demodulated\_wave = (modulated\_wave > threshold).astype(float) \* pulse\_amplitude  demodulated\_wave += (modulated\_wave < -threshold).astype(float) \* pulse\_amplitude  # Smoothing the demodulated signal  binary\_signal = (demodulated\_wave > 0).astype(int)  smoothed\_signal = binary\_erosion(binary\_dilation(binary\_signal))  return carrier\_wave, pulse\_signal, modulated\_wave, demodulated\_wave, smoothed\_signal  def plot\_waveforms(carrier\_wave, pulse\_signal, modulated\_wave, demodulated\_wave, smoothed\_signal, x):  # Create figure and axis for the plot  fig, ax = plt.subplots(5, 1, figsize=(15, 10))  # Plot carrier wave  ax[0].plot(x, carrier\_wave, color='red')  ax[0].set\_title('Carrier Wave')  ax[0].set\_ylim(-1.5, 1.5)  # Plot pulse signal  ax[1].plot(x, pulse\_signal, color='green')  ax[1].set\_title('Pulse Signal')  ax[1].set\_ylim(-0.5, 1.5)  # Plot modulated wave  ax[2].plot(x, modulated\_wave, color='blue')  ax[2].set\_title('OOK Modulated Wave')  ax[2].set\_ylim(-1.5, 1.5)  # Plot demodulated wave  ax[3].plot(x, demodulated\_wave, color='orange')  ax[3].set\_title('OOK Demodulated Wave')  ax[3].set\_ylim(-0.5, 1.5)  # Plot smoothed demodulated wave  ax[4].plot(x, smoothed\_signal, color='green')  ax[4].set\_title('Smoothed Demodulated Wave')  ax[4].set\_ylim(-0.5, 1.5)  for axes in ax:  axes.tick\_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)  # Adjust layout  plt.subplots\_adjust(hspace=0.5)  plt.show() |

Main python file:

|  |
| --- |
| import numpy as np  import OOK\_ani  import OOK\_static  def main():  choice = input("Enter 1 for static plot or 2 for animation: ")  x = np.linspace(10, 30, 1000)  if choice == '1':  # Static  carrier\_wave, pulse\_signal, modulated\_wave, demodulated\_wave, smoothed\_signal = OOK\_static.generate\_waves(  carrier\_frequency=20, x=x, pulse\_amplitude=1, pulse\_width=2.0)  OOK\_static.plot\_waveforms(carrier\_wave, pulse\_signal, modulated\_wave, demodulated\_wave, smoothed\_signal, x)  elif choice == '2':  # Animation  OOK\_ani.animate\_waveforms(x)  else:  print("Invalid choice. Please enter 1 or 2.")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

All of the codes used in the document can be found in the GitHub repo:

<https://github.com/Shubhranil-Basak/SummerSchool_2024/>

All the resources I referred to:

* <https://matplotlib.org/stable/api/widgets_api.html#module-matplotlib.widgets>
* <https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.FuncAnimation.html>
* <https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_erosion.html>
* [https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary\_dilation.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.binary_dilation.html#scipy.ndimage.binary_dilation)
* <https://electronics.stackexchange.com/questions/32310/what-exactly-are-harmonics-and-how-do-they-appear>
* <https://mathworld.wolfram.com/FourierSeriesSquareWave.html>
* <https://circuitdigest.com/calculators/adc-analog-to-digital-converter-calculator>
* <https://packaging.python.org/en/latest/guides/distributing-packages-using-setuptools/>
* <https://www.allaboutcircuits.com/textbook/experiments/chpt-2/ohmmeter-usage/>
* <https://www.tutorialspoint.com/digital_communication/digital_communication_amplitude_shift_keying.htm>
* <https://en.wikipedia.org/wiki/On%E2%80%93off_keying>
* <https://en.wikipedia.org/wiki/Envelope_detector>
* <https://dsp.stackexchange.com/questions/1841/how-to-smooth-a-signal>