MOVESENSE

MOVESENSE SENSOR PROGRAMMING

EDUCATION SESSION FOR CUSTOMERS

Petri Lipponen 2021-04-21

Content

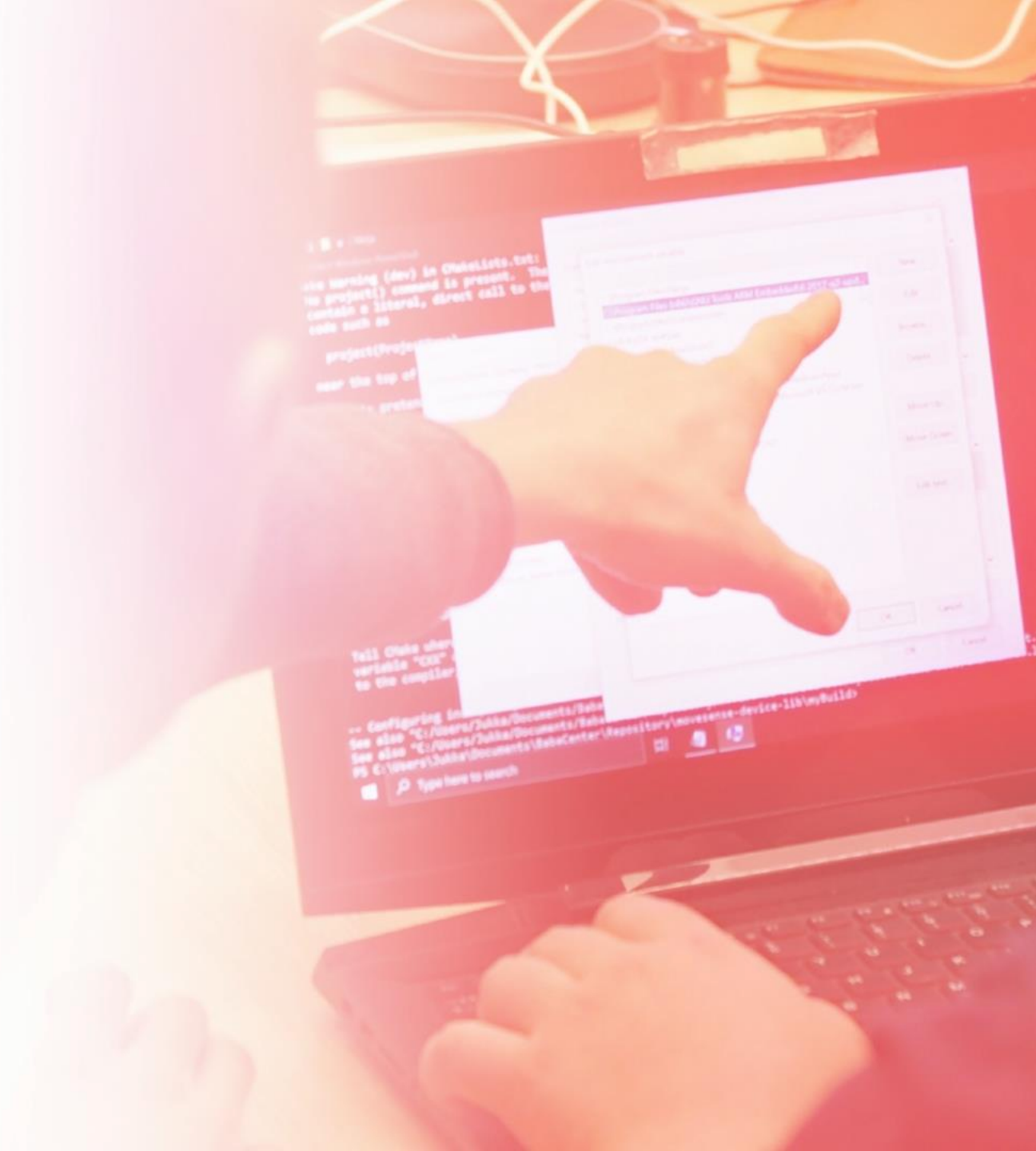
- What is Movesense?
- Sensor details
- Sensor programming environment (Docker)
- Sensor programming basics
- Movesense REST API
- Using data memory
- Security considerations
- Alternative communication methods
- Using multiple sensors
- Q & A

“What is Movesense?”

In Short:

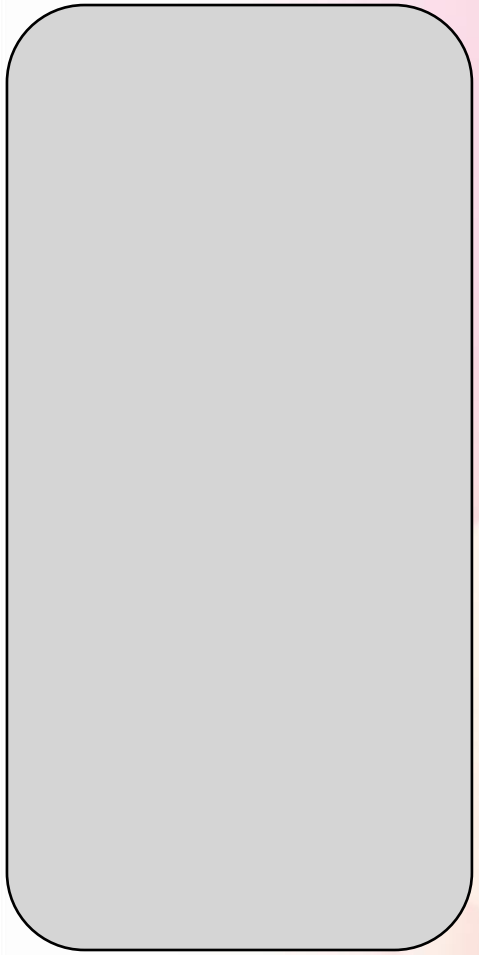
“Open Wearable Tech Platform”

- Programmable Sensor
- Mobile SDKs
- Embedded REST communication framework “Whiteboard”
- Physical accessories available



“What is Movesense?” Continued…

Bluetooth
Low Energy



- Whiteboard (REST)
- GATT
- BLE Advertising

https://www.movesense.com/specifications/

Movesense sensor details

- Waterproof: 30 m / 100 ft
- CR 2025 Coincell battery
- 64MHz NordicSemiconductor MCU (RAM: 64kB, FLASH 512kB)
- 9-axis IMU (Accelerometer, Gyroscope, Magnetometer)
- Maxim ECG Analog Frontend
  (ECG, HeartRate, RR-intervals, stud contact detection)
- Data memory: 384kB (EEPROM)
- Temperature measurement
- 1-wire Master (non-medical only)
  - Smart connector detection
  - 1-Wire communication support

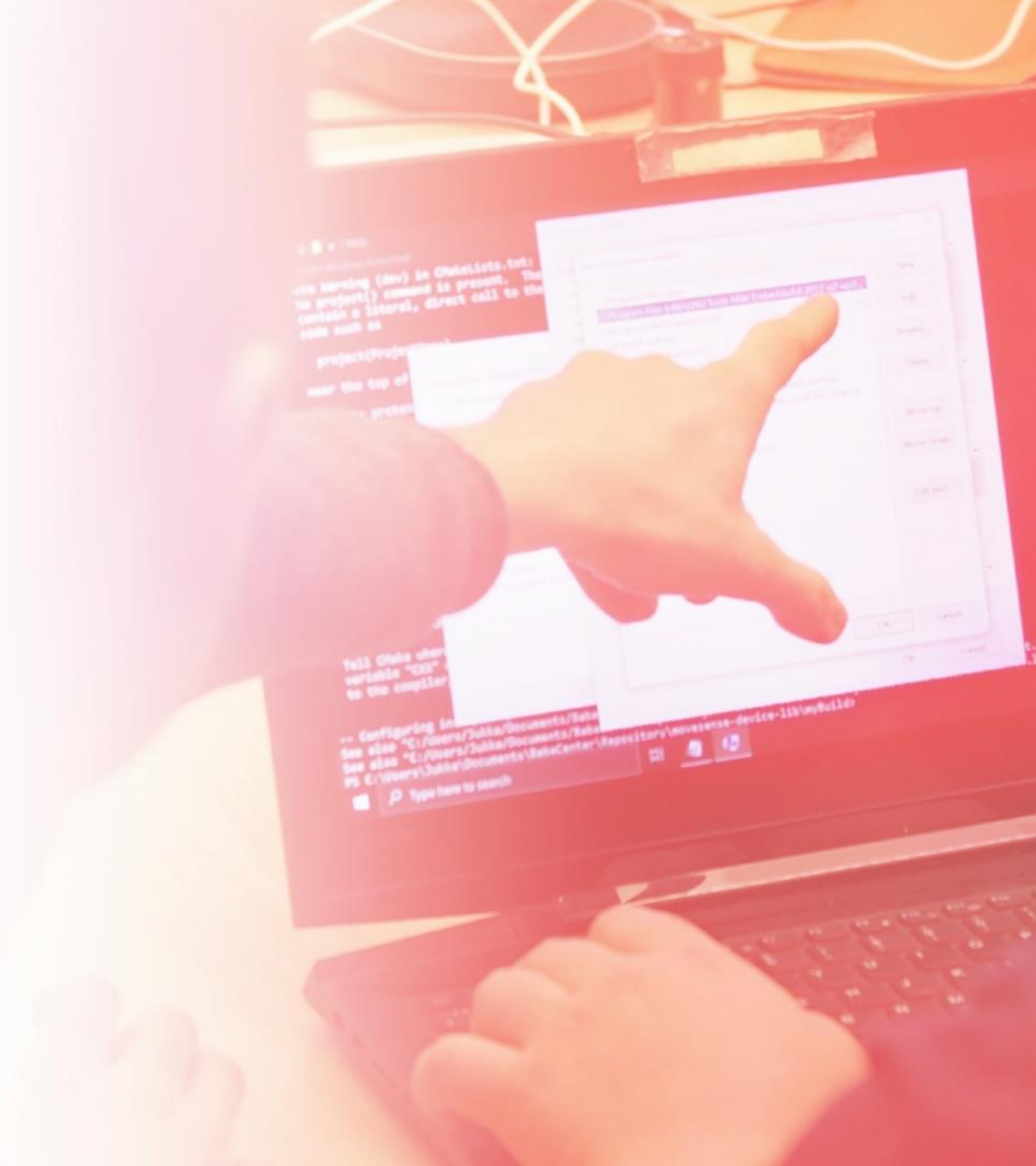
Sensor Programming Environment

- Docker based build environment for sensor hardware
- Simple build commands and many sample apps available

- “cmake” to initialize the build project
- “ninja pkgs” builds the hex files (for Jig or production line) and DFU (Firmware update) packages

- More details: https://www.movesense.com/docs/esw/getting_started/#toolchain-and-usage

Docker build demo



Sensor Simulator

”Movesense sensor software on Windows & Visual Studio”

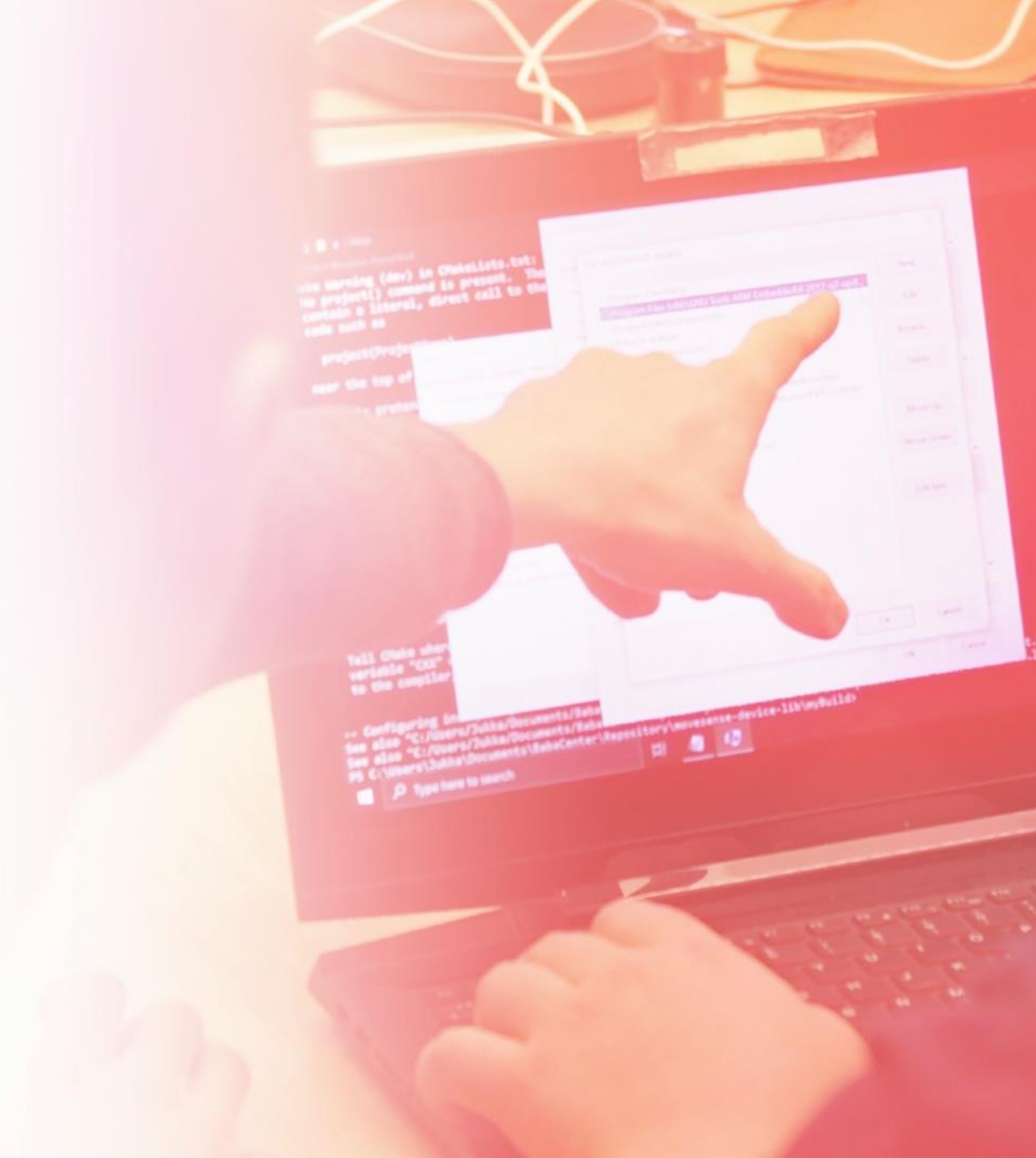
- Easier debugging and faster development cycle
- Simulated sensors with data import
- Whiteboard communication using *wbcmd.exe*

Limitations:

- No BLE
- No Mobile communication
- Not 100% accurate

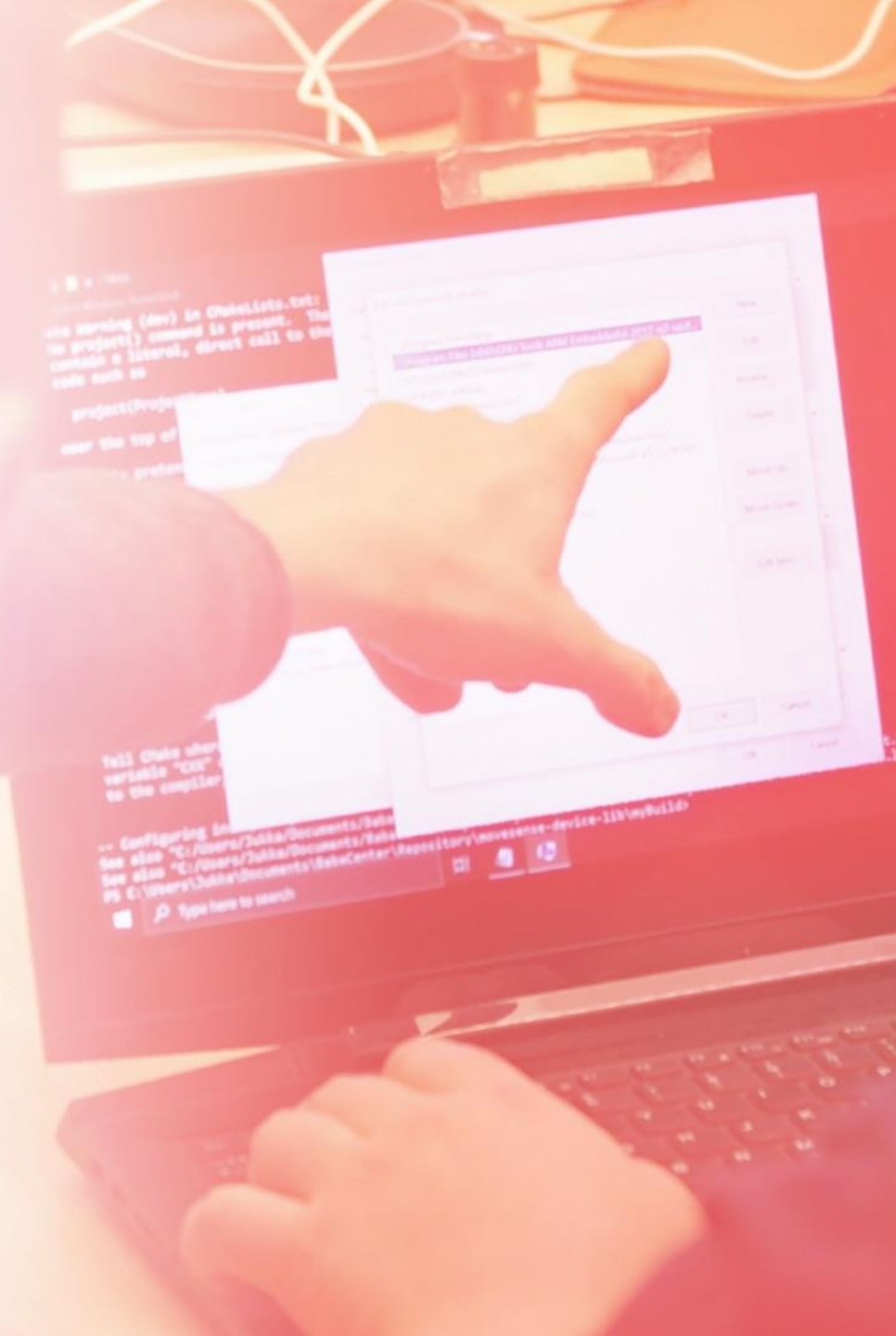
More details: https://www.movesense.com/docs/esw/sensor_simulator/

Sensor simulator demo



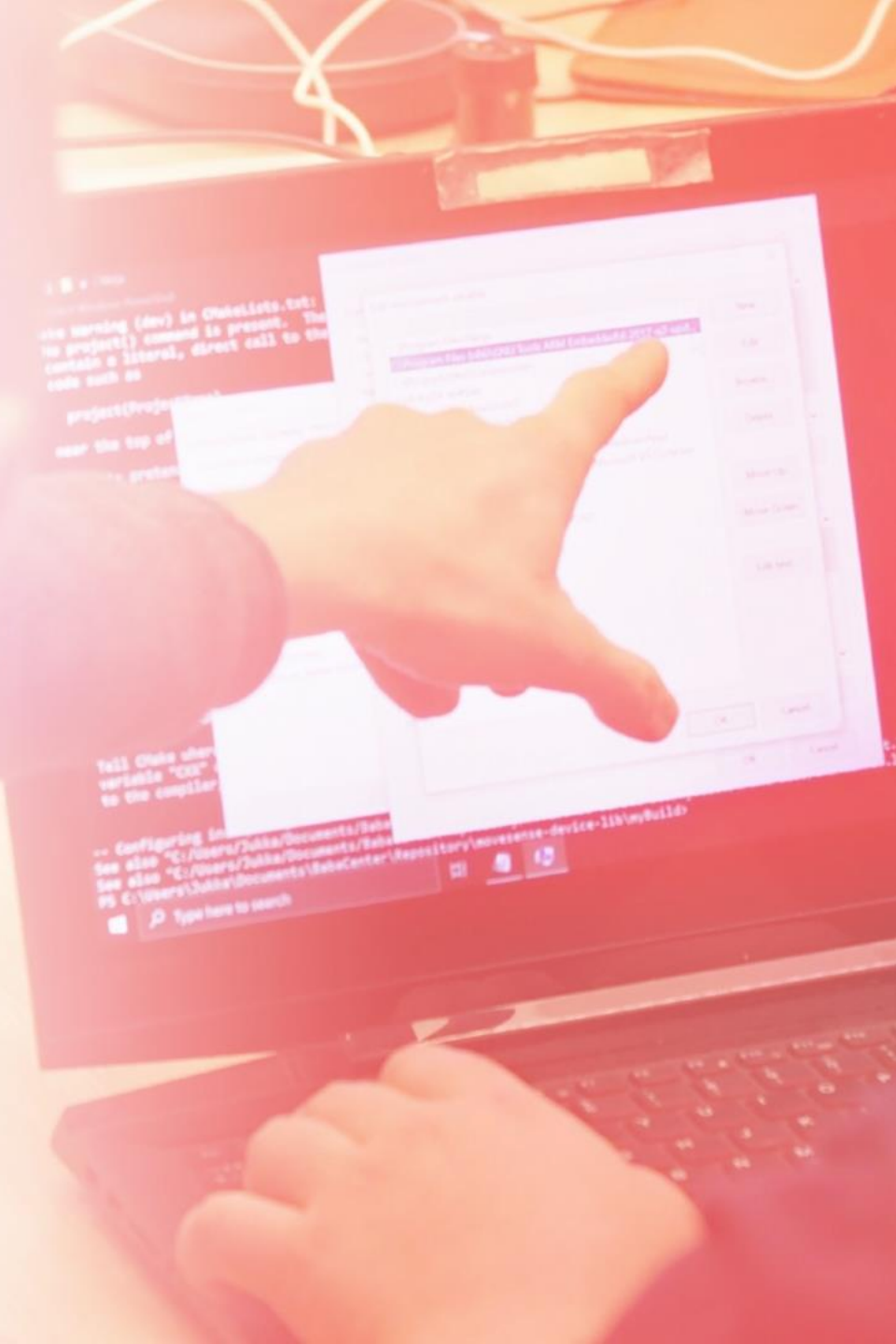
Sensor Programming Basics

- Language: C/C++
  - No dynamic memory allocation
  - No STL
  - Limited RAM & Flash
- All hardware and low level access is via Movesense REST API
  - GET/POST/PUT/DELETE
  - Publish-subscribe extension for datastreams
  - Same API inside the sensor and from mobile using MDS library!
- Fully asynchronous:
  - Code MUST NOT hog the execution => No busy-loops!
  - Automatic power optimization
  - Call – callback structure



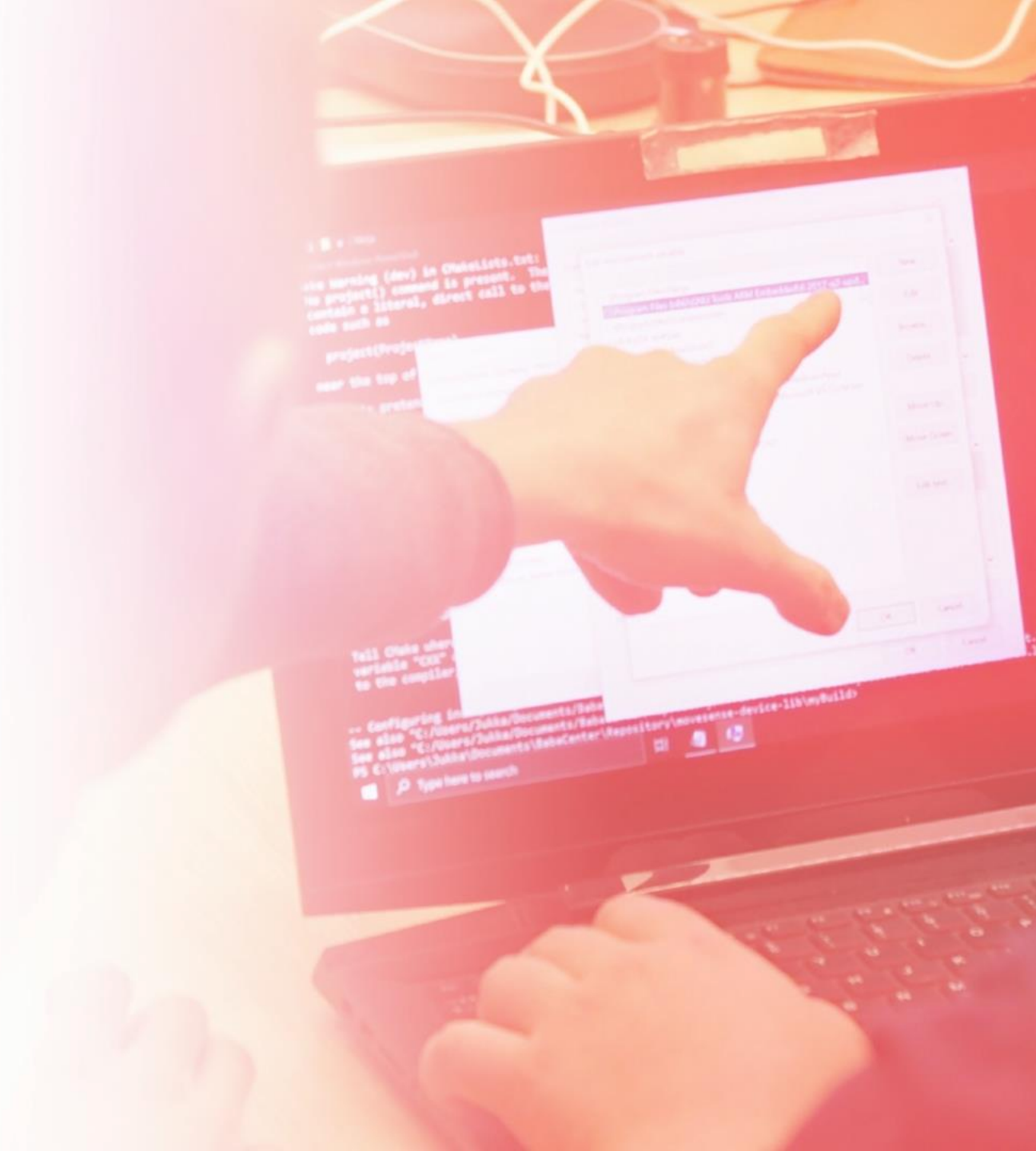
Sensor Programming Basics: Whiteboard REST Framework

- **Provides**: services, clients, timers, threading and external communication
- **ExecutionContext**: Whiteboard threads
- **LaunchableModule**: ”WB-aware class”
  - Runs in an ExecutionContext (WB thread)
  - Lifecycle callbacks (initModule, startModule, stopModule, deinitModule)
- **ResourceProvider**: WB REST service
  - API defined using Swagger 2.0 notation (yaml-file)
  - Request callbacks: onGetRequest, onPutRequest,…
- **ResourceClient**: WB REST client
  - Make requests to internal and external whiteboard services
  - Request methods: asyncGet, asyncPut,…



Sensor Programming Basics: project structure

- CMakeLists.txt: Main project file
- App.cpp:
  - Module list of the user application
  - Optional framework modules
  - Data memory layout
- app_root.yaml:
  - ExecutionContext definitions
  - Users API declarations
- wbresources –sub folder
  - APIs defined by the application



Movesense REST API

Main sections:

- */Meas* for sensor data (Acc, Gyro, Magn, Temp, HR, ECG)
- */Mem* for data memory access: DataLogger & Logbook
- */Comm* for communication protocols: BLE, 1Wire
- */Component* for low level features: LED, EEPROM, chip specific features
- */System for s*ystem features: Mode, Settings, Energy, Memory, States
- */UI* for user interface (LED blinks)
- */Misc* for the ones that do not fit in the above

and

- */Whiteboard* for Whiteboards own services

See: https://bitbucket.org/suunto/movesense-device-lib/src/master/MovesenseCoreLib/resources/movesense-api/

Movesense REST API: /Meas

Same pattern for all sensors

/Meas/*****/**Info**:

- Valid sample rates
- Sensitivity values (G-range etc.)

/Meas/****/**Config**:

- Sensitivity etc.

/Meas/**/<SampleRate>**:

- Data stream @ given sample rate (e.g. */Meas/Acc/13*)
- Note: when subscribing the /Subscribe is not given

Using Data Memory

- EEPROM (two chips: 256 kB + 128 kB)
- Low power consumption, max speed 400 kbps

High level API:

- /Mem/DataLogger for storing any subscribable data
- /Mem/Logbook for getting stored data
- Stored in a binary format ”sbem” in a ***ring buffer***
- Mobile MDS has automatic Logbook proxy with built-in SBEM to JSON conversion

Low level API:

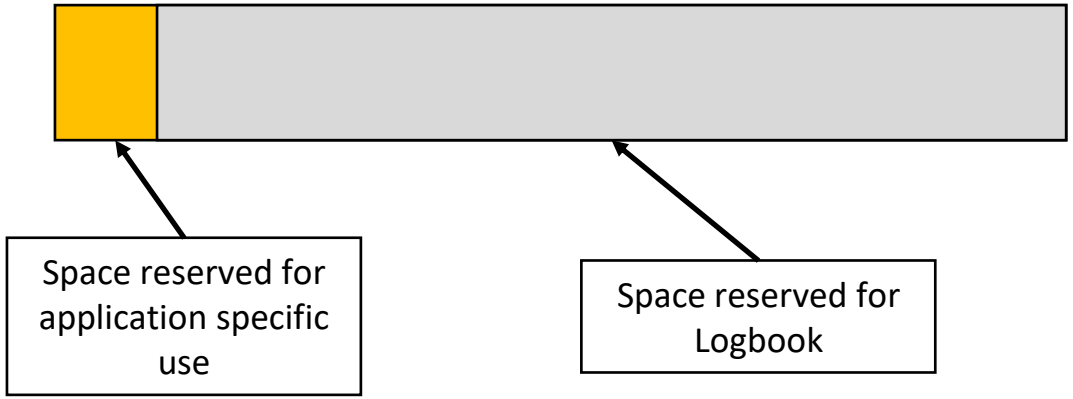
- /Component/EEPROM/<chip#>

See:

https://bitbucket.org/suunto/movesense-mobile-lib/src/master/android/samples/DataLoggerSample/

Logbook memory setup in ***App.cpp***:

- *LOGBOOK_MEMORY_AREA(start, size)*



Security considerations

Bluetooth Bonding (encrypted connection)

- “Just works” or PIN code
- Adjustable policy (allow/deny re-bonding)
- BLE-Keys stored in the internal FLASH memory
- Adjustable re-bonding time after powerup

DFU package signature

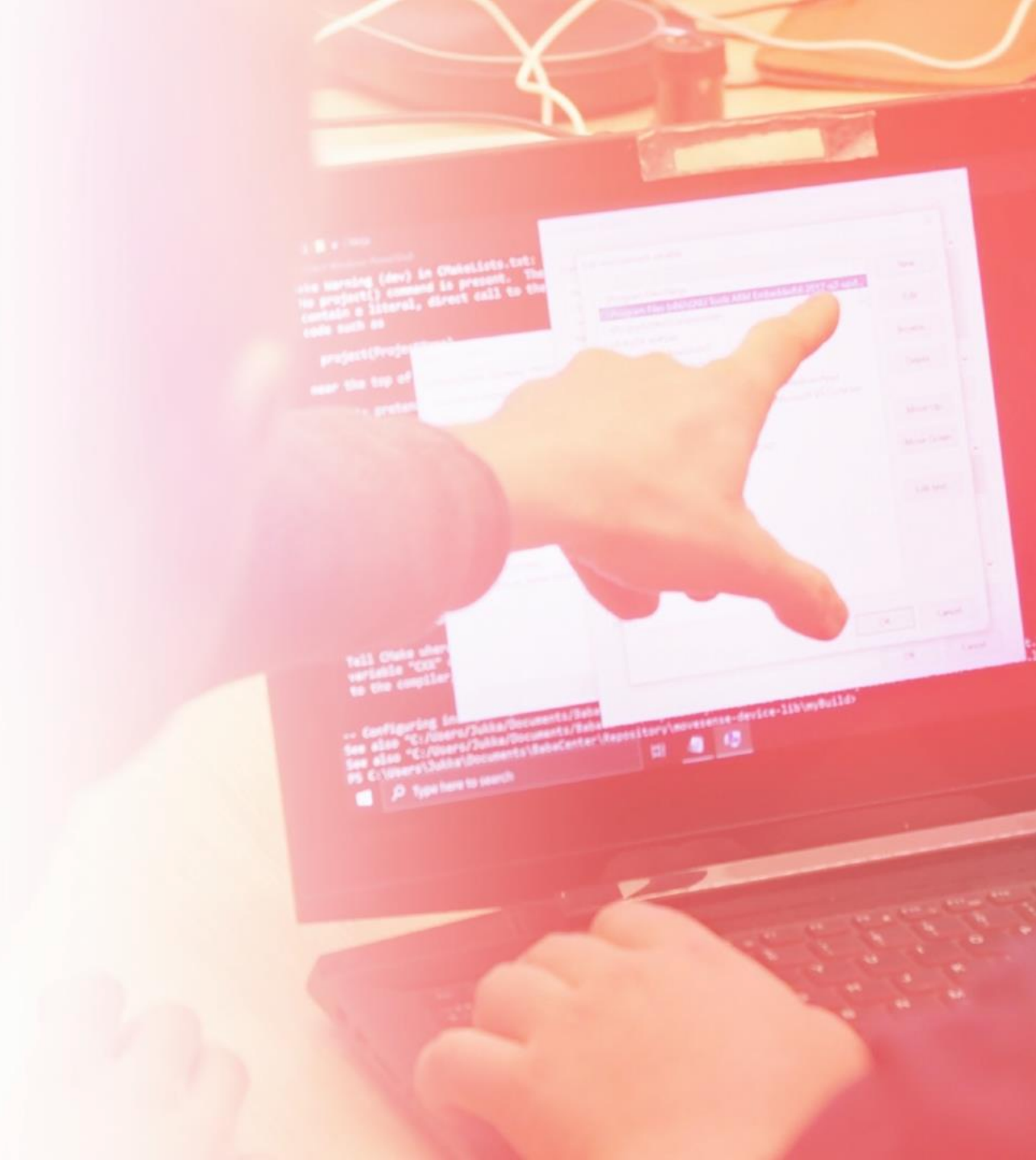
- Coming customer specific keys and bootloader (2.1 release)

App-specific authentication

- Authentication resource which required before data service access

Encrypted data memory

- Application responsibility



Alternative Communication Methods

“What if I want to use other than Android or iOS to talk to the sensors?”

Built in HRS & Nordic UART services

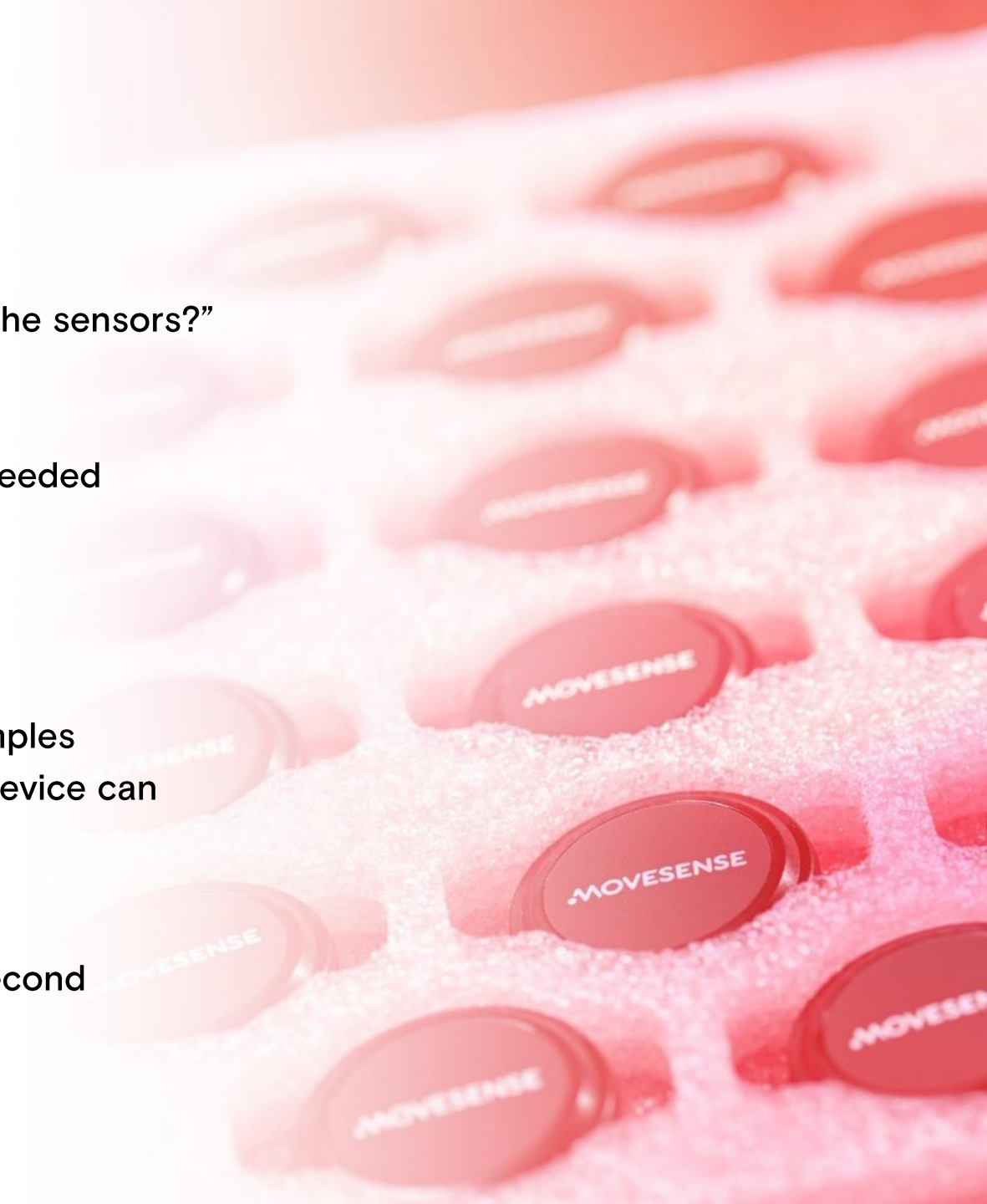
- Limited to those two, but easy to use if nothing else is needed

CustomGATTService

- Possible to create almost any kind of GATT service
- Up to 5 characteristics
- See: *gatt_sensordata_app* & *custom_gattsvc_app* in samples
- Samples include python and WebBLE clients, any BLE device can connect

Embedding data to BLE Advertising packet

- Up to 27 bytes per update, updated max ~5 times per second
- No limit to number of sensors
- See: *custom_bleadv_app* in samples



Using Multiple Sensors

BLE limitations:

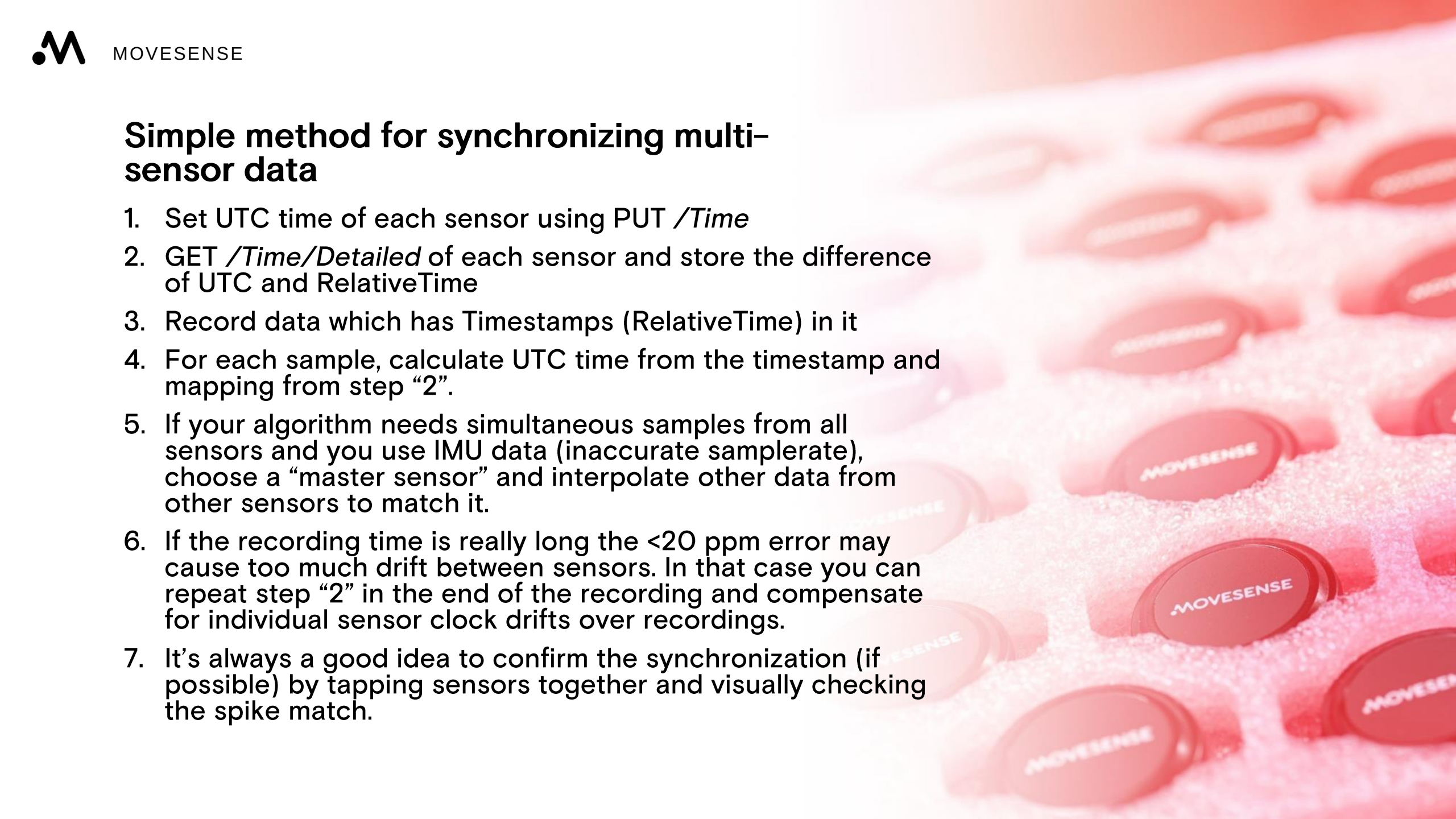
- Usually 4-7 connections allowed at the time
- Bandwidth can be a limiting factor

Synchronization of data from different devices:

- Each sensor has high accuracy (20 ppm) clock that provides the timestamps (“RelativeTime” = milliseconds since reset)
- **NOTE:** Not all sensors can provice accurate sample rates! E.g. Acc, Gyro & Magnetometer have up-to 10% error in sample rate (Hardware limitation)
- PUT */Time* allows setting universal time to each sensor
- GET */Time/Detailed* returns mapping between UTC and RelativeTime (=Timestamp) on each sensor

Simple method for synchronizing multi-sensor data

1. Set UTC time of each sensor using PUT */Time*
2. GET */Time/Detailed* of each sensor and store the difference of UTC and RelativeTime
3. Record data which has Timestamps (RelativeTime) in it
4. For each sample, calculate UTC time from the timestamp and mapping from step “2”.
5. If your algorithm needs simultaneous samples from all sensors and you use IMU data (inaccurate samplerate), choose a “master sensor” and interpolate other data from other sensors to match it.
6. If the recording time is really long the <20 ppm error may cause too much drift between sensors. In that case you can repeat step “2” in the end of the recording and compensate for individual sensor clock drifts over recordings.
7. It’s always a good idea to confirm the synchronization (if possible) by tapping sensors together and visually checking the spike match.



MOVESENSE

Questions?