```
// src/modulation/modulation.ts

import { WnspFrame } from "../protocol/frameTypes";

/**
 * A timeline segment describes how a frame will be rendered as light
 * between tStartMs and tEndMs.
 */
export type TimelineSegment = {
  tStartMs: number;
  tEndMs: number;
  wavelengthNm: number;
  intensityLevel: number;
};

/**
 * Convert a sequence of frames into a simple linear timeline.
 * Each frame is assigned a fixed duration.
 */
export function framesToTimeline(
  frames: WnspFrame[],
  frameDurationMs: number
): TimelineSegment[] {
  const segments: TimelineSegment[] = [];

  let currentStart = 0;
  for (const frame of frames) {
    const segment: TimelineSegment = {
      tStartMs: currentStart,
      tEndMs: currentStart + frameDurationMs,
      wavelengthNm: frame.wavelengthNm,
      intensityLevel: frame.intensityLevel,
    };
    segments.push(segment);
    currentStart += frameDurationMs;
  }

  return segments;
}

/**
 * A simple sample representation of detected optical state.
 */
export type OpticalSample = {
  tMs: number;
  wavelengthNm: number;
  intensityLevel: number;
};

/**
 * Conceptual demodulation: group samples into time
buckets corresponding
 * to frames and reconstruct approximate frames.
 *
 * In v1.0 this is primitive: we assume samples are already
aligned and
 * simply average wavelength & intensity per bucket.
 */
export function demodulateTimelineToFrames(
  samples: OpticalSample[],
  frameDurationMs: number
): WnspFrame[] {
  if (samples.length === 0) return [];
  const frames: WnspFrame[] = [];

  const startTime = samples[0].tMs;
  const endTime = samples[samples.length - 1].tMs;
  const frameCount = Math.ceil((endTime - startTime) /
frameDurationMs);

  for (let i = 0; i < frameCount; i++) {
    const bucketStart = startTime + i * frameDurationMs;
    const bucketEnd = bucketStart + frameDurationMs;

    const bucketSamples = samples.filter(
      (s) => s.tMs >= bucketStart && s.tMs < bucketEnd
    );

    if (bucketSamples.length === 0) continue;

    const avgWl =
      bucketSamples.reduce((sum, s) => sum +
s.wavelengthNm, 0) /
      bucketSamples.length;
    const avgIntensity =
      bucketSamples.reduce((sum, s) => sum +
s.intensityLevel, 0) /
      bucketSamples.length;
    const wavelengthNm = Math.round(avgWl);
    const intensityLevel = Math.round(avgIntensity);

    // For v1.0, we cannot reconstruct payloadBit and checksum reliably from samples,
    // so we set them to neutral values and allow higher-level logic to decide.
    const frame: WnspFrame = {
      sync: 0xaa,
      wavelengthNm,
      intensityLevel,
      checksum: 0,
      payloadBit: 0,
      timestampMs: bucketStart,
    };

    frames.push(frame);
  }

  return frames;
}
```