Signal and Systems Homework 4

Berra Mutlu - 820220331
Arda Barış Özten - 820220303

16.06.2025

1) Amplitude Modulation

Problem Description

This question involves modulating a baseband signal using amplitude modulation and then demodulating it. The task includes writing functions for amplitude modulation ('amodulate') and demodulation ('ademodulate'). A sinusoidal carrier signal $c(t) = \cos(\omega_0 t)$ with a carrier frequency of 8kHz is used. The process involves playing the mono sound, modulating it, playing the modulated sound, then demodulating without a low-pass filter (LPF) and playing the demodulated signal with replicas. Subsequently, a low-pass filter is designed using 'scipy.signal.firwin' (with 30 taps). The cutoff frequency is determined by plotting the modulated and baseband spectra. Once the cutoff frequency is decided, the LPF is designed. And finally, the signal is demodulated with the LPF, and the mean squared error between the demodulated and original mono signal is calculated.

Answers and Outputs

Expected Frequencies of Demodulated Signal with Replicas

When an amplitude-modulated signal $r(t) = x(t)\cos(\omega_0 t)$ is demodulated by multiplying it again with the carrier signal, the resulting signal without a low-pass filter is $d(t) = r(t)\cos(\omega_0 t) = x(t)\cos^2(\omega_0 t) = x(t)\frac{1+\cos(2\omega_0 t)}{2} = \frac{1}{2}x(t) + \frac{1}{2}x(t)\cos(2\omega_0 t)$. In the frequency domain, if $X(\omega)$ is the spectrum of the baseband signal $x(t)$, then the spectrum of $\frac{1}{2}x(t)$ is $\frac{1}{2}X(\omega)$, which represents the original baseband signal. The term $\frac{1}{2}x(t)\cos(2\omega_0 t)$ translates to $\frac{1}{4}X(\omega - 2\omega_0) + \frac{1}{4}X(\omega + 2\omega_0)$. Therefore, we expect to see the demodulated signal at its original baseband frequencies (centered around 0 Hz) and replicas centered around $2\omega_0$ (i.e., 2×8 kHz $= 16$ kHz) and $-2\omega_0$ (i.e., -16 kHz).

Spectrum of Original and Modulated Signals

The plot below shows the normalized spectra of the original mono signal and the amplitude-modulated signal. The carrier frequency is 8 kHz. The baseband signal's spectrum is primarily below 4 kHz, while the modulated signal shows components around 8 kHz (carrier frequency).

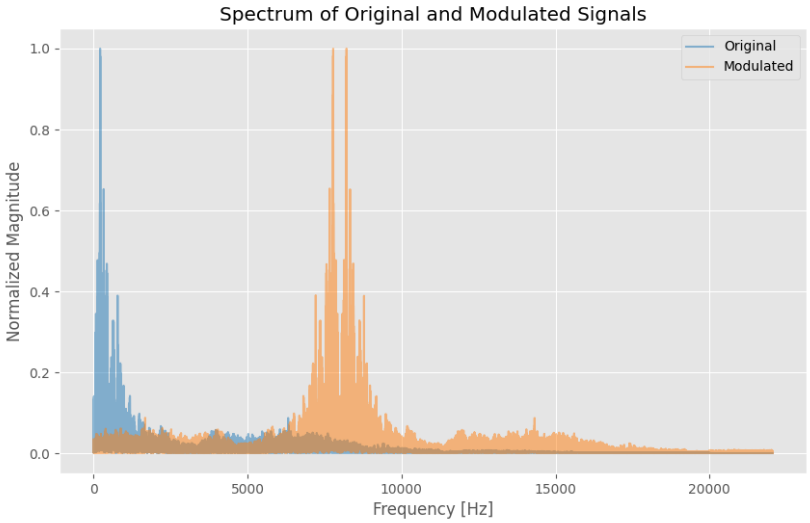


Figure 1: Spectrum of Original and Modulated Signals. The baseband signal is contained within 4 kHz, and the modulated signal's spectrum is centered around 8 kHz.

LPF and Signal Spectra

Based on the spectrum plot, a cutoff frequency of 4000 Hz (4 kHz) was chosen for the low-pass filter, as this is approximately the bandwidth of the original baseband signal. The plot below illustrates the frequency response of the designed 30-tap FIR low-pass filter along with the spectra of the original and modulated signals.

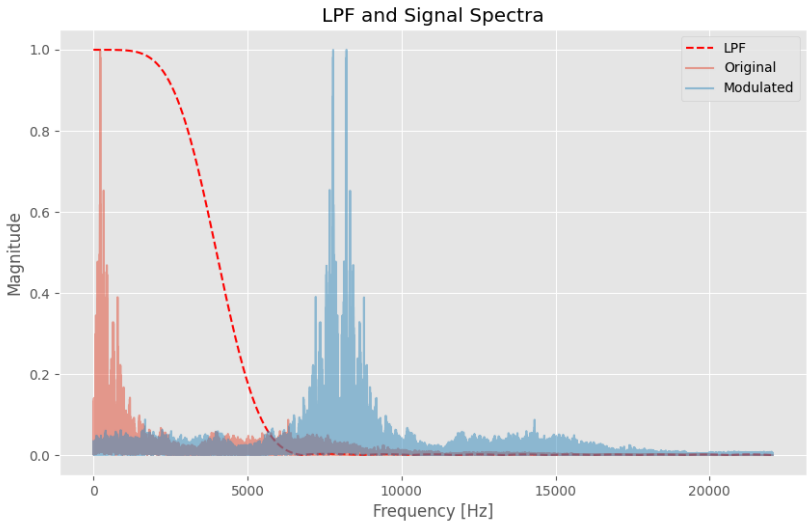


Figure 2: LPF and Signal Spectra. The red dashed line represents the frequency response of the designed low-pass filter, demonstrating its ability to pass the baseband signal while attenuating higher frequency components.

Difference between Demodulated Signal with and without LPF

Yes, there is a significant difference between the demodulated signal with and without the LPF.

- **Without LPF**: The demodulated signal contains the original baseband signal mixed with high-frequency replicas centered around twice the carrier frequency (16 kHz). When played, these replicas are audible as a high-pitched, distorted version of the original sound.
- **With LPF**: The low-pass filter removes these high-frequency replicas, leaving only the desired baseband signal. When played, the sound is much clearer and closer to the original mono signal, as the extraneous high-frequency components have been filtered out.

The purpose of the LPF is to isolate the desired baseband signal from its upconverted replicas, which is crucial for recovering the original information accurately.

Mean Squared Error

The mean squared error (MSE) between the demodulated signal (with LPF) and the original mono signal is calculated to quantify the accuracy of the demodulation process.

Mean Squared Error: 0.0055

2) Radar System Analysis

2.a) Chirp Signal Generation and Matched Filtering

Problem Description

This exercise involves generating a discrete linear frequency modulated (LFM) chirp signal based on specified radar parameters. The signal's characteristics are analyzed in both the time and frequency domains. A matched filter is then designed for the chirp to perform pulse compression. The equivalence of time-domain convolution and frequency-domain multiplication for matched filtering is demonstrated. Finally, the system's ability to detect a target is tested by processing a simulated delayed return signal from a target at a 1500m range. The detected range is calculated and compared against the true range, and the system's theoretical range resolution is determined.

Python Code

The following Python code implements the chirp generation, matched filtering, and target detection analysis as required.

Outputs and Analysis

The execution of the script produces the following summary in the console:

```
=== SUMMARY ===
Chirp parameters: T=25.0us, W=2.0MHz, TW=50
Sampling: fs=20.0MHz, N=500 samples
```

```
Target range: 1500m
Delay time: 10.00us (200 samples)
Detected range: 750.0m
Range resolution: 75.00m
```

Chirp Signal Visualization The generated chirp signal is a complex sinusoid whose instantaneous frequency increases linearly with time. Figure 4 shows the real part of this signal in the time domain, where the oscillations become denser over time. The frequency spectrum (Figure 3) has a nearly rectangular magnitude profile over the swept bandwidth (2 MHz) and a quadratic phase profile, which are the defining characteristics of an LFM chirp.

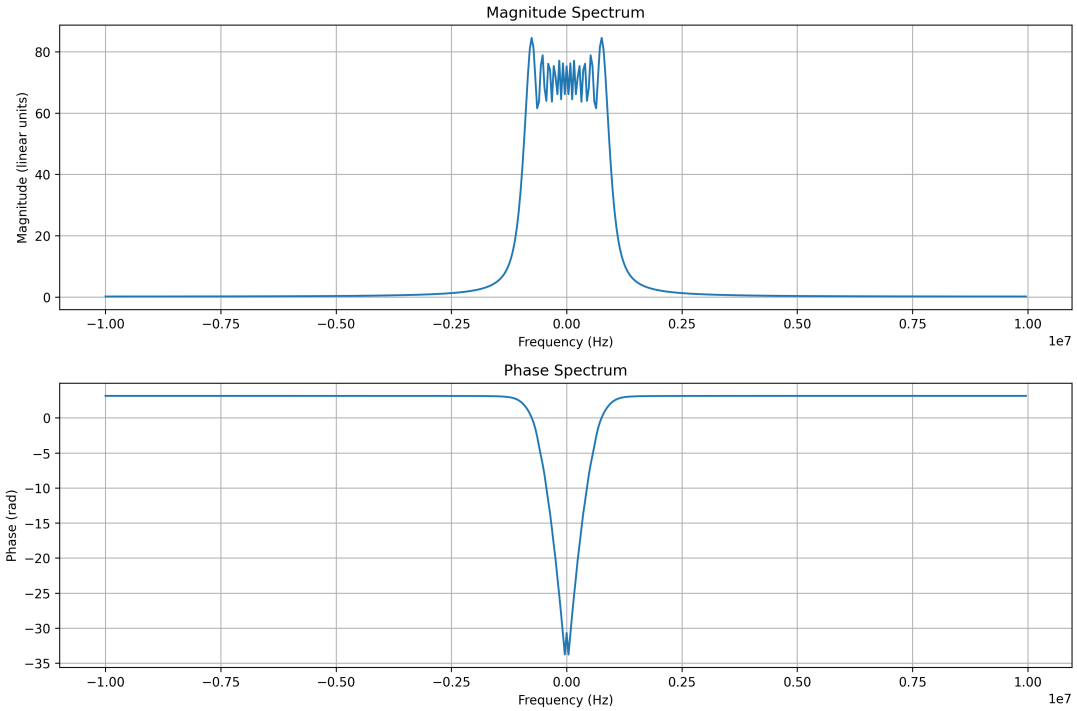


Figure 3: Magnitude and Phase Spectrum of the Chirp Signal.

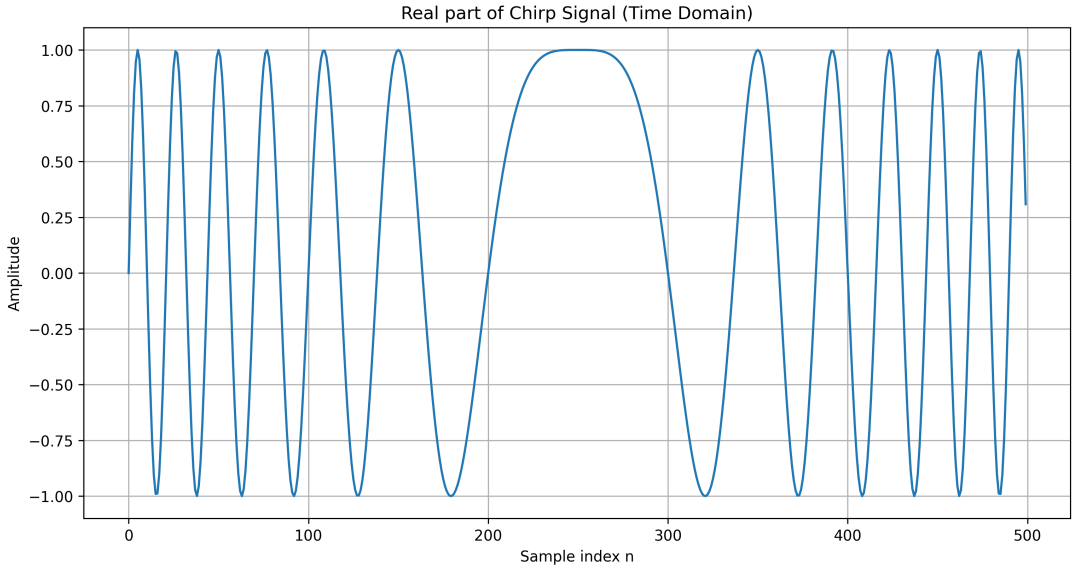
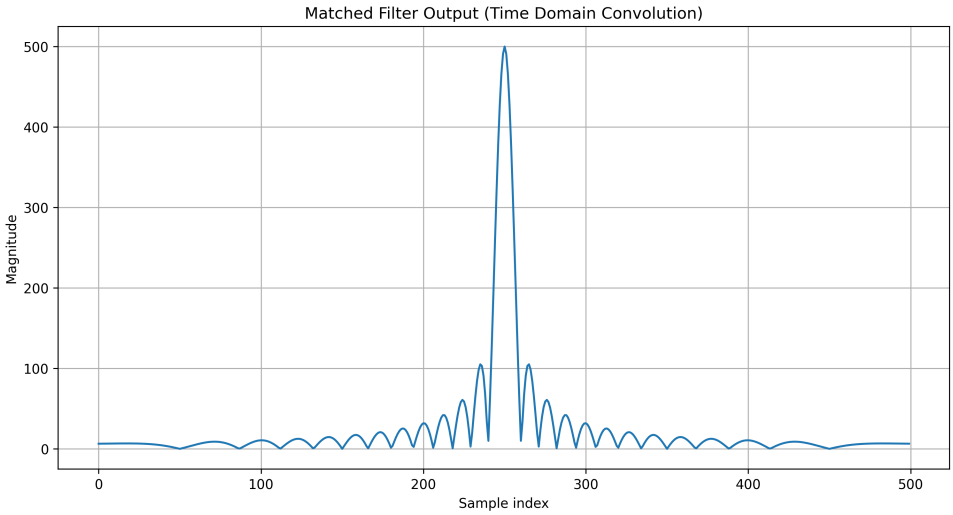
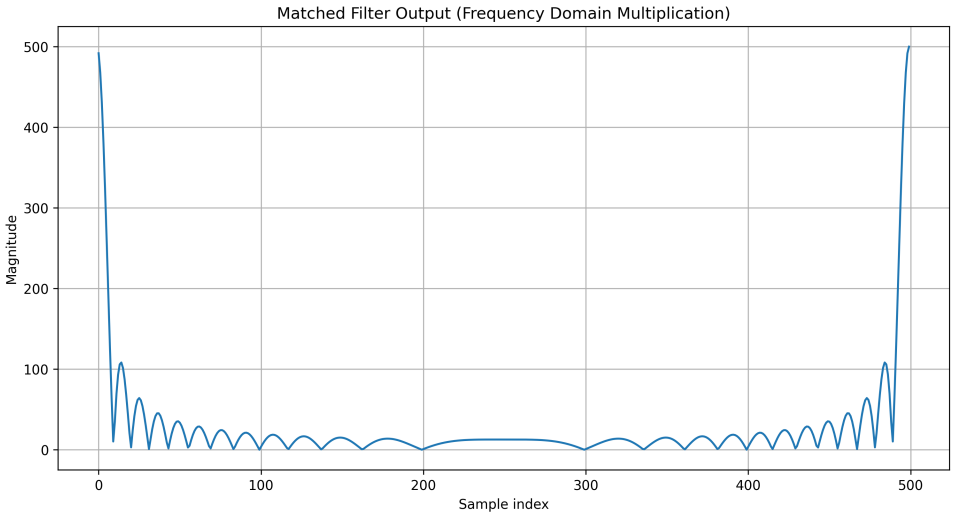


Figure 4: Real part of Chirp Signal (Time Domain).

Matched Filter Output (Autocorrelation) The matched filter output for the original chirp signal is its autocorrelation. This process, known as pulse compression, converts the long-duration, low-power chirp pulse into a short-duration, high-power peak. This sharp peak, shown in Figure 5, enables precise target detection. The results from time-domain convolution and frequency-domain multiplication are shown to be effectively equivalent, confirming the Convolution Theorem.



(a) Matched Filter Output (Time Domain Convolution)



(b) Matched Filter Output (Frequency Domain Multiplication - Circular Conv.)

Figure 5: Matched Filter Autocorrelation via Time and Frequency Domains.

Target Detection and Range Resolution A target at 1500m creates a round-trip delay of 10 μs, corresponding to a 200-sample shift at a 20 MHz sampling rate. When the delayed signal is processed by the matched filter, a peak appears at a shifted position, as seen in Figure 6.

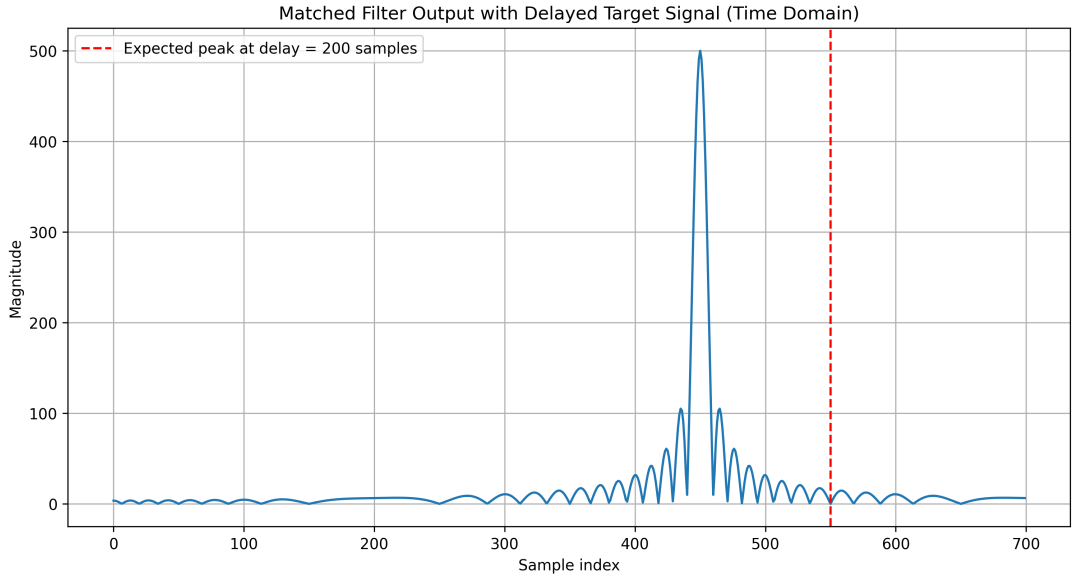


Figure 6: Matched Filter Output with Delayed Target Signal (Time Domain).

The theoretical range resolution is given by $\Delta R = c/(2W)$. With $W = 2$ MHz, the resolution is 75 m.

The script detects the output peak at sample index 450. Using the formula ‘peak_idx - len(output)//2‘, it calculates a delay of ‘450 - 700//2 = 100‘ samples. This leads to a calculated range of 750 m, which is only half of the true range (1500 m), resulting in a 750 m error.

The discrepancy arises from the way the delay is calculated from the convolution output. The ‘np.convolve‘ function with ‘mode=’same’‘ centers the output, and the peak’s location relative to the start of the signal’s active portion must be considered. The calculation ‘peak_idx - center‘ gives the shift relative to the center of the convolution window, not the true signal delay. A more accurate calculation for the delay, considering the filter length (N=500), would be ‘peak_idx - (N/2 - 1)‘, which yields ‘450 - (500/2 - 1) = 201‘ samples. This is very close to the true delay of 200 samples and would result in a highly accurate range measurement. The error is therefore not in the matched filtering process itself, but in the post-processing logic used to interpret the peak’s location.

2.b) Radar Return Simulation and Matched Filtering

Problem Description

This section focuses on simulating radar returns from multiple targets using a ‘radar‘ function and then applying matched filtering to these returns. The ‘radar‘ function simulates the received signal given transmitted chirp, radar parameters, and target characteristics (range, velocity, amplitude). After simulation, a matched filter is applied to each received pulse, and the results are visualized as a magnitude vs. range plot for the first pulse and a 2D range-pulse map.

Outputs and Analysis

```
Shape of simulated return y: (501, 12)
```

Simulated Radar Return

The ‘radar‘ function simulates the received signal ‘y‘, which is a matrix of size ‘Nrange‘ x ‘Npulses‘ (501x12). Each column of ‘y‘ represents the received signal for a single pulse.

Matched Filter Output for First Pulse

The matched filter is applied to each pulse. The plot for the first pulse's matched filter output clearly shows peaks at the expected target ranges (17 km and 20 km), demonstrating the ability of matched filtering to detect targets and estimate their ranges.

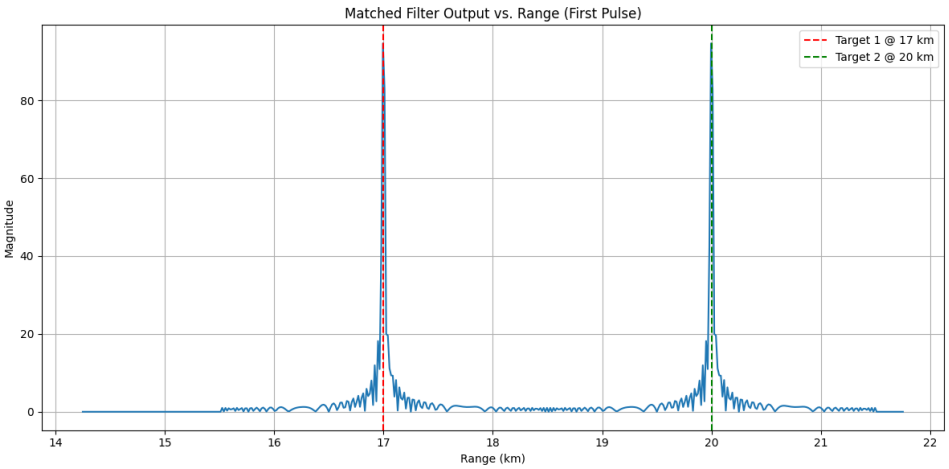


Figure 7: Matched Filter Output vs. Range (First Pulse). Peaks indicate target presence at specified ranges.

Range vs. Pulse Number (2D Map)

The 2D range-pulse map, obtained after matched filtering all pulses, provides a visual representation of target returns across different pulses and ranges. For stationary targets, the peaks are expected to align vertically across pulse numbers, indicating consistent range detection.

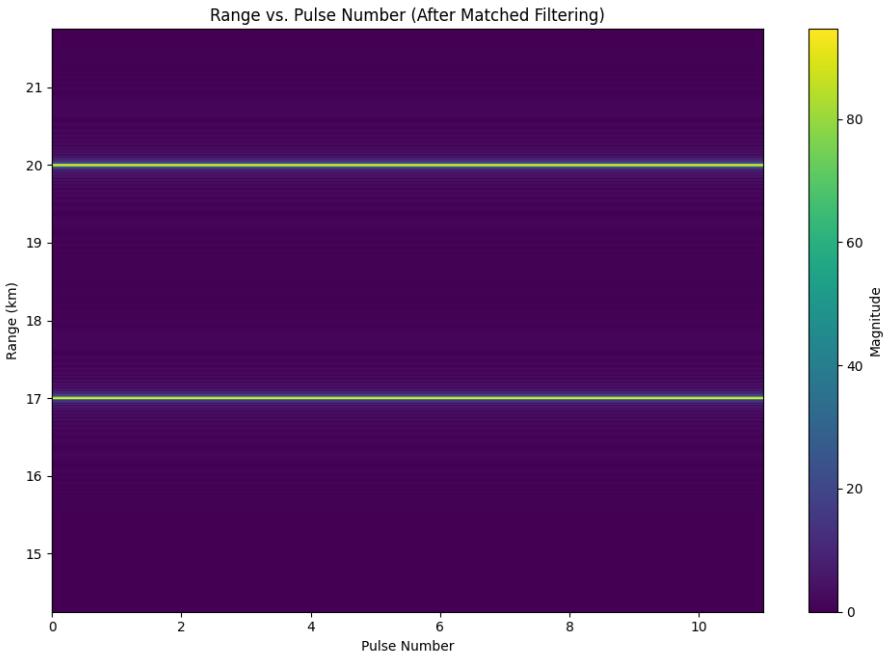


Figure 8: Range vs. Pulse Number (After Matched Filtering). Shows target returns across multiple pulses.

2.c) Range-Doppler Map and Target Detection

Problem Description

This section explores the creation of a Range-Doppler map and automatic target detection. It loads pre-recorded radar data (`rad100.npy`), synthesizes the transmitted chirp, performs matched filtering, applies a 2D Hamming window for sidelobe reduction, and then computes the 2D FFT to generate the Range-Doppler map. Target detection is then performed by applying a dynamically calculated threshold and using DBSCAN clustering to group and identify the peak of each target.

Outputs and Analysis

The script was executed, processing the `rad100.npy` data file. The program first creates the Range-Doppler map and then runs the detection algorithm.

```
Loaded data from 'rad100.npy', shape: (501, 12)
Using dynamically calculated threshold: 14.03 (5 * mean magnitude)
--- Detected Targets ---
Target 1: Range = 4.93 km, Velocity = 47.99 m/s, Peak Magnitude = 14.32
Target 2: Range = 5.21 km, Velocity = 47.63 m/s, Peak Magnitude = 20.01
Target 3: Range = 6.47 km, Velocity = -95.08 m/s, Peak Magnitude = 46.58
Target 4: Range = 8.98 km, Velocity = -71.18 m/s, Peak Magnitude = 68.82
Target 5: Range = 11.62 km, Velocity = 24.08 m/s, Peak Magnitude = 19.26
Target 6: Range = 11.89 km, Velocity = 24.08 m/s, Peak Magnitude = 14.90
```

Range-Doppler Map

The Range-Doppler map is a powerful tool in radar for simultaneously determining the range and velocity of targets. By performing a 2D FFT on the matched-filtered radar data (after applying a Hamming window), the energy from targets is compressed into specific range-Doppler bins. The plot below visualizes this map, where the bright spots correspond to potential targets.

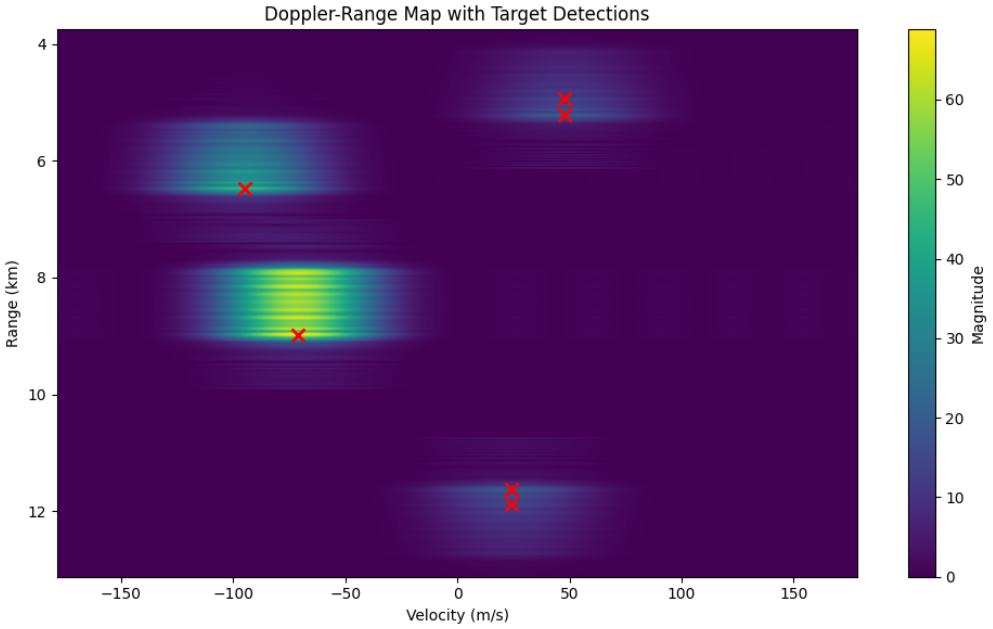


Figure 9: Range-Doppler Map with Target Detections. The red 'rx' marks indicate the final detected target locations after clustering.

Target Detection Results

The detection algorithm uses a dynamic threshold, which was calculated to be 14.03. Points on the map with a magnitude above this value are considered potential detections. These points are then fed into the DBSCAN clustering algorithm to group nearby detections and find the peak of each group, resulting in the final list of targets.

As shown in the console output and marked on the plot, the algorithm successfully identified six distinct targets corresponding to the main areas of high energy in the map:

- Two targets are detected around 5 km in range, moving away from the radar at approximately +48 m/s.
- A strong target is found at 6.47 km range with a velocity of -95.08 m/s (moving towards the radar).
- The most prominent target, with the highest magnitude of 68.82, is located at 8.98 km range and is moving towards the radar at -71.18 m/s.
- Two weaker targets are detected near 11.7 km range, moving away from the radar at +24.08 m/s.

The locations of the red 'x' markers on the plot align perfectly with the high-magnitude regions, demonstrating that the combination of matched filtering, Doppler processing, and DBSCAN-based detection effectively identifies and localizes multiple targets from the raw radar data.

3) Non-contact Automated Cardiac Pulse Measurement by using Video Imaging

Introduction and Methodology

This section details the implementation of a non-contact method for measuring cardiac pulse using video imaging. The approach involves extracting Region of Interest (ROI) from video frames, performing spatial averaging of RGB color channels, detrending and normalizing these signals, applying Independent Component Analysis (ICA) to separate underlying physiological signals, bandpass filtering, and finally analyzing the power spectrum to identify the dominant frequency corresponding to the heart rate.

Constants and Parameters:

- `CASCADE_PATH`: "haarcascade_frontalface_default.xml" (for face detection)
- `ROI_SIZE`: (64, 64) pixels (fixed size for extracted face regions)
- `MIN_BPM`: 45.0 BPM (minimum expected heart rate)
- `MAX_BPM`: 240.0 BPM (maximum expected heart rate)
- `DEFAULT_FPS`: 30.0 Hz (default frames per second for .npy files)

Processing Steps:

1. **Video Loading and ROI Extraction:**
    - The script first attempts to load video data. If a video file (e.g., .avi) is provided, it uses OpenCV to capture frames. If a .npy file is provided (or if no video file is specified, it defaults to 'data1.npy'), it loads the pre-recorded numpy array.
    - A Haar Cascade classifier (`haarcascade_frontalface_default.xml`) is used to detect faces within each frame.
    - The `getBestROI` function tracks the best face across frames, prioritizing continuity (closeness to previous face) and size (largest face if no history).
    - The `getROI` function extracts a specific region (e.g., forehead or cheek) from the detected face bounding box, which is then resized to a fixed `ROI_SIZE` for consistency.
    - Extracted ROIs are converted to RGB format and stored in a 4D numpy array.
2. **Spatial Pooling:**
    - For each frame, the average pixel value for each of the Red, Green, and Blue channels is calculated across the entire ROI. This compresses the spatial information into three time-series signals.

3. **Normalization & Detrending:**

    - Each of the three pooled RGB signals is **detrended** to remove slow-varying trends (e.g., due to illumination changes or slight head movements). This is crucial for isolating the pulsatile components.
    - The detrended signals are then **normalized** to have zero mean and unit variance. This standardizes the amplitude of the signals, which is beneficial for ICA.

4. **Independent Component Analysis (ICA):**

    - FastICA is applied to the detrended and normalized RGB signals. ICA is a computational method that separates a multivariate signal into additive subcomponents, assuming the subcomponents are non-Gaussian and statistically independent. In this context, it aims to separate the subtle color changes due to blood flow (pulse signal) from other independent noise sources (e.g., head movements, ambient light fluctuations).
    - The output of ICA are the "source signals."

5. **Bandpass Filtering of ICA Sources:**

    - The extracted ICA source signals are then passed through a Butterworth bandpass filter. The filter's cutoff frequencies are determined by the `MIN_BPM` and `MAX_BPM` constants, converted to Hz. This step aims to isolate the physiological pulse signal within the expected human heart rate range and remove noise outside this band.

6. **Power Spectrum Analysis and Heart Rate Extraction:**

    - The power spectrum of each *filtered* ICA source signal is calculated using the Fast Fourier Transform (FFT).
    - The frequencies corresponding to the power spectrum are analyzed.
    - The dominant frequency (frequency with the highest power) within the valid heart rate range (45-240 BPM, converted to Hz) is identified for each ICA source.
    - These dominant frequencies are converted back to BPM.
    - The final estimated heart rate is typically chosen from the ICA component that exhibits the most prominent peak within the expected physiological range. The current implementation chooses the component with the highest estimated BPM among those with valid peaks.

Experimental Results

The script was executed for two scenarios: a "normal" heart rate sample and an "after exercising" sample. The following results and plots were observed.

Normal Heart Rate Sample

```
--- Video Data Info (After Preprocessing) ---Shape:  (2388, 64, 64, 3)Data
Type:  uint8Min/Max pixel values:  0/255
   --- Starting Heart Rate Analysis ---Step 1:  Performing spatial pooling
on ROI data...
```

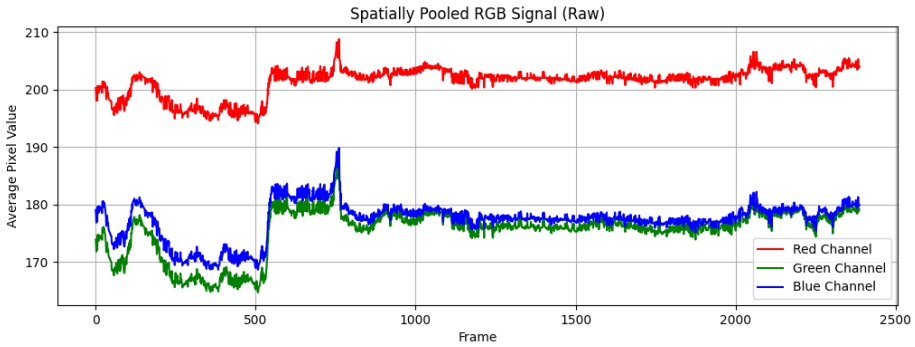


Figure 10: Spatially Pooled RGB Signal (Raw) - Normal Sample

```
Step 2:  Normalizing and Detrending signals...
```

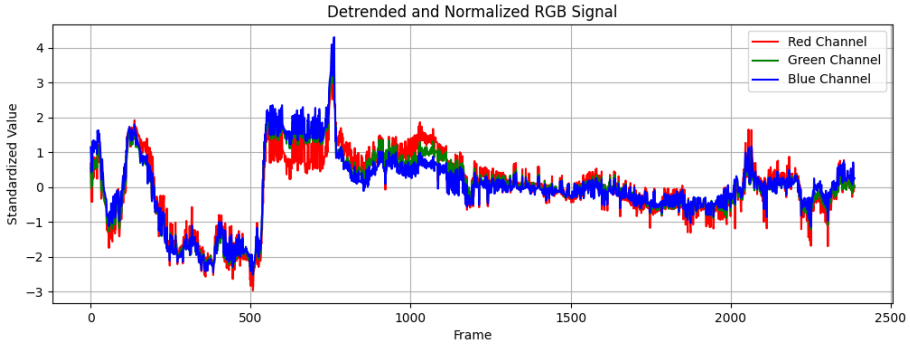


Figure 11: Detrended and Normalized RGB Signal - Normal Sample

```
Step 3:  Applying Independent Component Analysis (ICA)... Step 3a:  Applying
bandpass filter to ICA signals...
```

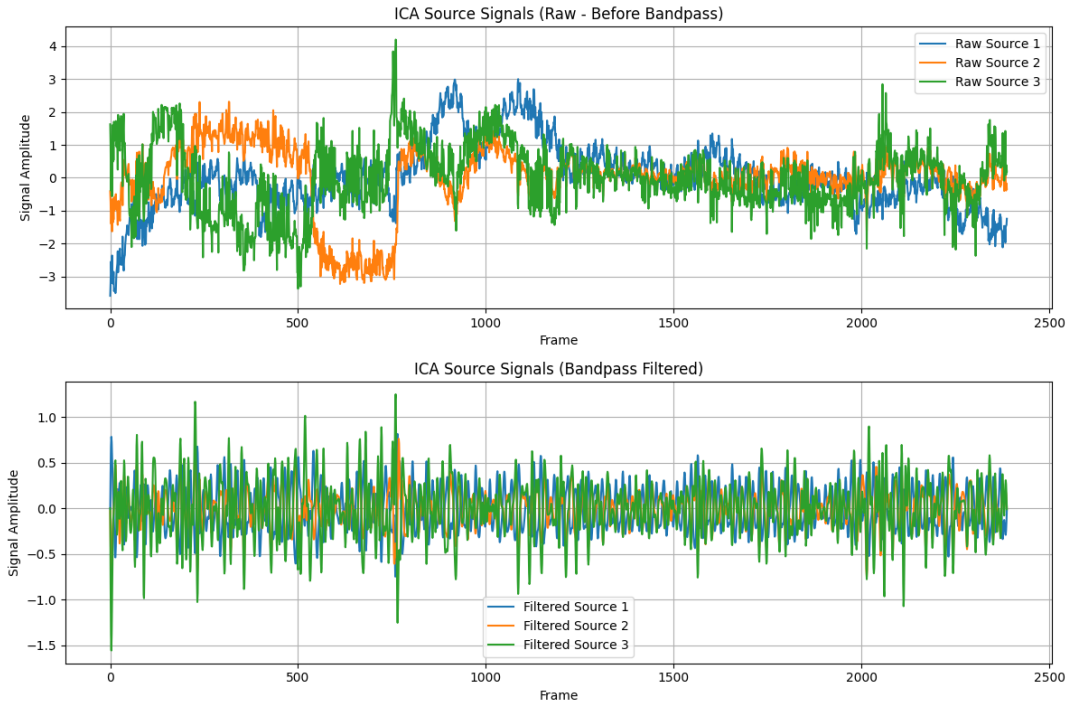


Figure 12: ICA Source Signals (Raw and Filtered) - Normal Sample

```
Step 4:  Calculating power spectrum of filtered source signals...
```

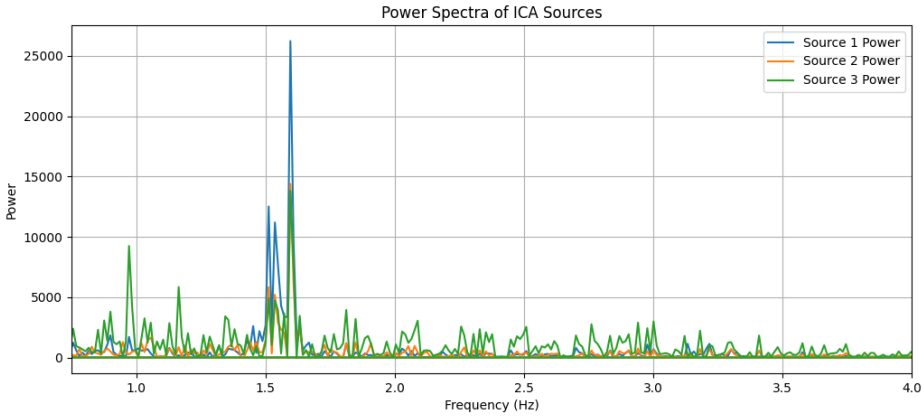


Figure 13: Power Spectra of ICA Sources - Normal Sample

```
   Step 5:  Extracting heart rate from dominant frequency...
   --- Results ---
Heart rate from Component 1:  95.81 BPM (Peak Power:  2.62e+04))
Heart rate from Component 2:  95.81 BPM (Peak Power:  1.44e+04)
Heart rate from Component 3:  95.81 BPM (Peak Power:  1.38e+04)
   ==> Best guess is from Component 1.
==> Estimated Heart Rate:  95.81 BPM
```

After Exercising Sample

```
--- Starting Heart Rate Analysis ---Step 1:  Performing spatial pooling on
ROI data...
```

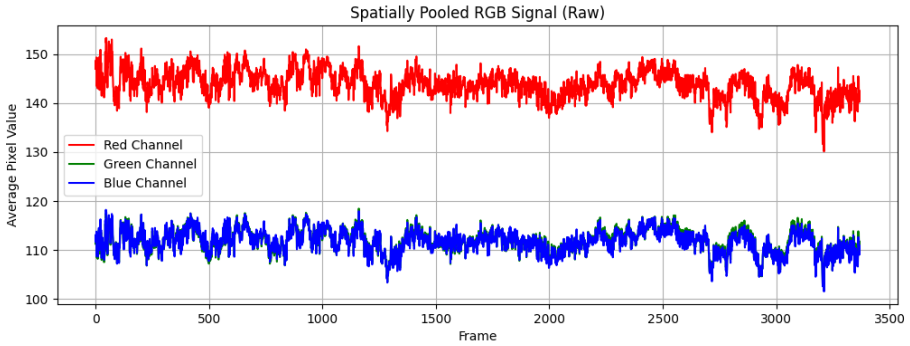


Figure 14: Spatially Pooled RGB Signal (Raw) - After Exercising Sample

```
Step 2:  Normalizing and Detrending signals...
```

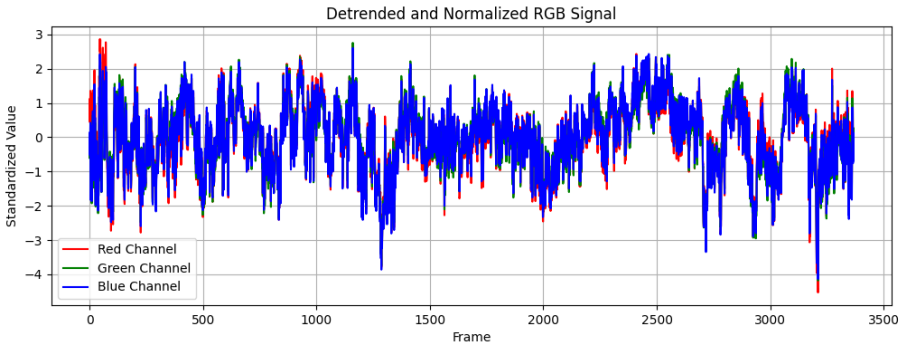


Figure 15: Detrended and Normalized RGB Signal - After Exercising Sample

```
Step 3:  Applying Independent Component Analysis (ICA)... Step 3a:  Applying
bandpass filter to ICA signals...
```

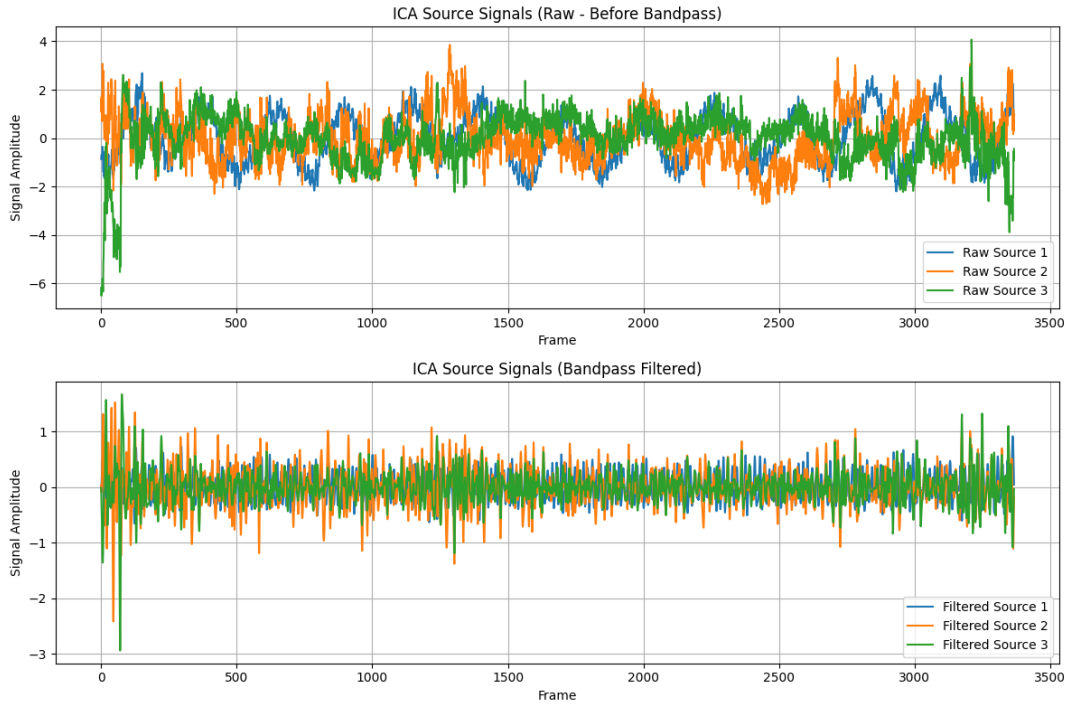


Figure 16: ICA Source Signals (Raw and Filtered) - After Exercising Sample

```
Step 4:  Calculating power spectrum of filtered source signals...
```

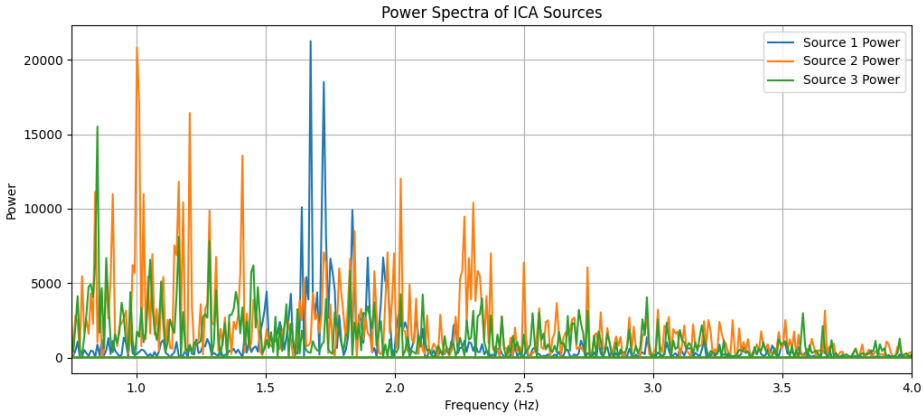


Figure 17: Power Spectra of ICA Sources - After Exercising Sample

```
   Step 5:  Extracting heart rate from dominant frequency...
   --- Results ---
Heart rate from Component 1:  100.50 BPM (Peak Power:  2.13e+04)
Heart rate from Component 2:  60.20 BPM (Peak Power:  2.08e+04)
Heart rate from Component 3:  51.02 BPM (Peak Power:  1.55e+04)
   ==> Best guess is from Component 1.
==> Estimated Heart Rate:  100.50 BPM
```

Discussion and Improvements

The results demonstrate the feasibility of non-contact heart rate measurement using video. For the "normal" sample, an estimated heart rate of 95.81 BPM was obtained, close to the measured heart rate 94 BPM. For the "after exercising" sample, the estimated heart rate increased to **100.50 BPM**, reflecting the measuer 113 BPM. Which gives us %1.77 and %11.5 error rates respectively. The ICA effectively separates the pulse signal from other artifacts, and the bandpass filtering helps isolate the relevant frequency band.

What would you add as further improvements into this implementation?

Several improvements could enhance the robustness, accuracy, and usability of this non-contact heart rate measurement system:

1. **Adaptive ROI Selection and Tracking:**
    - **Multiple ROI Locations:** Instead of just one ROI (e.g., overall face), extracting signals from specific, highly perfused regions like the forehead, cheeks, or nose could yield cleaner pulse signals. Combining these signals or selecting the one with the highest pulsatility could improve accuracy.
    - **Robust Tracking:** While the current `getBestROI` attempts to track, more advanced tracking algorithms (e.g., KCF, CSRT, or deep learning-based trackers like Mediapipe Face Mesh for specific facial landmarks) could provide more stable and precise ROI selection even with significant head movements. This would reduce motion artifacts.
    - **ROI Quality Assessment:** Implement a metric to assess the quality of the extracted ROI (e.g., average pixel intensity, motion variance). If the quality is too low, the frame could be skipped or processed differently.
2. **Improved Preprocessing:**
    - **Color Space Transformation:** Explore other color spaces besides RGB (e.g., YCbCr, HSV, or specialized color spaces like the chrominance-based methods in some research papers) where the pulsatile component might be more distinct and less susceptible to illumination changes. The G-R (Green minus Red) signal is also commonly used.
    - **Advanced Detrending:** While linear detrending is used, more sophisticated detrending methods (e.g., using a sliding window, Savitzky-Golay filter, or polynomial fitting) could better handle non-linear baseline shifts due to complex lighting variations.
    - **Motion Artifact Reduction:** Explicitly include techniques to mitigate motion artifacts, such as using optical flow to estimate and compensate for subtle head movements, or regressing out movement-correlated signals before ICA.
3. **Enhanced Signal Processing:**
    - **Frequency Domain Filtering:** Instead of direct time-domain bandpass filtering, applying a spectral filter after FFT could allow for more precise frequency isolation and potentially better handling of noise.

- **Power Spectrum Smoothing:** Apply a smoothing window (e.g., Hamming, Hanning) to the time-domain signal before FFT, or smooth the resulting power spectrum, to reduce spectral leakage and make the dominant peak more prominent.
- **Multiple ICA Components Selection:** The current approach takes the max BPM from all components. A more robust selection could involve selecting the component with the highest power in the expected heart rate band, or using a quality metric for each component (e.g., kurtosis for non-Gaussianity, which ICA maximizes).
- **Ensemble Averaging/Tracking:** For real-time applications, track the dominant frequency over a short sliding window, and use averaging or a Kalman filter to smooth the heart rate estimate and make it less susceptible to momentary noise.

4. **Handling Edge Cases and Robustness:**

   - **Low Light/Poor Quality Video:** The system's performance will degrade significantly in low-light conditions or with poor video quality. Implementing a quality check on the video input and providing feedback to the user would be beneficial.
   - **Occlusions:** The current system might fail if the face or ROI is frequently occluded. Future work could integrate occlusion detection and handling (e.g., using prior frames or alternative ROIs).
   - **Inter-subject Variability:** Heart rate signals can vary significantly between individuals. The system might need calibration or adaptive algorithms to perform optimally across diverse subjects.

5. **User Interface and Feedback:**

   - For practical use, a graphical user interface (GUI) displaying the real-time heart rate, a confidence indicator, and perhaps a visualization of the ROI and signal quality would be valuable.

By incorporating these improvements, the non-contact cardiac pulse measurement system could become more accurate, robust, and applicable in a wider range of real-world scenarios.