UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Design Engineering

Development of a simple, low-cost, touchless, short-distance gesture recognition system

End of Degree Project

Bachelor's Degree in Industrial Electronics and Automation Engineering

AUTHOR: Pinke , Angéla

Tutor: Rodríguez Ballester, Francisco

External cotutor: PASZTOR, DANIEL

ACADEMIC YEAR: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Angéla Pinke

SIMPLE, LOW-COST, TOUCHLESS, SHORT-DISTANCE GESTURE RECOGNITION SYSTEM

Bachelor's thesis

Degree

AUTHOR: ANGÉLA PINKE

Tutor: Francisco Rodríguez Ballester

External cotutor: Dániel Pásztor

ACADEMIC YEAR: 2022

Table of contents

HALLGATÓI NYILATKOZAT

Alulírott **Pinke Angéla,** szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 05. 20

...…….

Pinke Angéla

Összefoglaló

Az okos otthonok ígéretes jövő előtt állnak, és egy teljesen új területet nyitott a mérnökök számára. Az eszközök közötti kapcsolat érdekében a folyamatos aktív internetkapcsolat fenntartása nagyobb költségekkel járhat, ezért a hang- vagy gesztusvezérléssel működő otthoni automatizálási rendszer egy olyan megoldás, amellyel több embert lehet elérni. Míg a hangvezérlésű okosotthon termékek már népszerűnek mondhatóak a fogyasztók körében, addig a gesztusvezérlésű rendszerek csak nem olyan rég kezdtek el megjelenni az automatizálási iparágban, de az érintés nélküli koncepció máris terjedőben van, amit a Covid-19 járvány különösen felgyorsított az elmúlt néhány évben.

Egy olyan mikrokontroller-alapú beágyazott rendszert fejlesztettem, amely gesztusokat ismer fel, pontosabban kézmozdulatokat, például a kéz balról jobbra, jobbról balra vagy fel-le mozgását. A rendszer több infravörös fénykibocsátót és -vevőt használ. Amikor egy tárgy (vagy a felhasználó keze) megszakítja a sugárzott infravörös fényt,a vevőkhöz visszaverődő fény jelzi, hogy a tárgy az eszköz előtt van. A kéz megjelenésének és eltűnésének idejét a több infravörös vevővel kombinálva egy algoritmus képes felismerni néhány egyszerű, előre meghatározott gesztust.

A rendszernek elegendőnek kell lennie egy egyszerű intelligens háztartási készülék vezérléséhez, mint például ajtó vagy ablak nyitása/zárása, redőny fel- vagy lehúzása stb. A használat csak a felhasználó preferenciáitól függ.

Abstract

Smart-homes have a really promising future and it has opened a whole new field for engineers. For connectivity amongst devices, keeping an all-time active internet connection might not be budget friendly for all, so making a home automation system with voice or gesture control is a way to reach more people. While voice-activated smart home products have gained popularity among consumers already, gesture-controlled systems just started to permeate the industry of automation, but the touchless concept is already expanding especially that the Covid-19 pandemic accelerated it in the past few years.

The aim of the project is to develop a microcontroller-based embedded system to recognize gestures, more specifically hand movements as left-to-right, right-to-left or up-to-down. The system is using several infrared light emitters and receivers. The light reflected to the receivers when an object (or the hand of the user) interrupts the transmitted infrared light, which is used as an indication that the object is in front of the device. Combining the time of appearance and disappearance of the hand at the several infrared receivers, an algorithm can recognize some simple, pre-established gestures.

The system is enough to control a simple smart home appliance: opening/closing a door or window, moving up or down a blind etc. The usage only depends on the user preference.

Resumen

Los hogares inteligentes tienen un futuro realmente prometedor y han abierto un campo completamente nuevo para los ingenieros. Para la conectividad entre dispositivos, mantener una conexión a Internet activa en todo momento puede no ser económico para todos, por lo que hacer un sistema de automatización del hogar con control de voz o gestos es una forma de llegar a más personas. Mientras que los productos para el hogar inteligente activados por voz ya han ganado popularidad entre los consumidores, los sistemas controlados por gestos apenas comenzaron a penetrar en la industria de la automatización, pero el concepto "sin contacto" ya se está expandiendo, especialmente porque la pandemia de Covid-19 lo aceleró en los últimos años.

El objetivo del proyecto es desarrollar un sistema embedido basado en un microcontrolador para reconocer gestos, más específicamente movimientos como mover la mano de izquierda a derecha, de derecha a izquierda o de arriba a abajo. El sistema utiliza varios emisores y receptores de luz infrarroja. La luz reflejada a los receptores cuando un objeto (o la mano del usuario) interrumpe la luz infrarroja transmitida, que se utiliza como indicación de que el objeto está frente al dispositivo. Combinando el tiempo de aparición y desaparición de la mano en los distintos receptores de infrarrojos, un algoritmo puede reconocer algunos gestos sencillos y preestablecidos.

El sistema es suficiente para controlar un electrodoméstico inteligente simple: abrir/cerrar una puerta o ventana, subir o bajar una persiana, etc. El uso solo depende de la preferencia del usuario.

Resum

Les llars intelligents tenen un futur realment prometedor i han obert un camp completament nou per als enginyers. Per a la connectivitat entre dispositius, mantindre una connexió a Internet activa en tot moment pot no ser econòmic per a tots, per la qual cosa fer un sistema d'automatització de la llar amb control de veu o gestos és una manera d'arribar a més persones. Mentre que els productes per a la llar intel·ligent activats per veu ja han guanyat popularitat entre els consumidors, els sistemes controlats per gestos a penes van començar a penetrar en la indústria de l'automatització, però el concepte “sense contacte” ja s'està expandint, especialment perquè la pandèmia de Covid-19 ho va acelerar en els últims anys.

L'objectiu del projecte és desenvolupar un sistema embedido basat en un microcontrolador per a reconéixer gestos, més específicament moviments com moure la mà d'esquerra a dreta, de dreta a esquerra o de dalt a baix. El sistema utilitza diversos emissors i receptors de llum infraroja. La llum reflectida als receptors quan un objecte (o la mà de l'usuari) interromp la llum infraroja transmesa, que s'utilitza com a indicació que l'objecte està enfront del dispositiu. Combinant el temps d'aparició i desaparició de la mà en els diferents receptors d'infrarojos, un algorisme pot reconéixer alguns gestos senzills i preestablits.

El sistema és suficient per a controlar un electrodomèstic intel·ligent simple: obrir/tancar una porta o finestra, pujar o baixar una persiana, etc. L'ús només depén de la preferència de l'usuari.

1 Introduction

1.1 Smart homes today

A smart home can be termed as an automated system, which enables the user to have access over all its devices. We can think about anything which would make our life more comfortable. Arriving home and the bath is ready, it is not needed to push any buttons, just wave or talk and the TV, light or music is on and so on.

As I was saying the connectivity amongst devices can make this less relevant because of the fact that it can be not so affordable easily, that is why standalone devices are worth the effort.

It is already a wide palette of voice-activated smart home products in the market. A good example is Amazon's Alexa, which is the leader among the products. This is one of the most widely used and accepted home automation products that can work as a standalone device connecting to a cloud-based service to turn on music, set up alerts, make calls, and ask questions about the traffic and weather. [1]

Automated touchless systems had just started to enter the industry years ago and because of Covid, now we can see touchless hand sanitizer dispensers everywhere, and while they're likely not smart-enabled, the potential is there—and the touchless concept is expanding faster. For example, touchless doorbells are a new innovation for guests to announce their arrival without touching a shared surface that could potentially spread germs. [2]

Additionally as they also can be standalone devices, they have the same possibilities as smart speakers.

Figure 1.1: "Bixi" is the first portable hand-gesture recognition device [3]

The statistics about smart homes are also really promising:

- The worldwide connected home market is projected to grow 25% from 2020 to 2025.

  Analysts and smart home facts note that the growing emphasis on security, as well as wireless innovation and advancement in the internet of things (IoT), will drive the market forward at an astonishing growth rate.

- 86% of millennials would pay more for a connected home.

  Millennials—the generation that grew up with technology—are unsurprisingly the most enthusiastic about smart house technology. In fact, most of them are willing to pay 20% more for a home with smart devices, smart home statistics state. However, as the younger generation comes of age and start buying homes, smart home adoption will likely grow at an even faster rate.

- Smart security systems cut down home insurance rates.
- Smart heating and cooling systems can save smart home users 50% in energy use.

  It's especially because of the smart thermostats. Smart home energy usage statistics indicate that they are the most energy-efficient. In fact, some smart climate systems pay for themselves in less than two years.

  54% of homeowners believe that installing smart home systems will make the house sell faster.

- The global smart kitchen market is expanding fast

  Valued at approximately $14.5 billion in 2019, the smart kitchen market is forecast to reach an incredible $43 billion by 2027. Not just bigger in market size, smart kitchens will become more intelligent, too. From smart ovens and smart fridges that can scan food and even make orders for themselves, there are endless possibilities of what smart home appliances in the kitchen can do. [4]

As we already have all the capabilities to build these systems, it is really motivating to study or work in this topic.

1.2 Embedded systems

If we take a closer look into the technology, we can see that micro-computers make these to function. The word “computer” usually reminds us of PCs or laptops, but in reality computers are all around us. From the smart-homes to the ATMs we use regularly to access cash. These are called embedded systems and can be found everywhere in our society. In fact, a full 98% of microprocessors manufactured today will find their use in embedded systems. That leaves just 2% for use in computers! [5]

An embedded system is defined as a combination of computer hardware and software designed for a particular function. It is “embedded” because it may also function within a larger system, as within a car or a mobile phone for example. [6]

The world of embedded systems is a constantly developing industry, which always needs more and more workers.

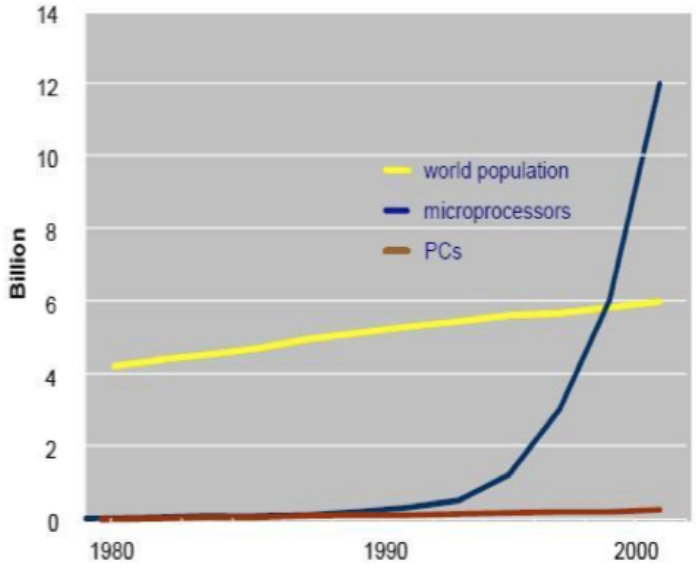


Figure 1.2: diagram of the growing number of microprocessors

- In 2000 it was around 10 billion microprocessors and in 2004 it was 20 billion, while the population of Earth was 7 billion.
- The growth rate is 14% in every year, while the market of PCs is 8%
- And the number of embedded software is growing 16% every year [7]

The main components of an embedded system:

- Connection to the physical environment (input) which can be buttons or sensors for example.

  Sensors are converting physical quantities into electrical signals. These can be either passive or active.

  Active requires an external source of power to operate, while the passive itself generates electrical output which means that it simply detects and responds to some type of input from the physical environment. [8]

  In my application I have one transmitter infrared sensor and three receivers. The receivers are passive, they are getting the reflected light from the transmitter.

  On the other hand the transmitter is active so it is needed to generate a digital signal in order to make it work.

- Communication unit

  It can be completely integrated, for example the UART, which I am also using (more on this later) or a completely external peripheral such as WiFi or Bluetooth.

  In between is when the protocol logic is integrated and an external signal level matching is needed for example to match the voltage levels.

  Software implementation is also possible, which means that we change the pins of the GPIOs in the software, but it is needed to be careful with the timings. [9]

- The processing unit

1.3 The microcontroller

One of the choices for computer hardware as a processing unit in an embedded system is the microcontroller. The microcontroller is a microprocessor with integrated peripherals: memory, timers, communication units, interface for debugging, GPIOs, LEDs, buttons. I will explain more about these in the next section.

Microcontrollers are widely used because of the lot of advantages it has.

- Programmable
- Low time required for performing operation.
- It is easy to use, troubleshooting and system maintenance is straightforward.
- Cheap (not only the chip, but the development environment and the debugger also)
- Easily accessible

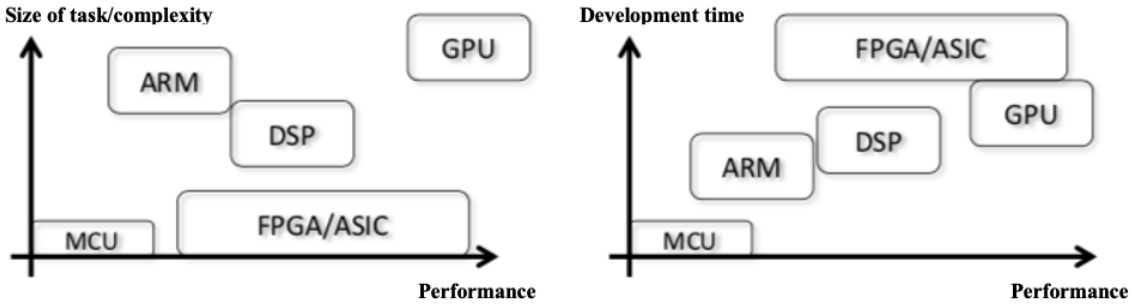


Figure 1.3: diagrams of processing units

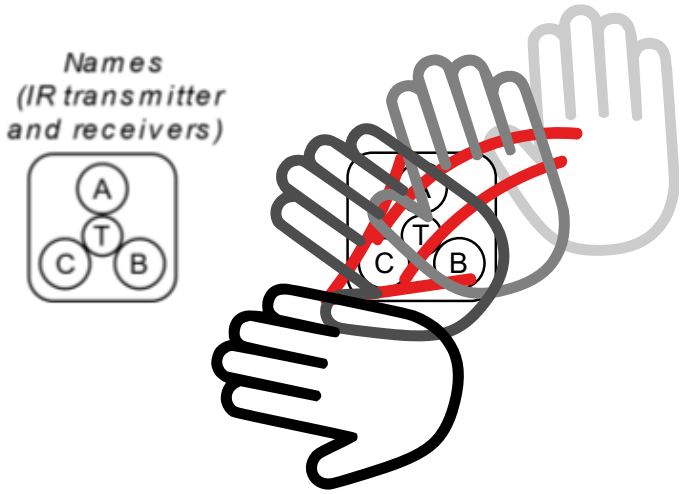
If we take a look at the picture above, we can see that microcontrollers (MCUs) need the least time to develop an application among the different processing units, but have the lowest performance capabilities and can only bear smaller tasks with less complexity.

At first glance it could be seen as not the best option, but the time is a really important factor. In fact, microcontrollers are the most used among them, because it is a good solution to put together a complex system with a lot of microcontrollers handling small tasks each or in a lot of cases we don't even need our application to be really complex. An automated coffee machine or the gesture detection, which I will introduce are good examples for that.

1.4 The goal of the gesture detection system

I was developing the gesture detection application for the microcontroller and testing the functionalities with the help of the microcontroller's LEDs, an oscilloscope and a console window through my computer. After that, the sensors have to be attached to the microcontroller and the whole system has to be tested altogether.

The internal representation of the attached sensors and a hand crossing in front of them can be seen below, if I am using 3 receivers in triangle shape.



The project can be separated by taking care of the transmitter and the receivers differently.

Firstly, the transmitter will need to have a digital signal, which will ride it and when the transmitter is working and generates a signal, the receivers can catch it if something is disturbing the infrared light (the hand of the user). The receivers will catch their own different signals and by the help of them, it is possible to calculate the different gestures.

In the next section I will introduce more deeply the most important technologies I was using and after that I will go through the steps that I was taking while I was developing the project up until the final result.

2 Technology

2.1 Inside the microcontroller

To understand microcontrollers better I will continue with the explanation of how they are built. Microcontrollers have every kind of peripheral, the same as a personal computer has a keyboard, which is also called peripheral. As many other microcontrollers, it also has timers, LEDs, I/O ports and UART communication ports.

I will explain these in this chapter, since I use them in my application.

2.1.1 GPIOs

The microcontroller has I/O (Input/Output) ports, and through these it's connected to the "real world". Inputs receive changes in the real world, from temperature sensing, to motion sensing, to push buttons, and much more. For the output ports we send a signal which can depend on our inputs or anything we want. The effect of our signal on the output can range from a simple LED light going on, to running a motor for a certain part, to many more. [11]

Each port has their own pins, so for example in my microcontroller the GPIOD13 stands for the pin 13 of the D port and it's also the orange LED light, so it can be a good choice to use as an output pin since it is easy to see the results.

As I am using 4 motion sensors, these are attached to the I/O ports of the microcontroller. One (the transmitter) as an output, and 3 (the receivers) as inputs.

2.1.2 Registers

One part of the memory of a microcontroller is organized into several "registers", each with its own address, so we can reach them. A register is just a location in memory that you can write data to or read data from and certain registers are reserved for setting the ports' values. [12]

Overall, if we want to turn a LED on, in our microcontroller, we would have to check at which port and pin it is and also where to set its value in the memory, but with the HAL functions we don't have to worry about memory locations and it's also making it sure that nothing gets messed up inside the microcontroller.

2.1.3 Timers

Timers allow us to measure time by counting. They can be 8,16 or 32 bit timers. For example, an 8-bit timer will count from 0 to 255. Then they “roll over” once they reach their max value. So, an 8-bit timer would start over again from 0 once it reaches 255.

A variety of settings can be applied to most timers to change the way they function. These settings are usually applied via other special function registers inside the microcontroller. For example, instead of counting to a maximum of 255, it can be told to the timer to roll over at 100 instead.

All timers require a clock of some sort, which will define how fast the timer counts. Most will be connected to the microcontroller’s main CPU clock. A timer will tick (increment by one) each time it receives a clock pulse. If the clock is 80 MHz then that might be too fast for many of our applications. A 16-bit timer can count to 65,535 before rolling over, which means that events can be measured no longer than about 819 microseconds.

To measure longer events, it’s needed to set a prescaler, which is a piece of hardware that divides the clock source. For example, a prescaler set to 80 would turn an 80 MHz clock into a 1 MHz clock. Now, the timer would tick once every 1 microsecond and, assuming a 16-bit timer, be able to time events up to a maximum of about 65.5 milliseconds. More on the calculations later. [13]

2.1.4 Communication Unit

A serial data connection is established to connect different functional units (in my case a PC and the microcontroller). Communication is mostly done with synchronous data transmission (the timer clock is also transmitted), which, although it involves an extra line, does not require all units to have a basic timing, match exactly and use synchronization methods. However, longer-distance communication between units is mostly asynchronous, as the clock providing the exact timing itself would be distorted during transmission (time delay, edge distortion). [15]

UART is a hardware module in microcontrollers for serial communication. It can be used to send data to a computer. The data format and transmission speeds are configurable, so when we want to read the data in our computer, we only have to pay

attention to set the same format and speed, as we set in the microcontroller configuration, so that is how we can read the same data, as we sent. This is a usual way to test microcontroller applications.

2.2 Software tools

The software I used is the STM32 CubeIDE. It is the original initiative to ease and accelerate the development of embedded systems. It provides the developer with all the low-level drivers, so it enables to dedicate most of the effort to develop the application layer and any required middleware (any other additional software which could be needed for the system).

The drivers:

- The STM32CubeMX
- The STM32Cube Hardware Abstraction Layer (HAL)

2.2.1 STM32Cube HAL

It is an STM32 abstraction layer embedded software ensuring maximized portability across the STM32 microcontroller. [10]

Basically this means, when we create a project, CubeIDE generates a lot of .h and .c files full of functions that we can use, so it makes our life a lot easier.

As an example:

The function below is inside the generated “stm32f4xx_hal_gpio.c” file.

```
/**
  * @brief  Sets or clears the selected data port bit.
  *
  * @note   This function uses GPIOx_BSRR register to allow atomic read/modify
  *         accesses. In this way, there is no risk of an IRQ occurring between
  *         the read and the modify access.
  *
  * @param  GPIOx where x can be (A..K) to select the GPIO peripheral for STM32F429X device or
  *                      x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
  * @param  GPIO_Pin specifies the port bit to be written.
  *          This parameter can be one of GPIO_PIN_x where x can be (0..15).
  * @param  PinState specifies the value to be written to the selected bit.
  *          This parameter can be one of the GPIO_PinState enum values:
  *            @arg GPIO_PIN_RESET: to clear the port pin
  *            @arg GPIO_PIN_SET: to set the port pin
  * @retval None
  */
```

Figure 2.1: comment above the HAL_GPIO_WritePin function declaration

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState
PinState)
  {
    /* Check the parameters */
    assert_param(IS_GPIO_PIN(GPIO_Pin));
    assert_param(IS_GPIO_PIN_ACTION(PinState));

    if(PinState != GPIO_PIN_RESET)
    {
      GPIOx->BSRR = GPIO_Pin;
    }
    else
    {
      GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;
    }
  }
```

As we can see there is a detailed comment above the function declaration, so we can know exactly what the function does and how we have to use it. For the usage of this function we only have to check which pin, and of which gpio we want to set and we don't have to worry about checking the GPIO's registers, because the function already contains the proper way to set the register.

2.2.2 STM32CubeMX

It is a graphical software configuration tool that allows generating C initialization code using graphical wizards.

As an example in CubeMX you can choose a timer and most importantly set its prescaler and period to adjust the time period you want to achieve. As it can be seen below in the picture, a lot of other other functions and settings a timer has.

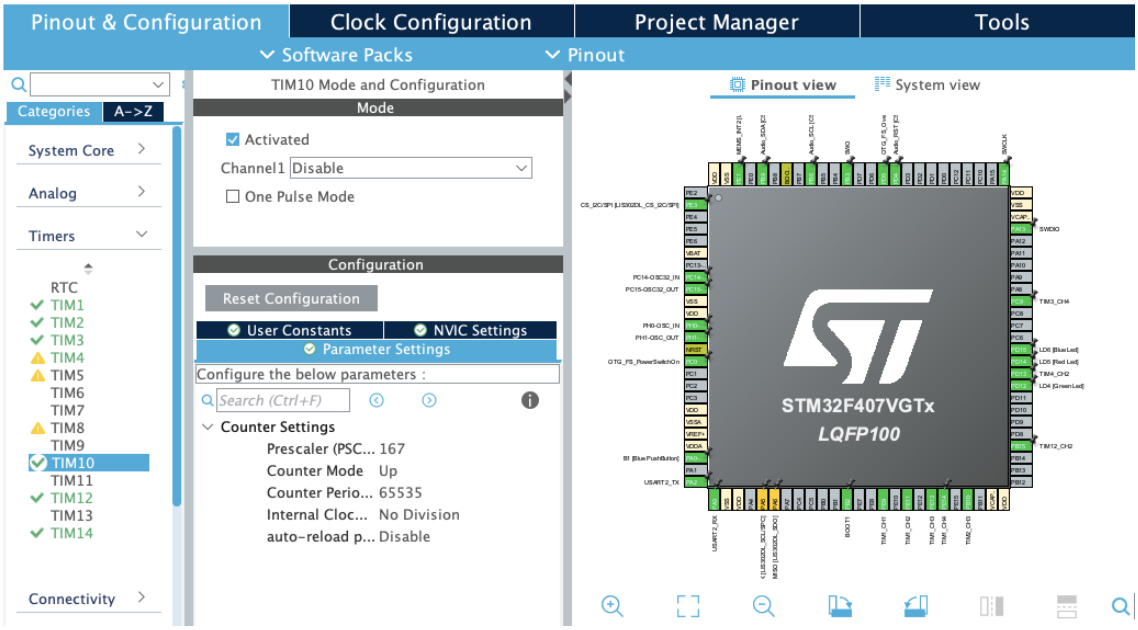


Figure 2.2: CubeMX window

As it can be seen in the picture in CubeMX we can also see the Pinout view and set the pins to be inputs, outputs or to make a connection with the timer channels.

At the connectivity tab the USART and the other integrated connection units can be set and at Clock Configuration we can change the clock frequencies and see how the whole clock configuration system is built.

After making changes, the CubeMX generates the initialization of the timer into the code, so in the code we only have to start the timer.

We can also see the initialization function in our main.c.

```
static void MX_TIM10_Init(void)
  {
    /* USER CODE BEGIN TIM10_Init 0 */
    /* USER CODE END TIM10_Init 0 */
    /* USER CODE BEGIN TIM10_Init 1 */
    /* USER CODE END TIM10_Init 1 */
    htim10.Instance = TIM10;
    htim10.Init.Prescaler = 167;
    htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim10.Init.Period = 65535;
    htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim10.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
    if (HAL_TIM_Base_Init(&htim10) != HAL_OK)
    {
      Error_Handler();
    }
    /* USER CODE BEGIN TIM10_Init 2 */
```

```
    /* USER CODE END TIM10_Init 2 */
  }
```

If we would want to overwrite these settings inside the code, the proper way is to create another function, which runs after the original initialization, because the modification of our .ioc file would always overwrite the initialization function, which the CubeMX created (the code above).

To start the timers the HAL library has more options. Some of these:

```
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
HAL_StatusTypeDef   HAL_TIM_PWM_Start(TIM_HandleTypeDef   *htim,   uint32_t
Channel)
HAL_StatusTypeDef  HAL_TIM_IC_Start_IT(TIM_HandleTypeDef  *htim,  uint32_t
Channel)
```

If it is needed to use the timer for more tasks then I have to call every function for every task. HAL_TIM_Base_Start can be used if I simply want my timer to start counting and I don't need any other functions. The HAL_TIM_Base_Start_IT enables interruptions, which means, all the time when the timer reaches its max counting value and rolls over, it calls a callback function.

So if we want our program to do something at every rollover, then we can write into the function below.

```
  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
```

HAL_TIM_IC_Start_IT is for input capture mode, which I will write about later. It also calls a Callback function, every time when it captures something in the corresponding input. The definition of the callback function is that we cannot call it directly, we call a function from the software library, which calls it. The picture above defines it and the timer's interruption is a good example.

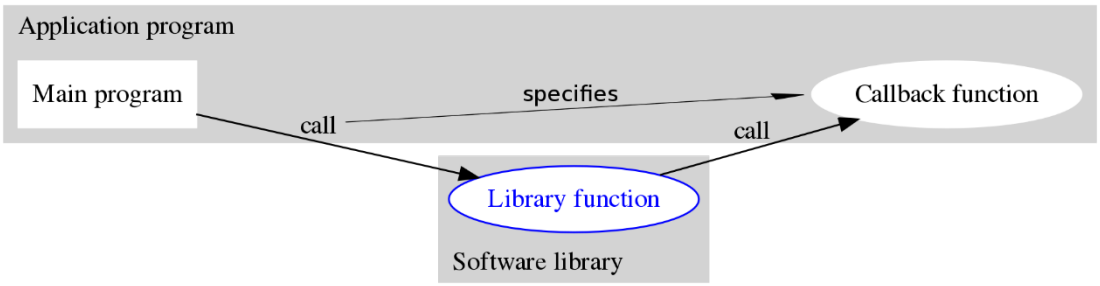


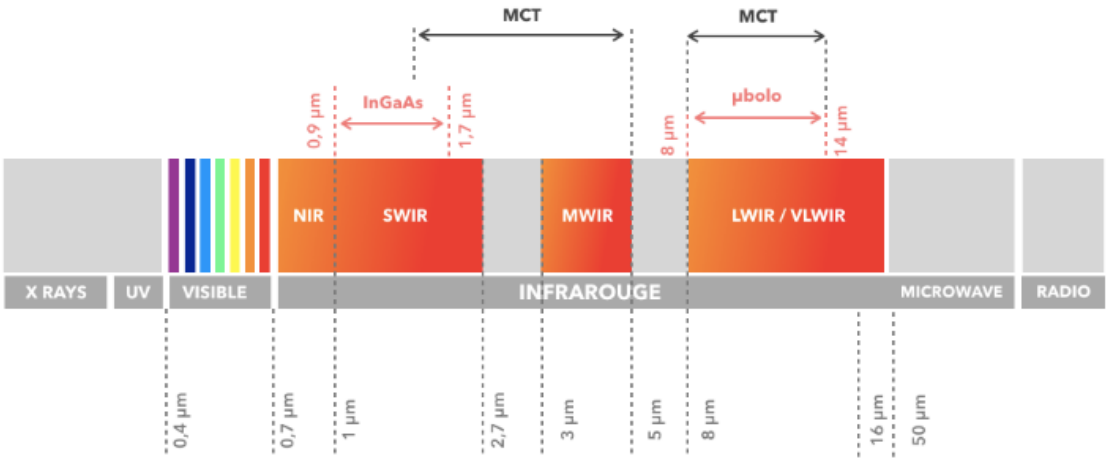
Figure 2.3: callback function [14]

2.3 Infrared sensors

The technology was first discovered in the early 19th century. However, it took quite some time to be able to actually use it and integrate it into marketable products. Today's powerful infrared technology is being used in a variety of novel ways, adding value to advanced systems for autonomous vehicles and smart buildings, for example.

Infrared can be integrated into existing systems to add new technical capabilities. And, as production volumes increase, costs will continue to come down, making the technology even more accessible for an even wider range of uses.

Radiation is characterized by its frequency and wavelength. And not all radiation is visible to the human eye.



There are several wavelengths in the electromagnetic spectrum, and each one has unique characteristics. The four types of infrared radiations: NIR (near infrared), SWIR (short wave infrared), MWIR (medium wave infrared), LWIR (long wave infrared). The difference is that thermal imaging begins from MWIR, where temperature gradients present in the scene being observed start to form, so it requires cryogenically-cooled technologies. Cooled detectors are bulkier and more expensive than uncooled detectors and since I am not using the sensors for image detection, it would be unnecessary to work in these spectrums.

The sensors I am using are working in the NIR spectrum. The underlying principle of NIR spectroscopy, for example, is molecular vibration caused by the excitation of molecules by the infrared source. The molecules absorb infrared waves, changing the degree of vibration of the electrons. This creates a measurable signal. [16]

3 Digital signal for the infrared transmitter

3.1 Representation of the infrared sensors

As an example I have a single IR transmitter and three IR receivers in a triangle shape as depicted below and I use different representations of them, so I can access them easier and the code is also clearer.

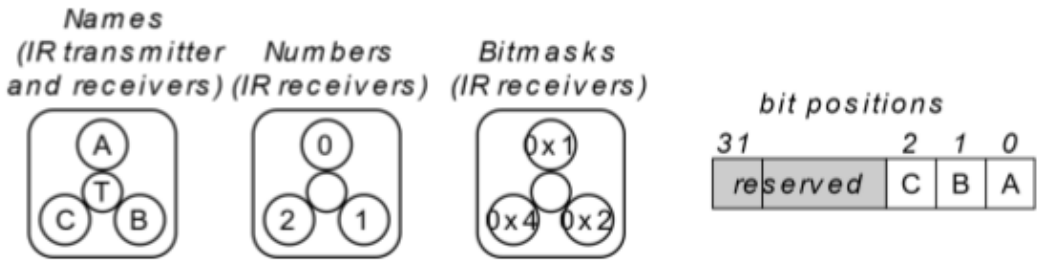


Figure 3.1: sensors and bit positions [16]

IR transmitter is represented by the letter T, IR receivers are represented by the letters A, B, and C. In the program, the transmitter will have no designator, and each IR receiver will be represented by a number (starting from 0) and a bitmask (1 << number). Thus, IR receivers A, B, and C will be represented by the numbers 0, 1, and 2, respectively, and by the bitmasks 0x01, 0x02, and 0x04, respectively.

The numbers will be used when accessing information stored into an array (for example the arrival time in one array element per IR receiver), the bitmasks will be used in the algorithm that determines the gesture; more on this later.

Different C structures are used to group all this information, as listed below.

An IR receiver is represented

- By a letter (A, B, C) in diagrams,
- As an unsigned number (0, 1, 2) when indexing arrays that contain one value per IR receiver, and
- As a bitmask (1 << number = 0x01, 0x02, 0x04) when integrating the obstacle detection of the IR receiver in a bitmask.

In a C structure:

```
typedef struct {
     uint32_t id;
     char *name;
     uint32_t bitmask;
} receiver_t;
```

And the three receivers in a triangle-shape device would be

```
static const receiver_t IRa = { .id = 0, .name = “A”, .bitmask = 0x01 };
static const receiver_t IRb = { .id = 1, .name = “B”, .bitmask = 0x02 };
static const receiver_t IRc = { .id = 2, .name = “C”, .bitmask = 0x04 };
```

A bitmask is

- Calculated as the bitwise-OR of the bitmasks of those IR receivers that detect an obstacle in front of it, and
- Internally represented by an uint32_t value. [16]

3.2 Generating the signal

For the IR transmitter I needed to generate a digital signal that contains trains of pulses as depicted below. The number and duty cycle of these pulses, as well as the duration of the inter-trains silence, had to be easily changed/customized in the programming to adjust these values during the experiments. I used #define’s macros for this purpose.

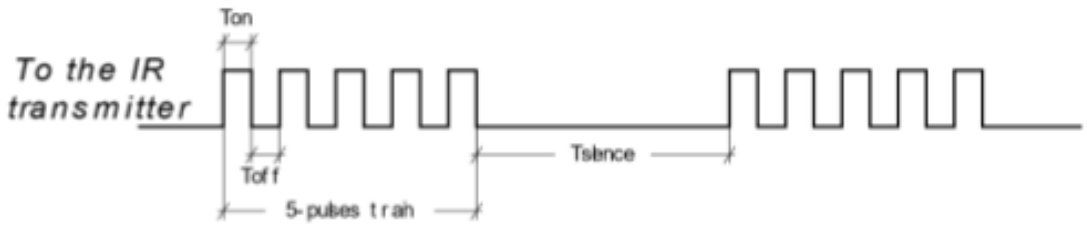


Figure 3.2: digital signal to drive the transmitter

My first idea was to use 3 timers, for each of the durations and handle them with interruptions. I wanted to turn on and off the timers after each other in the interruption function and count the pulses, while I am doing that. The problem was that the timing wouldn't have been precise enough and I also used more resources than necessary.

The proper solution is to use timers' PWM mode function, which can handle the whole task with only one timer and without the need to write the interruption function.

3.2.1 Pulse-width modulation

Pulse-width modulation (PWM) is a technique used to generate several pulses with different duty cycles in a given period of time at a given frequency.

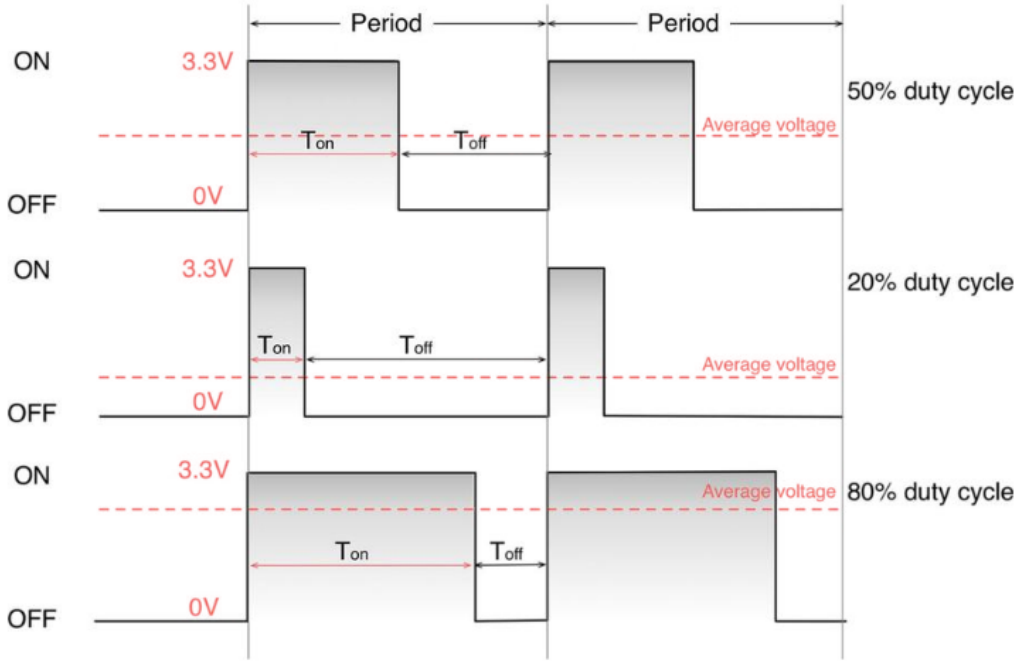


Figure 3.3: PWM with different duty cycles [17]

PWM has many applications in digital electronics, but all of them can be grouped in two main categories:

- Control the output voltage.
- Encoding (modulating) a message (that is, a series of bytes in digital electronics) on a carrier wave.

Those two categories can be expanded in several practical usages of the PWM technique. If only focusing on the control of the output voltage, here are several applications:

- generation of an output voltage ranging from 0V up to VDD (that is, the maximum allowed voltage for an I/O, which in an STM32 is 3.3V);
- dimming of LEDs;
- motor control;
- power conversion;
- generation of an output wave running at a given frequency (sine wave, triangle, square, and so on); [17]

In my case it is also used for the generation of output voltage, because that is how I drive my transmitter infrared sensor.

There are two PWM modes available:

- PWM mode 1: output channel starts in an inactive state and goes to active when it reaches the match value, which depends on the duty cycle we set. (in up-counting mode)
- PWM mode 2: output channel starts in an active state and goes to inactive when it reaches the match value, which depends on the duty cycle we set. (in up-counting mode)

The two modes are demonstrated in the picture above.

For my task it doesn't matter which PWM mode I use, I just have to decide and then not change it later, because it's important for the T_on, T_off and T_silence calculations.

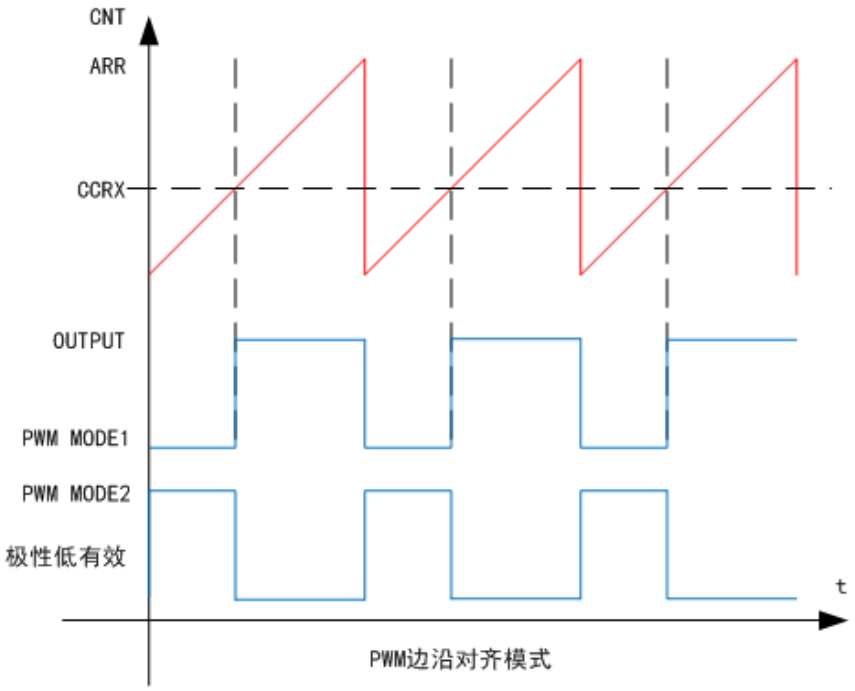


Figure 3.4: the PWM modes [18]

As it can be seen, it's easy to create a digital signal with a different T_on and T_off value, but I also needed a silence period after a given number of pulses. I solved this with **timer synchronization** and **one-pulse mode (OPM)**, using the repetition counter.

3.2.2 Timer synchronization

Some of the timers are linked together internally for **inter-timer synchronization** or chaining. It means that they can be configured as "master" and "slave", which means that the slave timer only starts counting when it is triggered by the master.

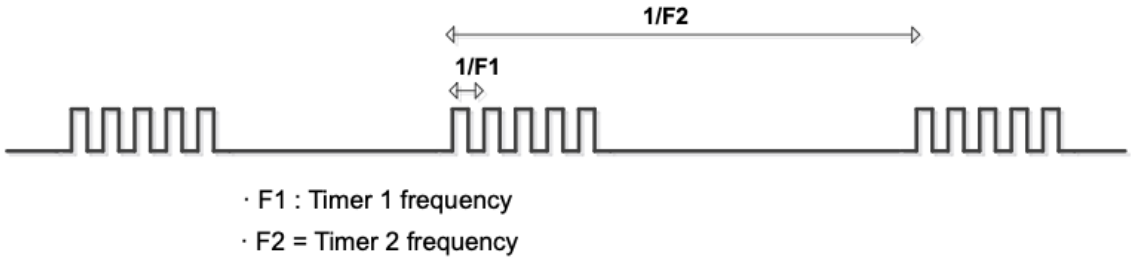


Figure 3.5: the frequencies for the timers [19]

The timer peripherals featuring the inter-timers synchronization capability have only one synchronization output signal named TRGO (abbreviation for "Trigger Out").

They also have four synchronization inputs (named ITRx where x ranges from 1 to 4) connected to the other timers synchronization outputs. The reference manual of the microcontroller lists the inter-timers connection matrix.

Slave TIM	ITR0 (TS = 000)	ITR1 (TS = 001)	ITR2 (TS = 010)	ITR3 (TS = 011)
TIM1	TIM5_TRGO	TIM2_TRGO	TIM3_TRGO	TIM4_TRGO
TIM8	TIM1_TRGO	TIM2_TRGO	TIM4_TRGO	TIM5_TRGO

Figure 3.6: inter-timers connection matrix for TIM1 and TIM8 [20]

As I have chosen the TIM1 and TIM2 timer, I have to set that the trigger input of the TIM1 is the ITR1.

3.2.3 One-pulse mode

The **one-pulse mode** of a timer peripheral is a feature that can be used together with the timer channels configured in output mode. It allows the timer to generate a pulse of a programmable width after/before a programmable delay on the timer channels

configured in PWM1 or PWM2 output compare modes. It is basically the PWM generation, which stops after one pulse.

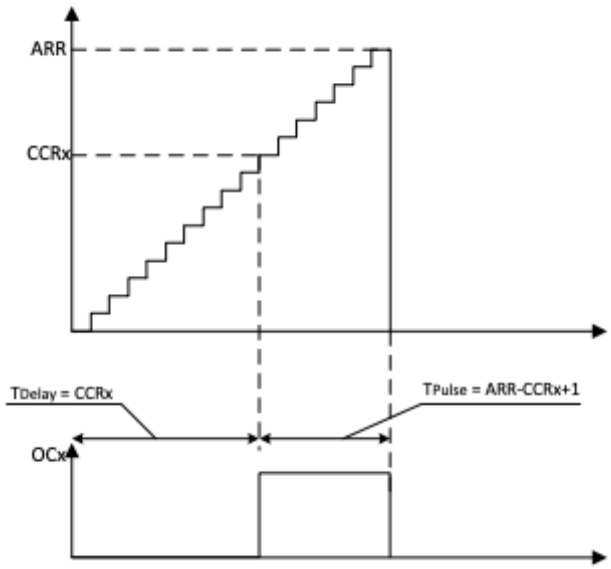


Figure 3.7: representation of one-pulse mode [19]

3.2.4 Repetition counter

A possibility to generate a series of pulses is to make the timer **repetition counter** register's content different from zero. For instance, to generate a series of five pulses, the repetition counter should be set to four. After the repetition counter content decreases down to zero, the fifth update event is the one that resets the CEN control bit. This way, the timer counter has overflowed five times before being disabled and the five required pulses are generated.

Not all the STM32 timer peripherals embed the repetition counter so as I checked in the reference manual, I could only choose either TIM1 or TIM8.

3.2.5 Implementation

The picture above shows how the synchronized timers allow us to get the desired waveform. As an example the TIM2 is the master and the TIM1 is the slave.

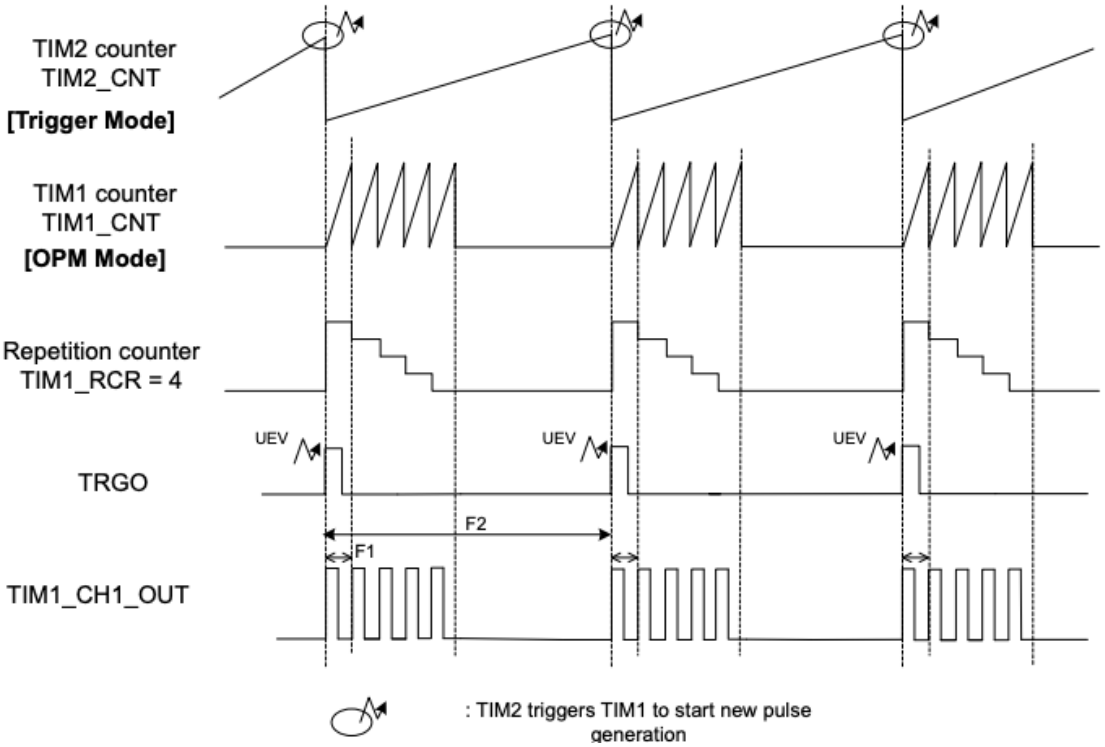


Figure 3.8: the desired waveform with the timer synchronization [19]

The TIM2 timer is configured as master to trigger the TIM1 timer ITR1 input via its internal TRGO signal, at each update event. The time between two “update event” is the T2 of the waveform targeted.

The TIM1 peripheral timer is configured as slave one pulse mode, to generate one pulse when triggered by TIM2 timer peripheral. To repeat the pulse generated, we use the repetition counter of TIM1 which should be configured to the desired number of pulses minus one.

After these settings the code was tested with an oscilloscope and at first my silence had a value of ‘1’, instead of ‘0’ and one less pulse, which is because of the fact that my silence should involve part of the signal generation from the other timer, because as it is in PWM1 mode, the last half-period is ‘0’. Because of this, the issue can be solved easily by changing the polarity. At the PWM generation settings in CubeMX there is also an option for polarity, which can be either high or low. After changing that, I could see in the oscilloscope that my signal was what I wanted.

3.3 Calculations for the defines

As I was saying, I wanted to make the T_on, T_off, T_silence and the number of pulses easily changeable. It makes the testing much more easier, because if we want to change these numbers, we have to know which timer is the corresponding one, calculate the prescaler and period and set in CubeMX. As a solution I made a function for the calculations and created #define macros, so the values can be changed easily through those.

At this part I had to take into account some important factors.

- I have to avoid floating points and use 'ns' because of that reason.
  As the timers are working as counters, the period cannot be set to 200.4 for example, because it can either count to 200 or 201. In between would not make sense.
  The timers' prescaler and period are 8, 16 or 32bit values, which are changeable by us, but as I was saying we cannot set the values as floating points, which we would not do normally, but as I wanted to write an algorithm, which calculates these, it could easily happen, if we don't pay attention.
- All the time, when I change something in CubeMX, a new code is generated.
  It is initializing all the timers, so the correct way to rewrite these is to make a function, which rewrites these values and I call it after the original initialization functions.

In the chapter before I was explaining how the timer peripherals in the microcontroller are built. It is leading us to the equation that

$$\frac{Period * Prescaler}{Clock\ frequency[Hz]} = Time\ of\ period[sec]$$

Time of period means that how much time does it take for the timer to count up to the "Period" number with the "Prescaler" number we also set before.

As I was explaining how I generate my digital signal, I am using two periods (master and slave) and they can be defined as:

```
slave_period_ns = Ton_+Toff_;
master_period_ns = (Ton_+Toff_)*pulses-Toff_+Tsilence_;
```

Inside the timer settings the “Pulse” defines the duty cycle, which for us defines the T_on and T_off values.

```
duty = Ton_/slave_period_ns;
pulse = htim1.Init.Period*duty;
TIM1_SetPulse(pulse);
```

The duty cycle of the master timer is not important, since I only need its rising edge as a trigger to the slave timer. To set the “Pulse” value, I copied the original initialization function’s pulse settings into a new function I created, because it is not as straightforward as setting for example the repetition counter to define how many pulses we need, which is:

```
htim1.Init.RepetitionCounter = pulses-1;
```

HAL library has a function to get the peripheral clock settings. Since the clock frequencies are also changeable, it is useful for us. In CubeMX the clock configurations can be seen and can be also configured.

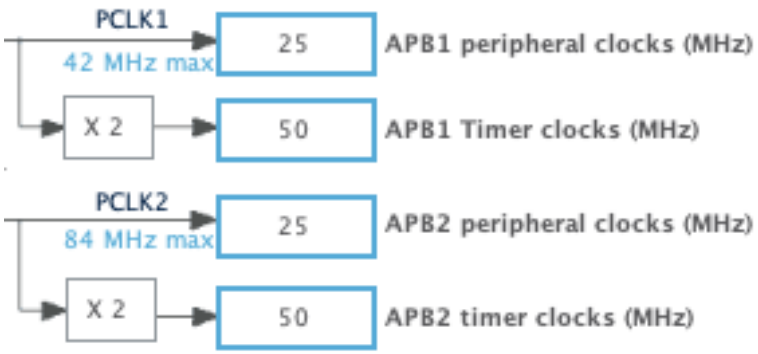


Figure 3.9: timer clock configuration

As we can see the peripheral clocks are set to 25Mhz and timer clocks have a $*2$ multiplier from that, which cannot be changed so using these information, I got the timer clock frequencies by:

```
Fpclk1_tim = HAL_RCC_GetPCLK1Freq()*2;
Fpclk2_tim = HAL_RCC_GetPCLK2Freq()*2;
```

To see which APB bus the timers are connected to, I checked it in the microcontroller's reference manual.

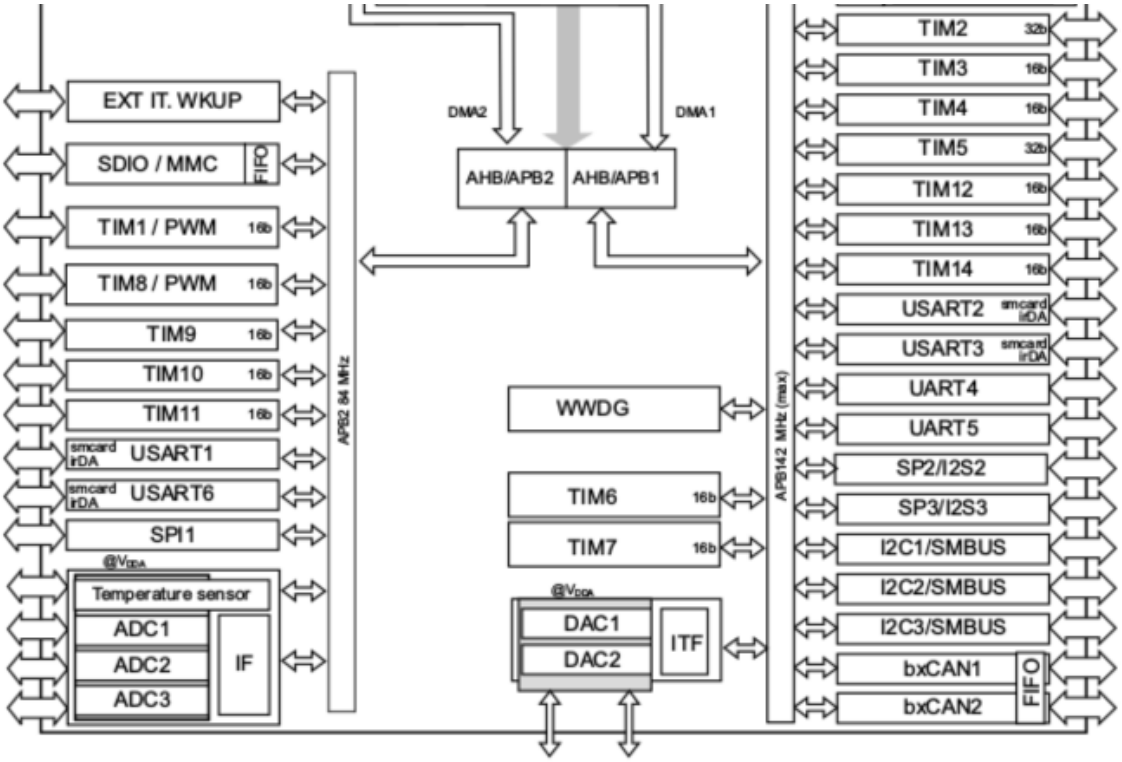


Figure 3.10: APB buses connected to timers [20]

To make sure that I won't set a value with floating point to the Period and Prescaler, I used "for" cycles and compared the values to each other. I am also using the iteration of the cycle.

```
for(int i = 0; i<65536; i++)
   {
     if(i==((period_ns*Fpclk_tim)*(htim->Init.Prescaler+1))/1000000000))
         {
           htim->Init.Period = i;
           done=1;
           break;
         }
   }
```

It's important that on the other part of the equation, the value can still appear to be not a cardinal number, if it is a small value and it has to be a small value in a lot of cases for example for an infrared sensor I have to use 25us period time and a half of it is

already 12.5us. It is the reason that I have to use nanoseconds for my calculations. Because of this, I am calculating with really big numbers, so I had to make sure that I am using enough bits and my variables won't overflow. Uint34 in that case is not enough, because it would overflow after 4 seconds, so it is important to use Uint64 variables.

The cycle above made in a way that if it cannot find a good period value then I increment the Prescaler by one and try again and this is until the Prescaler reaches its maximum value.

4 Capturing in the infrared receivers

4.1 Arrival time of IR receivers

In the absence of an obstacle in front of it, each IR receiver will produce a low-level digital signal (silence). Each IR receiver will produce, in the presence of an obstacle in front of it, the same waveform sent to the IR transmitter. If the obstacle is entering/exiting the area of the IR receiver, it may be the case the first/last train will contain less pulses and/or the first/last pulse is narrower. At the very least, each IR receiver will produce a digital signal different from silence, that is, a digital signal with some pulses (rising edges).

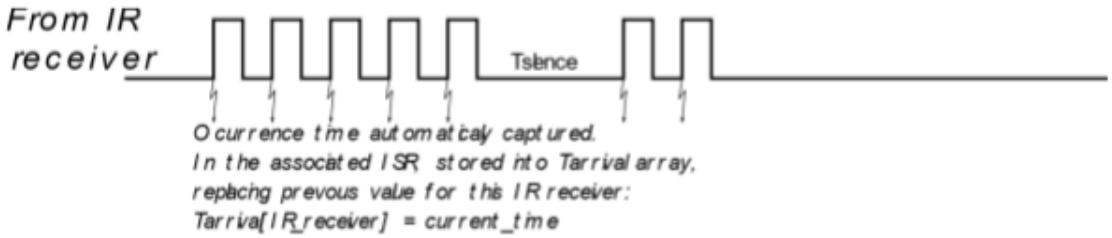


Figure 4.1: the captured signal [16]

4.1.1 Input capture mode

I configured a timer capture channel in the microcontroller per IR receiver. This capture channel is configured to detect rising edges in the incoming digital signal and capture the time these edges occur. With the associated interrupt service routine (ISR), these arrival times are stored into an array of values (one per IR receiver), replacing the previous stored value. See image above.

Below there is an example of arrival times for the three receivers A, B, and C.

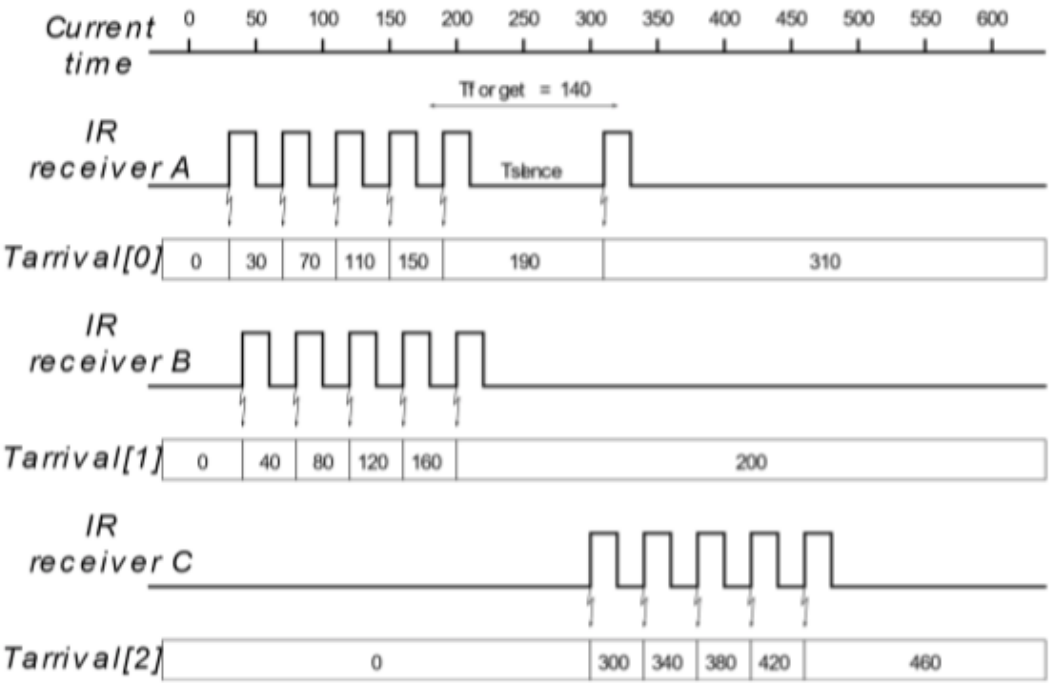


Figure 4.2: arrival times of the rising edges

Timers' **input capture mode** is used exactly for this purpose. This mode is important for external signal measurement or external event timing detection. The current value of the timer counts is captured when an external event occurs and an interrupt is fired. So this feature can be used for a wide range of measurement applications. [21]

Firstly, to test my code, I connected externally(with a cable) the digital signal's output pin, which I created for the transmitter to a pin which I set as an input and I configured the belonging timer's channel to input capture mode and also to detect the rising edges.

To capture the time it is a good way to simply ask for the present counter value of the timer and calculate the time from it, knowing the prescaler and the clock frequency. But it would only work if the timer would not roll over. A solution is to count how many times it rolled over so then we can measure the time. The HAL_TIM_PeriodElapsedCallback gets called in every rollover so I am counting the number of rollovers here.

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
     if ( htim->Instance == htim2.Instance )
          {
               epoch++;
          }
```

In HAL_TIM_IC_CaptureCallback every time a rising edge comes, I calculate the arriving time.

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef* htim)
  {
    gu32_Cnt = HAL_TIM_ReadCapturedValue(htim, htim->Channel);
    gu32_Ticks = gu32_Cnt + (epoch * (htim4.Init.Period+1));
     if ( htim->Instance == htim1.Instance )
     {
       if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
       {
        //*20 because of the 50Mhz clock and because of ns
        T_arrival[0] = gu32_Ticks*((htim4.Init.Prescaler+1)*20);
        HAL_UART_Transmit(&huart2,(uint64_t*)uart_buf2,uart_buf_len2,100);
       }
     }
  }
```

I also configured an USART, so I could write the values to the console.

As in here I am using the USART only for testing, the settings are not so important so I only have to pay attention to set the same values on both sides. In CubeIDE there is a console to present serial communication, I used that.

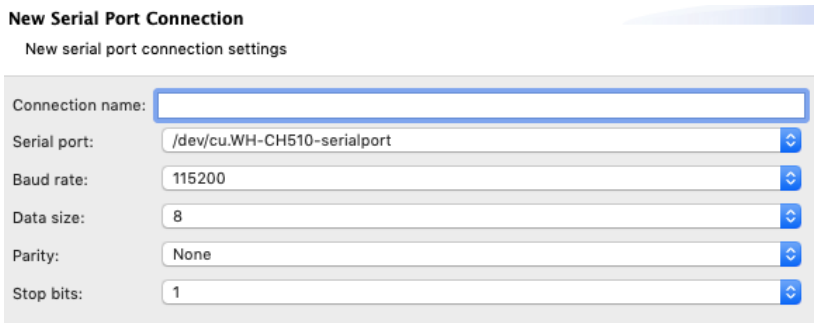


Figure 2.4: Window of making a new serial port connection

I set the same values to configure the UART in CubeMX and I also set it to be Asynchronous.

.

4.1.2 Filter time

Simultaneously, a periodic filter timer is used to filter out ("forget") arrival times too far in the past.

To this extent, in the associated interrupt service routine to this filter timer I subtract the current time minus the Tarrival[IR_receiver] to obtain the elapsed time since I received the last rising edge in the IR receiver. If this time lapse is less than a configurable Tforget time, the system determines there is still an object in front of the IR receiver; otherwise the system considers there is no obstacle in front of the IR receiver. This is depicted below:

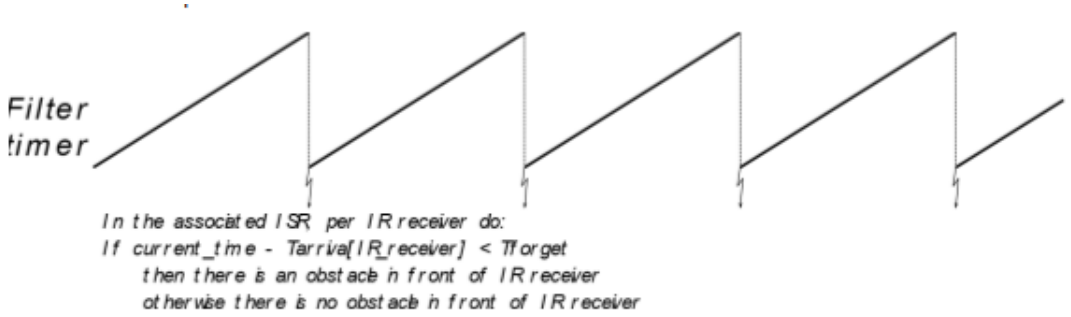


Figure 4.4: filter timer

The Tforget time period must be set large enough to accommodate the maximum distance between two consecutive rising edges: max (Ton+Toff, Ton+Tsilence).

If it was smaller than it would "forget" too many rising edges and miss the presence of the obstacle.

The result of the subtract is stored in another array, Tdiff, see below:

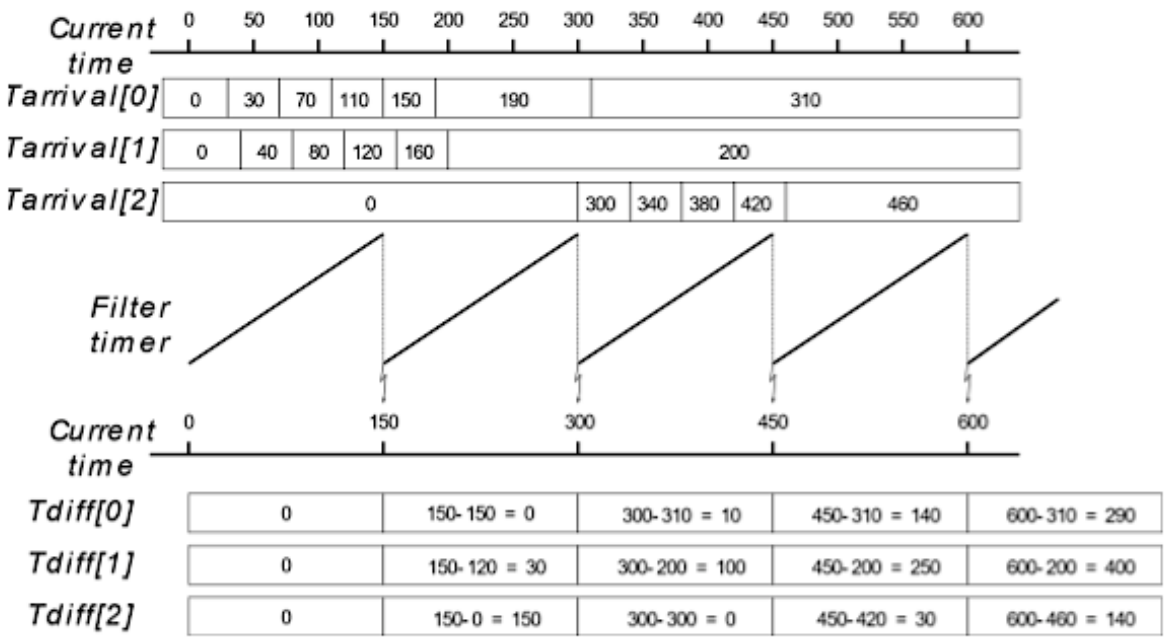


Figure 4.5: the usage of filter time

Assuming an example value of Tforget = 140, these Tdiff values translate into the following bitmasks. Determining in front of which IR receiver there is an obstacle or not the system composes a bitmask of this information gathering all IR receivers.

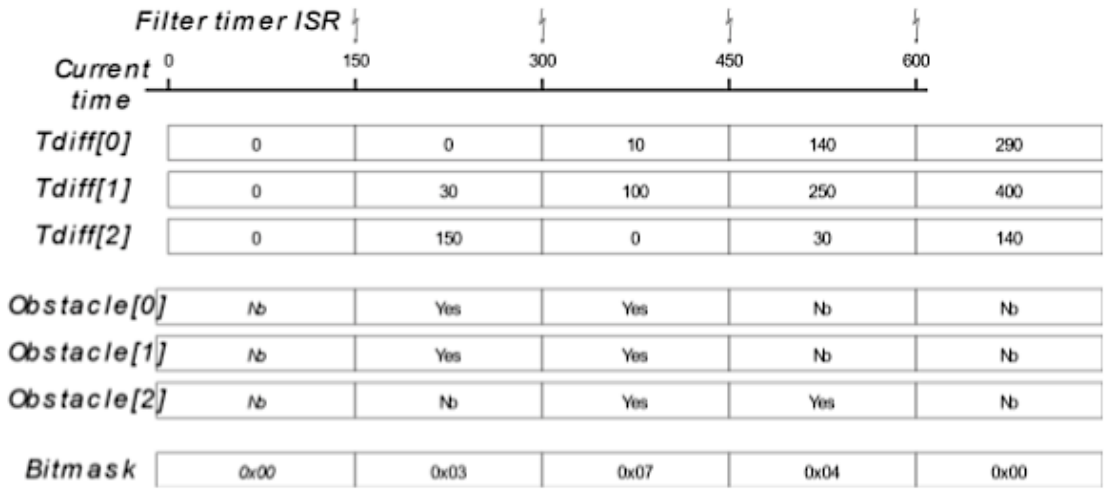


Figure 4.6: bitmasks from the presence of obstacles

If the forget time is smaller than the “current time – last time of rising edge arrival”, I set the Obstacle[IR_receiver] variable’s value to the corresponding IR receiver’s bitmask, indicating the presence of the obstacle by that.

Then with the “OR” logical operator, I created one bitmask out of the three, so it indicates the position of the obstacle.

```
for(int i=0; i<3 ;i++)
     {
     if((T_diff[i] < Tforget) && (T_arrival[i]!=0))
     {
       if(i==0)
         obstacle[i] =  1 << i; //0x01
       if(i==1)
         obstacle[i] =  1 << i; //0x02
       if(i==2)
         obstacle[i] =  1 << i; //0x04;
     }
     else
       obstacle[i] = 0x00;
     }
 uint32_t bitmask= obstacle[0] | obstacle[1] | obstacle[2];
```

These bitmask values can be translated into the set of IR receivers with an obstacle in front, represented in the image below with a grey colour.

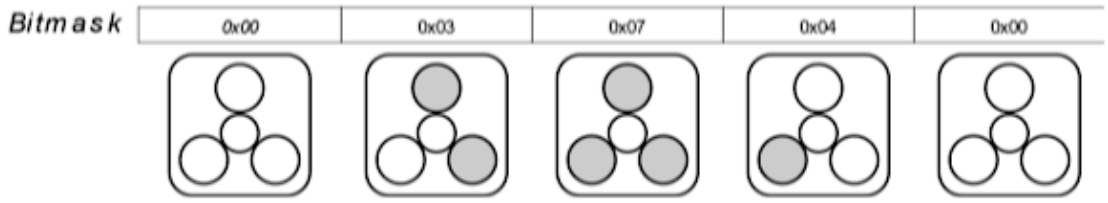


Figure 4.7: representation of the bitmasks

An algorithm (more on this later) tries to match the sequence of bitmask values against a set of pre-defined gestures the system can detect. A gesture may be defined as a sequence of bitmasks starting and ending in the bitmask 0x00; the 0x00 bitmasks are used as markers to isolate a gesture but are not part of the gesture itself.

From the series of bitmask values this algorithm would determine that the corresponding gesture is a hand crossing diagonally from top-right to bottom-left, as shown below.

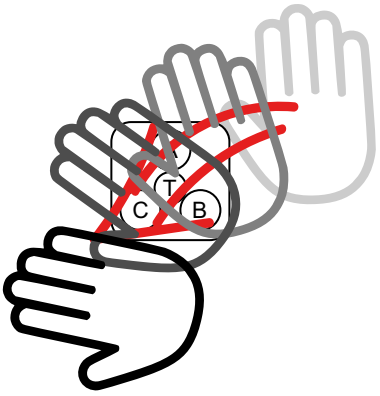


Figure 4.8: hand crossing in front of the sensors [16]

4.2 Bitmask operations

4.2.1 Bitmask duplicate removal

It is assumed the period of the filter timer is small, as small as possible, so the system is able to detect a quick gesture from the user. But also, slow gestures must be detected. To accommodate this variability in the user gesture speed, the filter timer must be set to high frequency, and I used a bitmask duplicate removal step between the bitmask detection described above and the gesture detection algorithm.

To develop this compression step the assumption is that, as the filter timer is set to a relatively high frequency, the same bitmask will be detected several times. To solve this issue, I used a duplicate removal algorithm, see example below.

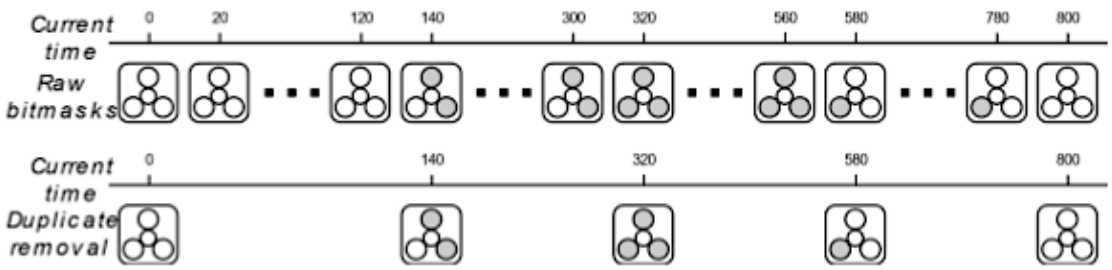


Figure 4.9: result of the bitmask duplicate removal

Assuming the time units in the image above are milliseconds, the filter timer detects a bitmask every 20 ms. When a series of identical bitmasks are detected the duplicate removal step considers only the first of them.

For this a sequence struct can be used.

A bitmask sequence is

- A sequence of bitmasks of variable length, and
- Internally represented by a sequence_t value:

```
#define MAX_SEQ_LENGTH  6
typedef struct {
  uint32_t length;
  uint32_t bitmasks[MAX_SEQ_LENGTH];
} sequence_t;
```

The MAX_SEQ_LENGTH macro is defined as an upper bound of the maximum number of bitmasks in a gesture; the bitmasks array field stores the bitmasks of the gesture, and the length field indicates how many entries of the bitmasks array are used, because it is not necessarily the maximum value and in this way it is kept in track how many bitmasks are in the sequence.

In every filter time a bitmask gets created, so I call the algorithm all the time when it happens through the timer callback function. I will not put every bitmask into an array and then do a bitmask removal, rather I check every bitmask before I put it in an array. I save the first bitmask into a variable and when the next one is coming, I am comparing the two with each other.

Since the bitmask 0 is indicating the beginning and the end of a sequence, firstly I check if it is 0 or not.

```
if(bitmask==0 && first==0)
   {
     bitmask_first = bitmask;
     seq.length=0;
      first=1;
   }
```

Then I call my algorithm, which starts with

```
if(bitmask != bitmask_first)
```

The algorithm will go through only if the two bitmasks are different. If that is and the second one is 0, then that is the indication of the end of the sequence. At this point I clear the sequence and the corresponding variables, so the algorithm can start a new sequence at the next filter time.

The arrival time of this first-in-a-row bitmask (values 0, 140, 320, and 560 in the example above) are important for the next step, the removal of spurious of bitmasks, so I am also saving the arrival times as it can be seen later in the next section.

4.2.2 Spurious bitmasks removal

The system must also consider that slight variations in the angle of the user hand when moved in front of the device may produce spurious bitmasks.

Compare the two examples depicted below. The user gesture is the same in both cases: the user hand (represented by a pink area in the image) is crossing diagonally from top-right to bottom-left. Theoretically the bitmask sequence for this gesture would be A-B, A-B-C, C. This is what happens in the first case, where the user hand is perfectly aligned with the pair of receivers A-B: The sequence starts at time 140, when a bitmask different from 0x00 (A-B) is detected, then A-B-C is detected at 320, and C at 600; the sequence ends at 800 when an empty bitmask 0x00 is detected.

However, due to slight differences in the angle of movement of the user hand with respect to the angle of the pair of IR receivers A-B, some spurious bitmasks (marked in red) are detected in the second case: The A receiver perceives a bit earlier than receiver B that the obstacle appears and disappears, so the sequence is A at 140, A-B at 160, A-B-C at 320, B-C at 600, and C at 620. The two spurious bitmasks, A at 140 and B-C at 600 last much less time than the rest, only 20 ms.

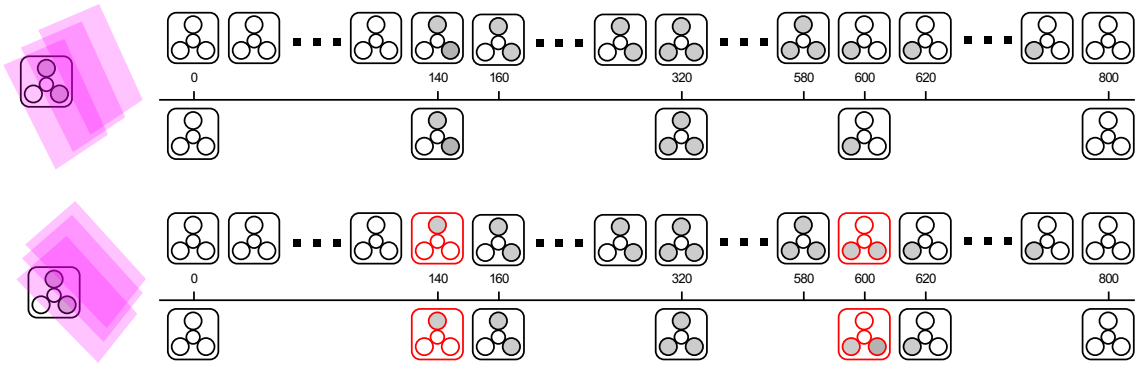


Figure 4.10: representation of spurious bitmasks

A spurious bitmask removal step is required to eliminate those from a sequence. The first issue to solve here is determining how to decide if a detected bitmask is spurious or not. For this decision, I used the time duration of the bitmasks.

A time threshold, Tspurious, is defined and easily customized in the program as I added to the #defines-s. If the duration of a bitmask is less than Tspurious, the bitmask is discarded and replaced by the next bitmask in the sequence.

The minimum time for Tspurious is the period of the filter timer; this time is the absolute minimum duration of a bitmask.

The actual value of Tspurious must be selected as a trade-off between detection robustness and the maximum gesture speed and angle deviation the device is able to manage:

- If Tspurious is too small, no bitmasks will be considered spurious, and the user will have to execute the gesture perfectly aligned with the angle formed by the IR sensors in order to be correctly detected.
- If Tspurious is too large then too many bitmasks will be purged, and the gesture won't be recognized.

From the example above, if Tspurious is set to 50 ms, any sequence with a duration less than 50 ms will be considered spurious and eliminated from the sequence of bitmasks sent to the gesture detection algorithm. The result of this algorithm for the above example sequences is shown in the figure below.

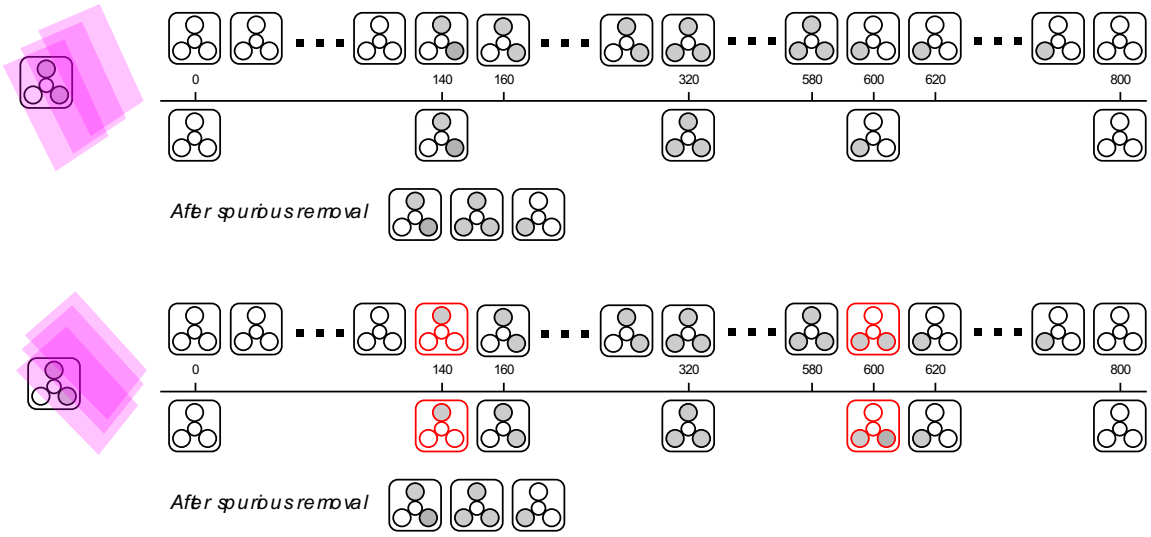


Figure 4.11: spurious bitmask removal

In both cases, after the removal of those bitmasks with a duration less than 50 ms, the same sequence of bitmasks (the same gesture) is sent to the gesture detection algorithm, which I will write about in the next chapter.

I implemented the spurious bitmask removal into the duplicate removal, so firstly,

```
if(T_bitmask_arrival1==0){
                T_bitmask_arrival1 = T_filter_current;
                bitmask_first = bitmask;
}
```

As a first arrived bitmask, I will save its arrival time without any further calculations because even if it appears to be spurious, I still need to make a comparison with the next arriving one. So when the next one arrives, I will make time calculations to find out if it is spurious.

```
else
    {
      T_bitmask_arrival2 = T_filter_current;
      if(T_bitmask_arrival2 - T_bitmask_arrival1 > Tspurious)
      {
        seq.bitmasks[seq.length]=bitmask_first;
        seq.length++;
      }
      bitmask_first=bitmask;
      T_bitmask_arrival1 = T_filter_current;
```

```
    }
```

As it can be seen, if it is spurious then I just move on to the next bitmask and I don't add it to the sentence.

4.3 Gesture detection algorithm

A sequence of bitmasks is finished as the bitmask values different from 0x00, after removal of duplicate and spurious bitmasks. This sequence is a simple array of bitmask values; this array is the input argument to the gesture detection algorithm.

The gestures are identified by a name and a number. Numbers start from 1, as 0 is reserved for the "unknown/unrecognized" gesture. Names are acronyms built from T (Top), B (Bottom), L (Left), R(Right), and follows the template <from>-<to>; for example T-B means "from top to bottom" (vertical) and TR-BL means "from top-right to bottom-left" (diagonal). The gesture detection algorithm manages the detection of the 8 sequences (vertical, horizontal, and 45º diagonal).

Different 'struct' types can be used, so the code is more clear and understandable.

A gesture is

- A valid sequence that represents a recognizable user gesture, and
- Internally represented by a gesture_t value:

```
typedef struct {
    uint32_t id;
    char *name;
    uint32_t length;
    uint32_t bitmasks[MAX_SEQ_LENGTH];
} gesture_t;
```

The id field is a numerical value that uniquely identify the gesture, starting from 1 (0 is reserved for the "unknown/unrecognized" gesture); the name value is a textual

unique name for the gesture, and length and bitmasks fields have the same meaning as in the sequence_t type.

A gesture pool is

- The set of gestures a device may detect, and
- Internally represented by a gesture_pool_t value:

```
#define MAX_POOL_GESTURES 8
typedef struct {
     uint32_t length;
     gesture_t gestures[MAX_POOL_GESTURES];
} gesture_pool_t;
```

The MAX_POOL_GESTURES macro must be defined as an upper bound of the maximum number of gestures a device may detect; the gestures array field stores the set of gestures the device detects, and the length field indicates how many entries of the gestures array are used. A gesture_pool_t value is expected to be const, as the gestures a program can detect can't change during runtime.

What follows is the definition of the gesture pool used to detect the 8 possible gestures (vertical, horizontal, and 45º diagonal) described in the previous section:

```
static const gesture_pool_t gesture_pool_triangle = {
.length = 8,
.gestures = {
   {.id = 1, .name = "T-B", .length = 3, .bitmasks = {0x01, 0x07, 0x06} },
   {.id = 2, .name = "B-T", .length = 3, .bitmasks = {0x06, 0x07, 0x01} },
   {.id = 3, .name = "R-L", .length = 5, .bitmasks = {0x02, 0x03, 0x07, 0x05, 0x04} },
   {.id = 4, .name = "L-R", .length = 5, .bitmasks = {0x04, 0x05, 0x07, 0x03, 0x02} },
   {.id = 5, .name = "TR-BL", .length = 3, .bitmasks = {0x03, 0x07, 0x04} },
   {.id = 6, .name = "TL-BR", .length = 3, .bitmasks = {0x05, 0x07, 0x02} },
   {.id = 7, .name = "BL-TR", .length = 3, .bitmasks = {0x02, 0x07, 0x05} },
   {.id = 8, .name = "BR-TL", .length = 3, .bitmasks = {0x04, 0x07, 0x03} }
          }
};
```

When I already have an array of bitmasks out of the spurious removal algorithm, I only have to compare with the gesture pool and then I can have the name of the movement.

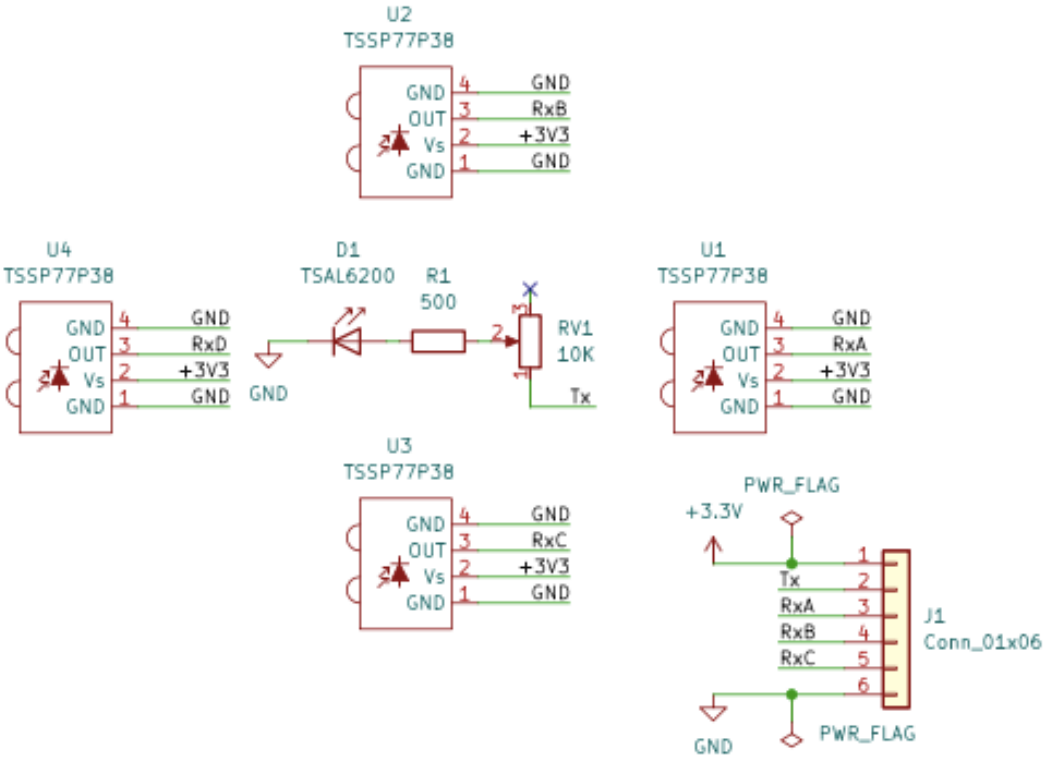
The receiver sensors can be in other shapes so I created other gesture pools for the different shapes. If I change the placement of sensors then it can be tested on those too.

5 Schematic

First the application was planned with three sensors in a triangle shape and also that is how I explained the application in this documentation.

I was testing the application with an FPGA, which was programmed to simulate the gestures, when there are three sensors in triangle shape and also four sensors in a diamond shape, but the code is easily changeable for other formations too.

Since the receivers were too small to solder on a regular board, we had to design a new one and it is designed for four sensors in a diamond shape. The schematic can be seen below.



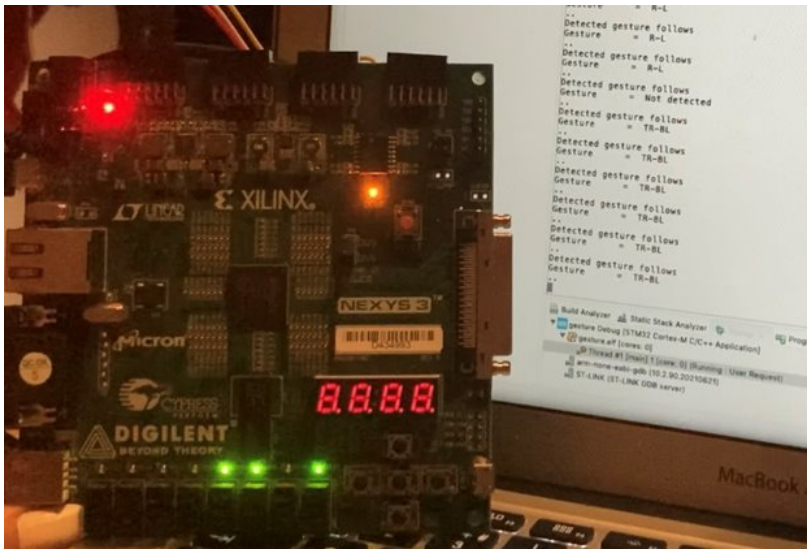
6 Testing and result

As it was not possible to solder the sensors to a regular board and it was needed to design and order a new one, we ran out of time, so I was only able to test the application with an FPGA, which was simulating the gestures but even with spurious bitmasks, so I could see that my algorithms are working properly.

Through the console I can see the name of the movements after each other. As I was testing the best time for spurious bitmask removal was 60ms for me, as a trade-off between detection robustness and the maximum gesture speed and angle deviation the device is able to manage, so with this number it seemed that the algorithm chooses the correct amount of bitmasks to be spurious.

```
..
Detected gesture follows
Gesture      =  TR-BL
..
Detected gesture follows
Gesture      =  Not detected
..
Detected gesture follows
Gesture      =  R-L
..
Detected gesture follows
Gesture      =  R-L
```

As a result the system is working properly. Taking into consideration some special cases, it is still possible to make further improvements.



Video link, in which I present the application with the FPGA: https://www.youtube.com/watch?v=MVezzy_Fexo&t=111s

7 Budget

1. Unitary Prices						
Ref	Unit	Description	Price (€)			
1. Materials						
m1	u.	Infrared transmitter diode TSAL6200	0,8400€/unit			
m2	u.	Infrared receiver TSSP77P38	2,2500€/unit			
m3	u.	STM32-F4 discovery MCU board	22€/unit			
m4	u.	500ohm resistor	0,0708€/unit			
m5	u.	10K potentiometer	1,2600€/unit			
m6	u.	Custom board to JLCPCB	18,33€/unit			
m7	u.	USB cable to power the testing board	3€/unit			
m8	u.	USB cable for UART communication	12€/unit			
m9	u.	USB cable for debugging	3€/unit			
m10	pack	Cable for connecting pins	4,75€/pack			
m11	u.	STM32CubeIDE software licences	0€			
m12	m.	Xilinx Nexys 3 board for testing	31,1€/month			
m13	m.	Digital oscilloscope	16,66€/month			
m14	m.	Macbook Air	25€/month			
2. Labor						
w1	h.	Engineer	17€/h			

2. Measurements						
Ref	Unit	Description	Price(€)	Quantity	Estimated life	Cost
Depreciation						
m12	u.	Xilinx Nexys 3 board for testing	311€/unit	1	10 years	31,1€/month
m13	u.	Digital oscilloscope	500€/unit	1	30 years	16,66€/month
m14	u.	Macbook Air	1500€/unit	1	5 years	25€/month
Materials						
m1	u.	Infrared transmitter diode TSAL6200	0,84	1		0,84 €
m2	u.	Infrared receiver TSSP77P38	2,25	4		9,00 €
m3	u.	STM32-F4 discovery MCU board	22	1		22,00 €
m4	u.	500ohm resistor	0,0708	1		0,07 €
m5	u.	10K potentiometer	1,26	1		1,26 €
m6	u.	Custom board to JLCPCB	18,33	1		18,33 €
m7	u.	USB cable to power the testing board	3	1		3,00 €
m8	u.	USB cable for UART communication	12	1		12,00 €
m9	u.	USB cable for debugging	3	1		3,00 €
m10	pack	Cable for connecting pins	4,75	1		4,75 €
m11	u.	STM32CubeIDE software licences	0	0		- €
m12	m.	Xilinx Nexys 3 board for testing	31,1	4		124,40 €
m13	m.	Digital oscilloscope	16,66	4		66,64 €
m14	m.	Macbook Air	25	4		100,00 €
			Total materials cost			365,29 €
Labor						
w1	h.	Engineer	13,59	300		4.077,00 €
			Total labor cost			4.077,00 €
			Total assembly cost			4.442,29 €

3. Valuation			
Ref.	Unit	Cost	
Total materials cost	€	365,29 €	
Total labor cost	€	4.077,00 €	
Total manufacturing cost	€	4.442,29 €	
Manufacturing overheads	0,13	577,50 €	
Industrial benefit	0,06	266,54 €	
VAT	0,21	1.054,16 €	
Total budget	€	6.340,48 €	

The monthly income for an engineer is from the the recently-published salary tables in Valencia. [23]

8 Evaluation

I have successfully developed the embedded application for an STM32 microcontroller, then tested my application on an FPGA, which simulated the gestures. I was able to see the results through the console and after making some corrections, it was working as expected. As a next step, additional improvements could still make a difference.

Further developments to improve the system could be to take more tests in different sensor positions and also to test in everyday life, inside a home with the actual sensors. Tests can discover the application more and it is leading to make the application more precise and to fix potential mistakes if they occur.

The project itself was leading me to learn a lot of new things. Even though the STM32 microcontrollers are widely used, the task itself was more unique and I had to think more about how to solve certain issues without the help of the internet. I feel that I have also realized that microcontrollers' reference manuals really do have every bit of information, it is just needed to dedicate time to look through it. I also understand now much better how microcontrollers are working and it includes that the using of timers has become much more convenient to me throughout the development process.

I can really say that the project was very interesting for me and I would like to thank my supervisor, Francisco for this as he was providing me a lot of help, information and clear explanations, saving me from a lot of struggle and making the project enjoyable for me this way.

9 Bibliography

[1] S. Muthyala, “TOP 10 INTELLIGENT HOME AUTOMATION PRODUCTS IN USE TODAY,” 23 November 2021. [Online]. Available: https://www.analyticsinsight.net/top-10-intelligent-home-automation-products-in-use-today/.

[2] S. A. Meredith Hirt, “10 Smart Home Trends This Year,” 19 April 2022. [Online]. Available: https://www.forbes.com/advisor/home-improvement/smart-home-tech-trends/.

[3] L. Shanesy, “ARE GESTURE-CONTROL DEVICES THE NEXT BIG THING IN HOME TECH?,” 27 October 2016. [Online]. Available: https://www.builderonline.com/products/home-technology/are-gesture-control-devices-the-next-big-thing-in-home-tech_o.

[4] L. Cvetkovska, “30 Smart Home Statistics for All High-Tech Enthusiasts,” 10 January 2022. [Online]. Available: https://comfyliving.net/smart-home-statistics/.

[5] S. Writer, “How Embedded Systems Impact Your Everyday Life,” 2 January 2018. [Online]. Available: https://www.totalphase.com/blog/2018/01/embedded-systems-impact-everyday-life/.

[6] B. Lutkevich, “Embedded system,” December 2020. [Online]. Available: https://www.techtarget.com/iotagenda/definition/embedded-system.

[7] D. G. Tevesz, “"Mikrokontroller alapú rendszerek" electronic note - chapter 1,” 26 September 2021. [Online]. Available: https://www.aut.bme.hu/Upload/Course/VIAUAC06/hallgatoi_jegyzetek/Mar_ea_1.pdf.

[8] T. Contributor, “Active sensor,” September 2014. [Online]. Available: https://www.techtarget.com/iotagenda/definition/active-sensor.

[9] D. o. M. a. I. Systems, “"Beágyazott és információs rendszerek" electonic note - Introductory lecture,” 10 December 2021. [Online]. Available: https://www.mit.bme.hu/oktatas/targyak/vimiac06/jegyzetek.

[10] M. Gudino, “Introduction to Microcontrollers,” 26 February 2018. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller.

[11] A. Kalnoskas, “A beginner’s guide to microcontrollers,” 2 December 2015. [Online]. Available: https://www.microcontrollertips.com/a-beginners-guide-to-microcontrollers-faq/.

[12] S. Hymel, “Getting Started with STM32 - Timers and Timer Interrupts,” 17 August 2020. [Online]. Available: https://www.digikey.com/en/maker/projects/getting-started-with-stm32-timers-and-timer-interrupts/d08e6493cefa486fb1e79c43c0b08cc6.

[13] D. G. Tevesz, “"Mikrokontroller alapú rendszerek" electronic note - chapter 3,” 12 November 2021. [Online]. Available: https://www.aut.bme.hu/Upload/Course/VIAUAC06/hallgatoi_jegyzetek/Mar_ea_3a.pdf.

[14] K. Magdy, “STM32 HAL Library Tutorial,” 2 June 2020. [Online]. Available: https://deepbluembedded.com/stm32-hal-library-tutorial-examples/.

[15] Wikipedia, “Callback,” 23 May 2022. [Online]. Available: https://en.wikipedia.org/wiki/Callback_(computer_programming).

[16] Lynred, “FIVE THINGS YOU NEED TO KNOW ABOUT INFRARED TECHNOLOGY,” 6 May 2020. [Online]. Available: https://lynred.com/blog/five-things-you-need-know-about-infrared-technology.

[17] F. R. Ballester, “Development discussion”.

[18] C. I. Out, “STM32 timer,” 2021. [Online]. Available: https://www.codeinsideout.com/blog/stm32/timer/.

[19] P. Clic, “Desarrollo STM32, la función HAL realiza la función de interrupción de tecla / temporizador / PWM,” [Online]. Available: https://www.programmerclick.com/article/63111486552/.

[20] STMicroelectronics, “General-purpose timer cookbook,” 11 July 2019. [Online]. Available: https://www.st.com/resource/en/application_note/dm00236305-generalpurpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf.

[21] STMicroelectronics, “Reference manual - STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm®-based 32-bit MCUs,” 25 February 2021. [Online]. Available: https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf.

[22] K. Magdy, “STM32 Timers Tutorial | Hardware Timers Explained,” 19 June 2020. [Online]. Available: https://deepbluembedded.com/stm32-timers-tutorial-hardware-timers-explained/.

[23] “Salary tables in Valencia” 1 June 2021.

Available: https://upvedues-my.sharepoint.com/personal/frodrigu_upv_edu_es/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Ffrodrigu%5Fupv%5Fedu%5Fes%2FDocuments%2FMicrosoft%20Teams%20Chat%20Files%2Fnuevas%2Dtablas%2Dindustria%2Dmetal%2Dvalencia%5F2021%2Epdf&parent=%2Fpersonal%2Ffrodrigu%5Fupv%5Fedu%5Fes%2FDocuments%2FMicrosoft%20Teams%20Chat%20Files&ga=1