Optical force sensing – design considerations

Application Note

Published by ams-OSRAM AG
Tobelbader Strasse 30,
8141 Premstaetten Austria
Phone +43 3136 500-0
ams-osram.com

Optical force sensing – design considerations

Application Note No. AN001061

Valid for:
TCS3701, SFH 5721, SFH 4053, SFH 4253, TMD3725, TMD3765, AS7057, SFH 2704

Abstract

An optical force sensing system uses a light emitter and a light sensor that are located side-by-side below a flexible (and reflective) surface. The more forceful a press on that surface, the greater its deflection, the higher the value reported by the sensor. In other words, optical force sensors implement a (force aware) button. This application note reviews the design aspects of such a system, covering optical, electrical, mechanical and even driver and application software. It discusses various component selections from discrete solutions to fully integrated and suggest applications like keypads or even sliders.

Table of contents

1 Introduction

This application note describes *optical force sensing*. Optical force sensing uses a light emitter and a light detector that are located side-by-side below a flexible (and reflecting) surface. This combination allows the sensor to measure how forceful the surface is pressed. In other words, optical force sensing implements a (force aware) button.

Several designs have been made; they are not available for ordering, but they have been used in this document to explain design choices, guiding the readers in designing their own solutions.

1.1 Context

There are different ways to implement touch-based interface functions. Conventional implementations deploy *mechanical buttons*, more advanced ones are based on *magnetic position sensing*, or *capacitive* or *resistive* touch detection. The latter are often used in combination with control menus on touch screens. Cost and stylish design aspects often constrain the selection process.

These options rely on established technologies, but they suffer from various disadvantages, which typically require additional design and implementation effort. In addition, new usage constraints emerged from the COVID-19 pandemic: areas which are exposed to frequent touching must be disinfected by using alcohol or alkali suds. Suddenly it is also important that the switch is hermetically sealed to prevent liquids entering as well as color stability of the control knob itself. Conventional implementations can only address these constraints if special coating or shielding measures are being realized. This leads to higher design and manufacturing costs.

To summarize, here is a brief – yet not exhaustive – overview of the various dis-/advantages the various traditional switching technologies bear.

Table 1: Dis-/advantages of various traditional switching technologies

Technology	Mechanical Switching	Magnetic Position Sensing	Capacitive or Resistive Touch Sensing
Advantages	No additional electronics required.	Fine-tunable control action (e.g., 3D position sensing for joystick function)	Close integration into product design possible.
	Lowest possible power consumption.		Convenient for implementing flexible, context related switching options (especially if paired with display)
	Operation without any additional supply.		
Disadvantages	Moving parts which require special care taking during integration.	Complementary magnetic materials required	Problematic if operated by person having wet hands or wearing (e.g., latex) gloves
	Exposed to mechanical / electrical stress, resulting in fast wear-out and malfunction	Moving parts which require special care taking during integration	Sensitive to electromagnetic radiation
	Additional coating required if e.g., water proof implementation required.	Sensitive to stronger magnetic fields	Extra design efforts and cost for provisioning e.g., electrodes
	Expensive if robust solution (e.g. automotive) is required	Difficult to clean or disinfect	
	Difficult to clean or disinfect		
	Stylistically old fashioned		

The above overview table shows a relatively long list of possible disadvantages coming along with conventional mechanical switches, which made product designers investigate magnetic, resistive, or capacitive sensing, or combinations of these. Nevertheless, even those alternatives - although having their dominant use within different applications – could be further improved.

- Hermetically sealed, dirt-/water-proof surface integration which is easy to clean or disinfect without need for special coating measures.

- Cost efficient integration directly with the product's housing without need for extra materials that could show different aging behavior with respect to color, shape or brittle.
- Proper touch detection for reliable switching functionality under all circumstances.
- Context sensitive switch appearance for focused, less distracted, control actions.
- Compelling switch designs which are fully integrated with the product's surface.
- Lower environmental footprint by reducing plastic usage, parts usage, volume, and weight.

The answer is *optical force sensing*, which introduces new differentiating applications and design opportunities based on an innovative switching concept. This application note introduces the general concept of optical force sensing and hints to different implementation options, which are all enabled by ams OSRAM portfolio. As the implementation of optical force sensing is closely connected to the mechanical design of the end product's housing, which makes it very cost efficient for the implementation, the application note explains the different design constraints to be considered in order to take maximum advantage of this new switching concept in next generation product designs.

Besides the challenges listed above, sometimes existing switch functions need to be backed by redundant implementations to meet functional safety requirements. For example, a capacitive sensing technology could be complemented with optical force touch to ensure intent through enough force.

1.2 Concept

An optical force sensor consists of two key components: a light emitter and a light sensor. Both are mounted in a chamber, below a reflective ceiling. The emitter floods light towards the ceiling. The ceiling reflects the light back, and the sensor measures the amount of light received, see Figure 1. This reading is referred to as the (no-press) baseline.

When the sensor is pressed, the mechanical design should be such that it results in a ceiling displacement, see Figure 2. More light reaches the sensor, so the sensor readout increases. This readout is known as the *count*, informally this is the number of photons that hit the sensor. The more forceful the press, the lower the ceiling, the more light reaches the sensor, the higher the count. In other words, the change in sensor count correlates to the applied force.

Technically, the optical force sensing technology measures ceiling *displacement*. From an application point of view, it is a user interface element (a button) that measures force: the system can determine that sufficient force is applied to activate a function. This is much harder to achieve with, for example, capacitive touch sensors.

Figure 1: Optical force sensor in no-press state

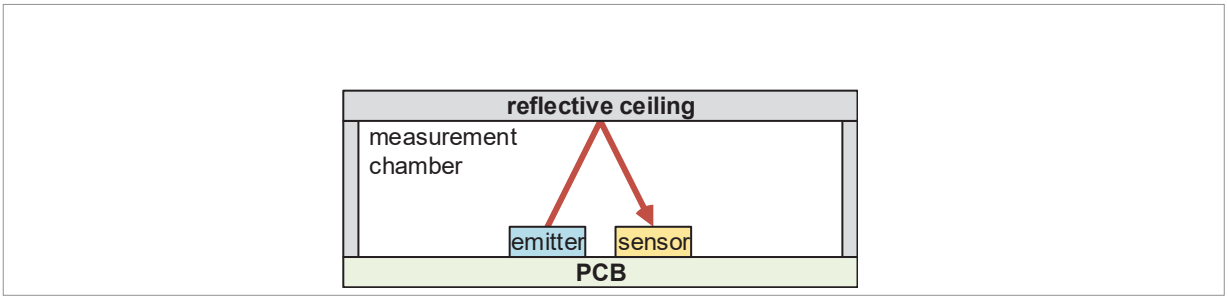
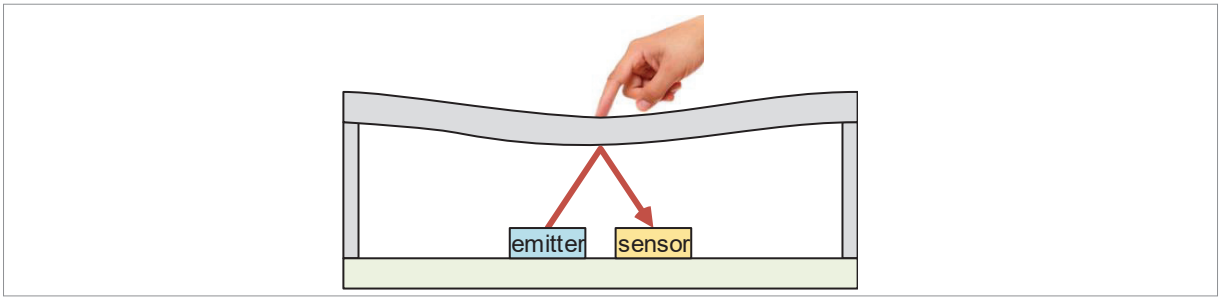
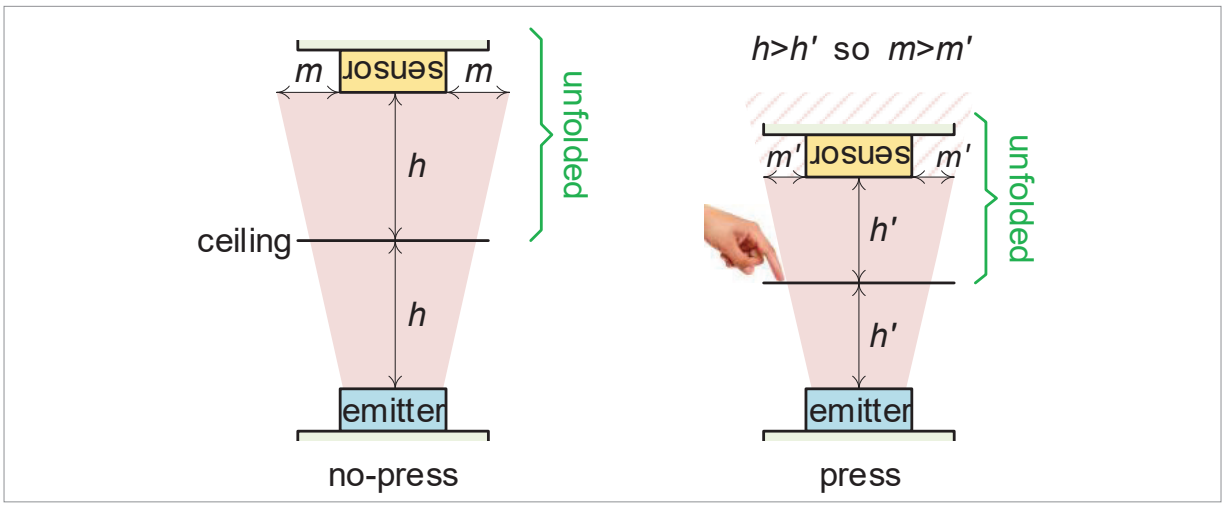


Figure 2: Optical force sensor being pressed



To understand why a ceiling displacement causes an increase in the count, consider the diagrams in Figure 3. These diagrams show the light paths unfolded around the reflective ceiling ("mirror"). The left diagram shows the light cone (red) emanating from the emitter in the no-press case. The ceiling is at a distance of h, so the emitter-sensor distance is $2h$. When the ceiling is pressed (right diagram), the ceiling distance is decreased to h', and the emitter-sensor distance is then $2h'$. The light cone is shorter, so a larger part of its area hits the sensor; "irradiance decreases with distance from the source and follows the inverse square law": a gain of $(h/h')^2$. As a result, the sensor has a higher readout. For a more thorough explanation, see Chapter 2.

Figure 3: Light paths (unfolded around mirroring ceiling) in no-press (left) and press state (right)



1.3 Applications

The simplest application of an optical force sensor is a single button in a user interface panel. Since there is a digital system behind it, the button can distinguish quick tap and long press. The digital system can also detect double tap or actually measure force (for example, a confirmation of some radical action could require a certain force level). Multiple optical force sensors together can even form a high resolution 1-dimensional or 2-dimensional touch pad, a setup that can also detect gestures (flick left, flick right).

Buttons in wet environments (hair dryers, switches in bathrooms, outdoor equipment), buttons that need cleaning (medical), buttons that need to be robust (gloves, high impact functions) are clear candidates for optical force sensing. Other applications include collision detection in robotics (“bumpers”) and force measuring (“scales”).

Advantages of optical force sensing solutions include:

- The buttons are hidden behind a surface; therefore, they are easy to clean and disinfect, and they are dirt and waterproof.
- There are no separate materials for the buttons, which mitigates aging problems (e.g. local discoloration).
- Since the buttons are hidden behind a surface, they could be invisible until needed, at which point a backing light highlights the button icon (shy tech). This backing light could be user-interface state dependent, or based on detection of an approaching hand.
- The system measures force (relying on the elasticity of the ceiling) which is robust compared to some other user interface mechanisms: Wet hands or gloves are no issue.
- Optical force sensing is a digital solution (micro-electronics), which reduces volume, weight, and parts (e.g., wiring harness in automotive), and gives more design flexibility.
- The mechanical properties of the ceiling determine resilience against shock and vibration, but in general that is good (ceiling has low mass and high support).

High level design constraints of optical force sensing solutions include:

- The surface needs to displace when pressed, so either the surface itself needs to be flexible (e.g., plastic) or the surface needs to be mounted flexibly (glass plate on rubber ring).
- The sensor must be able to register the change in light, imposing optomechanical requirements on the design (enough displacement, close enough to the surface, reflective surface).
- The sensor measures light, so care must be taken that the measured light emanates from the emitter and not from the sun or some other user interface light (button backing light). There are software workarounds when this happens, but the easiest system solution is to shield the sensor from these sources by having sufficiently opaque “ceilings” and “walls”.
- The sensor relies on a digital measurement (“integration time”) which causes a slight delay (e.g., 25 ms) compared to, for example, mechanical switches.

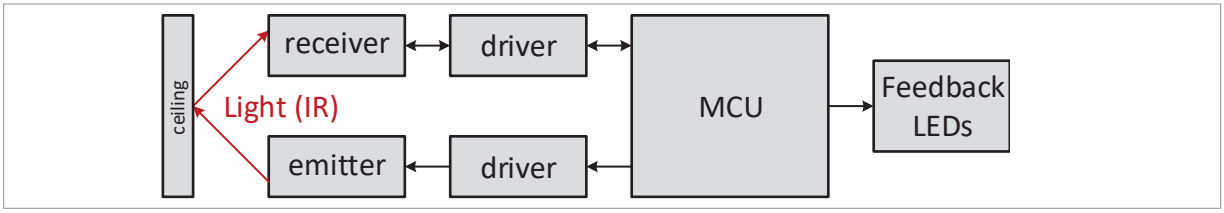
The switching precision and reliability of the intended optical force sensing solution strongly depends on the performance of the deployed components. To allow operation at the lowest possible power levels highly efficient LEDs and detectors must be used. Especially the detector must provide leading-edge signal-to-noise ratio (SNR) for proper decoding of active switching actions. SNR performance around 80 dB or better is required to achieve stable operation at low power consumption.

1.4 System diagram

Figure 4 depicts a minimalistic optical force sensing demonstration system. It provides one “button” (place to assert some force on) and a series of LEDs to indicate the amount of force.

Such a system consists of an element (IRED) that emits light (“emitter”) and element that detects light (“receiver”). A microcontroller unit (“MCU”) controls the emitter, configures the receiver, invokes measurements, performs signal processing on the measurement results and finally, it gives feedback via some LEDs.

Figure 4: System diagram



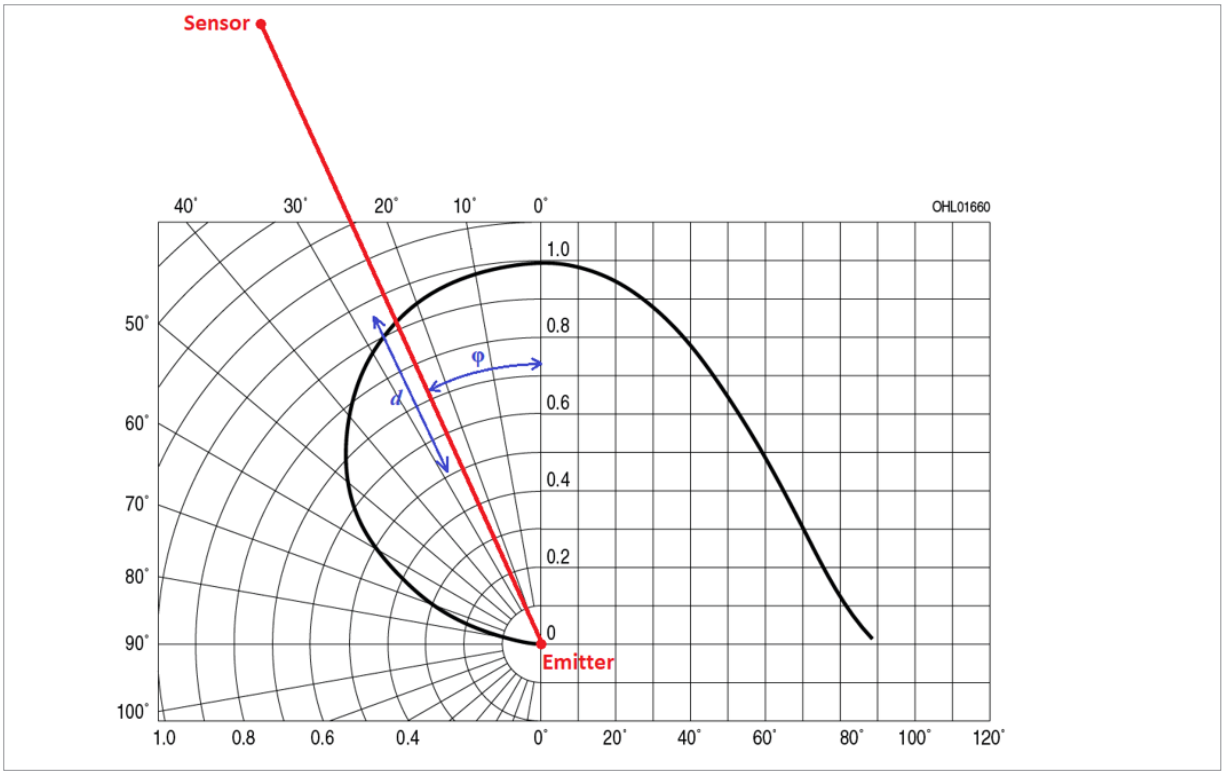
2 Physical model

This chapter presents a physical/mathematical model of an optical force sensor. The aim is to better understand the behavior of the system and the influence of the key parameters. Feel free to skip this chapter, it is not required for the understanding of the rest of this document.

2.1 Phenomena

The model is based on two physical phenomena: illumination *distance* and *angle*. The received radiant intensity decreases with distance from the source following an inverse square law. The radiant intensity also decreases when the line of sight has an increasing angle from the surface’s normal vector. An example of emitter angular dependency is illustrated in Figure 5.

Figure 5: Modeling distance and angle

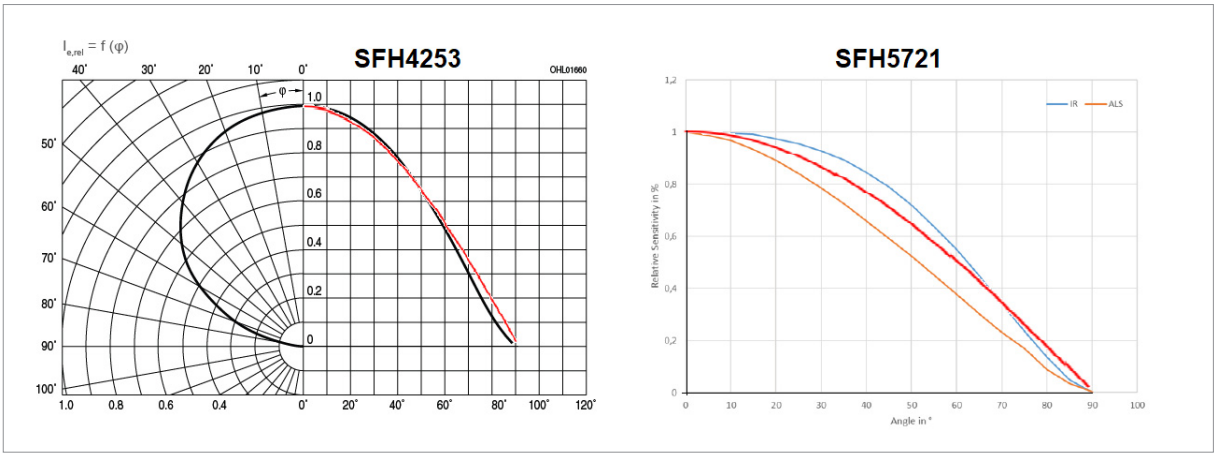


The black curve in Figure 5 is a typical *radiation characteristic* of an LED as can be found in its datasheet (here from SFH 4253). The emitter is at the origin of the graph. The red line is an example of a line of sight to a sensor. The key parameters are distance d (blue) and the angle from normal φ (also blue). This example shows that with an angle of 25º, radiation intensity is still above 90%.

The left-hand side of the graph shows the radiation in polar coordinates, the right-hand side shows the radiation in Cartesian coordinates. The black curve in Figure 5 follows a so-called Lambertian radiation pattern: radiation is proportional to the cosine of the angle φ between the line of sight and the surface's normal vector. Typically, LEDs have this radiation characteristic, but the optics of their package may alter this. Also, sensors have this (incoming) radiation characteristic, and also there, the package may alter this.

Figure 6 left shows the radiation curve (black) overlayed with a cosine (red) function, a near perfect match for this emitter (SFH 4253). Figure 6 right shows the radiation curve of the SFH 5721 sensor (IR in blue) it also has a good match with the cosine (red).

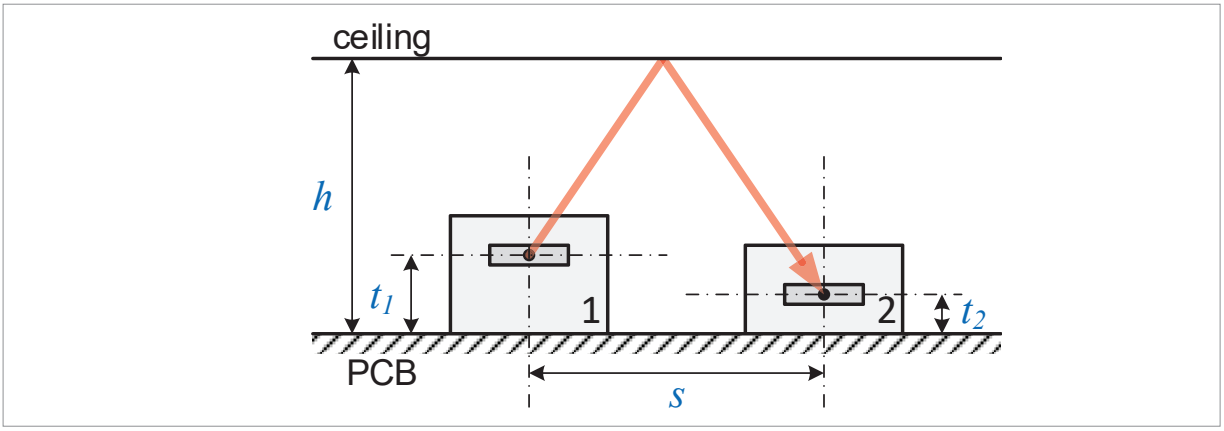
Figure 6: Radiation curve (black respectively blue) versus cosine (red)



2.2 Parameters

Figure 7 shows the model parameters. There are two components (gray), labeled 1 and 2. It doesn't matter for the model which is the emitter and which the sensor, because we only look at the distance and angle of the light path (red line). For ease of discussion, we refer to component 1 as the emitter, and 2 as the sensor.

Figure 7: Key model parameters

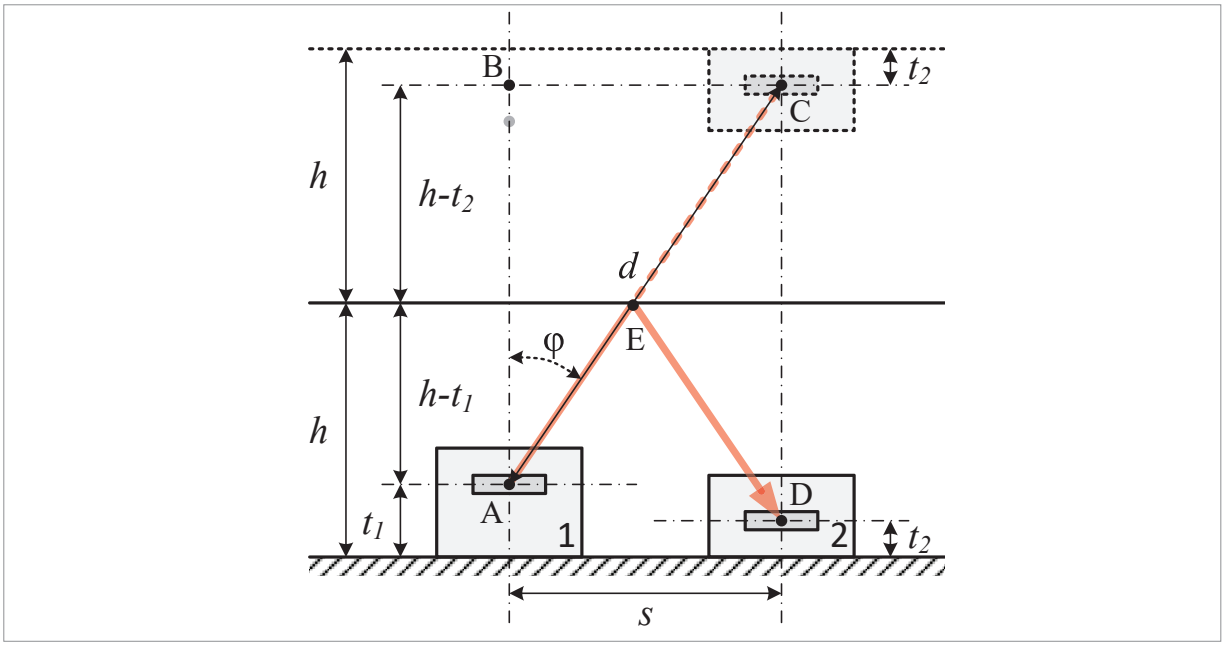


The components have physical dimensions, of interest are the height t_1 of the emitter and the height t_2 of the sensor (we will later define $t = t_1 + t_2$). These heights are measured from the bottom of the PCB to the center of the die inside the component. The two components have a lateral (center to center) spacing of s. Finally note that the ceiling height (the reflector above the PCB) is h.

2.3 Model

Figure 8 is derived from Figure 7. It depicts components 1 and 2 with lateral spacing s (vertical center lines) and heights t_1 and t_2 (horizontal center lines) between their centers A and D. The diagram also shows component 2 mirrored in the ceiling, the mirrored objects are dotted. The mirror of D (center of component 2) is point C. This allows constructing the light ray (red) from A to D by drawing a line from A to C, and where it intersects with the ceiling at E, drawing a line to D.

Figure 8: The model



Note that distance BC is s and $AB = (h\text{-}t_1 + h\text{-}t_2) = (2h\text{-}t_1\text{-}t_2) = (2h\text{-}t)$, where we define $t = t_1 + t_2$. We can now determine d and φ from the triangle ABC: $d^2 = (2h\text{-}t)^2 + s^2$ and $\cos(\varphi) = (2h\text{-}t)/d$.

As mentioned in the introduction, the model is based on distance and angle. The intensity I at the sensor is thus characterized as in Equation 1, where d is the distance the light traverses from emitter to sensor, φ is the angle of the ray of light with respect to the surface's normal vector, $f_1(\varphi)$ is the radiation characteristic of device 1 (the emitter), $f_2(\varphi)$ is the radiation characteristic of device 2 (the sensor), and I_0 is nominal radiation of the emitter.

The model makes several simplifications:

- The emitter is a point source, and the sensor is a point sink. The model takes the centers of the respective dies as those points. A more accurate model would integrate over the surfaces, but it does not seem worth the trouble.
- The ceiling is fully reflective, non-diffusing and flat, and it reflects the one ray of interest from emitter to sensor. In practice the ceiling attenuates the signal, but this is likely a constant factor. This seems discounted for the unknown nominal value of I_0 anyhow.
- Crosstalk (direct light from emitter to sensor) is ignored. In reality, crosstalk is a large contributor to the sensor readout. In practice, this is likely a constant offset for $I(h)$.

Equation 1: Physics

$$I = \frac{f_1(\varphi) \cdot f_2(\varphi)}{d^2} I_0$$

We assume a Lambertian emitter, so $f_1(\varphi) = \cos(\varphi)$ and similarly $f_2(\varphi) = \cos(\varphi)$ for the sensor. Substituting these radiation characteristics in the intensity formula and then using $\cos(\varphi) = (2h\text{-}t)/d$ gives Equation 2.

Equation 2: Intermediate

$$I = \frac{\cos(\varphi) \cdot \cos(\varphi)}{d^2} I_0 = \frac{(2h-t)^2}{d^4} I_0$$

As a last step, we substitute $d^2 = (2h\text{-}t)^2 + s^2$ leading to the final expression Equation 3 for the light intensity as function of ceiling height h.

Equation 3: Model

$$I(h) = \frac{(2h-t)^2}{((2h-t)^2 + s^2)^2} I_0$$

The derivative of Equation 3 is given by Equation 4.

Equation 4: Derivative

$$I'(h) = 4\frac{(2h-t)(s^2 - (2h-t))}{((2h-t)^2 + s^2)^3} I_0$$

The derivative of Equation 4 is given by Equation 5.

Equation 5: Second derivative

$$I''(h) = 8\frac{((2h-t)^2 + s^2)((s^2 - (2h-t)^2) - 2(2h-t)^2) - 6(2h-t)^2(s^2 - (2h-t)^2)}{((2h-t)^2 + s^2)^4} I_0$$

Setting Equation 4 to zero gives three solutions for h, see Equation 6. Recall that best-case t_1 equals t_2 so that $\frac{1}{2}t$ is at the die height of both emitter and sensor, inside the packages, so too low for h. This discards the third solution. Since s is positive the second solution is an even lower h, so we can also discard that one. Only the first solution gives a physically possible solution for h.

Equation 6: Peaks

$$h_{peak} = \frac{1}{2}t + \frac{1}{2}s$$

$$h_{peak} = \frac{1}{2}t - \frac{1}{2}s$$

$$h_{peak} = \frac{1}{2}t$$

Setting Equation 5 to zero gives four solutions, see Equation 7. Again, the ones below ½t can be discarded. From the remaining two, we pick the one higher than h_{peak}. That is the one with both “+”s.

Equation 7: Max slopes

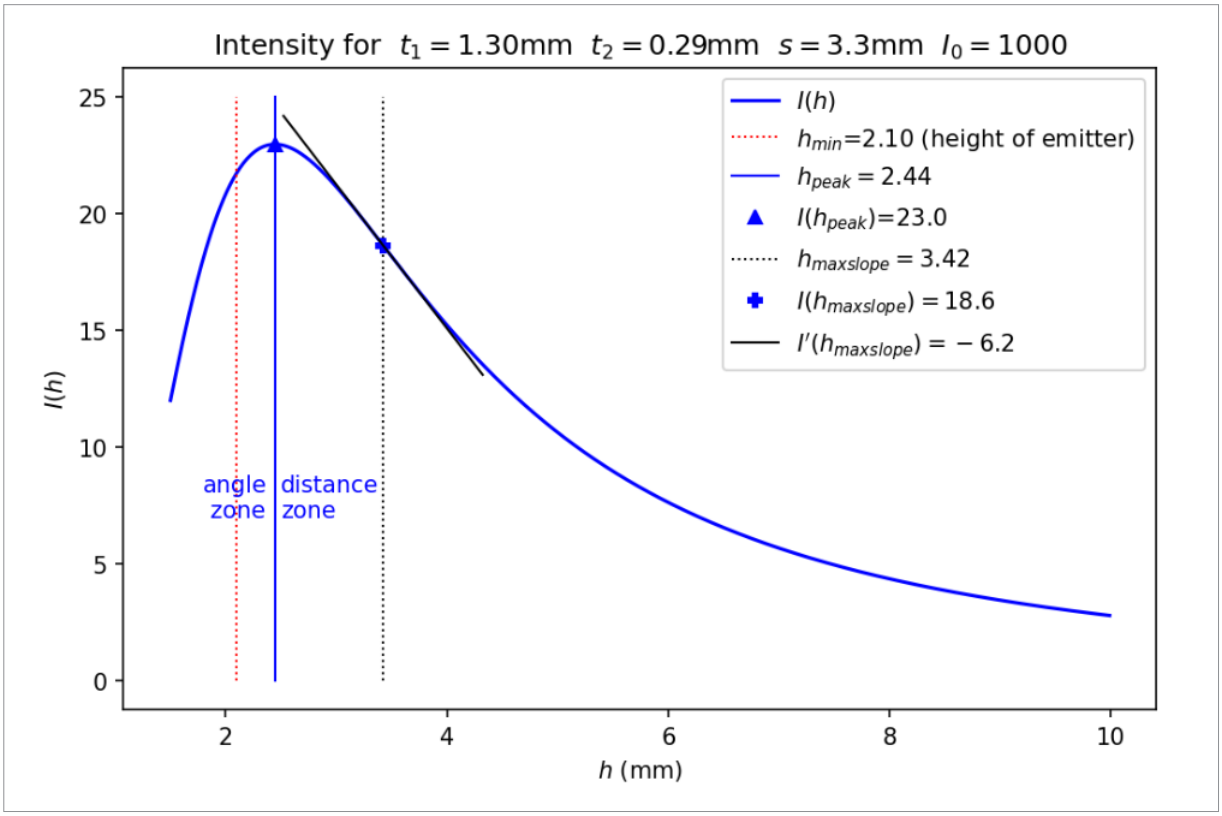
$$h_{maxslope} = \frac{1}{2}t \pm \frac{1}{2}s\sqrt{\frac{4}{3} \pm \frac{1}{3}\sqrt{13}}$$

2.4 Findings

Figure 9 shows $I(h)$. The parameters s, t_1 and t_2 are based on the key components used in Section 3.1. The nominal irradiance I_0 is given an arbitrary value of 1000.

Note the red line h_{min}; it plots the height of the tallest component (here the SFH 4253 emitter), which means that it is physically impossible to have a ceiling lower than that.

Figure 9: Intensity as function of ceiling height



Note that on the right side of the peak, the *distance* phenomenon is dominant (when h increases, $1/d^2$ approaches 0), whereas on the left side the angle phenomenon is dominant (when h decreases, $\cos(\varphi)$ approaches 0). Observe that these two phenomena are opposite: higher ceiling (higher h), means longer light path (higher d), means less radiation on sensor (smaller I). But higher ceiling also means smaller angle of the light ray (smaller φ), means more radiation from emitter, means more radiation on sensor (higher I). The cases are illustrated in Figure 10 and Figure 11.

Figure 10: Low ceiling: angle is dominant

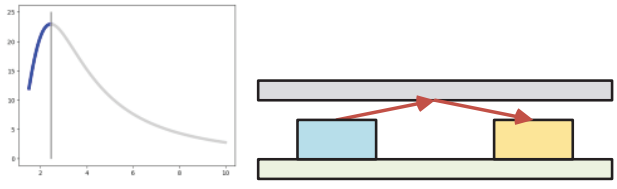
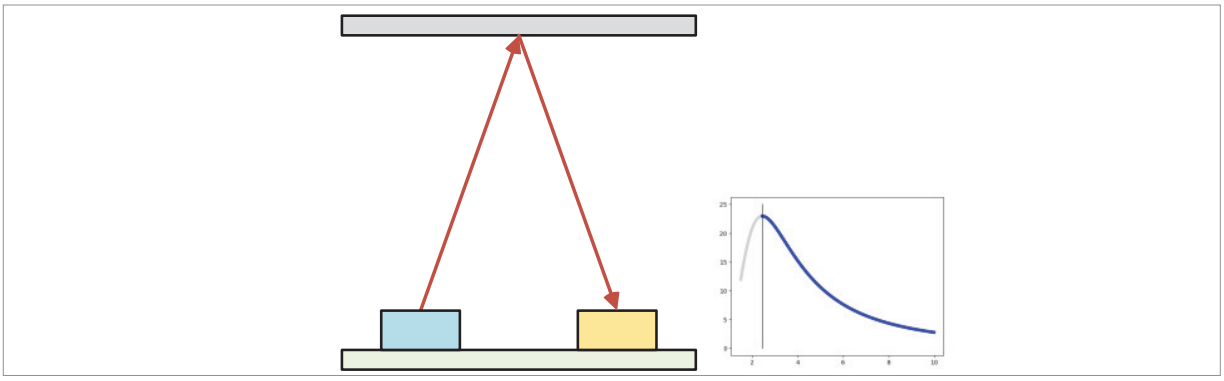


Figure 11: High ceiling: distance is dominant



Between the two 0-approaching extremes, there is a peak. We find it by setting the first derivative of $I(h)$ to 0, see Equation 4 and Equation 6: $h_{peak}=½t+½s$. This value is not particularly interesting.

More interesting is $h_{maxslope}$. It is computed by setting the second derivative $I''(h)$ to 0, see Equation 5 and Equation 7. This determines the point where $I(h)$ has a maximum slope, that is, it identifies the h-region where the ceiling displacements cause the biggest changes in I. It is suggested to design the OFS system with the ceiling at this height.

3 Electrical design

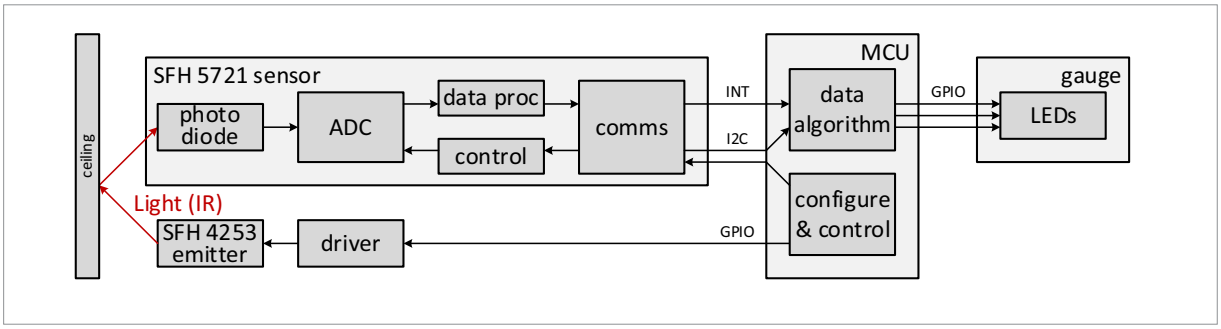
This chapter describes electrical design aspects of an optical force sensing system. To make this document more concrete, specific (key) components have been chosen.

3.1 Key components

An optical force sensor is made up from the following key components: an emitter and its driver, and a sensor and its A/D converter. ams OSRAM offers solutions with different levels of integration.

In this application note we have selected discrete components and not a fully integrated one: as sensor the SFH 5721 *digital light sensor* and as emitter the SFH 4253 *high power infrared emitter*. At the moment of writing this application note, these ams OSRAM components result in a system with the highest SNR, about 80 dB. However, other components are also available to build an optical force system (see Section 3.5). The system diagram is show in Figure 12.

Figure 12: System diagram



The SFH 5721 sensor contains three channels: photopic (mimicking human eye sensitivity), infrared (850 nm, which we use with the SFH 4253 IR emitter) and a dark channel (for dark current compensation). The sensor is accessible via I²C. The MCU writes to configure its integration time, (analog and digital) gain and (single shot or continuous) mode, and reads the measurement values (infrared channel). The SFH 5721 also has an INT pin with dual use. It is possible to program high and low thresholds for the channels, and if the channels exceed them the INT line is asserted. The other use of the INT pin is as input, to trigger a measurement ("synchronous mode"). The SFH 5721 is not automotive grade.

The SFH 4253 is an automotive grade infrared emitting diode (IRED), with a center frequency of 850 nm. We use the SFH 4253-R variant, it has a higher intensity of 11.2 mW/sr to 18.0 mW/sr at 70 mA forward current.

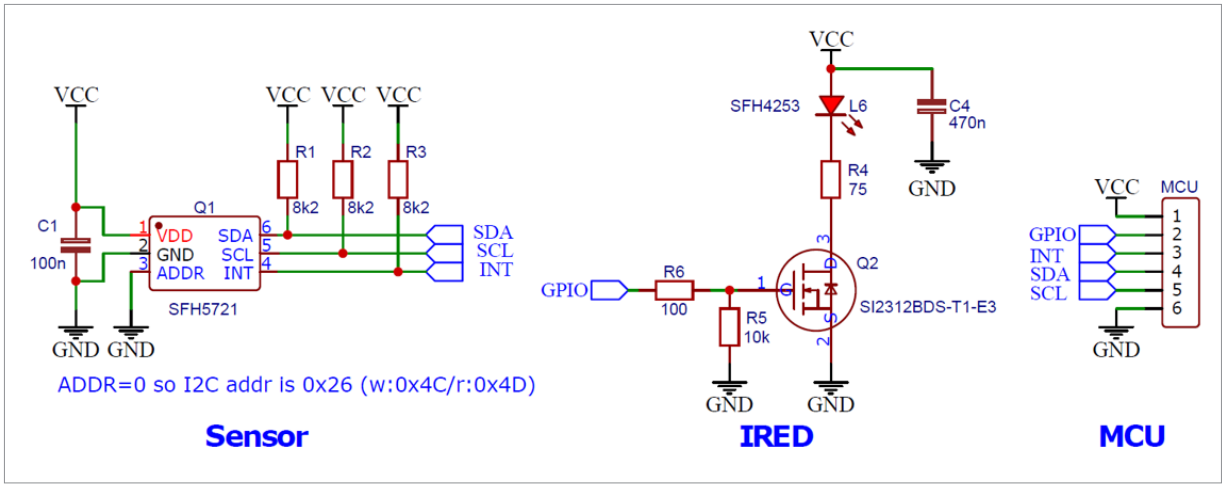
An MCU, for example an ESP32, is used to control the operation and interpret the data and show the measured force level on the gauge LEDs.

3.2 Schematics

The SFH 5721 is rated for a supply voltage of 3V3, so we use a 3V3 microcontroller unit (MCU), an ESP32 in our design. The MCU needs to have a GPIO pin to control the emitter. It needs an I²C block with the SDA and SCL pins to communicate with the sensor, and optionally it needs a GPIO pin for the INT from the sensor.

As can be seen in Figure 13, the SFH 5721 schematics follows the application diagram from its datasheet, with pull-ups (R1, R2, R3) for the communication pins, and a decoupling capacitor (C1). The ADDR pin is tied to GND giving the SFH 5721 a 7-bit I²C address of 0x26, so an I²C write needs 0x4C and a read needs 0x4D as first byte after the I²C start condition. With this ADDR pin, we can assign two different I²C addresses to the SFH 5721, which means that we can only put two SFH 5721 sensors on a single I²C bus.

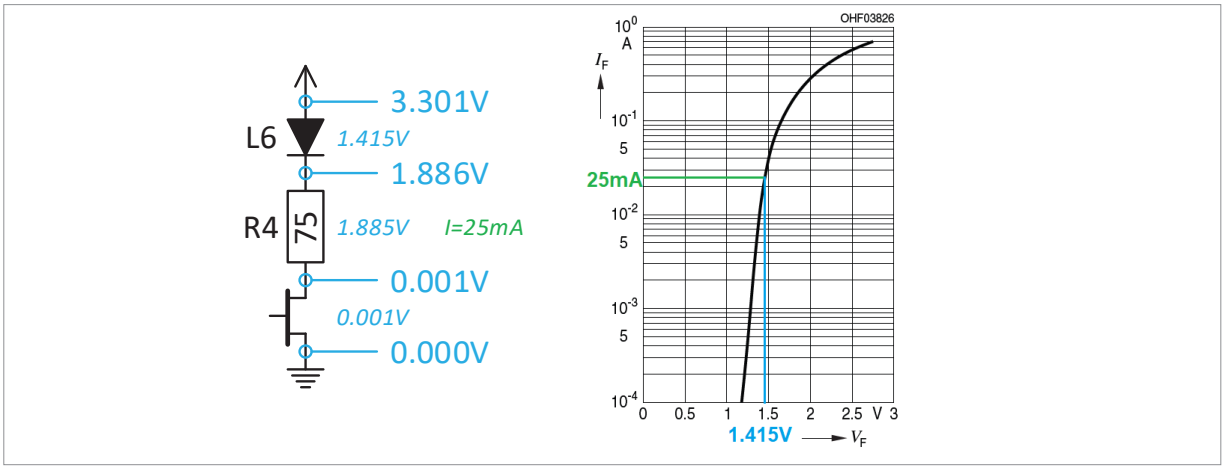
Figure 13: Schematics



As stated in its datasheet, the SFH 4253 IRED can be driven up to 70 mA. To implement a reliably constant drive current we added MOSFET Q2 and current limiting resistor R4 to drive IRED L6, see Figure 13. This configuration (R4=75 Ω) drives the IRED at 25 mA as can be seen in Figure 14 – left side shows a measurement from an actual demoboard, the right side shows the V/I curve of the IRED from its datasheet.

If the MCU can sink 25 mA, the external MOSFET Q2 may be skipped. It would probably lead to bigger board-to-board differences in the IRED current than with the external MOSFET. If board-to-board current differences are of big concern, it is an option to use a dedicated driver like the AS1170 from ams OSRAM. Another reason to add a dedicated driver would be an unstable VCC.

Figure 14: IRED current



3.3 Bill of materials

The schematics of Figure 13 leads to the BOM depicted in Table 2.

Table 2: Bill of materials

ID	Name	Designator	Quantity
1	100 nF	C1	1
2	470 nF	C4	1
3	MCU	J3	1
4	SFH 4253-R	L6	1
5	SFH 5721	Q1	1
6	SI2312BDS-T1-E3	Q2	1
7	8k2 Ω	R1, R2, R3	3
8	75 Ω	R4	1
9	10k Ω	R5	1
10	100 Ω	R6	1

3.4 Layout

Capacitor C1 should be close to Q1, C4 should be close to L6. For mass production, R4 must have a low tolerance rating (±1%) since it determines the IRED brightness, and hence the operating point of the sensor.

The most relevant characteristic is the physical spacing between the SFH 4253 emitter L6 and the SFH 5721 sensor Q1. On our board the center-to-center spacing is 3.3 mm, see Figure 15. This spacing influences the no-press count of the sensor and the range of possible ceiling heights that can be supported (the ceiling height should greater than h_{peak} from Figure 9; if it is too close the ceiling displacement is not reflected in the sensor count).

Figure 15: Sensor and emitter to scale

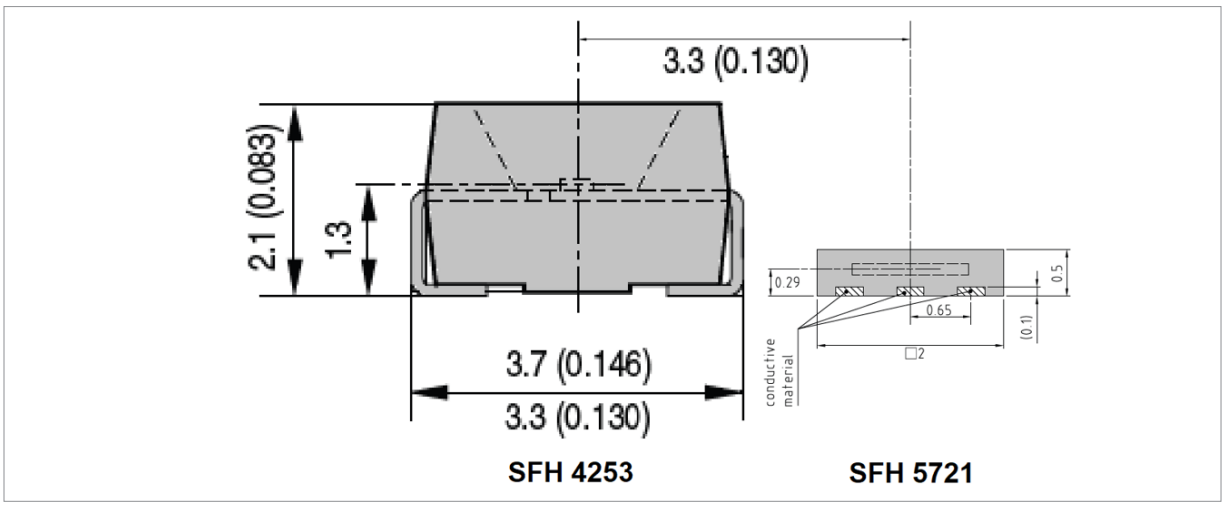
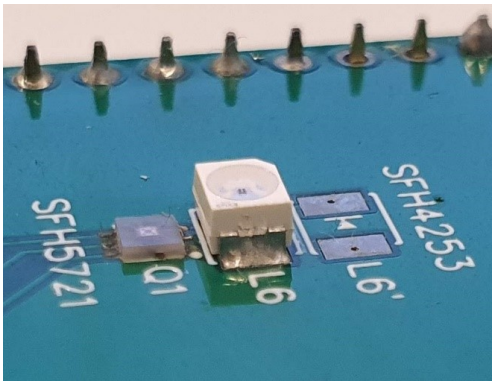


Figure 16 shows our PCB. Since this was a prototype board, we gave the footprint of Q1 longer leads, making it easier to hand-solder. L6 was placed at 3.3 mm from Q1. Other components are moved elsewhere (and on bottom) to prevent light artefacts. There is a second emitter footprint (L6') for trying larger sensor-emitter spacing.

Figure 16: Sensor and emitter

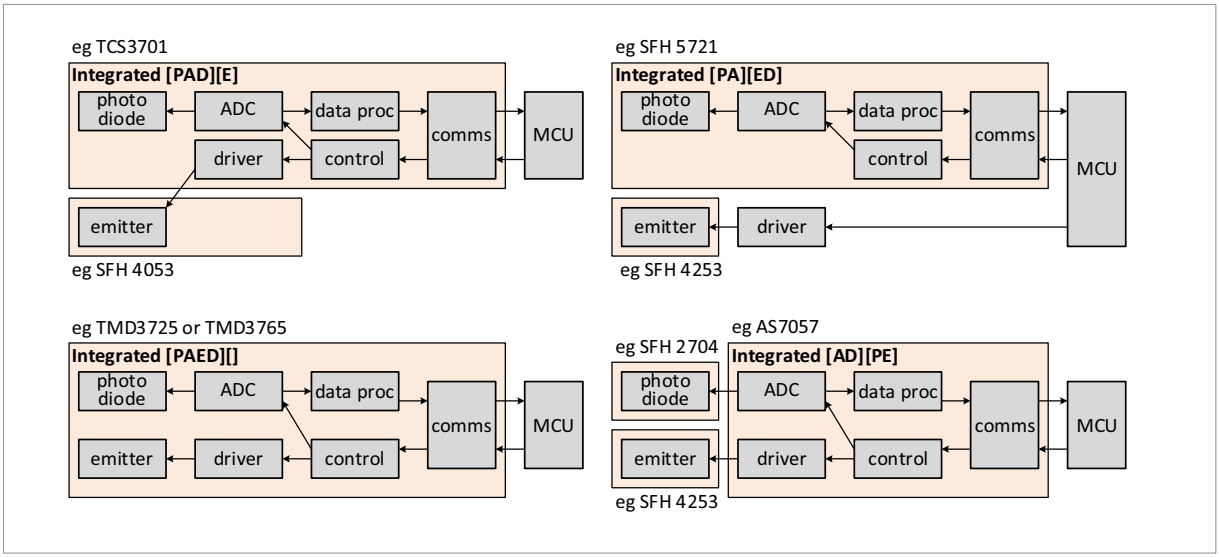


Sensor Q1 should measure the emission from IRED L6 that is reflected via the ceiling. Direct light from L6 to Q1 is unwanted. The SFH 4253 has relatively high package, so that the direct light path to Q1 is minimal.

3.5 Other partitioning's

This document is centered around a design based on an SFH 5721 sensor and an SFH 4253 emitter. However, ams OSRAM offers other components leading to a different partitioning of the functional blocks. The functional blocks we distinguish are the emitter and receiver ("photodiode"). The photodiode needs an ADC to convert the analog signal to a digital one. The emitter needs a driver to convert a digital control signal to a drive current. Furthermore, there are supporting blocks for control, data processing and communications. Figure 17 shows different partitioning's; on the left of the orange blocks example ams OSRAM products are listed.

Figure 17: Different integration of elements (P=photodiode, A=ADC, E=emitter, D=driver)



Each of these partitioning's comes with its own advantages:

- **[PAD][E] TCS3701 + SFH 4053**
  Balance between integration, compactness and freedom in LED selection and positioning.
- **[PA][ED] SFH 5721**
  Suitable for applications requiring good sensitivity at ceiling heights larger than a few millimeters; also suitable where emitter wavelengths other than infrared may be beneficial.
- **[PAED] TMD3725 or TMD3765**
  Highly integrated, hence provides ease of design and manufacturing, in addition to enabling compact geometries.
- **[AD][PE] AS7057**
  Perfect for multi-button applications requiring excellent sensitivity and compact geometries.

4 Mechanical design

Many parameters influence the sensor readout (the count):

- The height of the ceiling with respect to the PCB (emitter and sensor).
- When a force is applied on the ceiling: its elasticity.
- The space between the emitter and the sensor on the PCB.
- The brightness of the emitter.
- The radiation characteristics of the emitter.
- Temperature, since it influences emitter brightness and sensor sensitivity.
- Measurement configuration like integration time and gain.

Also, there are issues in the opto-mechanical design:

- The transmissivity and reflectivity of the ceiling (and the scattering).
- The above two in combination with the chosen wavelength (infrared?).
- Direct crosstalk between emitter and sensor (as opposed to a path via the ceiling).
- Crosstalk with nearby light sources (typically, and optical force sensor implements a user interface element, and that has backing and/or activation lights).

All these parameters are discussed in the sections of this chapter.

4.1 Height of the ceiling

The height of the ceiling greatly influences the (no-press) count and sensitivity. Figure 18 plots the count of the IR channel given the distance between the PCB and the ceiling – keeping all others (e.g., integration time, gain, ceiling reflectivity, sensor-emitter-spacing) unchanged. Notice that the count drops from 10k to 2k when the distance increases from 3 mm to 8 mm. Note that the lab measurements from Figure 18 match well with the theoretical model in Figure 9.

The figure to the right, Figure 19, is the derivative of the count-distance graph (albeit negated). Figure 18 shows a maximum (of nearly 10k) for a distance of about 2.4 mm and as a result, Figure 19 is 0 for that distance (blue triangle). What is more interesting is that Figure 19 itself has a maximum of 2.5 counts/µm for a distance of 3.5 mm (red plus). That is important, because it means that when the ceiling is 3.5 mm from the PCB, the sensitivity (the change in counts when the ceiling is pressed) is biggest.

We also learn from Figure 19 what the sensitivity actually is. The sensor count increases by 2.5 when the ceiling is displaced by 1 µm.

Figure 18: Sensor count versus distance to ceiling

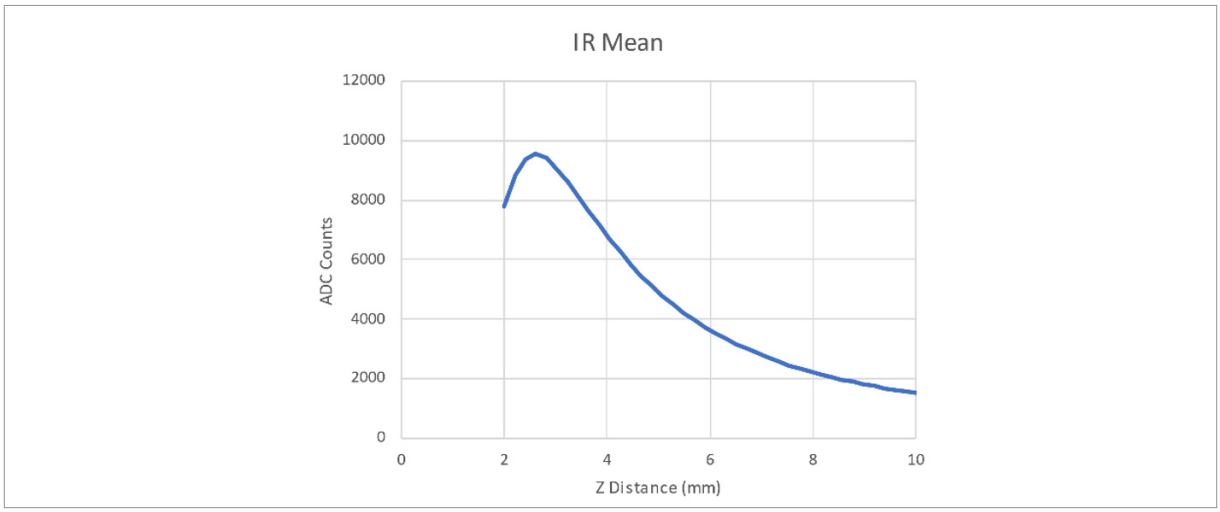
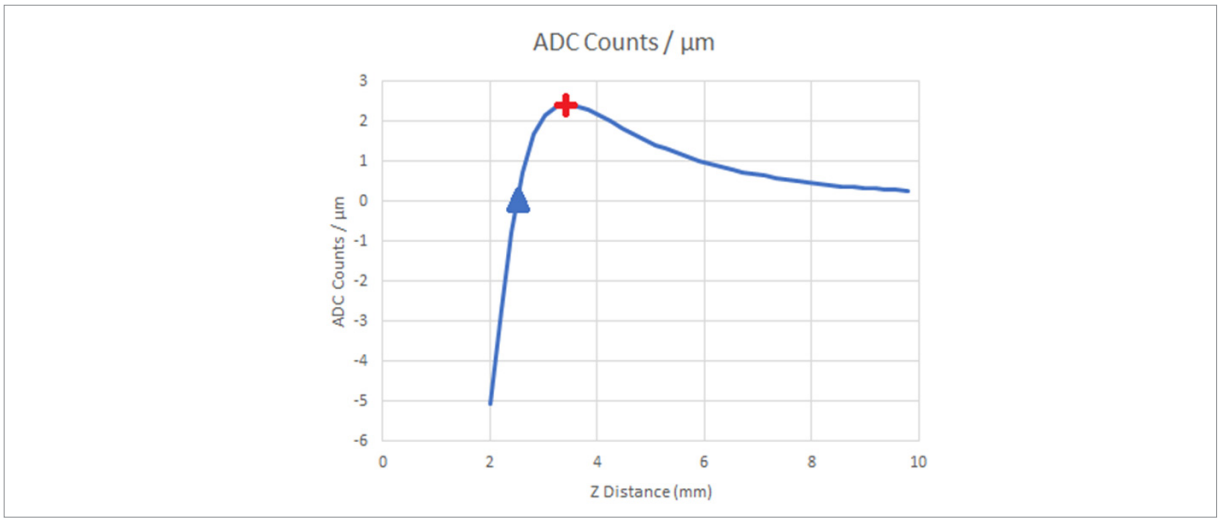
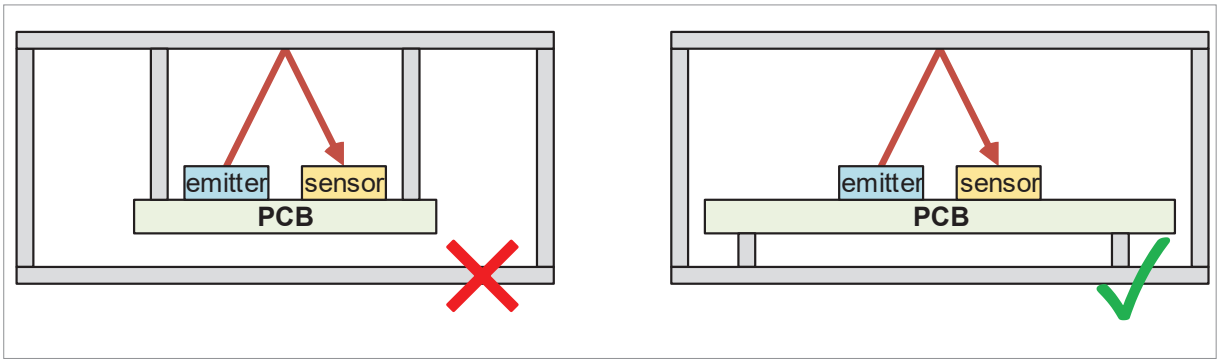


Figure 19: Sensor count sensitivity



It goes without saying that the distance to the ceiling is measured from the PCB. So, in Figure 20, attaching the PCB as shown on the left gives bad results: the PCB moves down with the ceiling when it is pressed. The rightmost fixture is preferred: ceiling displacement is decoupled from the PCB.

Figure 20: PCB support



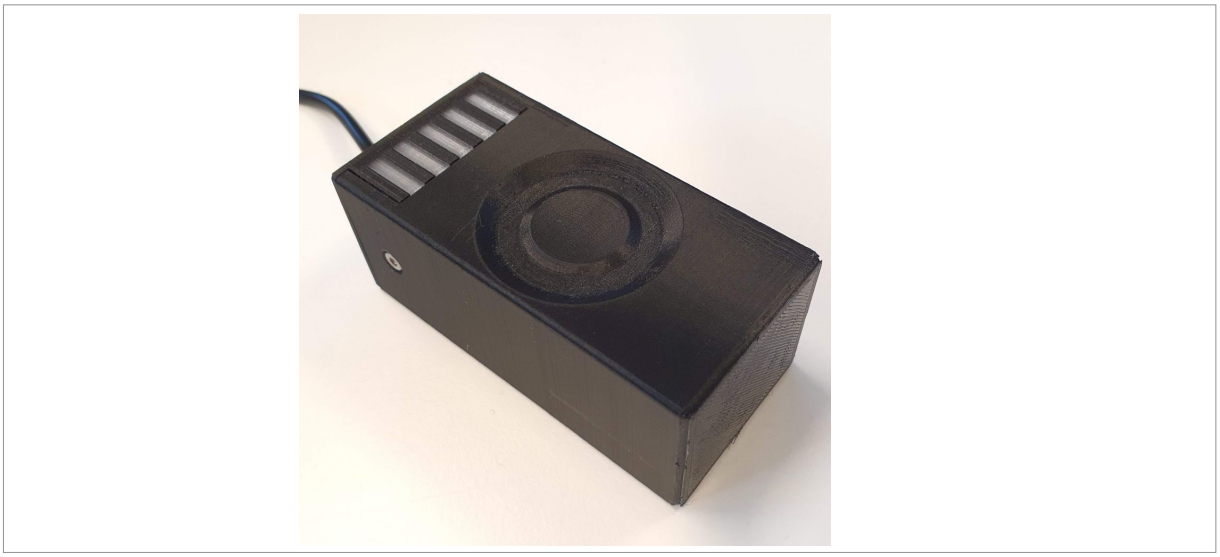
4.2 Elasticity of the ceiling

The previous section shows that a ceiling displacement of about 1 µm is measurable. The choice of ceiling material, the ceiling area and its thickness determine the required force that the user needs to apply.

In our design, the box is 3D printed in PLA – black since that has low transmissivity. The ceiling area is 33×68 mm², which is relatively small and thus deflects little on press. All sides, including the ceiling are 2 mm thick, which again gives little deflection.

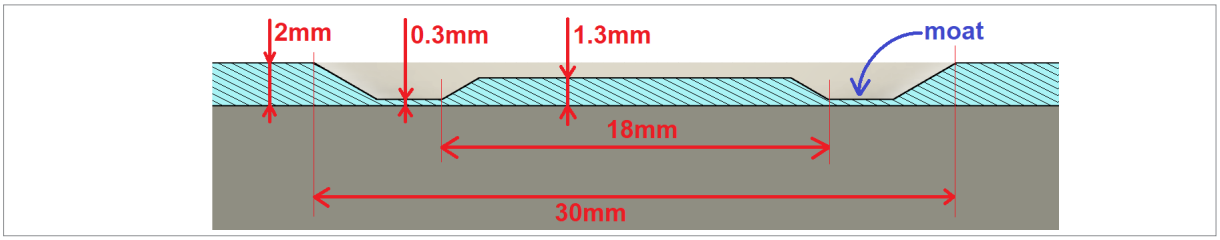
Figure 21: Sunken touch point

Figure 22: Touch point with moat



To mitigate the problem of small deflection, make a sunken touch point, see Figure 21. The inner circle is 25 mm and sunken by 1 mm, so the remaining ceiling thickness is 1 mm. An alternative approach is shown in Figure 22, the so-called *moat* design. The area to press is surrounded by a “moat” that has a considerable width (5 mm) but a small thickness (0.3 mm), see the sectional view in Figure 23. Of course, optical force sensing works on a completely flat surface, but tactile structures help the user (and may increase displacement).

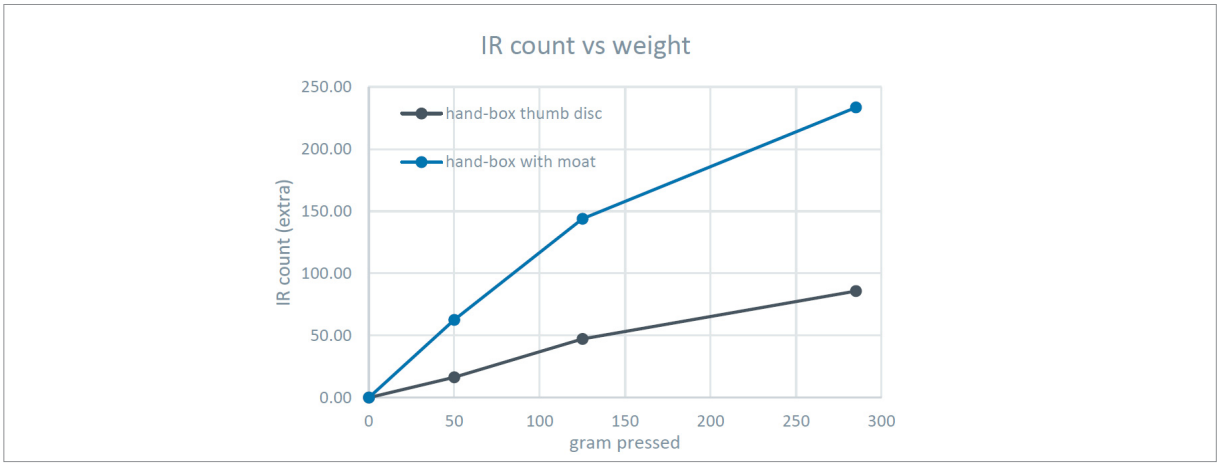
Figure 23: Sectional view of the moat



The moat could also be placed at the bottom side of the ceiling, so that it is out of sight for the user. However, a bit of tactile structures on the user side makes it easy for the user to locate buttons – even blindly.

Figure 24 shows the count increase of the IR channel for the two designs, with 4 different forces applied to the touch point. The moat design roughly has a 3× higher change of count for a given force.

Figure 24: Effectiveness of the moat



4.3 Emitter-sensor spacing

The typical operating point of the system is the blue cross in Figure 9. Here the $I(h)$ curve has the biggest slope, so the system is most sensible to ceiling displacements. The system operates on the “distance” side of the $I(h)$ curve, the side where the *distance* between sensor and emitter is dominant, and not the radiation *angle*.

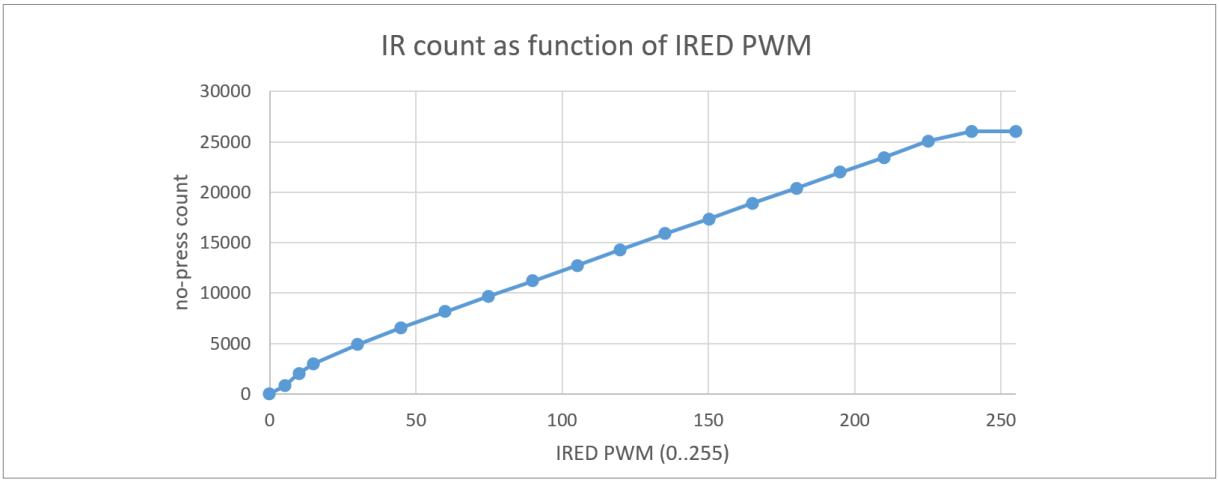
These two phenomena are conflicting: lower ceiling, means shorter light path, means more radiation on sensor. But lower ceiling also means bigger angle of the light ray, means less radiation from emitter, means less radiation on sensor. Therefore, it is advised to have the emitter-sensor spacing as small as the PCB design rules allow.

4.4 Emitter brightness

The schematics of Figure 13 shows the IRED is controlled by a GPIO pin from the MCU. However, the MCU can also drive the IRED using PWM, which gives brightness control over the IRED. Note that the PWM frequency should be a couple of orders higher than the measurement frequency, otherwise the measurements will be disturbed.

Figure 25 shows that when we vary the PWM of the IRED, this has a near-linear effect on the no-press count.

Figure 25: IR count as function of IRED PWM



In practice, IRED PWM appears useless; it saves power, but that can also be achieved by using a higher current limiting resistor (R4). In Figure 13 resistor R4 is relatively high: the IRED is driven with 25 mA, but the IRED could be driven up to 70 mA. If the optical force system is enclosed in a light isolated chamber, a single fixed current is sufficient.

One option of power saving is to only drive the IRED while the sensor is measuring. In the architecture of Figure 13, with a standalone sensor and IRED, the MCU must implement that. Some sensors, for example TMD3xxx, have a built-in IRED driver that is auto synced with measurements.

If ambient light can leak into the measurement chamber, one might apply correlated double sampling: measure once with the IRED off and once with the IRED on and subtract the former from the latter. Sensors like TMD3xxx have this function integrated.

Finally, there is an option to trigger the SFH 5721 by using its INT pin as an input. This might be useful to synchronize two SFH 5721, or to sync it with an emitter flash.

4.5 Emitter radiation characteristics

The emitter radiation characteristics start contributing with smaller heights and bigger emitter-sensor spacing. The reason is that for small h and large s (see the model in Section 2.3) less of the light from the emitter top is used, and more of the light from the emitter side. Most emitters are brighter at the top and dimmer from the side, see for example the curve in Figure 6, a typical Lambertian characteristic – the curve follows a cosine.

4.6 Temperature

The IRED is temperature dependent; the SFH 4253 has a temperature coefficient of brightness is -0.5 %/K. Also, the sensor is temperature dependent; according to a graph in its datasheet the SFH 5721 infrared temperature coefficient is (1.175-0.875) / (85 - -40) = 0.24%. However, the

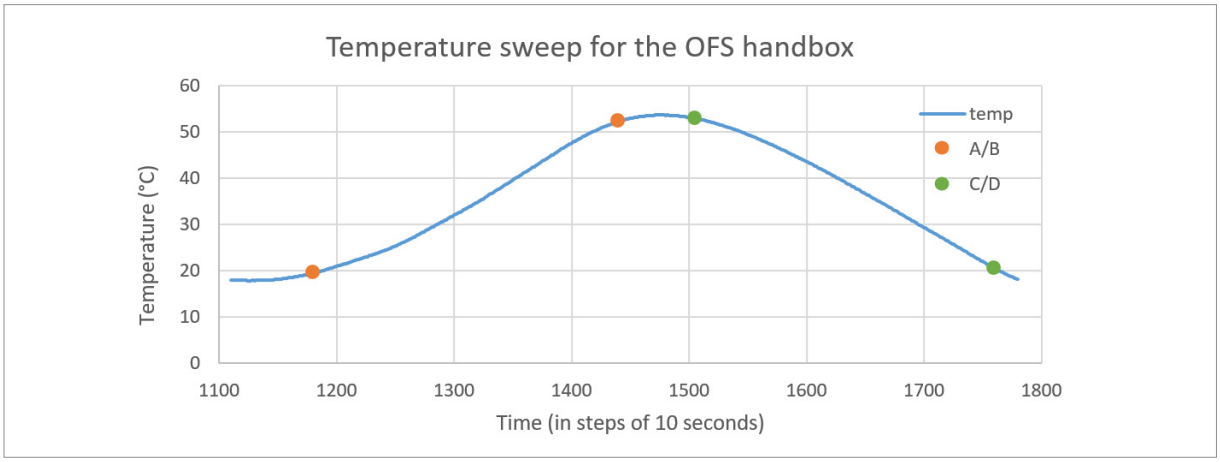
rest of the system is also temperature dependent: the MCU, the MOSFET, and the current limiting resistor (R4) of the IRED.

Figure 26: Temperature sweep in climate chamber

We have put an optical force sensing demo in a climate chamber. Figure 26 shows the setup: the optical force sensing PCB with the white ceiling (forming the measurement chamber) and the ENS210 temperature probe (ENS210) wedged in the measurement chamber. The whole fixture was placed inside a black aluminum box (closed during measurements).

We swept the temperature from 0°C to 60°C and back, taking a temperature measurement every 10 seconds during the total 2 hours. Figure 27 shows the temperature sweep, with A/B markers during the ramp-up and C/D markers during the ramp-down. Note that due to self-heating and thermal capacitance, the system did not reach the minimum 0 °C nor the maximum 60 °C of the climate chamber. Note that PLA has a glass transition temperature between 50 °C and 80 °C, so doing this experiment with a PLA casing is not wise.

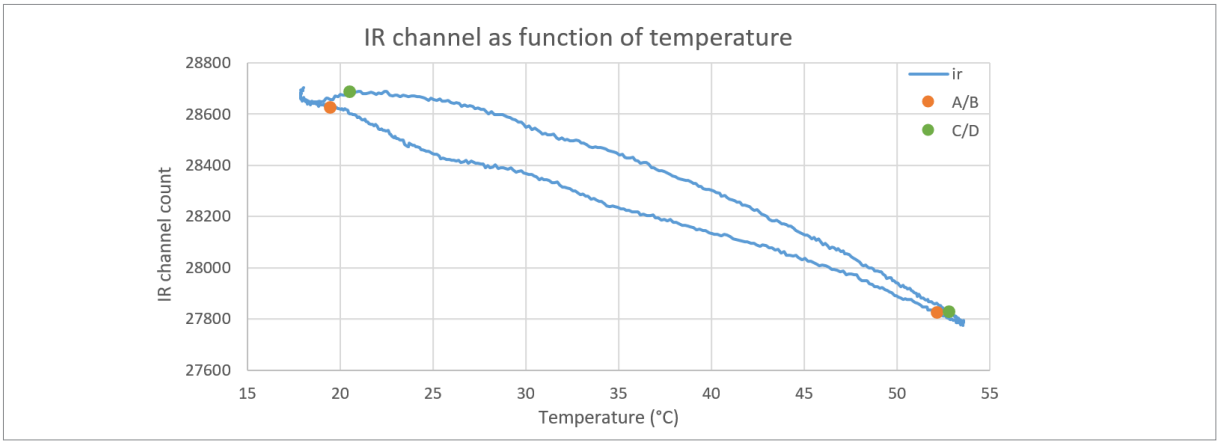
Figure 27: Temperature sweep



During the sweep, the optical force system was left untouched i.e., in the no-press state. Figure 28 shows the counts from the IR channel during this sweep, with the matching A/B and C/D markers. Observe that the curve shows some hysteresis.

In this figure, point A has an IR count of 28624 at 19.5 °C, and B 27822 at 52.2 °C. This is a count decrease of 802 over 32.7 °C, or -25 count/K, or not even -0.1 %/K on 28000 counts. Similarly, when looking at slope C/D we find that C has 27826 counts at 52.85 °C, and D has 28685 counts at 20.55 °C, so here a decrease of 859 over 32.3 °C, or -27 count/K, also not even -0.1 %/K on 28000.

Figure 28: IR counts versus temperature

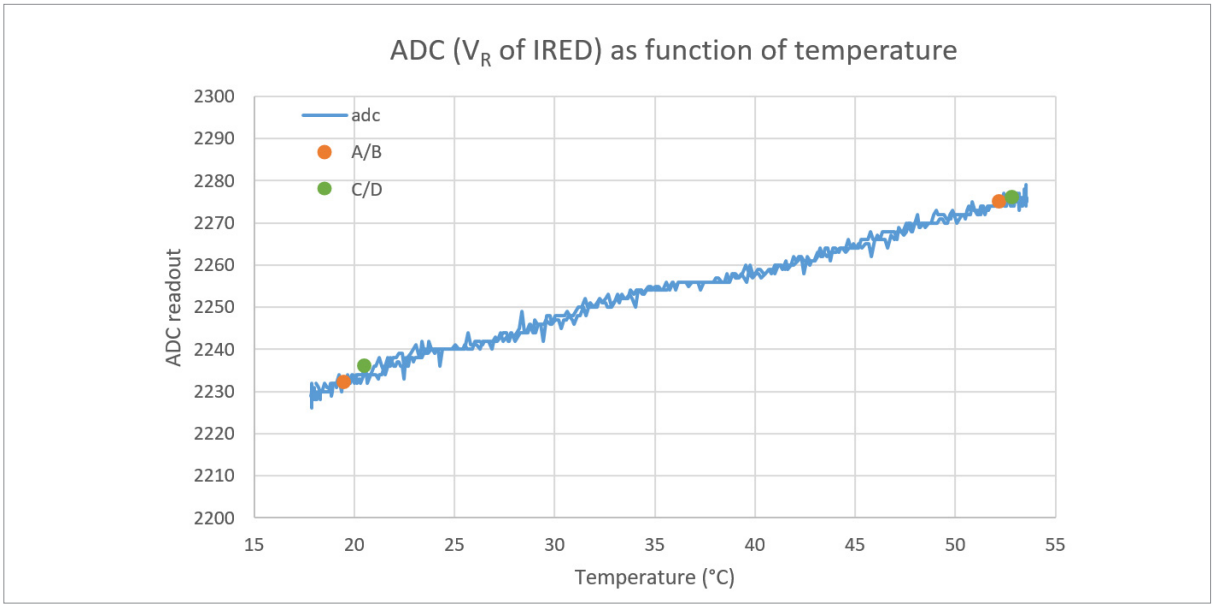


We expected a temperature coefficient of -0.5 %/K for the IRED and +0.24 %/K for the sensor, so -0.26 %/K. However, the entire system shows a temperature coefficient of -0.1 %/K. That might be different for other systems.

Figure 24 shows that a tap of 100 gram causes a count increase of ~100 (depending on the casing). A temperature decrease of 4 K also causes a count increase of 100. Fortunately, a tap is much faster than 4 degrees temperature increase.

The anode of IRED L6 (Figure 13) is at 3V3, the voltage on the cathode is determined by current limiting resistor R4. We connected that net to an ADC of the microcontroller and measured that voltage as function of temperature. Figure 29 shows a linear relation. Implementing temperature compensation in this way could be an option if absolute counts are important.

Figure 29: Measuring voltage as function of temperature

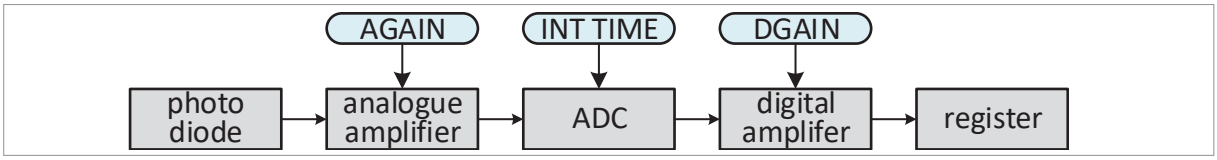


This section discusses the behavior of the electrical system under various temperatures, under the no-press condition. It did not discuss the rigidity of the casing. Naturally the casing should be able to withstand the temperature ranges it is subjected to. For a simple push-release style button application, that is enough. If the optical force system is intended to measure the force, the elasticity of the ceiling plays a role, but also how it changes with varying temperatures. That is beyond the scope of this document.

4.7 Measurement configuration

Light sensors have a measurement pipeline like the one depicted in Figure 30. The photodiode converts photons to electrons, an amplifier then amplifies the electrical signal (by an AGAIN factor). The analog-to-digital converter makes the signal digital, by “counting” the number of photons during a preset interval (“integration time”). An optional amplifier can amplify the digital signal (by multiplying by a DGAIN factor). Finally, the result is stored in a register to be picked up by a read over the communication channel (e.g., I²C) from the MCU.

Figure 30: Measurement pipeline



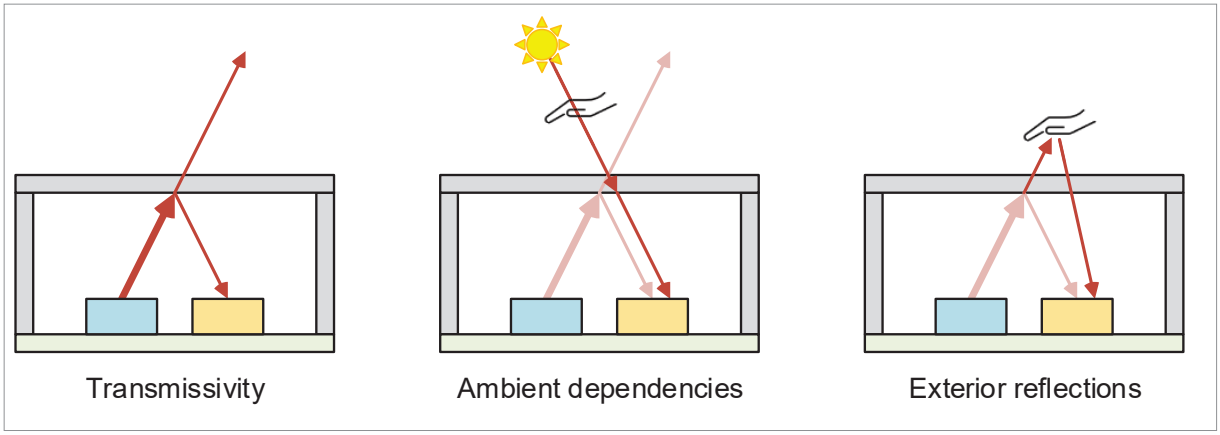
For the SFH 5721, the AGAIN, INTTIME and DGAIN are all linear. It is suggested to keep DGAIN to 1 (the host MCU can do that if necessary). For SFH 5721, an integration time of 25 ms is suggested, because that is the shortest measurement time that results in a 16-bit measurement. Only if the 25 ms delay is unacceptable for the application it is suggested to pick a lower value.

The analog gain should be such that the no-press baseline count of the sensor is at 50% of its range. That results in a headroom of 50% when the ceiling is pressed.

4.8 Transmissivity, reflectivity, and scattering

Figure 31 illustrates transmissivity and two issues coupled to that. The left most diagram sketches the phenomenon: only part of the light emanating from the emitter is reflected by the ceiling, another part is transmitted through the ceiling out of the measurement chamber. In other words, the ceiling is partially transparent for the wavelength the sensor registers.

Figure 31: Transmissivity



The middle diagram in Figure 31 illustrates the issue of ambient dependencies. If the ceiling is transmissive for the sensor's wavelength, ambient light (e.g., from the sun) can also transmit from the outside into the measurement chamber, reaching the sensor. If something temporarily interrupts that light (waving hand), the sensor might erroneously register that as a ceiling displacement. Correlated double sampling (see section 4.4) might mitigate this issue.

The right most diagram of Figure 31 illustrates the issue of exterior reflections. If the ceiling is transmissive, a strong reflector above the ceiling sends an extra amount of light back onto the sensor. Without the ceiling being displaced, the sensor registers an extra amount of light.

These issues are not mere theoretical, as the examples below illustrate.

Figure 32: White PLA partially transmissive for IR

Since the ceiling needs to reflect enough light for the system to be sensitive for small ceiling displacements, one might be tempted to make the ceiling white. We did that and found out that a ceiling of 2 mm white PLA (Polylactic Acid, a filament often used in FFF – fused filament fabrication printing) is partially transmissive for 850 nm infrared: Figure 32 shows that the camera picked up the light (here purple) emitted by the IRED in the center.

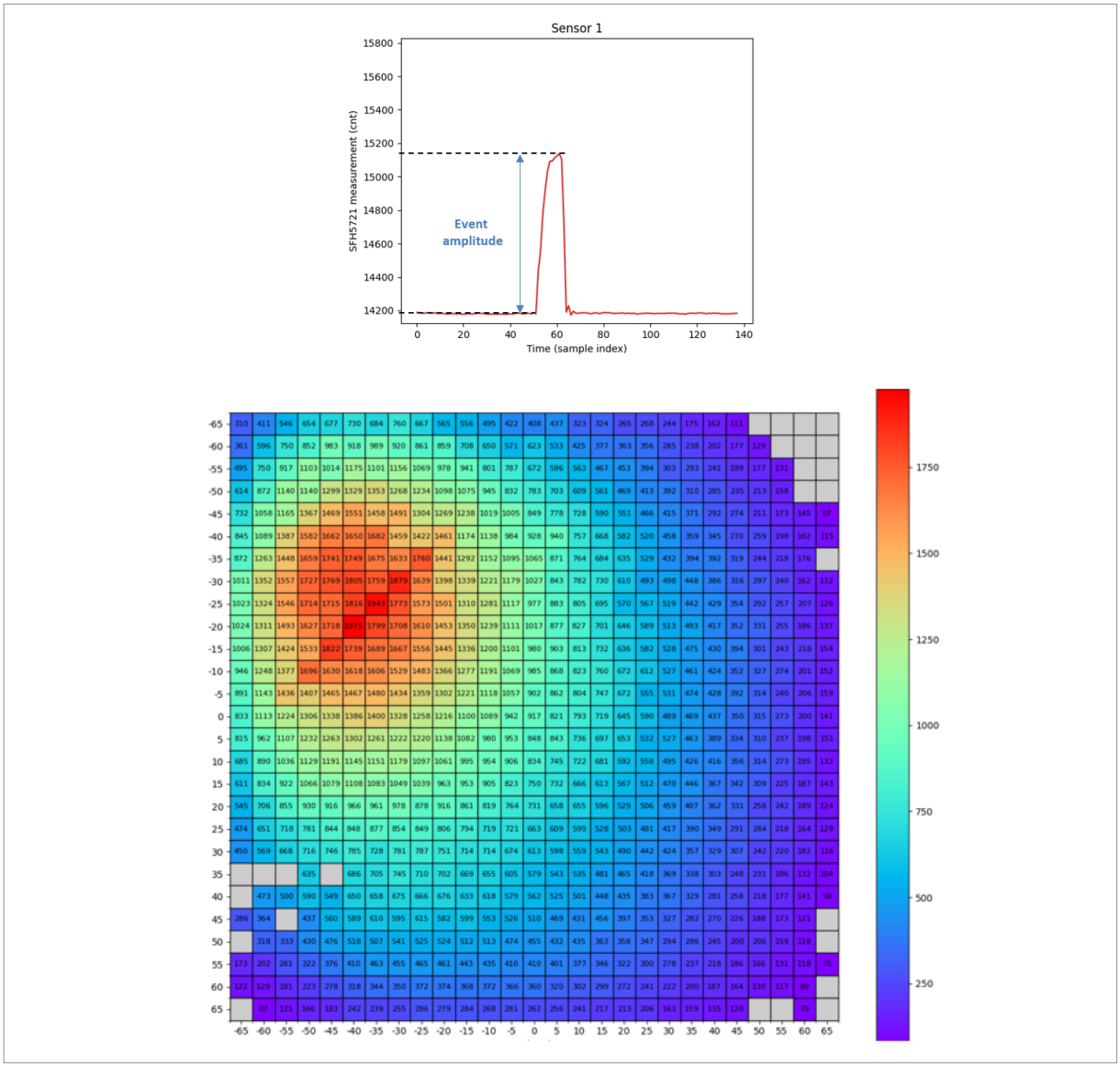
Figure 33 illustrates exterior reflections in action. A microfiber cleaning cloth for glasses reflects so much light that the gauge level is 4 out of 5 LEDs.

Figure 33: Exterior reflections

Considering transmissivity alone, it seems black PLA is preferable over white PLA. However, the reflectivity of black PLA is about a factor 10 below white PLA. The final solution was black PLA (to reduce transmissivity) with a white coating on the inside (to increase reflectivity).

The reflector (the reflective coating) does deserve some attention as the next practical example illustrates. On a 160×160 mm² box with a sensor in the upper left corner, a robot systematically tapped each patch of 5×5 mm², measured the counts, subtracted the no-press baseline, and plotted this event amplitude on a 2D heatmap, see Figure 34. We observe a diagonal bias, and we believe this may be due to the 3D fused filament fabrication (FFF) printer favoring printing diagonal structures.

Figure 34: Heatmap of event amplitude showing diagonal bias



Do not use a 3D FFF printer for the reflective coating at the inside of the ceiling. A more homogeneous scattering is preferred – start by gluing a piece of paper, like a self-adhesive notebook label.

4.9 Wavelength

The optical force sensing system itself has no preference for a wavelength. Of course, when using two discrete components their wavelengths should match. One reason to choose for infrared is its invisibility to the human eye. That is only an argument when the chamber is not (completely) light-isolated from the environment.

Some materials are translucent for infrared, but much less so for blue. In that case it might be beneficial to use a blue emitter and a sensor with a photodiode with filter passing blue.

In general, we do not recommend using the emitter also for backlighting. It is a mix of functions, and also introduces the ceiling-transmissivity problems into the system (see for example Figure 33).

4.10 Emitter-sensor crosstalk

The IRED we have chosen (SFH 4253) is not only higher than the sensor, but also has a build-in reflector – see Figure 15. Consequently, there is little direct light from the emitter to the sensor. That is good, because we need the path via the ceiling to measure ceiling height.

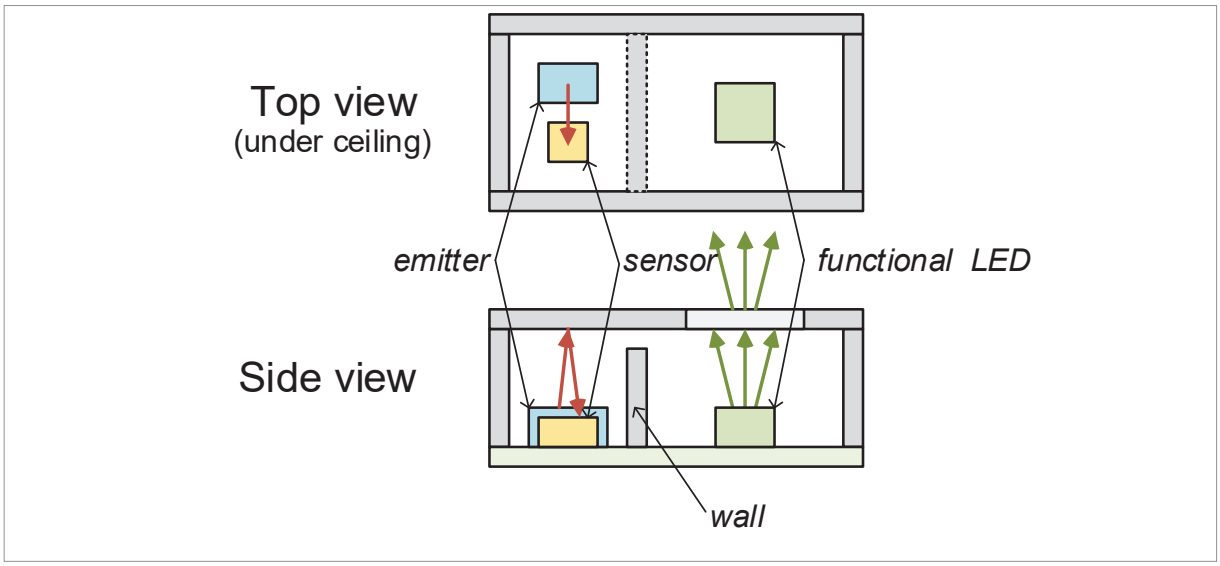
If the IRED is lower or has a wider field-of-illumination (cone), it would still be acceptable as long as the crosstalk is sufficiently smaller than the reflected light intensity.

4.11 Crosstalk with nearby light sources

An optical force sensing system might be used as part of an HMI system containing feedback to the user. The button may have some tactile structures, and a press might be supplemented by visual confirmation to the user via a functional LED that lights up at that location. This means the ceiling is transparent for the functional LED, and with that very likely also for the emitter wavelength. This means the measurement chamber is no longer isolated from ambient light.

One way to solve this problem is to widen the measurement chamber and put the emitter and sensor on one side and the functional LED on the other side, see Figure 35. This works since a press on the ceiling at the position of the function LED also lowers the ceiling above the emitter and sensor and can thus be measured. A separating wall prevents ambient light reaching the sensor. The wall should not reach up to the ceiling, or presses will no longer lower the ceiling above the emitter and sensor.

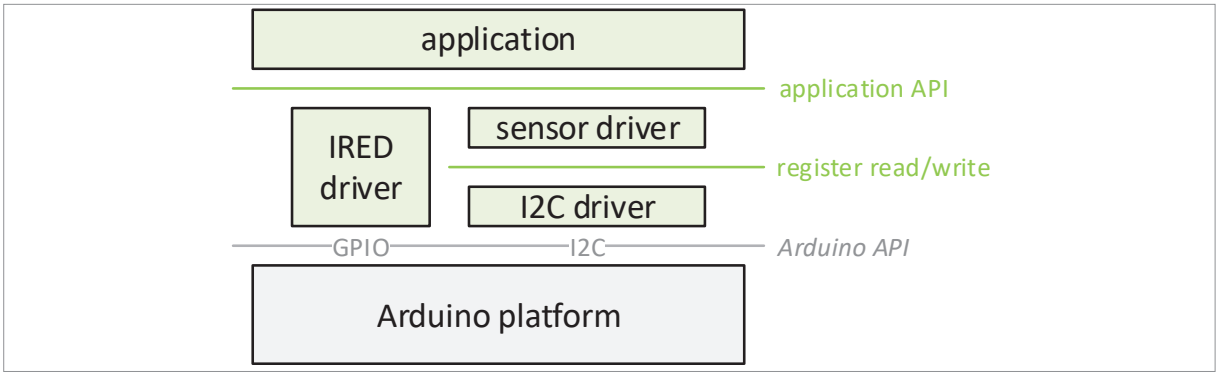
Figure 35: Separating the emitter and sensor from the functional led in the measurement chamber



5 Software

This chapter describes the software needed to control an optical force sensor. This example application aims at clarity and thus also simplicity; it is not meant as production code. Again, we focus on an SFH 5721 sensor paired with an SFH 4253 IRED. We assume a microcontroller with I²C and GPIO.

Figure 36: Software architecture



This chapter gives the code for a simple optical force sensing system; all green code in Figure 36 is presented. It assumes a lower API for I²C (to control the SFH 5721 sensor) and GPIO (to control the SFH 4253 emitter). Such API's typically come with the MCU platform, and in our case, we have chosen to build on top of the well-known Arduino API (actually, ESP32). However, we have split the I²C/sensor in two layers, where the bottom layer offers an I²C read and write register and the upper layer is the actual SFH 5721 driver. This should make it easier to port to other MCU platforms.

To help debugging and error solving, the code performs many prints over the serial port using `Serial.printf()`.

If all code from all subsections is copy-and-pasted into one source file, it should compile under Arduino for ESP32.

5.1 IRED driver

The SFH 4253 is a plain IRED (infrared LED) that is controlled by a GPIO pin via a MOSFET. In our board the MOSFET is connected to GPIO pin 32, and the MOSFET does not inverse: GPIO high means IRED on. The driver is built on top of the Arduino API.

```
#define IRED_PIN      32

// Switches IRED full on
void ired_on() {
  digitalWrite(IRED_PIN, HIGH);
}
```

```
// Switches IRED off
void ired_off() {
  digitalWrite(IRED_PIN, LOW);
}

// Configures GPIO pin that controls the IRED (and ensures it's off)
void ired_init() {
  pinMode(IRED_PIN, OUTPUT);
  ired_off();
  Serial.printf("ired : init\n");
}
```

5.2 I²C driver

To keep the I²C driver in the application note simple, it is hardwired to an I²C bus on pins 21 and 22.

```
#include <Wire.h>
#define I2C_BUS_FREQ  (400*1000) // or 100kHz
#define I2C_SCL_PIN   22
#define I2C_SDA_PIN   21

// Configures I2C bus
void i2c_init() {
  Wire.begin( I2C_SDA_PIN, I2C_SCL_PIN, (uint32_t)I2C_BUS_FREQ);
  Serial.printf("i2c  : init\n");
}
```

A second simplification is that the device address (of the SFH 5721) is also hardwired.

```
#define SFH5721_I2CADDR7BITS_0              0x26 // ADDR pin to GND
#define SFH5721_I2CADDR7BITS_1              0x27 // ADDR pin to VDD
#define SFH5721_I2CADDR                     SFH5721_I2CADDR7BITS_0
```

The write function takes a register address and a byte buffer. It returns 0 on success, for non-zero check the Arduino error codes.

```
// Write `size` bytes from `buf` to SFH5721 register at `addr`.
// Possible results
//   0: success.
//   1: data too long to fit in transmit buffer.
//   2: received NACK on transmit of address.
//   3: received NACK on transmit of data.
//   4: other error.
//   5: timeout
int  sfh5721_i2c_write(uint8_t addr, uint8_t*buf, int size) {
```

```
  Wire.beginTransmission(SFH5721_I2CADDR);     // START, I2CADDRESS
  Wire.write(addr);                            // send register address
  for( int i=0; i<size; i++)
    Wire.write(buf[i]);                        // send all bytes in buf
  int result= Wire.endTransmission(true);      // STOP
  return result;
}
```

The read function also takes a register address and a buffer that will be used to store the read register value into;

```
// Read `size` bytes from SFH5721 register at `addr` into `buf`.
// Possible results, see sfh5721_i2c_write();
// adds 10 if number of read bytes has mismatch.
int sfh5721_i2c_read (uint8_t addr, uint8_t*buf, int size) {
  Wire.beginTransmission(SFH5721_I2CADDR);     // START, I2CADDRESS
  Wire.write(addr);                            // send address of register DEVID
  int result= Wire.endTransmission(false);     // Repeated START
  Wire.requestFrom(SFH5721_I2CADDR,size,1);    // Read size byte, STOP
  if( Wire.available()!=size ) result += 10;
  for( int i=0; i<size; i++) {
    buf[i] = Wire.read();                      // Copy all bytes from Wire to buf
  }
  return result;
}
```

We have two convenience functions. One is an overloaded write function: since all registers are one byte, it is convenient to pass a single byte instead of a byte buffer.

```
// Write `val` to SFH5721 register at `addr`.
// Shorthand for buf write with a size of 1, the most common call.
int sfh5721_i2c_write(uint8_t addr, uint8_t val) {
  uint8_t buf[1] = {val};
  return sfh5721_i2c_write(addr,buf,1);
}
```

The second convenience function is an empty transaction, a ping, to see if the device acknowledges. This ensures the device is present, without changing its state.

```
// Pings the device to check if I2C is wired well.
// 0 is success (see write for others)
int sfh5721_i2c_ping() {
  Wire.beginTransmission(SFH5721_I2CADDR);     // START, I2CADDRESS
  int result= Wire.endTransmission(true);      // STOP
  return result;
}
```

This completes the I²C register access abstraction. Next section is the actual sensor driver.

5.3 Sensor driver

The first part of the SFH 5721 sensor driver is a list of all registers and their bit-field values. We have as naming convention for registers: SFH5721_R*NN*_*RRR*, where SFH5721 is fixed, after the R follows a (hexadecimal) register address NN, then the register name *RRR*. The bit-fields of a register follow this naming SFH5721_R*NN*_*RRR*_*FFF*_*VVV*, where the appended *FFF* is the bit-field name and *VVV* is the value (name). So, for example, there is configuration register at address 9: `SFH5721_R09_MCONFA`, it contains a bit-field known as “integration time” with several values, and `SFH5721_R09_MCONFA_IT_25M` is one of them, for an integration time of 25 ms.

The bit-fields of a register are listed immediately below the register with an extra indent.

```
#define SFH5721_R00_OPSEL                    0x00
#define   SFH5721_R00_OPSEL_RESET              0b10000000
#define   SFH5721_R00_OPSEL_SMODE_SYNC         0b00100000
#define   SFH5721_R00_OPSEL_SMODE_ASYNC        0b00000000
#define   SFH5721_R00_OPSEL_OMODE_SINGLE       0b00010000
#define   SFH5721_R00_OPSEL_OMODE_CONT         0b00000000
#define   SFH5721_R00_OPSEL_EN_ALS             0b00000100
#define   SFH5721_R00_OPSEL_EN_IR              0b00000010
#define   SFH5721_R00_OPSEL_EN_DARK            0b00000001
#define   SFH5721_R00_OPSEL_EN_NONE            0b00000000
#define SFH5721_R01_STAT                     0x01
#define   SFH5721_R01_STAT_DRY_DARK            0b00000001
#define   SFH5721_R01_STAT_DRY_IR              0b00000010
#define   SFH5721_R01_STAT_DRY_ALS             0b00000100
#define SFH5721_R09_MCONFA                   0x09
#define   SFH5721_R09_MCONFA_IT_0M2            0b00000000
#define   SFH5721_R09_MCONFA_IT_0M4            0b00001000
#define   SFH5721_R09_MCONFA_IT_0M8            0b00010000
#define   SFH5721_R09_MCONFA_IT_1M6            0b00011000
#define   SFH5721_R09_MCONFA_IT_3M1            0b00100000
#define   SFH5721_R09_MCONFA_IT_6M3            0b00101000
#define   SFH5721_R09_MCONFA_IT_12M5           0b00110000
#define   SFH5721_R09_MCONFA_IT_25M            0b00111000
#define   SFH5721_R09_MCONFA_IT_50M            0b01000000
#define   SFH5721_R09_MCONFA_IT_100M           0b01001000
#define   SFH5721_R09_MCONFA_IT_200M           0b01010000
#define   SFH5721_R09_MCONFA_IT_400M           0b01011000
#define   SFH5721_R09_MCONFA_IT_800M           0b01100000
#define   SFH5721_R09_MCONFA_IT_1600M          0b01101000
#define   SFH5721_R09_MCONFA_AGAIN_1           0b00000000
#define   SFH5721_R09_MCONFA_AGAIN_4           0b00000001
#define   SFH5721_R09_MCONFA_AGAIN_16          0b00000010
#define   SFH5721_R09_MCONFA_AGAIN_128         0b00000011
#define   SFH5721_R09_MCONFA_AGAIN_256         0b00000100
#define SFH5721_R0A_MCONFB                   0x0A
```

```
#define   SFH5721_R0A_MCONFB_DGAIN_1          0b00000000
#define   SFH5721_R0A_MCONFB_DGAIN_2          0b00100000
#define   SFH5721_R0A_MCONFB_DGAIN_4          0b01000000
#define   SFH5721_R0A_MCONFB_DGAIN_8          0b01100000
#define   SFH5721_R0A_MCONFB_DGAIN_16         0b10000000
#define   SFH5721_R0A_MCONFB_SUB_DISABLED     0b00000000
#define   SFH5721_R0A_MCONFB_SUB_ENABLED      0b00010000
#define   SFH5721_R0A_MCONFB_WAIT(n)          (n)
#define SFH5721_R0B_MCONFC                   0x0B
#define   SFH5721_R0B_MCONFC_EN_DARK           0b00000001
#define   SFH5721_R0B_MCONFC_EN_IR             0b00000010
#define   SFH5721_R0B_MCONFC_EN_ALS            0b00000100
#define   SFH5721_R0B_MCONFC_EN_NONE           0b00000000
#define   SFH5721_R0B_MCONFC_LPFDEPTH_2        0b00000000
#define   SFH5721_R0B_MCONFC_LPFDEPTH_3        0b00010000
#define   SFH5721_R0B_MCONFC_LPFDEPTH_4        0b00100000
#define   SFH5721_R0B_MCONFC_LPFDEPTH_5        0b00110000
#define SFH5721_R0C_DATA1DARK                0x0C
#define SFH5721_R0E_DATA2IR                  0x0E
#define SFH5721_R10_DATA3ALS                 0x10
#define SFH5721_R14_DEVID                    0x14
```

The datasheet of the SFH 5721 prescribes a start-up sequence. That is implemented in the `init()` routine. Before the initialization starts, the device is reset but first pinged to make sure it is present on the bus. After the initialization finishes, we check if the device has the correct id. Problems are printed over serial.

```
// Pings to device to check I2C connection,
// then issues startup sequence from datasheet, and checks DEVID.
void sfh5721_init() {
  int result;
  uint8_t buf;

  // Probably the SFH5721 has had its 5ms to boot, but just to be sure
  delay(5); // Give time to read OTP

  // Try to ping (check SFH5721 is correctly mounted on PCB, with correct address)
  result = sfh5721_i2c_ping();
  if( result!=0 ) Serial.printf("5721 : ERROR init: ping (%d)\n", result);

  // If the host was reset, the SFH5721 might still be measuring
  // and then the startup sequence doesn't ACK. So reset SFH5721
  result = sfh5721_i2c_write(SFH5721_R00_OPSEL,SFH5721_R00_OPSEL_RESET);
  if( result!=0 ) Serial.printf("5721 : ERROR init: reset failed (%d)\n", result);

  // Datasheet has recommended startup sequence
  // step 1: power up or reset
  delay(5); // step 2: Give time to read OTP
```

```
  result=sfh5721_i2c_write(0x62,0x55); // step 3: single byte to register
  if( result!=0 ) Serial.printf("5721 : ERROR init: step 3 failed(%d)\n", result);
  result=sfh5721_i2c_write(0xEC,0,0); // step 4: buffer with size 0 to register
  if( result!=0 ) Serial.printf("5721 : ERROR init: step 4 failed (%d)\n",result);
  delay(5); // step 5

  // Read and check DEVID
  result=sfh5721_i2c_read(SFH5721_R14_DEVID,&buf,1);
  if( result!=0 ) {
    Serial.printf("5721 : ERROR init: reading DEVID failed (%d)\n", result);
  } else {
    if( buf!=0x01 )
      Serial.printf("5721 : ERROR init: wrong devid (0x%02x i.o. 0x01)\n",buf);
  }

  // Done
  Serial.printf("5721 : init\n");
}
```

The following function performs the sensor configuration – again a hardwired configuration for simplicity. It uses the shortest integration time (25 ms) that gives full 16-bit resolution. Assuming the ceiling is close and reflective, an analog gain of 4× brings the no-press baseline at half the range (a count of about 32k). It also configures a waiting time of 0 ms, which is only applicable for the operational mode “continuous” (as opposed to “single shot”). It does not engage the low-pass filter.

```
// Performs a hardwired configuration
// Note: ceiling is assumed to be white and close (~4mm) to sensor
// this ensures IR count is approximately 50% of 65535.
void sfh5721_conf() {
  int result;

  // IT at 25ms is shortest time with full 16 bit resolution,
  // AGAIN at 4x brings count roughly at halfway 65535
  result = sfh5721_i2c_write(SFH5721_R09_MCONFA, SFH5721_R09_MCONFA_IT_25M
         | SFH5721_R09_MCONFA_AGAIN_4 );
  if( result!=0 ) Serial.printf("5721 : ERROR conf: MCONFA failed (%d)\n",result);

  // No digital gain, no dark count subtraction, no wait (for omode continuous)
  result = sfh5721_i2c_write(SFH5721_R0A_MCONFB, SFH5721_R0A_MCONFB_DGAIN_1
         | SFH5721_R0A_MCONFB_SUB_DISABLED | SFH5721_R0A_MCONFB_WAIT(0) );
  if( result!=0 ) Serial.printf("5721 : ERROR conf: MCONFB failed (%d)\n",result);

  // Disable low pass filter
  result = sfh5721_i2c_write(SFH5721_R0B_MCONFC, SFH5721_R0B_MCONFC_EN_NONE );
  if( result!=0 ) Serial.printf("5721 : ERROR conf: MCONFC failed (%d)\n",result);

  Serial.printf("5721 : conf\n");
```

```
}
```

Once configured, there are two ways to measure: single shot and continuous. In our simple system, we ignore “synchronous” measurements where an external pulse on INT starts measurement(s). We are only interested in the infrared channel.

```
// Starts one single shot measurements, then gets and prints IR data
void sfh5721_single() {
  int result = sfh5721_i2c_write(SFH5721_R00_OPSEL, SFH5721_R00_OPSEL_SMODE_ASYNC
             | SFH5721_R00_OPSEL_OMODE_SINGLE | SFH5721_R00_OPSEL_EN_IR );
  if( result!=0 ) Serial.printf("5721 : ERROR single: OPSEL failed(%d)\n",result);
}

// Starts continuous measurements, a separate loop should get the IR data
void sfh5721_cont() {
  int result = sfh5721_i2c_write(SFH5721_R00_OPSEL, SFH5721_R00_OPSEL_SMODE_ASYNC
             | SFH5721_R00_OPSEL_OMODE_CONT | SFH5721_R00_OPSEL_EN_IR );
  if( result!=0 ) Serial.printf("5721 : ERROR cont: OPSEL failed (%d)\n",result);
}
```

Finally, we need a function to get the measurement data. Getting data is one, making sure the “data is ready” is non-trivial. The SFH 5721 has an INT pin, but no straightforward way to flag data ready. There are low and high thresholds (ILTL/ILTH and IHTL/IHTH), an interrupt enable register (IEN) and an interrupt configuration (ICONF) with a persistence (IPERS) bit-field. According to the datasheet “Interrupt persistence can be configured in a way that an interrupt is generated after every measurement.” – presumably setting a low threshold to 0, persistence to 0 and enabled. But this only works for continuous mode, not for single shot.

The function below works in all cases, it just polls the status register for data ready. It is suboptimal in the sense that I²C traffic while the sensor is measuring might somewhat disturb the measurements. As a development feature the function prints measurement time and has a time-out mechanism (hardwired to two seconds). For simplicity, read data is not returned but printed over serial.

```
// Polls STAT.DRY to see if measurement is available (with timeout),
// then reads and returns IR data.
// Instead of polling we could use a fixed wait (measurement time is known)
// or we could use the INT feature (but only in CONT omode)
// "Interrupt persistence can be configured in a way that an
// interrupt is generated after every measurement").
uint16_t sfh5721_get_ir() {
  int result;
  uint32_t t1, t0 = micros(); // micros() returns clock time in micro seconds
  uint8_t stat;
  do {
    result = sfh5721_i2c_read(SFH5721_R01_STAT,&stat,1);
    if( result!=0 ) Serial.printf("5721 : ERROR data: STAT failed (%d)\n",result);
    t1 = micros();
  } while( (stat & SFH5721_R01_STAT_DRY_IR)==0 && (t1-t0<2*1000*1000) );
```

```
  if( (stat & SFH5721_R01_STAT_DRY_IR)==0 ) Serial.printf("5721 : DRY timeout\n");

  uint8_t data2ir[2];
  result = sfh5721_i2c_read(SFH5721_R0E_DATA2IR,data2ir,2);
  if( result!=0 ) Serial.printf("5721 : ERROR data: DATA2IR failed(%d)\n",result);

  uint16_t ir= data2ir[0] + 256*data2ir[1];
  return ir;
}
```

This concludes the sensor driver. In the next section we use it in a simple application.

5.4 Application

The application initializes all drivers: I²C, IRED, SFH 5721, starting with the serial port. Next it configures the sensor driver and performs five measurements without and then five with the IRED on. Then it switches to continuous modes.

```
void setup() {
  Serial.begin(115200);
  delay(500);
  Serial.printf("\n\nWelcome to SFH5721 (and SFH4253) driver (ofstest)\n\n");

  i2c_init();
  ired_init();
  sfh5721_init();
  sfh5721_conf();

  Serial.printf("\nSINGLE\n");
  for( int i=0; i<10; i++ )
    if( i==5 ) ired_on();
    sfh5721_single();
    Serial.printf("ir:%d\n", sfh5721_get_ir() );
  }

  delay(10*1000);
  Serial.printf("\nCONT\n");
  sfh5721_cont();
}

void loop() {
  Serial.printf("ir:%d\n", sfh5721_get_ir() );
}
```

5.5 Output

The application generates the below output. The PCB is enclosed in a box therefore the first five measurements (with IRED off) have a count of 0, and once the IRED switches on the count goes to 26k.

Then the application switches to continuous measurements. With continuous measurements the counts start at 26k, but then we press the ceiling (count goes to 30k) and release (count back to 26k).

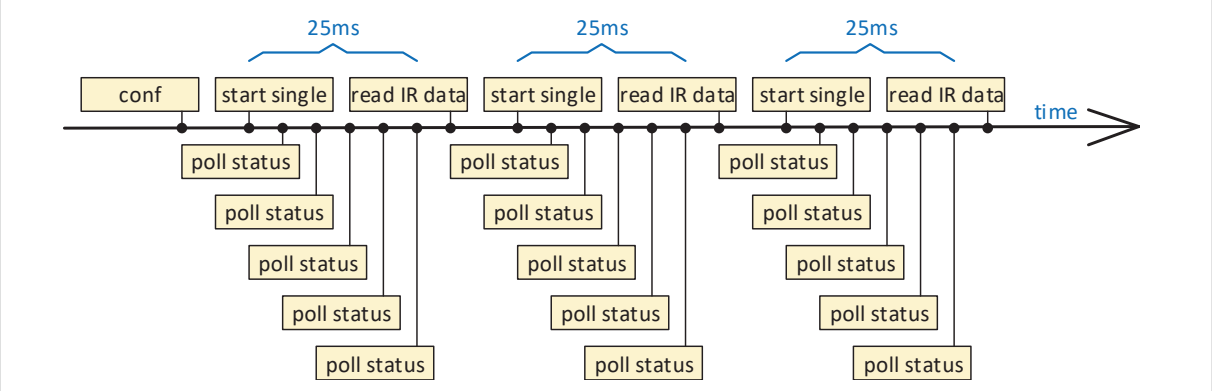
```
Welcome to SFH5721 (and SFH4253) driver (ofstest)

i2c  : init
ired : init
5721 : init
5721 : conf

SINGLE
ir:0
ir:0
ir:0
ir:0
ir:0
ir:25993
ir:25991
ir:25985
ir:25985
ir:25980

CONT
ir:25875
ir:25899
ir:25940
ir:26026
ir:26467
ir:27884
ir:28059
ir:28537
ir:29105
ir:29836
ir:29955
ir:29885
ir:29523
ir:28075
ir:27464
ir:26679
ir:26068
ir:25997
```

```
ir:25970
ir:25971
ir:25959
```

5.6 Timing

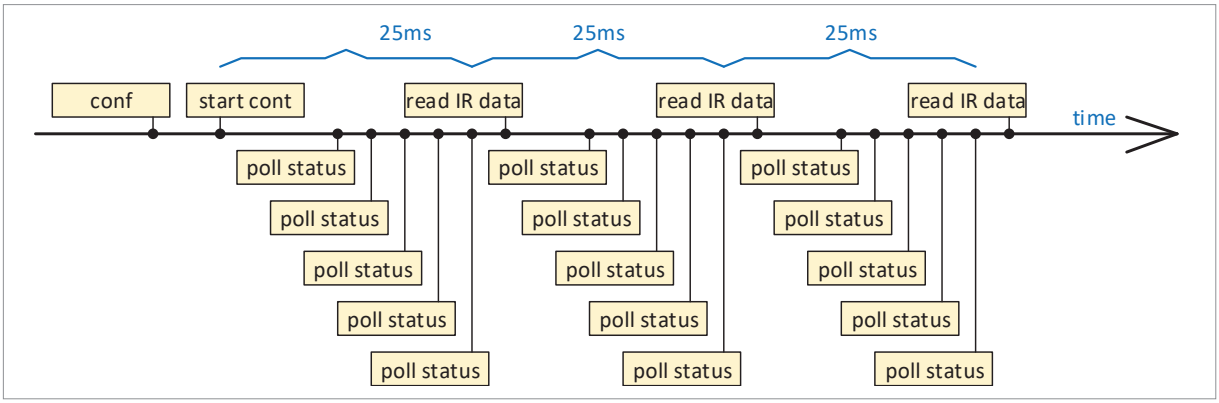
The executing architecture of the demo application is straightforward. In single shot mode, a configuration is followed by a series of measurements, each measurement consists of a single shot activation, polling the status to detect data ready and finally measuring the IR data, see Figure 37.

Figure 37: Series of single shot measurements, polling for data ready



The demo then switches to continuous mode. In continuous mode, configuration is followed by one continuous mode activation, followed by a series of measurements, each measurement consists of polling the status to detect data ready and finally measuring the IR data, see Figure 38.

Figure 38: Series of continuous measurements



5.7 Code size

Compiling the application (with the drivers) into a running binary gives the following memory summary from the compiler.

```
Sketch uses 258013 bytes (19%) of program storage space. Maximum is 1310720 bytes.
Global variables use 17200 bytes (5%) of dynamic memory, leaving 310480 bytes for
local variables. Maximum is 327680 bytes.
```

When compiling the (ESP32) platform, without our code, but with stub calls to all used API points – like `Serial.printf(0); Wire.begin(0,0,0UL);` results in the following summary.

```
Sketch uses 256021 bytes (19%) of program storage space. Maximum is 1310720 bytes.
Global variables use 17200 bytes (5%) of dynamic memory, leaving 310480 bytes for
local variables. Maximum is 327680 bytes.
```

Most of the code (256 kB) is for the (ESP32) platform itself; the demo application is only a fraction, namely 258013 - 256021 = 1992 bytes.

The code has several (development) `printf` strings; together they are 825 bytes, which means the actual code size approaches 1 kB. For a source file of about 250 lines that means 4 bytes per code line, which makes sense.

5.8 Signal processing

The software presented thus far is basically a driver with a minimalistic application: it just prints the measurement data. For optical force systems we are not so much interested in the raw measurement data, but more in the force *events* (the button presses).

There are several approaches to detect force events.

- **Fixed Threshold**
  The log of the test application showed a no-press baseline of about 26k and a press readout of 30k, a change of 4k. Figure 28 shows that a temperature increase of 40 degrees has a reduction of 1k in the sensor data. So, the counts will shift to a no-press of 25k and a press of 29k. This means that a threshold between 26k and 29k gives us plenty of margin. This would be the simplest algorithm to detect a press.

- **Shape Detector**
  One could write an algorithm that finds jumps in the signal, checks pre and post jump levels to determine if the press is forceful enough.

- **Baseline**
  One could write a low pass filter, where only really low frequencies pass, say a moving average window of 10 minutes. The output of this filter approximates the (no-press) baseline. Subtracting this baseline from the current sensor reading gives a relative reading, the event amplitude. If the event amplitude is high, this is a press.

  The advantages of this approach is that there is no signal delay and that a hard press following a soft press can be distinguished (e.g. making a scale where more and more weights are added). The biggest problem is that the baseline is a guesstimate; if the press takes a long

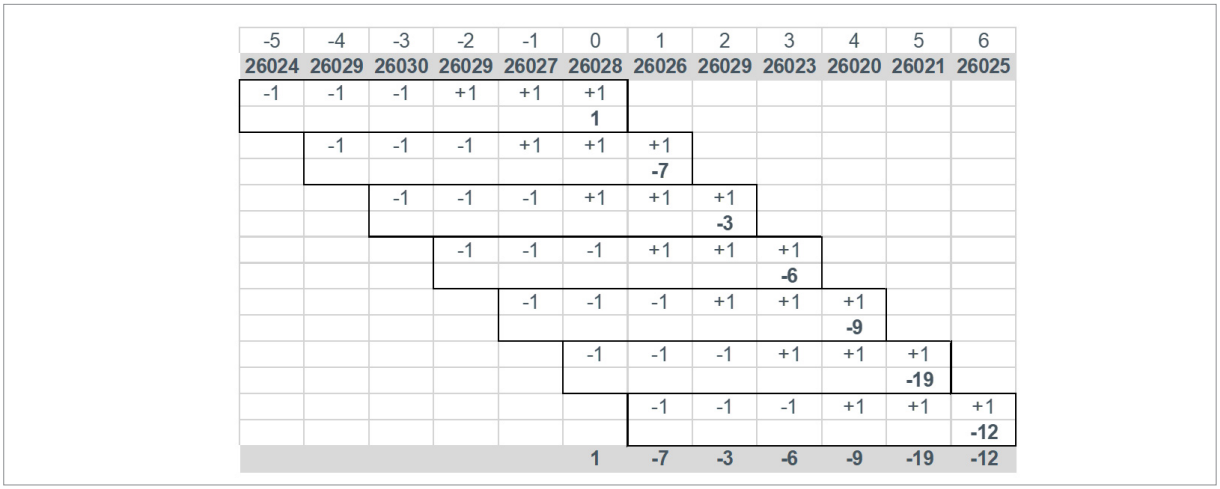
time, the baseline becomes less trustworthy.

- **Convolutional Filter**
  The rest of this section shows how a convolutional filter can detect edges. This is not dependent on a (assumed baseline), but it has a lag of a couple of measurements.

A convolutional filter slides a window with factors, referred to as the kernel, along the measurements, each time computing a dot product between the kernel and the measurements. We will use a window with length 6, and weights -1, -1, -1, +1, +1, +1. Such a kernel is known as an edge detector.

Figure 39 shows an example of convolution in action. The top row gives the time stamps. The second row (gray) the actual (ir) measurements at the time stamps as input to the convolutional filter. The next rows show the kernel slide, at each time slot computing the dot product of the weights with the input values. The last row repeats the computed dot products; it is the output of the convolutional filter (1, -7, -3, -6, -9, -19, -12).

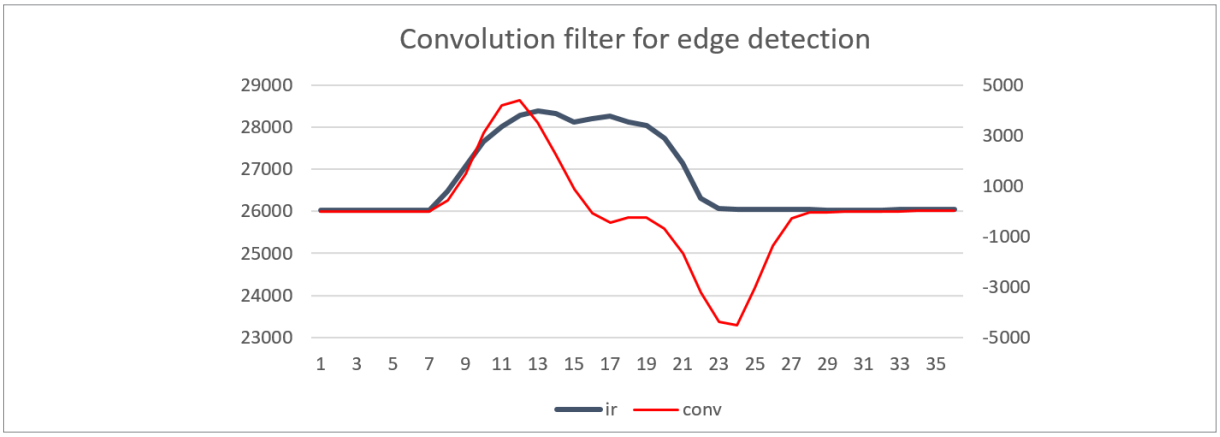
Figure 39: Convolution example



An important observation is that if the input sequence is flat (say with values v), the output is zero, because the weights in the kernel sum to 0 (-1×*v*-1×*v*-1×*v*+1×*v*+1×*v*+1×*v*=0). But as soon as there is a step in the signal (say from *v* to *v*+d), the output spikes at the moment it reaches this step: -1×*v*-1×*v*-1×*v*+1×(*v*+d)+1×(*v*+d)+1×(*v*+d) = 3×d. Note that if the step is down (negative *d*) the output is a negative spike.

Figure 40 shows actual measurements of an OFS system being pressed; this is the gray curve (with y-axis on the left). The red line is the output of the convolution filter with y-axis on the right. Notice that the red line is indeed 0, it spikes up at the beginning of the press event, it is again 0 during the press event, and finally spikes negative when the press stops. From then onwards, the filter output is 0 again.

Figure 40: Convolution filter for edge detection



The implementation is shown below. It needs a buffer to store the input values. We suggest using a circular buffer (sized a power of two to make wrap around efficient on CPUs with poor division). The elements are 16 bits, because that is what our sensor produces.

```
#define FILTER_BUFFERSIZE  8 // power of 2 gives efficient wrap
#define FILTER_KERNELSIZE  6 // the smaller, the lower the lag
#if FILTER_KERNELSIZE>FILTER_BUFFERSIZE
  #error Size mismatch
#endif

// buffer[] used from head-FILTER_KERNELSIZE (incl) up to head (excl) with wrap
uint16_t filter_buffer[FILTER_BUFFERSIZE];
int filter_head=-1; // -1 special: buffer is empty

// the kernel weights (for an edge detector; must sum to 0)
int filter_weights[FILTER_KERNELSIZE] = {-1,-1,-1,+1,+1,+1};
```

When a measurement becomes available, it is added to the buffer. The function has one special case: when the buffer is empty, it duplicates the measured value KERNELSIZE times, so that there is enough data to run a convolution. This means the filter immediately outputs without any initial delay.

```
void filter_add(uint16_t ir) {
  if( filter_head==-1 ) { // first measurement, fill buffer
    for( int i=0; i<FILTER_KERNELSIZE; i++ ) {
      filter_buffer[i] = ir;
    }
    filter_head = (0+FILTER_KERNELSIZE)%FILTER_BUFFERSIZE;
  } else { // append new measurement to buffer
    filter_buffer[filter_head] = ir;
    filter_head = (filter_head+1)%FILTER_BUFFERSIZE;
  }
}
```

To keep this code clean, we do not maintain partial dot products; the following function recomputes that every call. The variable p points to the tail of the input window, whereas i is the index in the kernel weights. One warning: below function assumes that int is 32 bits. This is not only necessary to ensure that acc cannot overflow, it also ensures that in the multiplication the unsigned buffer value is promoted to signed.

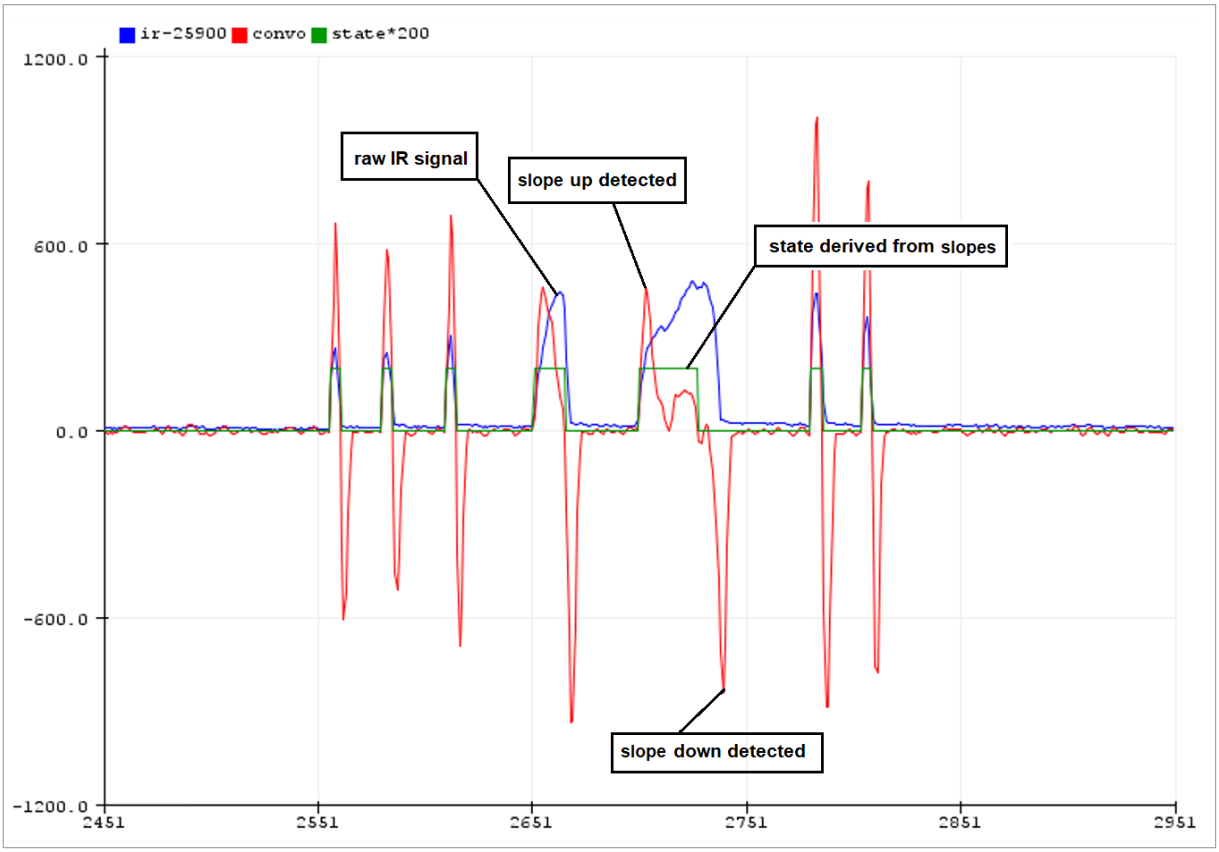
```
int filter_convo() {
  int acc = 0;
  int p = (filter_head+FILTER_BUFFERSIZE-FILTER_KERNELSIZE)%FILTER_BUFFERSIZE;
  for( int i=0; i<FILTER_KERNELSIZE; i++ ) {
    acc += filter_buffer[p]*filter_weights[i];
    p = (p+1)%FILTER_BUFFERSIZE;
  }
  return acc;
}
```

All we need now is to replace the loop function in the application to the following. The code maintains a state variable which indicates if the OFS is in pressed state (1) or released (0). The threshold for the state transitions (±25) depends on the kernel size and the increase in count when pressing.

```
int state=0;
void loop() {
  uint16_t ir = sfh5721_get_ir();
  filter_add( ir );
  int convo = filter_convo();
  if( convo>+25 ) state=1;
  if( convo<-25 ) state=0;
  Serial.printf("ir-25900:%d\tconvo:%d\tstate*200:%d\n",
    ir-25900, convo, state*200 );
}
```

The `printf` shifts the *ir* by 25900 and scales state by 200 just to have the Arduino make a nice graph (Tools > Serial Plotter) for our settings. Figure 41 shows the result. The blue line is the raw IR measurements, the red line the output of the convolutional filter detecting edges (slopes) in the IR signal, and the green line the button-press state.

Figure 41: Convolutional filter (red) maintaining press state (green curve) of raw IR signal (blue)



6 Outlook

This application note described an optical force sensing system centered around two discrete components, implementing a basic button. Here we have an outlook for other solutions.

To reduce cost or increase reliability, an optical force sensing system could be combined with a capacitive touch sensor from ams OSRAM (AS85xx family). The capacitive sensor cannot register force, but it supports multiple channels in one sensor. Combined, the capacitive sensor can detect where the press is – using a grid of capacitive plates – and the optical force sensor can detect the force of the press.

Another option is to combine an optical force sensor with a second optical force sensor. If they are placed at a distance (for example 10 cm apart) a central MCU receiving the raw data from both sensors can compute the location of a finger press between the two sensors. In other words, a (1D) slider can be made – refer to the optical force sensing slider application note of ams OSRAM for details. With a third optical force sensor a (2D) touch pad can be made.

Where a slider or touch pad is “continuous” both can also quantize their positions, forming discrete press positions, that is, forming a keypad.

If the force itself is a desired property of the optical force sensor system, it is suggested to pair it with a temperature sensor from ams OSRAM (AS62xx family).

Contact sales at ams-osram.com/support/ if support is needed on any of these products.

7 Glossary

- **API** – Application Programming Interface: A software interface, offering a service to other pieces of software, typically documented.
- **Arduino** – An open-source (hardware and) software company and user community that designs and manufactures microcontrollers together with a software framework.
- **Baseline** – (also known as no-press baseline) The value reported by the sensor if it is (or were not) pressed. Typically, this value drifts in time due to environmental conditions (temperature).
- **Count** – (in the context of this document) The unitless measurement result of a light sensor ("number of photons hitting the sensor during the integration time").
- **Convolution** – A mathematical operation on a signal (series of numbers) producing a "filtered" output signal (also a series of numbers); the dot product of a fixed size vector (the kernel) moving as a sliding window over the values of the input signal.
- **Crosstalk** – The unwanted coupling between signals; in this document the coupling between the light from a local LED or the emitter and sensor signal.
- **ESP32** – Low-cost high-performance microcontroller supported by Arduino.
- **HMI** – Human-machine interface; a user interface that connects a human to a machine (control and feedback).
- **I²C** – Inter Integrated Circuit: Digital communication protocol between ICs.
- **IR** – Infrared: A region of the electromagnetic radiation spectrum with wavelengths from 700 nm to 1000 nm (invisible to the human eye).
- **IRED** – IR LED: A semiconductor diode that emits infrared light.
- **Lambertian emission** – The typical radiation profile of a LED: proportional to the cosine of the angle φ between the line of sight and the surface's normal vector.
- **LED** – Light Emitting Diode: A semiconductor diode that emits light.
- **MCU** – Micro Controller Unit: An IC that contains an embedded processor, memories, and some peripherals like I/O ports and/or I²C.
- **Moat** – (from the broad ditch dug around a castle) Deliberate weakening of the ceiling of an

optical force sensor to make it more sensitive to light pressures.

- **OFS** – Optical Force Sensing: a technology that uses a light emitter and a light detector located side-by-side below a surface, measuring surface displacement from the changes in reflected light.
- **PCB** – Printed Circuit Board: A board connecting electronic components.
- **PWM** – Pulse Width Modulation: A modulation technique that generates square pulses with fixed frequency and amplitude; the pulse-width encodes the amplitude.
- **Reflectivity** – A measure of the ability of a surface to reflect light.
- **Scattering** – A change in the direction of motion of a particle because of a collision with another particle; here light being diffused when it hits the ceiling.
- **Transmissivity** – The degree to which a medium (here the ceiling) allows something (here light), to pass through it.

8 Revision information

Changes from previous version to current revision v2-00	Page
Added Abstract to the cover page	

- Page and figure numbers for the previous version may differ from page and figure numbers in the current revision.
- Correction of typographical errors is not explicitly mentioned.

ABOUT ams OSRAM Group (SIX: AMS)

The ams OSRAM Group (SIX: AMS) is a global leader in optical solutions. By adding intelligence to light and passion to innovation, we enrich people's lives. This is what we mean by Sensing is Life. With over 110 years of combined history, our core is defined by imagination, deep engineering expertise and the ability to provide global industrial capacity in sensor and light technologies. Our around 24,000 employees worldwide focus on innovation across sensing, illumination and visualization to make journeys safer, medical diagnosis more accurate and daily moments in communication a richer experience. Headquartered in Premstaetten/Graz (Austria) with a co-headquarters in Munich (Germany), the group achieved over EUR 5 billion revenues in 2021. Find out more about us on https://ams-osram.com

For further information on our products please see the Product Selector and scan this QR Code.

Published by ams-OSRAM AG
Tobelbader Strasse 30, 8141 Premstaetten, Austria

Published by ams-OSRAM AG
Tobelbader Strasse 30,
8141 Premstaetten Austria
Phone +43 3136 500-0
ams-osram.com