University of Southern California

ECE 105: Introduction to Electrical Engineering

Guest Lecture

Wireless Communication: Symbol Generation & Sensor Fusion

Walter V. Unglaub

11/20/2025

Arduino code jointly developed by Gary Chen, Samuel Ye, and Walter Unglaub and available for download at: https://github.com/Garbear008/Sensor-Fusion-Project.

Sensor Fusion: An Introduction

USC University of Southern California

- What is **sensor fusion**?
  - The art and science of combining data from multiple sensors to produce an *estimate* that's better than any single sensor *alone*.
    - More accurate, stable, complete, and therefore *robust*.
    - "Teamwork for sensors"

https://www.youtube.com/watch?v=hyGJBV1xnJI

- Why perform sensor fusion?
  - ✓ Completeness
  - ✓ Noise & drift
  - ✓ Redundancy & safety

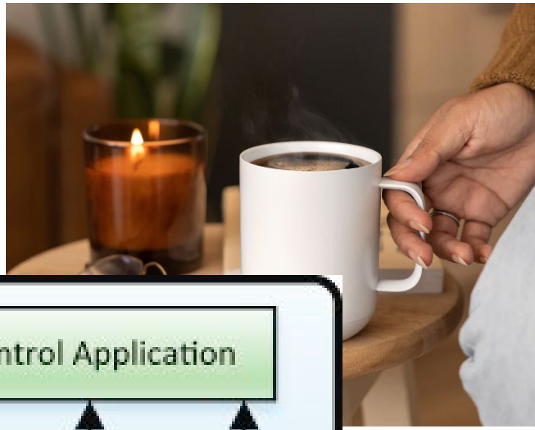
- Everyday examples:
  - **Smartphones:** Accelerometer + gyroscope + magnetometer (screen rotation, navigation)
  - **Cars & robots:** Cameras + lidar + radar for detecting lanes, obstacles, and distances.
  - **Wearables:** Optical heart-rate + accelerometer to filter motion artifacts.

Sensor Fusion: A relatable mental model + core behaviors

USC University of Southern California

❖ Imagine a coffee mug filled with hot coffee; you try telling the temperature by:

- Touching the cup (noisy but instant feel), and
- Watching the steam (slow cue but less noisy)



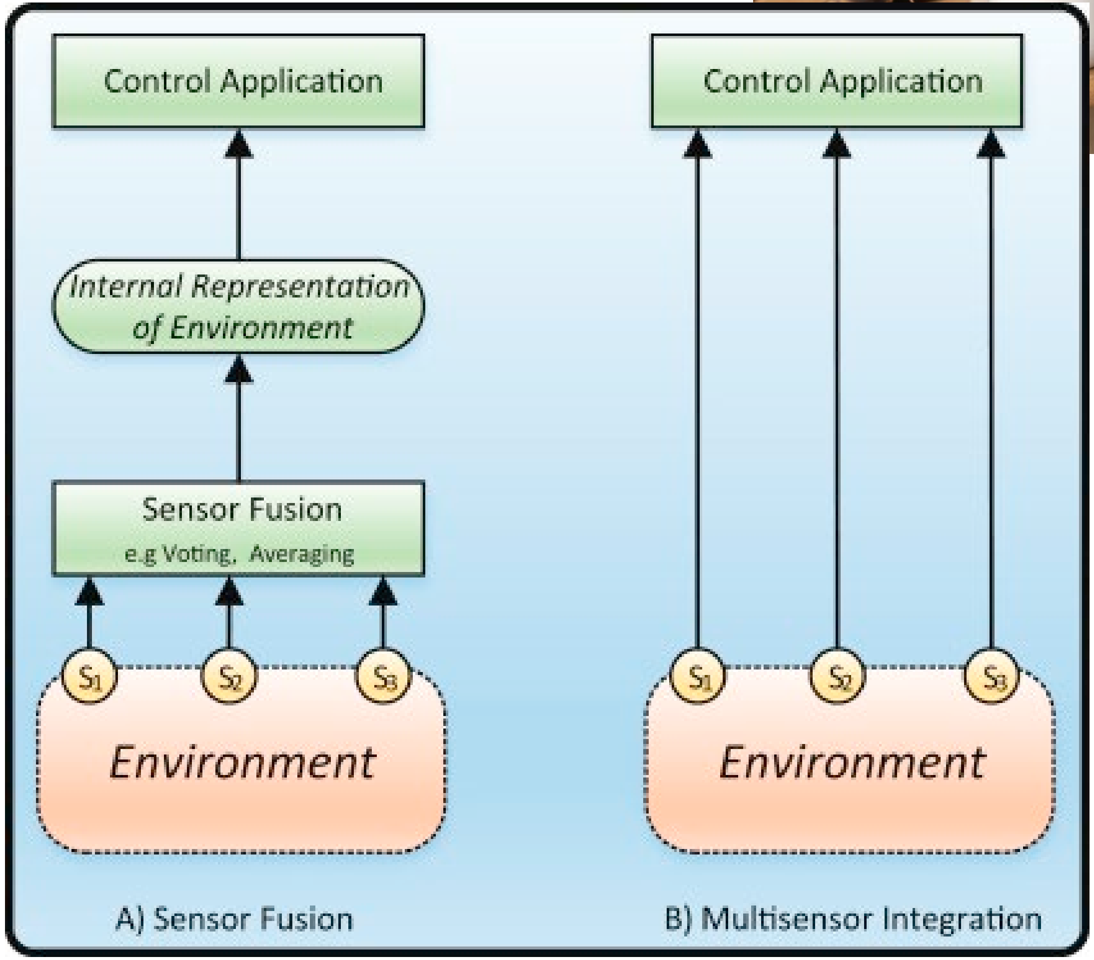
➢ A good estimate of temperature blends both methods: *quick responsiveness* from your touch and *stable confirmation* from the steam.

➢ Sensor fusion *formalizes* this idea with math.

❖ **Core** sensor behaviors (the various parts):

- Bias (offset)
- Noise
- Drift
- Resolution & range
- Bandwidth/latency
- Sampling rate

❖ **Key idea:** Fusion works only as well as your understanding of each sensor's *imperfections*.



https://doi.org/10.1016/B978-0-12-811153-6.00001-4

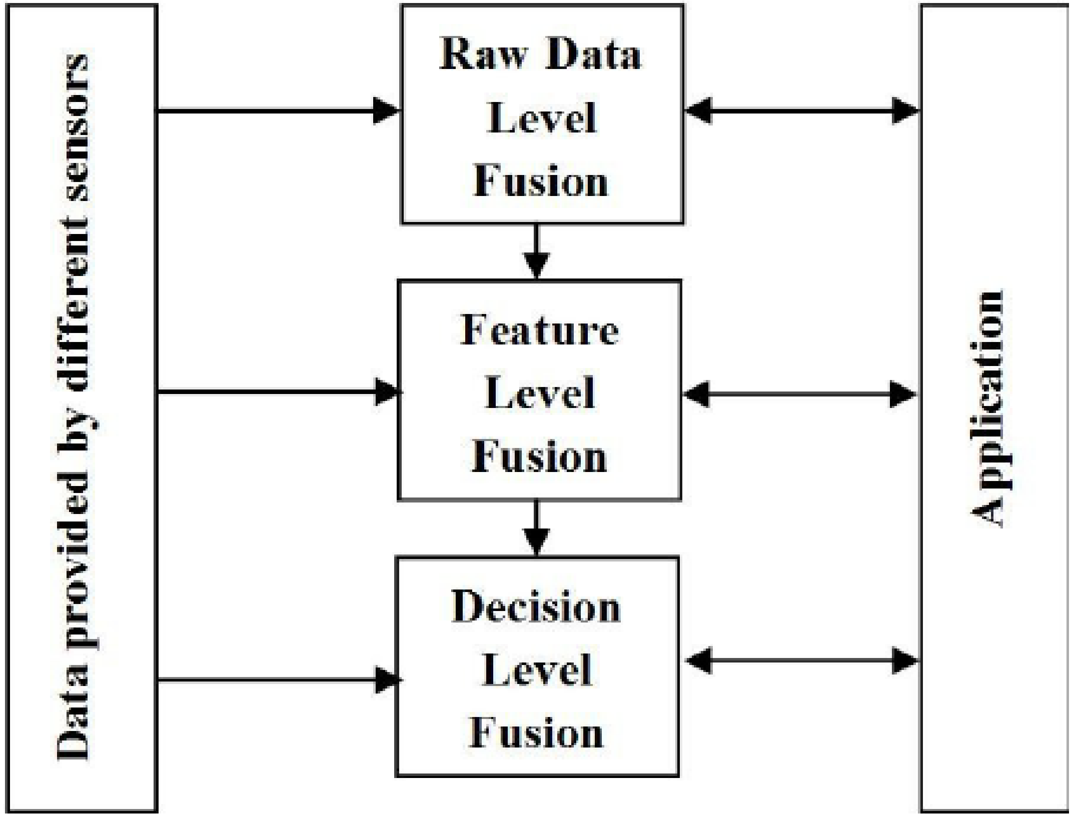
Levels of fusion and alignment of signal data

USC University of Southern California

❖ Levels of fusion:

1. **Raw data level:** Combine measurements directly.
2. **Feature level:** Combine extracted features .
3. **Decision level:** Combine independent decisions.

For first projects, you'll usually start at the **raw** level.



https://tinyurl.com/yc5bupfx

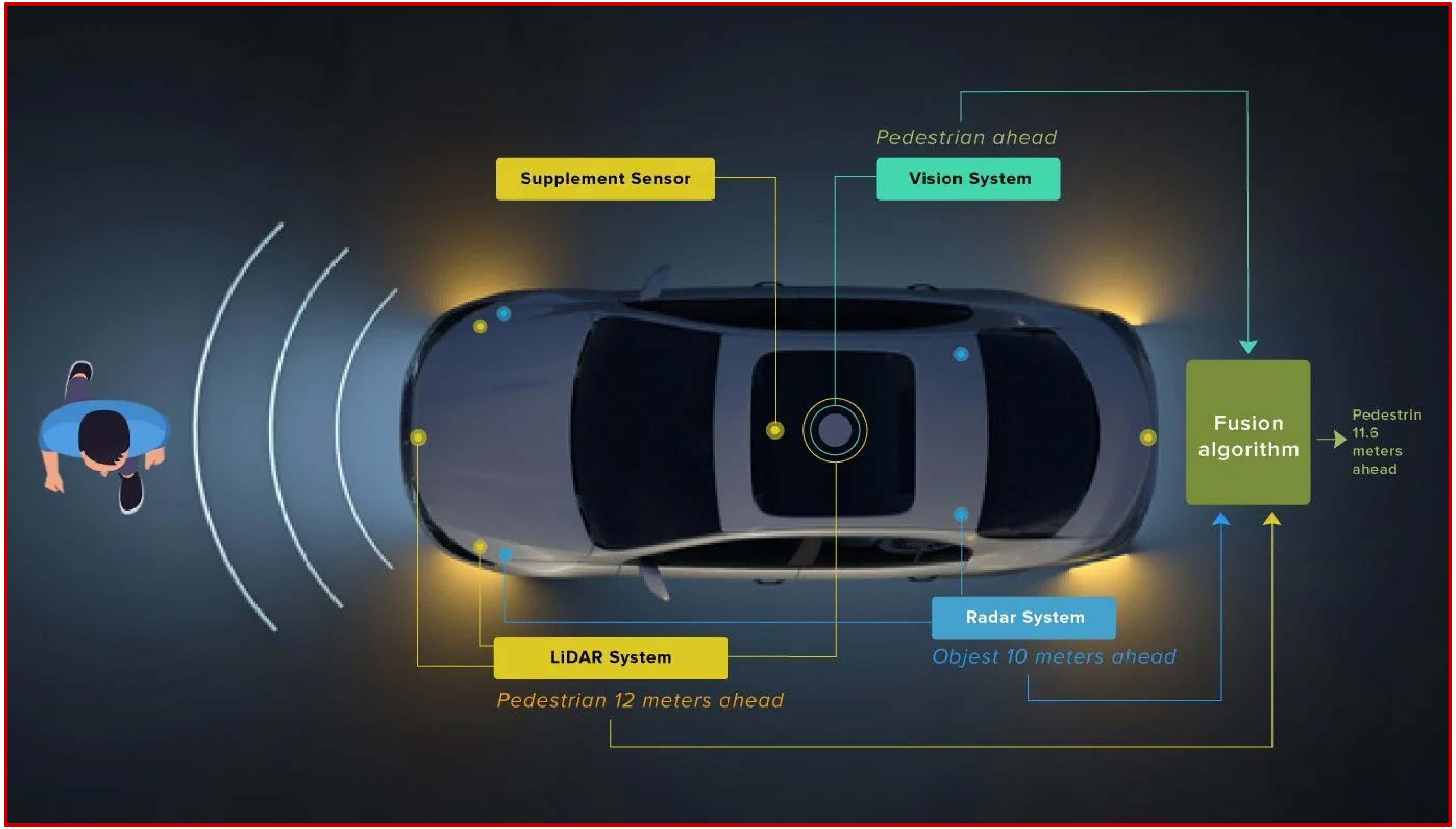
❖ Before blending data, you must ***align*** it:

✓ **Coordinate frames:** Express all measurements in a common frame.

✓ **Calibration:** Determine scale factors, axis misalignment, and bias.

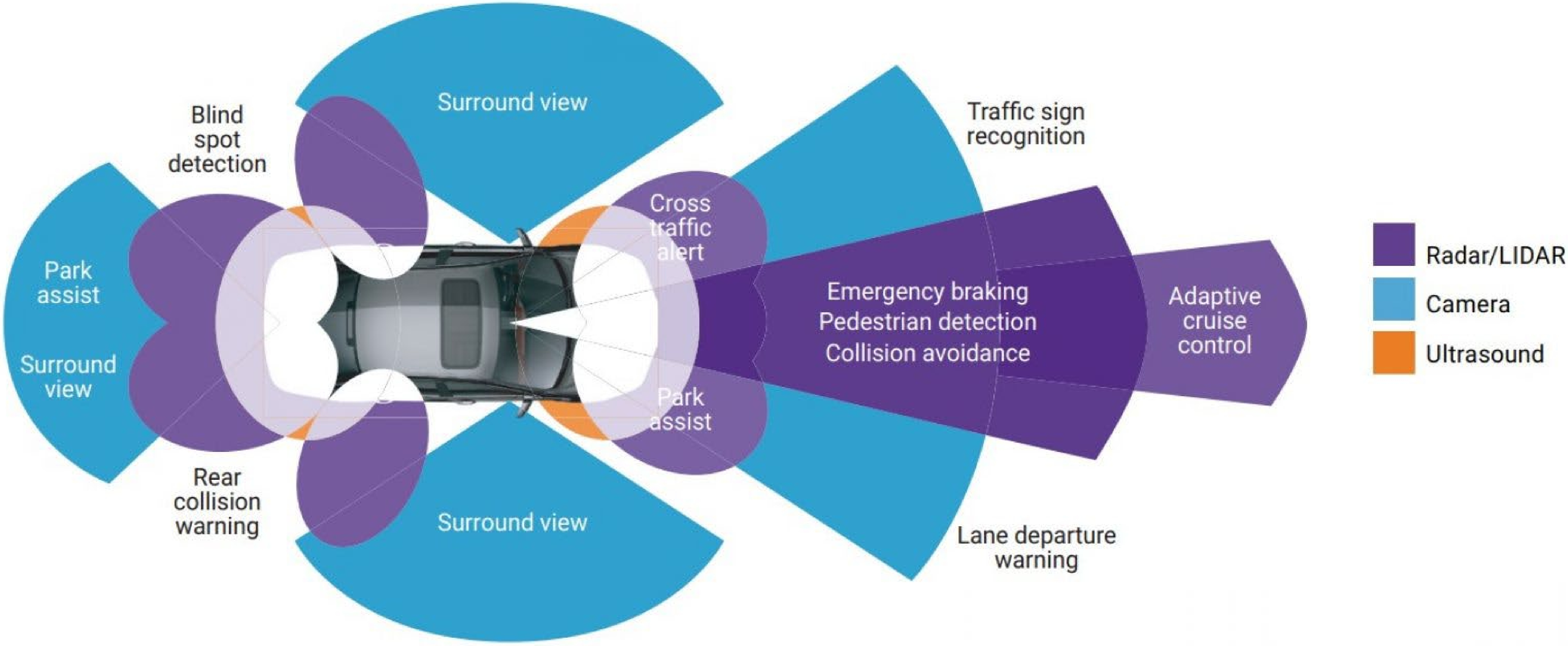
✓ **Time synchronization:** If two sensors don't share a clock, you must align timestamps; otherwise, fusion can "smear" fast motions.

Multi-sensor data fusion in autonomous vehicles

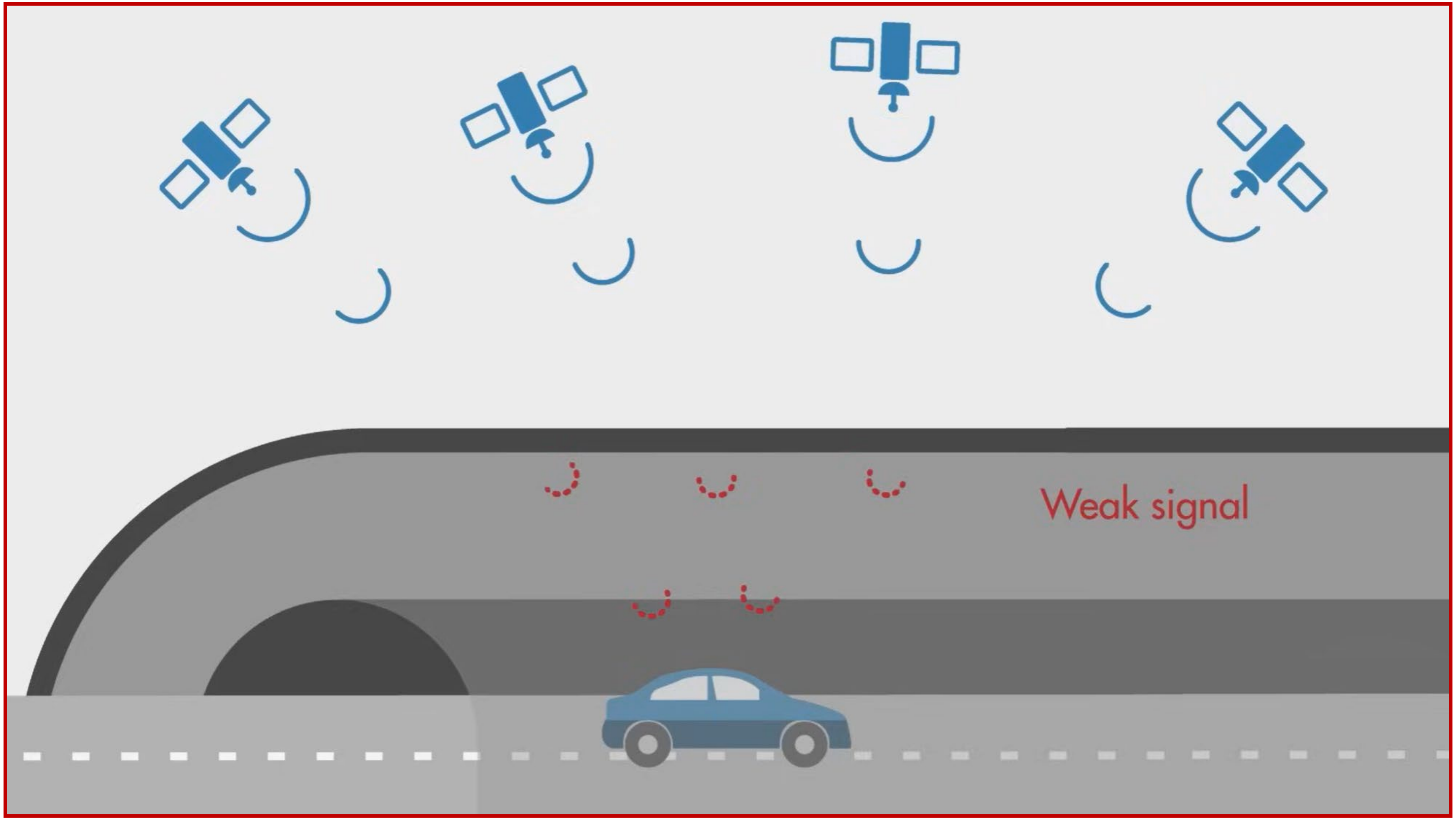


https://www.digitaldividedata.com/blog/multi-sensor-data-fusion-in-autonomous-vehicles

Multi-sensor data fusion in autonomous vehicles

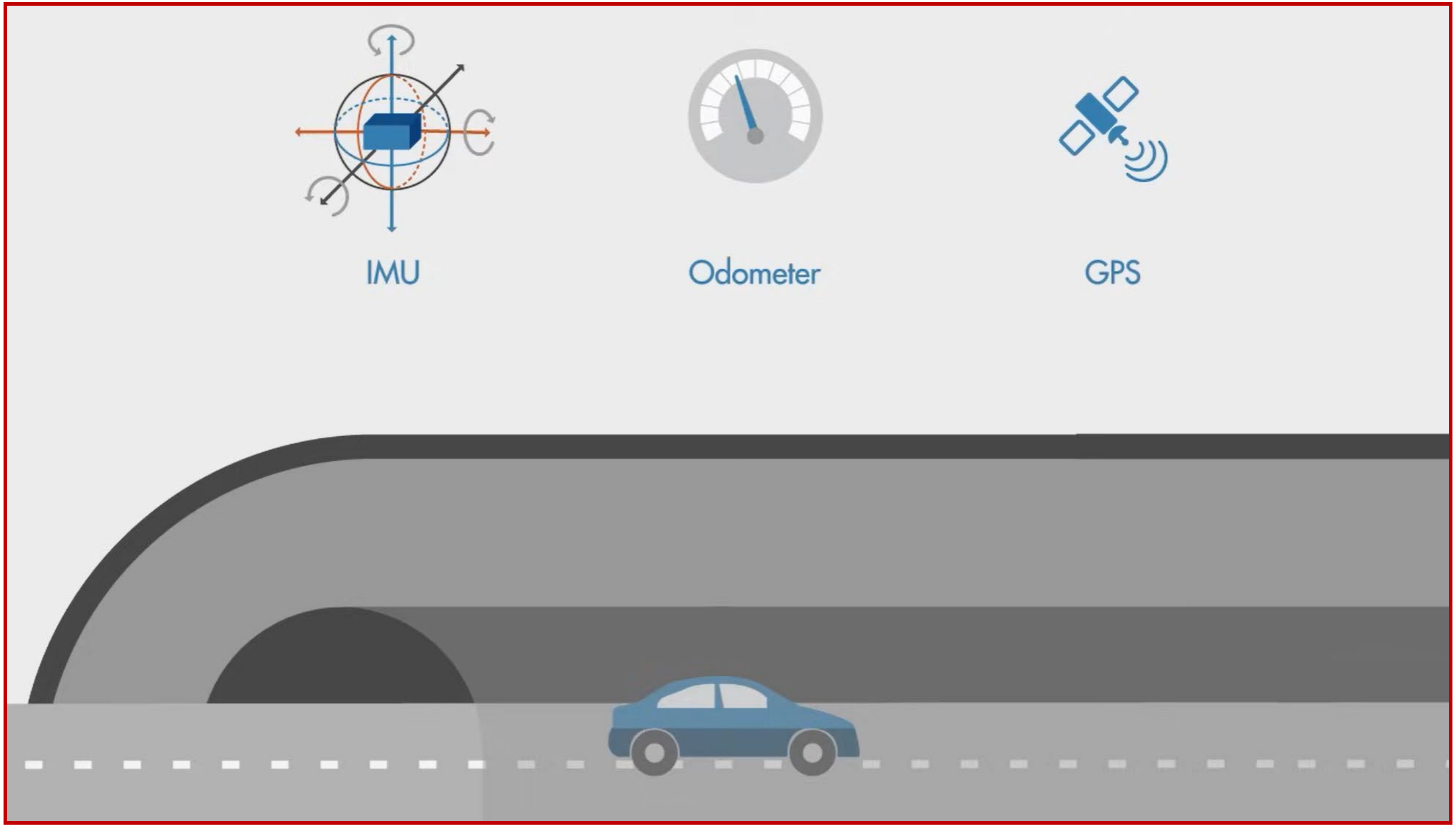


Kalman Filtering (aka Linear Quadratic Estimation)



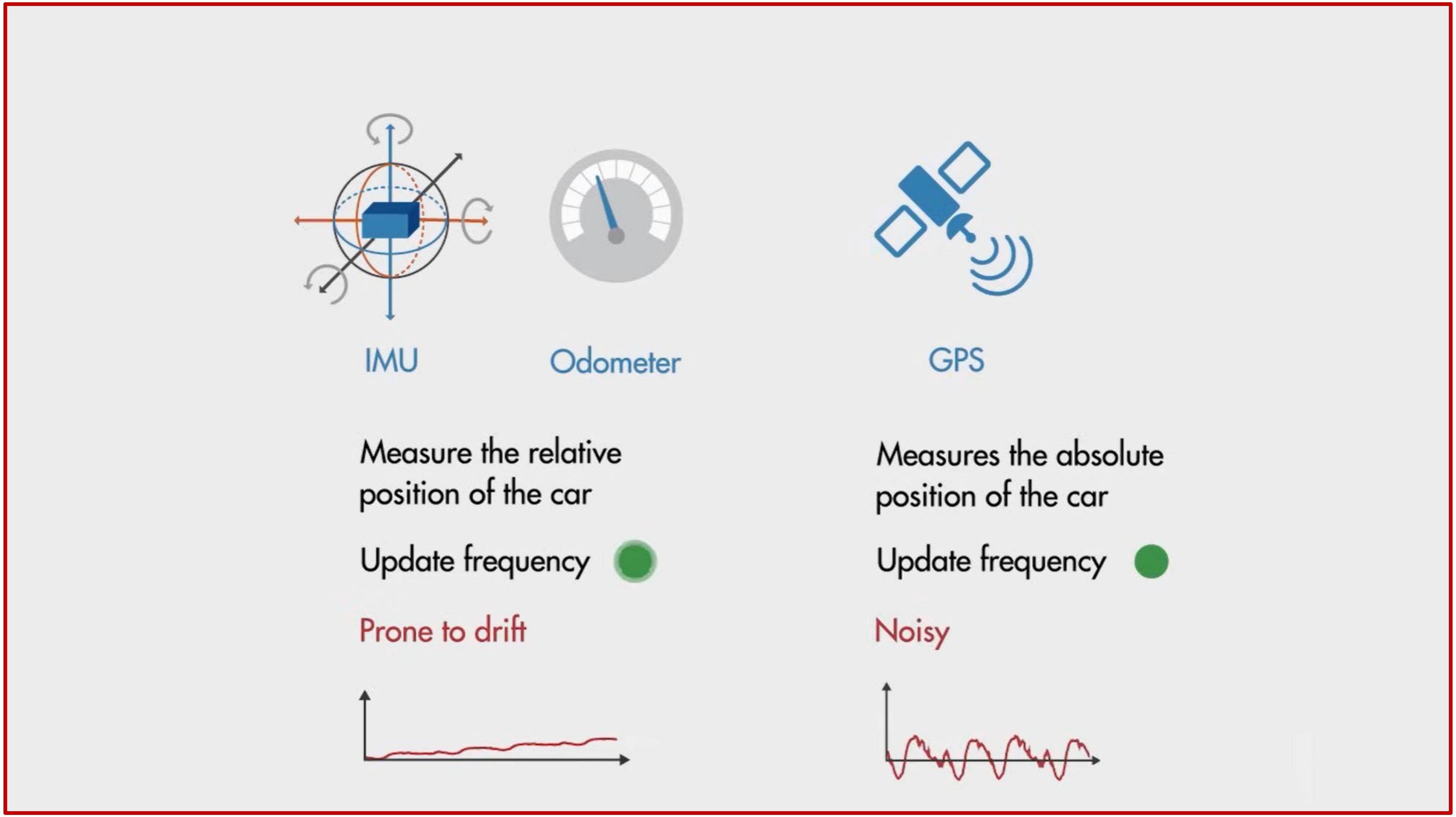
YouTube: *MATLAB:* Why Use Kalman Filters? | Understanding Kalman Filters

Kalman Filtering (aka Linear Quadratic Estimation)



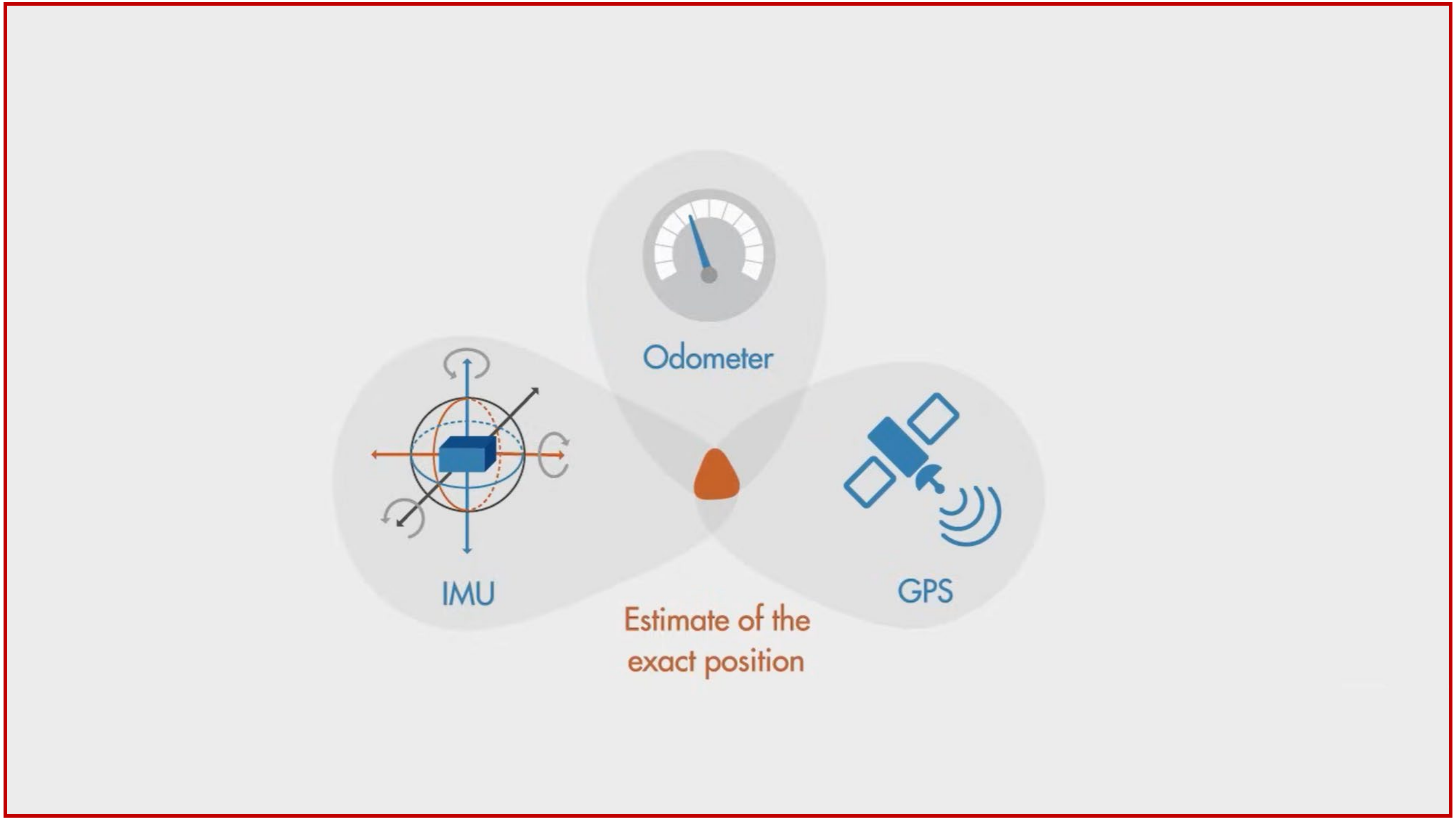
YouTube: *MATLAB:* Why Use Kalman Filters? | Understanding Kalman Filters

Kalman Filtering (aka Linear Quadratic Estimation)



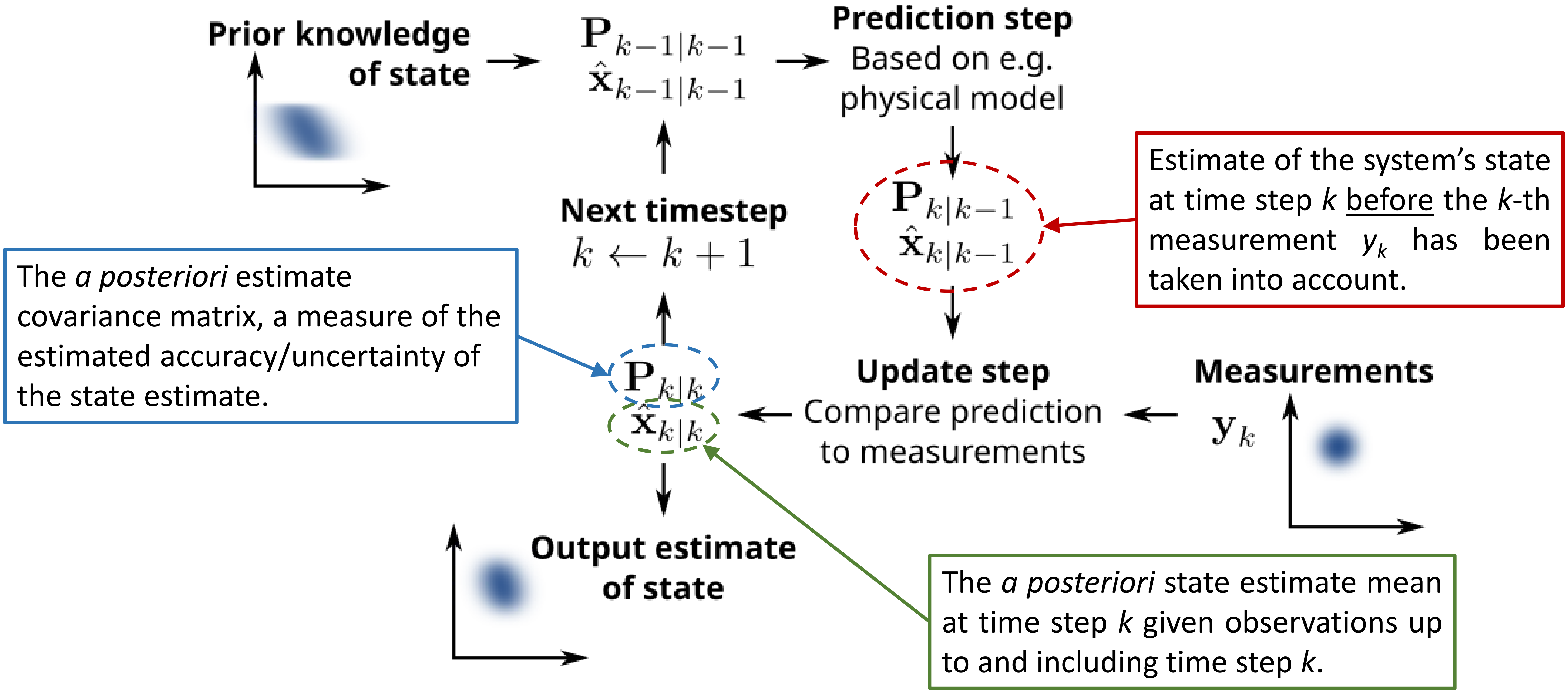
YouTube: [*MATLAB:* Why Use Kalman Filters? | Understanding Kalman Filters]

Kalman Filtering (aka Linear Quadratic Estimation)



YouTube: [*MATLAB:* Why Use Kalman Filters? | Understanding Kalman Filters]

Kalman Filtering: State estimation via state transition model

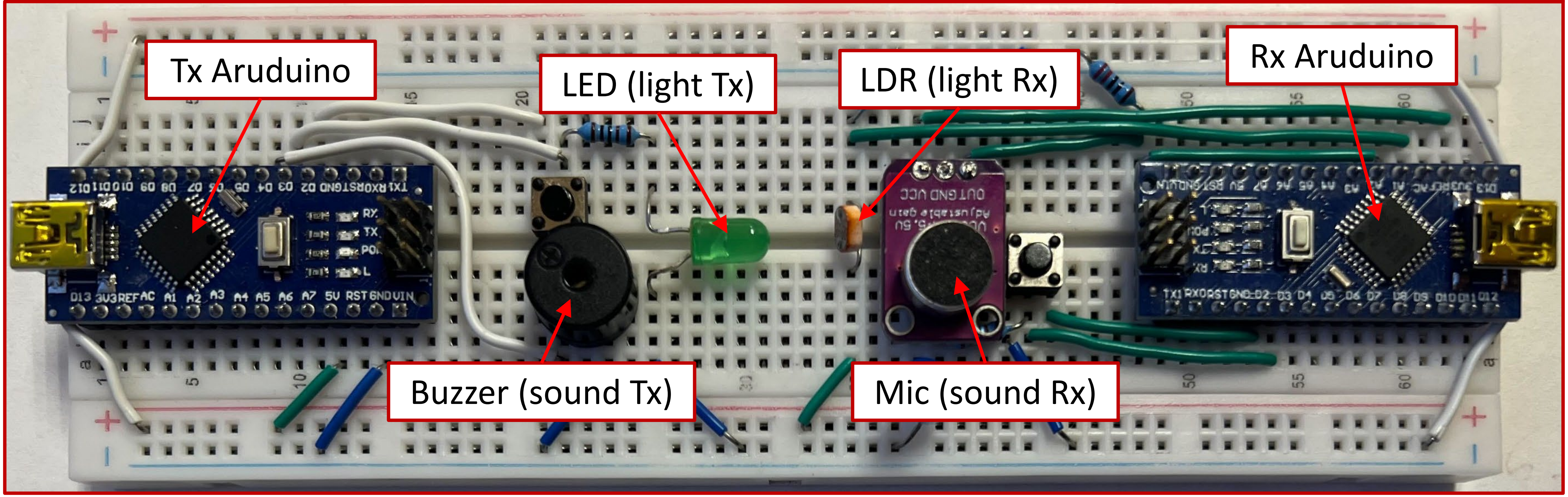


Arduino Demo: Wireless Sensor Fusion w/Light & Sound

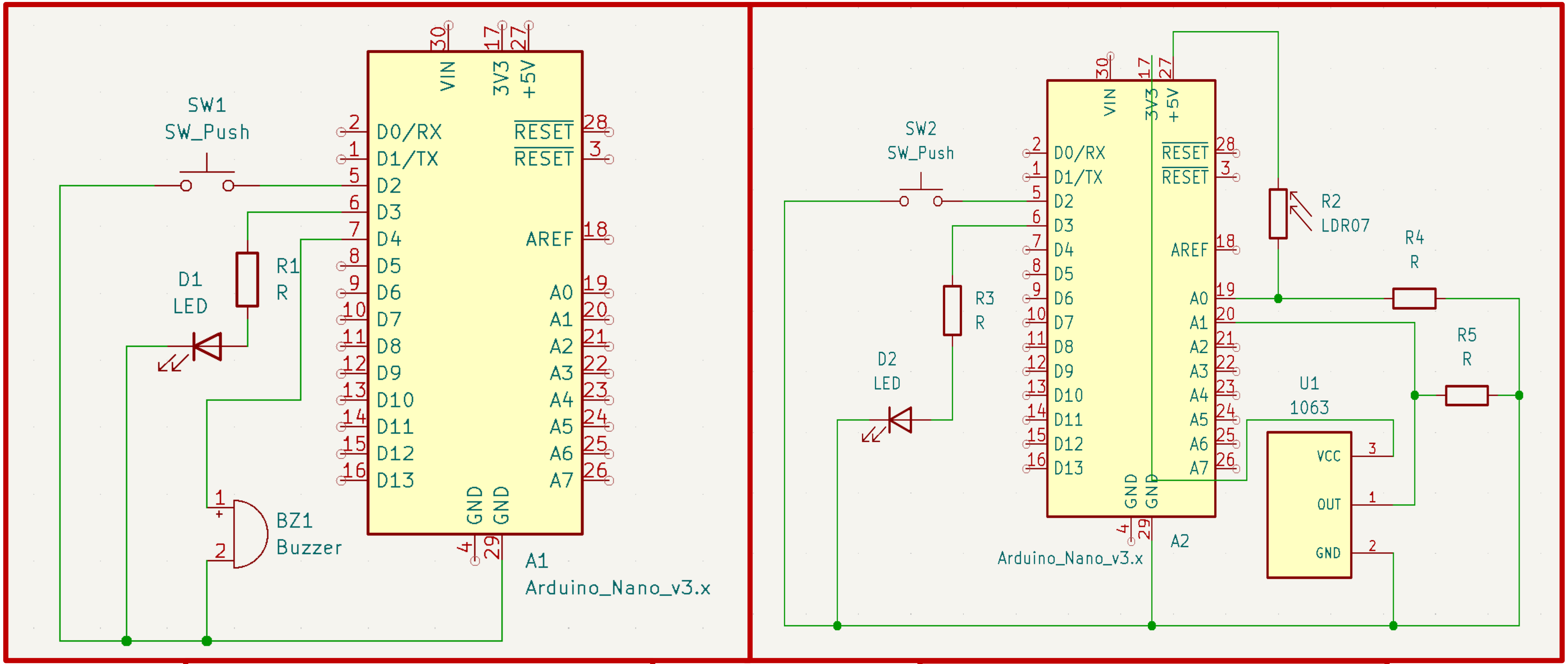
Wireless channels to be utilized:

- ❖ **LiFi (light)**
  - LED on/off encodes bits.
  - LDR samples and thresholds to recover bits.
- ❖ **Audio (sound)**
  - Buzzer emits short/long tones representing binary on/off (1/0) states
  - Can utilize multiple tones to generate *symbols*
- ❖ **Hardware overview**

IEEE: Qualcomm Buys Arduino



Sensor Fusion: Schematics



Transmitter (Tx) schematic

Receiver (Rx) schematic

Writing/assembling sketch blocks across different model versions

Version 1 core receiver functions:

- **`pickUpSound()`:** Measures peak amplitude vs. defined threshold.
- **`senseLight()`:** Measures instant threshold.
- **`getLightByte()`:** Performs mid-bit sampling.
- **`decodeBinary()`:** Assemble 8-bit characters.

Version 2: Speed tuning

- Reduce `dotDuration`, `dashDuration`, `bitGap`, `charGap`.
- Button behavior updates.

Version 3: Auto-thresholding

- Includes *startup calibration* for ambient light & noise .
- 2-second moving averages/peaks + buffers).

Version 4: Stable silence

- Avoid misreading sine-wave near-zero crossings by requiring *periods* of silence.
- Move timing to **`micros()`** for 1 ms → 1 μs-scale control.

Arduino code jointly developed by Gary Chen, Samuel Ye, and Walter Unglaub and available for download at: https://github.com/Garbear008/Sensor-Fusion-Project.

Arduino codes: **Fuse** models (from 2 channels → fusion)

USC University of Southern California

- **fuseV1: OR-fusion concept**
  - Receive characters via light AND sound in parallel.
  - Take the first *printable* char per position (light-dominant or sound-dominant views), with a *clear buffers* command.

- **fuseV2: Speed-up carried over**
  - Stable-silence integrated → dotDuration down to ~750 µs.

- **fuseV3: Symbol design**
  - Multi-symbol audio: combine durations + multiple frequencies to encode 4 bits/packet.
  - Practical mic & buzzer limits.
  - Zero-crossing frequency estimate (±200–300 Hz).

Arduino codes: **Comb** models (three channels & integrity checks)

- **Comb 1 concept:**
  - Return to 1-bit audio to run *three channels at once* (light + 1-bit audio + multi-tone audio) for redundancy.
  - Structured displays: `displayString`, `realString`, `trashString` to visualize error screening.

- **Error detection:**
  - **Parity bit (even or odd)** on ASCII bytes; quick to compute, detects odd-bit flips but not even.
  - **Checksum** (Comb 1): simple bit-position sums → remainder product; appended after a stop byte.

- **Error correction (Comb 2):**
  - Uses **Hamming (11,7)** and **extended (12,7) SECDED** error-correction code intuition.
  - This means locating and fixing single-bit errors; double-bit detect.

- **Performance engineering (Comb 3):** ADC pre-scaler change
  - 16 MHz clock, ADC 13 cycles.
  - Default prescaler 128 → ~112 µs/read.
  - Lower to 8 → ~8 µs/read.
  - Enables dotDuration ≈ 100 µs.
  - Light period ~2 ms.
  - Tighter audio freq bands (≈1.5–3 kHz).

Arduino Demo: Summary and Lessons Learned

USC University of Southern California

❖ How the fusion algorithm "thinks"

➢ **Timing alignment**
- Consistent start bit for light; silence windows for audio.
- Per-char synchronization before fusion.

➢ **Validity gates**
- Parity/checksum/Hamming pre-filter what's *eligible* to fuse.

➢ **Decision logic**
- Per-position printable-char check (light- or sound-dominant) → build display string.

➢ **Failure handling**
- Timeouts, sensor health (saturation, no-change), fallbacks (choose surviving channel).

❖ Shortcomings & lessons learned

➢ **Physical limits:**
- Buzzer frequency range ceiling.
- Mic amplitude dependence.
- Ambient light variability.
- LDR latency (light period choice).

➢ **Algorithmic tradeoffs**
- Duration vs. throughput
- Simple zero-crossing vs. FFT (speed).
- Parity/checksum coverage gaps (even-bit errors).

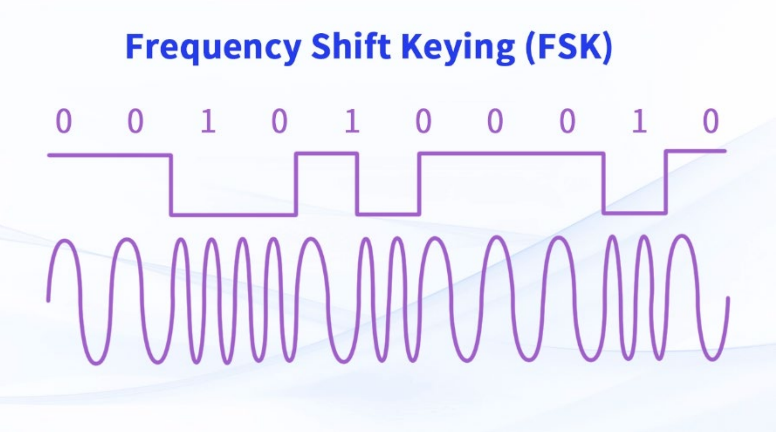
➢ **Implementation realities**
- ADC speed vs. accuracy.
- Timer granularity.
- Serial monitor throughput.

Arduino Demo: Extensions & Research Directions

USC University of Southern California

❖ **Richer symbols / more throughput**

- Higher-Q audio front-end or different mic + true **M-FSK** (frequencies spaced by filter bandwidth) or **PSK/FSK hybrid** with better tone generation.
- **Line coding** (Manchester) to stabilize timing; **framing** and **CRC** for stronger detection.
- **Interleaving + Hamming/CRC** to handle bursts.



❖ **More modalities**

- **Ultrasonic** (HC-SR04 receiver mod/MEMS mic), **IR**, **laser**, **RF (LoRa/FSK)**, **vibration** (piezo), **magnetic** (Hall).
- Extend OR-fusion to **N-of-M voting** or **confidence-weighted fusion** (e.g., lower weight for noisy mic frames).

❖ **Better fusion**

- Decision-level majority vote across channels; **soft decisions** using SNR estimates
- Time alignment via **preambles** and **cross-correlation**
- **Kalman-style** reliability weighting when channels provide continuous estimates (e.g., RSSI + lux)

❖ **Applications**

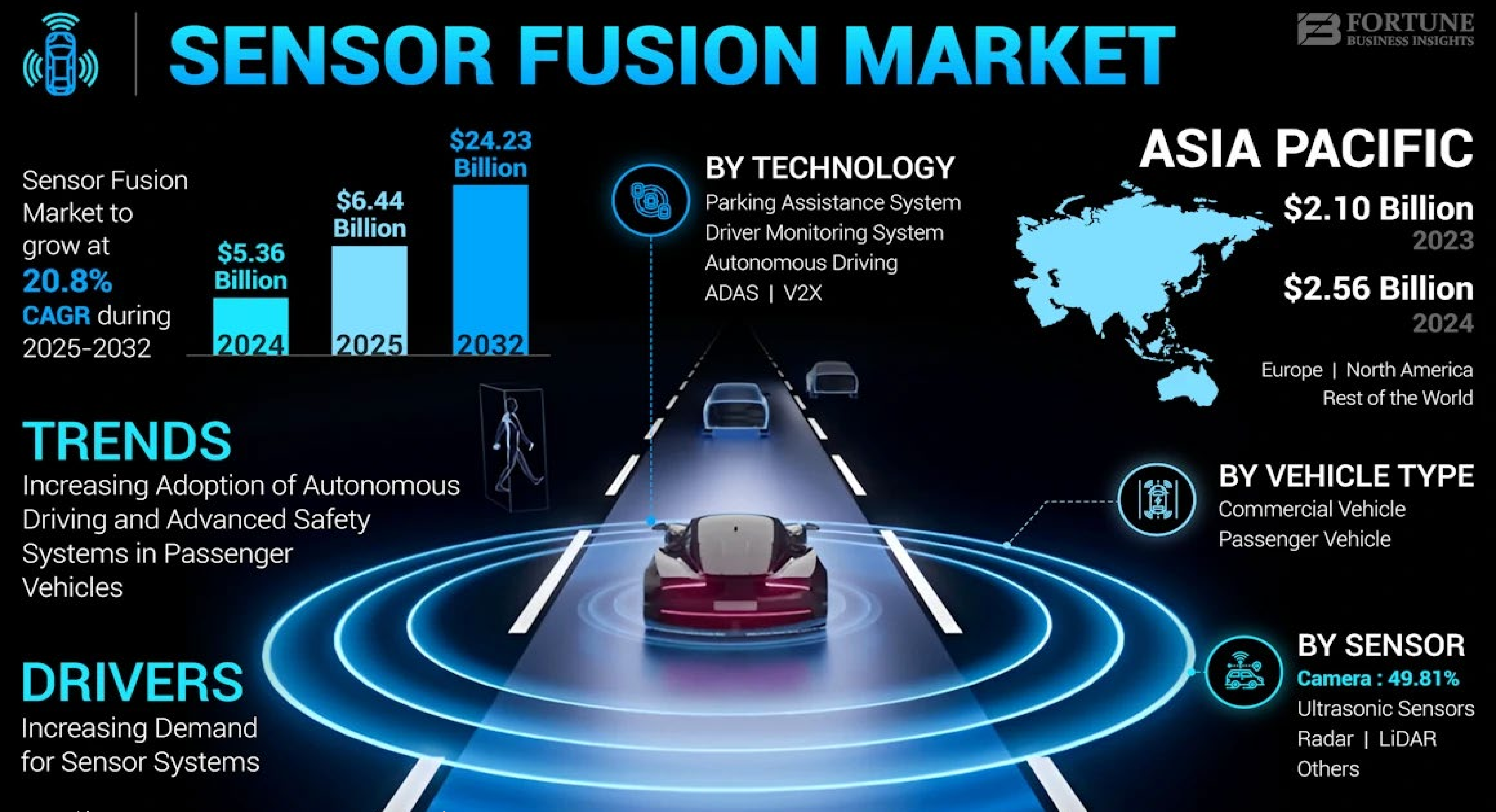
✓ Low-power redundant signaling for IoT in noisy/occluded spaces.
✓ Robotics comms in environments where RF is restricted (labs, hospitals).
✓ Visualizing redundancy, symbols, and error-control coding live.

Common pitfalls and some tips to avoid them!

- Some general rules of thumb:
  - **Outlier rejection:** Ignore obviously bad readings.
  - **Gating:** Accept a measurement only if it's "close enough" to the prediction.
  - **Normalization & unit checks:** m/s vs. km/h errors are common!
  - **Health checks:** If a sensor freezes or saturates, reduce its weight or drop it temporarily.
  - **Latency compensation:** Delay (or extrapolate) other signals to align with a late sensor.
- Typical pitfalls (and some tips on how to avoid them!)

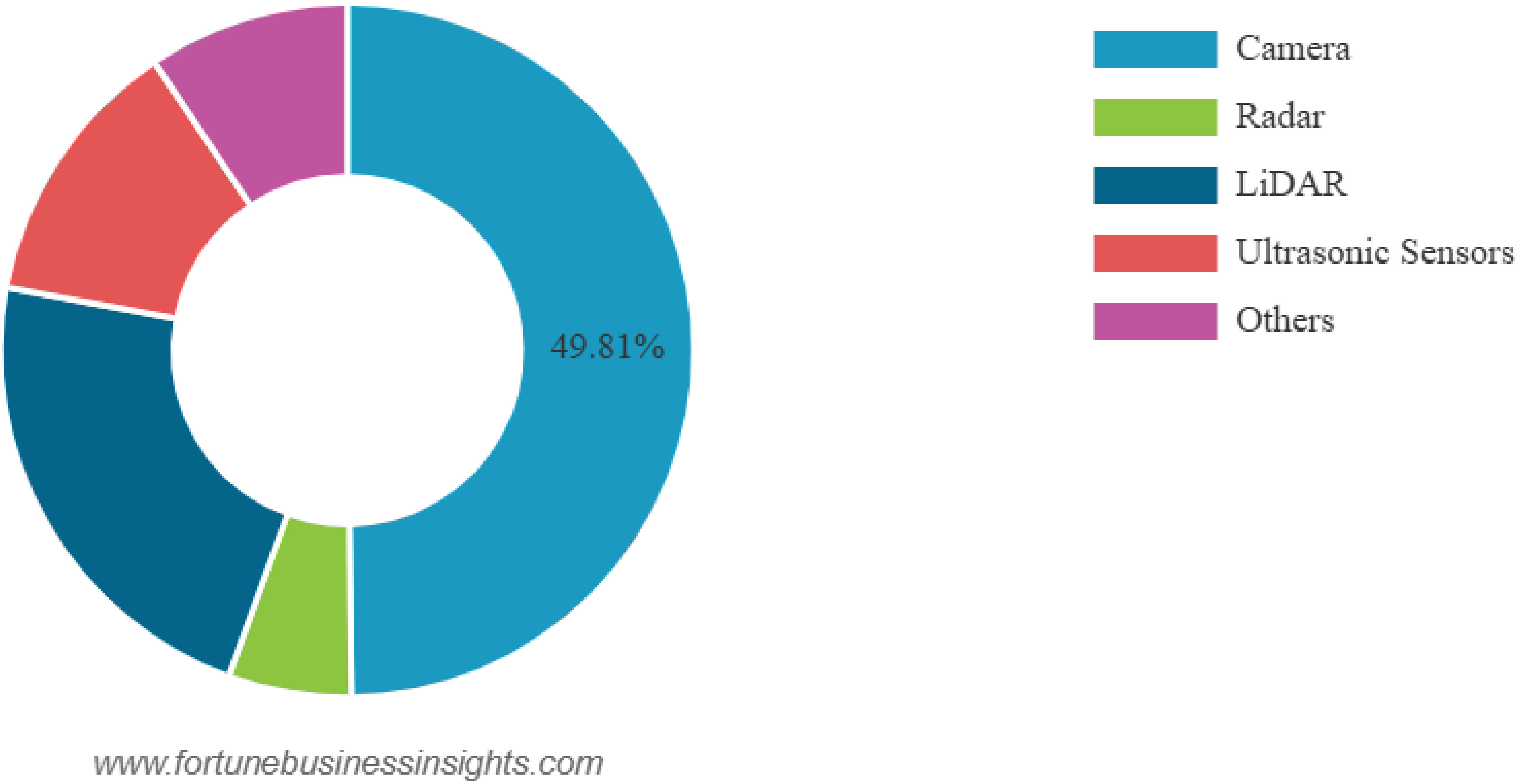
➢ **Not calibrating biases**	✓ Do a quick still-calibration on startup.
➢ **Ignoring dynamics**	✓ When accelerating, trust the gyro more than the accelerometer.
➢ **Bad timing/sync:**	✓ Use consistent timestamps.
➢ **Mismatched frames**	✓ Name your axes consistently; keep track of rotations!

- Sensor fusion is also about trusting the *right sensor* at the *right time*!
- Start simple (*accuracy before speed!*), calibrate carefully, and let data quality guide the weights.



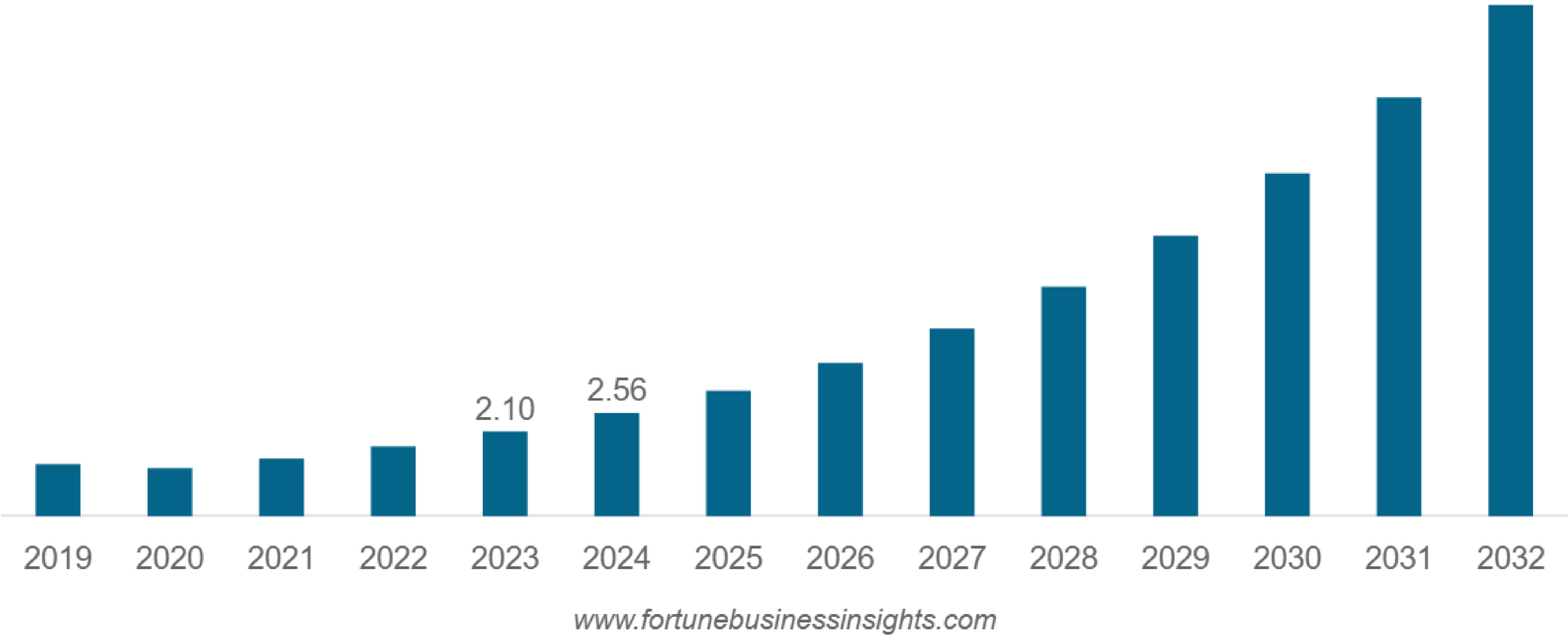
https://www.fortunebusinessinsights.com/sensor-fusion-market-107822

Global Sensor Fusion Market Share, By Sensor, 2024



https://www.fortunebusinessinsights.com/sensor-fusion-market-107822

Asia Pacific Sensor Fusion Market Size, 2019-2032 (USD Billion)



https://www.fortunebusinessinsights.com/sensor-fusion-market-107822

Extra/Backup Slides