# Gesture software remarks

## Algemeen

Code is goed opgedeeld in aparte modules: GPIO, I2C, … staan in aparte bestanden.

Code is goed opgesplitst in aparte functies. Alleen de main functie is wat lang. Deze zou wat meer opgesplitst kunnen worden.

Bijvoorbeeld

* Perform\_calibration
* Handle\_sensors
* Handle\_led\_timer
* Handle\_command

De code maakt wel veel gebruik van globale variabelen. Bij grotere projecten kan dit snel onduidelijk worden en problemen veroorzaken. Bijvoorbeeld race condities of interrupt safety issue.

Main.c: lijn 150:

Variable hebben random data als deze op de stack staan (gedefinieerd in functie). Static/globale variabelen zijn wel 0 bij opstart.

### Struct met data

Ik zie ook verschillende arrays met data voor elke sensor (Bv isReady[amountSensor]). Dit zou je kunnen herwerken naar 1 array van sensor data structs. Het lijkt erop dat je ook de sensor definities zou kunnen integreren in deze struct.

Ik denk dat de functieaanroepen dat ook simpeler kunnen worden door deze struct mee te geven. Zo kan ook een mogelijke bug vermeden worden (zie verder).

typedef struct

{

bool isReady;

bool hasRead;

struct resultaat resultaat;

VL53L3CX\_Object\_t sensor;

Sensor\_Definition\_t pos;

} sensorData\_t;

Static sensorData sensorData;

Functieaanroep

getData(&sensor[left.id], &left, resultaat, (uint8\_t \*)isReady);

wordt dan

getData(&sensorData[id]);

### State variabele

In de code worden op verschillende plaatsen booleans gebruikt om een state bij te houden.

if (objectPresent && !prevObjectPresent)

Op den duur kan dit complex worden. Op dat moment kan een state machine de code duidelijk en begrijpbaar houden.

typedef enum

{

STATE\_SENSOR\_START = 0,

STATE\_SENSOR\_SAMPLING,

STATE\_SENSOR\_STOP,

} sensorState\_t;

static sensorState\_t sensorState = STATE\_SENSOR\_START;

switch (sensorState)

{

case STATE\_SENSOR\_START:

Start\_Sensor(&sensor[center.id], center.gpioPin);

Start\_Sensor(&sensor[right.id], right.gpioPin);

sensorState = STATE\_SENSOR\_DETECTED;

break;

case STATE\_SENSOR\_DETECTED:

// Handle sensor data

// Update presence

If (sensorPresent)

{

sensorState = STATE\_SENSOR\_DETECTED;

}

else

{

sensorState = STATE\_SENSOR\_STOP;

}

break;

case STATE\_SENSOR\_STOP:

Stop\_Sensor(&sensor[center.id]);

Stop\_Sensor(&sensor[right.id]);

break;

default:

break;

}

## Naamgevingen

Specifieke consistente naamgevingen kunnen snel duidelijk maken wat voor type een variable of term is

* Defines in hoofdletters:

#define AMOUNT\_SENSOR\_USED 3

* Typedefs met “\_t” suffix

typedef enum

{

…

} commands\_t;

* Functienamen met bestandsnaam als prefix

bool Sensor\_getData(…)

## Mogelijke bugs

* Main.c: lijn 203
  + getData(&sensor[left.id], &left, resultaat, (uint8\_t \*)isReady);
  + Moet resultaat niet &resultaat[left.id] zijn? En zelfde voor isReady
* Main.c: line 230:
  + moet isReady niet isReady[left.id] zijn?
* Main.c: lijn 93/99: Dubbel prototype van void SystemClock\_Config(void);