Time Synchronous Averaging for Condition-Based Monitoring

This notebook demonstrates the use of time synchronous averaging (TSA) for extracting periodic components from rotating machinery signals. TSA is a fundamental technique in condition-based monitoring for enhancing gear mesh frequencies and suppressing random bearing noise.

Background

Time synchronous averaging is used to:

- Extract periodic components from noisy machinery signals
- Enhance gear mesh frequencies
- Suppress random noise and bearing fault signatures
- Prepare signals for further analysis (e.g., discrete/random separation)

The technique requires signals to be resampled to the angular domain first (samples per revolution), then averages multiple revolutions to enhance periodic content.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import shmtools

# Set random seed for reproducible results
np.random.seed(42)
print("SHMTools Time Synchronous Averaging Demo")
print("=========================================")
```

```
SHMTools Time Synchronous Averaging Demo
=========================================
```

```
/Users/eric/repo/shm/shmtools-python/venv/lib/python3.9/site-packages/urllib
3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports OpenSSL 1.1.1
+, currently the 'ssl' module is compiled with 'LibreSSL 2.8.3'. See: http
s://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

Generate Synthetic Machinery Signal

We'll create a synthetic signal representing:

- Gear mesh frequency (periodic component)
- Bearing fault impulses (random component)
- Background noise

In [2]:
```python
# Signal parameters
samples_per_rev = 256  # Angular resolution
n_revolutions = 20     # Number of revolutions to simulate
n_channels = 1         # Single accelerometer
n_instances = 2        # Healthy vs damaged bearing

# Create angular time vector
n_samples = samples_per_rev * n_revolutions
theta = np.linspace(0, n_revolutions * 2 * np.pi, n_samples)

# Gear mesh frequency components (periodic)
# Fundamental gear mesh + harmonics
gear_signal = (2.0 * np.sin(10 * theta) +      # 10th order (gear mesh)
               1.0 * np.sin(20 * theta) +      # 2nd harmonic
               0.5 * np.sin(30 * theta))       # 3rd harmonic

# Random bearing fault impulses (for damaged case)
bearing_fault_rate = 15.3  # Ball pass frequency outer race
bearing_impulses = np.zeros_like(theta)

# Add random impulses at approximately bearing fault frequency
fault_phases = np.arange(0, n_revolutions * 2 * np.pi, 2 * np.pi / bearing_f
for phase in fault_phases:
    # Add some randomness to impulse timing and amplitude
    actual_phase = phase + 0.1 * np.random.randn()
    impulse_idx = np.argmin(np.abs(theta - actual_phase))
    if impulse_idx < len(bearing_impulses) - 10:
        # Create decaying impulse
        decay = np.exp(-0.5 * np.arange(10))
        amplitude = 1.0 + 0.3 * np.random.randn()
        bearing_impulses[impulse_idx:impulse_idx + 10] += amplitude * decay

# Background noise
noise = 0.3 * np.random.randn(n_samples)

# Create signal matrix
X_angular = np.zeros((n_samples, n_channels, n_instances))

# Instance 0: Healthy (gear + noise only)
X_angular[:, 0, 0] = gear_signal + 0.2 * np.random.randn(n_samples)

# Instance 1: Damaged (gear + bearing faults + noise)
X_angular[:, 0, 1] = gear_signal + 0.8 * bearing_impulses + 0.3 * np.random.

print(f"Generated signal matrix: {X_angular.shape}")
print(f"Signal length: {n_samples} samples ({n_revolutions} revolutions)")
print(f"Angular resolution: {samples_per_rev} samples per revolution")
```

```
Generated signal matrix: (5120, 1, 2)
Signal length: 5120 samples (20 revolutions)
Angular resolution: 256 samples per revolution
```

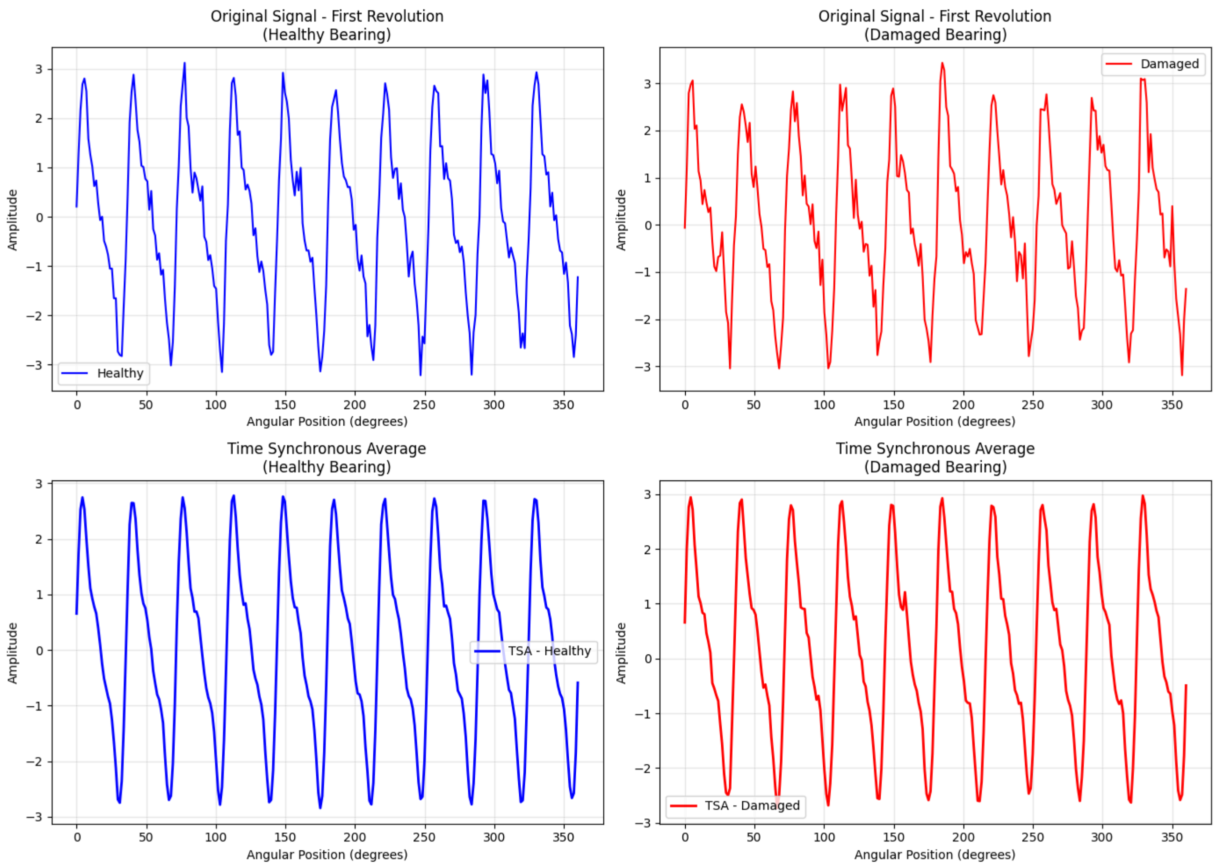
Apply Time Synchronous Averaging

In [3]:
```python
# Compute time synchronous average
X_tsa = shmtools.time_sync_avg_shm(X_angular, samples_per_rev)

print(f"TSA result shape: {X_tsa.shape}")
print(f"Reduced from {X_angular.shape[0]} samples to {X_tsa.shape[0]} sample
print(f"Averaging over {n_revolutions} revolutions")
```

```
TSA result shape: (256, 1, 2)
Reduced from 5120 samples to 256 samples
Averaging over 20 revolutions
```

Visualize Results

In [4]:
```python
# Create angular position vector for one revolution
theta_rev = np.linspace(0, 2 * np.pi, samples_per_rev)
theta_deg = theta_rev * 180 / np.pi

# Plot comparison of original signals vs TSA
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Original signals - first revolution only
axes[0, 0].plot(theta_deg, X_angular[:samples_per_rev, 0, 0], 'b-', linewidt
axes[0, 0].set_title('Original Signal - First Revolution\n(Healthy Bearing)'
axes[0, 0].set_xlabel('Angular Position (degrees)')
axes[0, 0].set_ylabel('Amplitude')
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].legend()

axes[0, 1].plot(theta_deg, X_angular[:samples_per_rev, 0, 1], 'r-', linewidt
axes[0, 1].set_title('Original Signal - First Revolution\n(Damaged Bearing)'
axes[0, 1].set_xlabel('Angular Position (degrees)')
axes[0, 1].set_ylabel('Amplitude')
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].legend()

# Time synchronous averaged signals
axes[1, 0].plot(theta_deg, X_tsa[:, 0, 0], 'b-', linewidth=2, label='TSA - H
axes[1, 0].set_title('Time Synchronous Average\n(Healthy Bearing)')
axes[1, 0].set_xlabel('Angular Position (degrees)')
axes[1, 0].set_ylabel('Amplitude')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].legend()

axes[1, 1].plot(theta_deg, X_tsa[:, 0, 1], 'r-', linewidth=2, label='TSA - D
axes[1, 1].set_title('Time Synchronous Average\n(Damaged Bearing)')
axes[1, 1].set_xlabel('Angular Position (degrees)')
axes[1, 1].set_ylabel('Amplitude')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].legend()

plt.tight_layout()
plt.show()
```

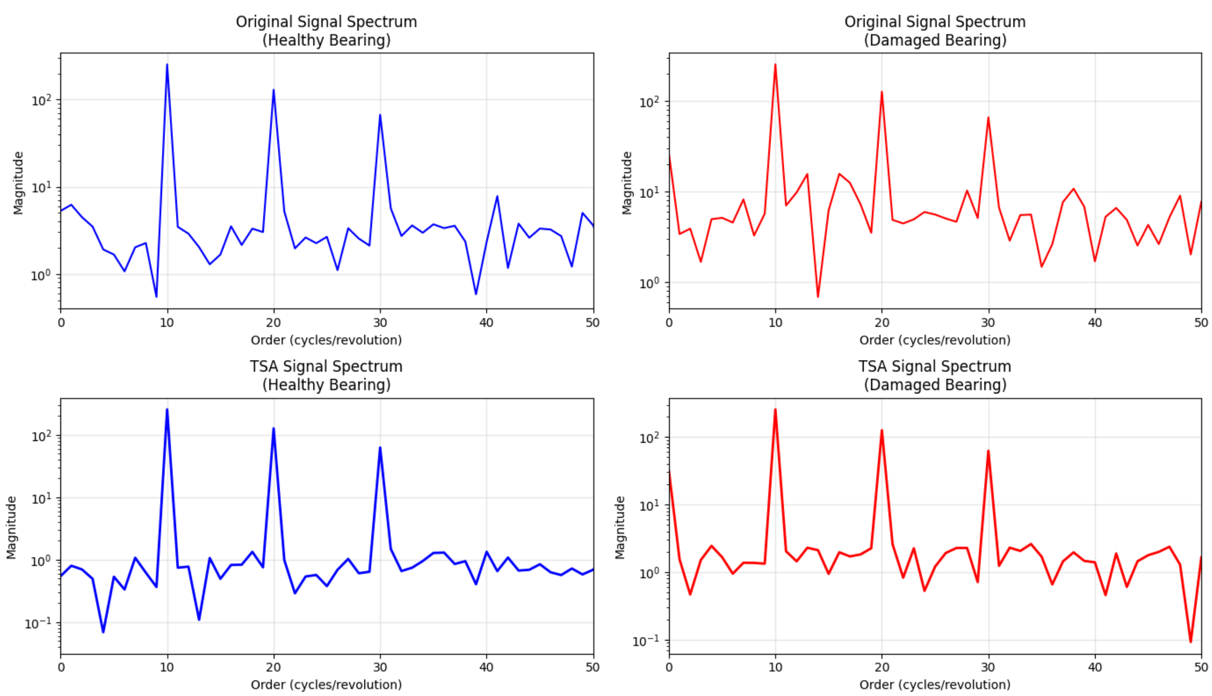


Compare Frequency Content

In [5]:

```
# Compute frequency spectra
def compute_spectrum(signal, samples_per_rev):
    """Compute spectrum in orders (cycles per revolution)"""
    fft_result = np.fft.fft(signal)
    magnitude = np.abs(fft_result[:len(fft_result)//2])
    orders = np.arange(len(magnitude)) * samples_per_rev / len(signal)
    return orders, magnitude

# Compute spectra for original and TSA signals
fig, axes = plt.subplots(2, 2, figsize=(14, 8))

# Original signal spectra (first revolution)
orders_orig, mag_orig_healthy = compute_spectrum(X_angular[:samples_per_rev,
orders_orig, mag_orig_damaged = compute_spectrum(X_angular[:samples_per_rev,

axes[0, 0].semilogy(orders_orig, mag_orig_healthy, 'b-', linewidth=1.5)
axes[0, 0].set_title('Original Signal Spectrum\n(Healthy Bearing)')
axes[0, 0].set_xlabel('Order (cycles/revolution)')
axes[0, 0].set_ylabel('Magnitude')
axes[0, 0].grid(True, alpha=0.3)
axes[0, 0].set_xlim([0, 50])

axes[0, 1].semilogy(orders_orig, mag_orig_damaged, 'r-', linewidth=1.5)
axes[0, 1].set_title('Original Signal Spectrum\n(Damaged Bearing)')
axes[0, 1].set_xlabel('Order (cycles/revolution)')
axes[0, 1].set_ylabel('Magnitude')
```

```python
axes[0, 1].grid(True, alpha=0.3)
axes[0, 1].set_xlim([0, 50])

# TSA signal spectra
orders_tsa, mag_tsa_healthy = compute_spectrum(X_tsa[:, 0, 0], samples_per_r
orders_tsa, mag_tsa_damaged = compute_spectrum(X_tsa[:, 0, 1], samples_per_r

axes[1, 0].semilogy(orders_tsa, mag_tsa_healthy, 'b-', linewidth=2)
axes[1, 0].set_title('TSA Signal Spectrum\n(Healthy Bearing)')
axes[1, 0].set_xlabel('Order (cycles/revolution)')
axes[1, 0].set_ylabel('Magnitude')
axes[1, 0].grid(True, alpha=0.3)
axes[1, 0].set_xlim([0, 50])

axes[1, 1].semilogy(orders_tsa, mag_tsa_damaged, 'r-', linewidth=2)
axes[1, 1].set_title('TSA Signal Spectrum\n(Damaged Bearing)')
axes[1, 1].set_xlabel('Order (cycles/revolution)')
axes[1, 1].set_ylabel('Magnitude')
axes[1, 1].grid(True, alpha=0.3)
axes[1, 1].set_xlim([0, 50])

plt.tight_layout()
plt.show()

print("Key observations:")
print("1. TSA enhances periodic components (gear mesh at orders 10, 20, 30)"
print("2. TSA suppresses random noise and bearing fault impulses")
print("3. Healthy and damaged signals show similar TSA results (gear dominat
print("4. For bearing fault detection, the random component (original - TSA)
```



```
Key observations:
1. TSA enhances periodic components (gear mesh at orders 10, 20, 30)
2. TSA suppresses random noise and bearing fault impulses
3. Healthy and damaged signals show similar TSA results (gear dominates)
4. For bearing fault detection, the random component (original - TSA) is ana
lyzed
```

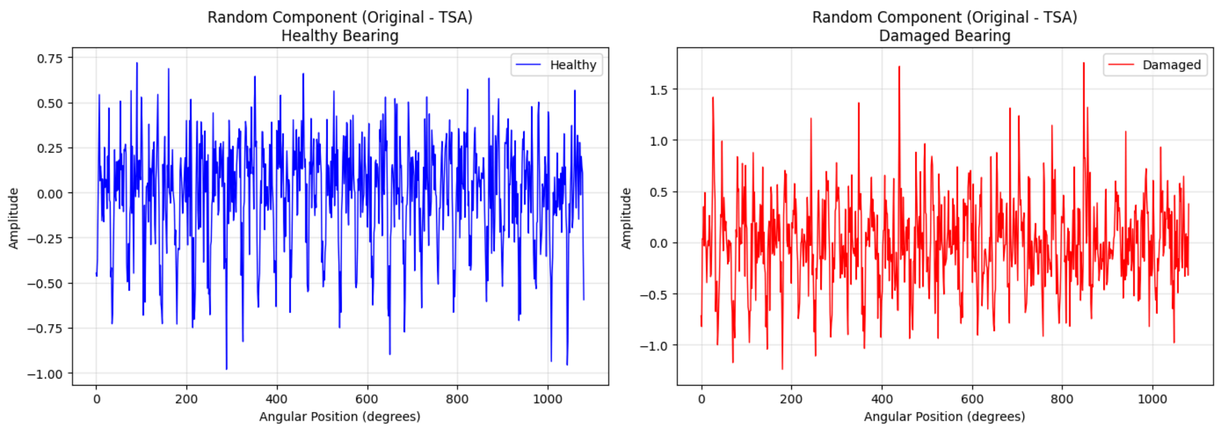
Extract Random Component

The random component (original signal minus TSA) contains the bearing fault information.

In [6]:
```
# Compute random component by subtracting TSA from original
# Note: We need to replicate TSA for each revolution
X_tsa_replicated = np.tile(X_tsa, (n_revolutions, 1, 1))
X_random = X_angular - X_tsa_replicated

# Compute RMS values to quantify the effect
rms_original_healthy = np.sqrt(np.mean(X_angular[:, 0, 0]**2))
rms_original_damaged = np.sqrt(np.mean(X_angular[:, 0, 1]**2))
rms_tsa_healthy = np.sqrt(np.mean(X_tsa[:, 0, 0]**2))
rms_tsa_damaged = np.sqrt(np.mean(X_tsa[:, 0, 1]**2))
rms_random_healthy = np.sqrt(np.mean(X_random[:, 0, 0]**2))
rms_random_damaged = np.sqrt(np.mean(X_random[:, 0, 1]**2))

print("RMS Analysis:")
print(f"Original signal RMS - Healthy: {rms_original_healthy:.3f}")
print(f"Original signal RMS - Damaged: {rms_original_damaged:.3f}")
print(f"TSA signal RMS - Healthy: {rms_tsa_healthy:.3f}")
print(f"TSA signal RMS - Damaged: {rms_tsa_damaged:.3f}")
print(f"Random component RMS - Healthy: {rms_random_healthy:.3f}")
print(f"Random component RMS - Damaged: {rms_random_damaged:.3f}")
print(f"Damage detection ratio (Random): {rms_random_damaged/rms_random_heal

# Plot random components
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot first few revolutions of random component
n_plot = 3 * samples_per_rev
theta_plot = np.linspace(0, 3 * 2 * np.pi, n_plot) * 180 / np.pi

axes[0].plot(theta_plot, X_random[:n_plot, 0, 0], 'b-', linewidth=1, label='
axes[0].set_title('Random Component (Original - TSA)\nHealthy Bearing')
axes[0].set_xlabel('Angular Position (degrees)')
axes[0].set_ylabel('Amplitude')
axes[0].grid(True, alpha=0.3)
axes[0].legend()

axes[1].plot(theta_plot, X_random[:n_plot, 0, 1], 'r-', linewidth=1, label='
axes[1].set_title('Random Component (Original - TSA)\nDamaged Bearing')
axes[1].set_xlabel('Angular Position (degrees)')
axes[1].set_ylabel('Amplitude')
axes[1].grid(True, alpha=0.3)
axes[1].legend()
```

```
plt.tight_layout()
plt.show()
```

```
RMS Analysis:
Original signal RMS - Healthy: 1.635
Original signal RMS - Damaged: 1.670
TSA signal RMS - Healthy: 1.615
TSA signal RMS - Damaged: 1.620
Random component RMS - Healthy: 0.252
Random component RMS - Damaged: 0.407
Damage detection ratio (Random): 1.62
```



Summary

This demonstration shows the effectiveness of time synchronous averaging for condition-based monitoring:

Key Results:

1. **Periodic Enhancement**: TSA successfully extracts and enhances periodic gear mesh components
2. **Noise Suppression**: Random noise and bearing fault impulses are significantly reduced in the TSA signal
3. **Damage Isolation**: The random component (original - TSA) isolates bearing fault signatures
4. **Quantitative Analysis**: RMS analysis shows clear differences between healthy and damaged conditions in the random component

Applications:

- **Gear Analysis**: Use TSA signal to analyze gear mesh frequencies and detect gear faults
- **Bearing Analysis**: Use random component to detect bearing faults and compute damage indicators

- **Preprocessing**: TSA serves as preprocessing for advanced techniques like discrete/random separation

Next Steps:

For complete condition-based monitoring analysis, additional techniques would include:

- Angular resampling from time domain signals
- Discrete/random separation for more sophisticated gear/bearing isolation
- Spectral kurtosis analysis for optimal frequency band selection
- Envelope analysis and bearing fault frequency identification

The implemented `time_sync_avg_shm` function provides a solid foundation for these advanced CBM techniques.