Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Fiabilidad de la sincronización de señales utilizando GPSs como referencia global

Autor: Javier Custodio Pérez

Tutor: Ignacio Parra Alonso

Cotutora: Carlota Salinas Maldonado

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo de Fin de Grado

FIABILIDAD DE LA SINCRONIZACIÓN DE SEÑALES UTILIZANDO GPSs COMO REFERENCIA GLOBAL

Autor: Javier Custodio Pérez

Tutor: Ignacio Parra Alonso

Cotutora: Carlota Salinas Maldonado

Tribunal:

Presidente: Iván García Daza

Vocal 1º: Noelia Hernández Parra

Vocal 2º: Ignacio Parra Alonso

FECHA: 21 de Septiembre de 2021

Índice

Resumen 4
Abstract 5
Palabras clave 6
1. Introducción 7
1.1. Motivación del Trabajo de Fin de Grado 7
1.2. Estado del arte 9
2. Proyecto 10
2.1. Análisis 10
2.1.1. Módulo GPS u-blox NEO-6M 11
2.1.2. Placa Arduino UNO 14
2.1.3. Sensores del grupo INVETT 15
2.2. Especificación 18
2.2.1. Electrónica de acondicionamiento 21
2.3. Implementación 24
2.3.1. Programación de señales con Arduino a través del puerto serie 24
2.3.2. Comportamiento del Módulo GPS en función del sensor 35
2.4. Experimentación 39
2.4.1. Funcionamiento de las cámaras Grasshopper3 con la señal PPS 39
2.4.2. Funcionamiento del iluminador infrarrojo con la señal PPS 40
2.4.3. Funcionamiento de cámara e iluminador con la misma señal PPS (1 Módulo GPS) 41
2.4.4. Funcionamiento de cámara e iluminador con señales PPS distintas (2 Módulos GPS) 42
2.4.5. Envío de cadenas UBX prediseñadas al Módulo GPS a través de la Placa Arduino 43
2.4.6. Funcionamiento de las cámaras Grashooper3 comandadas por la Placa Arduino 44
2.4.7. Funcionamiento del iluminador infrarrojo comandado por la Placa Arduino 46
2.4.8. Comprobación de fiabilidad de sincronización entre los sensores *in-door* 47
2.4.9. Comprobación de fiabilidad de sincronización entre los sensores en el exterior 48
3. Conclusiones 52
3.1. Trabajo futuro 52
4. Listado de referencias 52
5. Bibliografía 53

Resumen

En este Trabajo de Fin de Grado se comprobará la fiabilidad de un sistema para conseguir que todas las señales de un vehículo autónomo perteneciente al grupo INVETT puedan estar sincronizadas independientemente de su localización o naturaleza.

Para lograrlo, se utilizará la señal de reloj atómica que captan los GPSs de los satélites. De esta manera, se podrán utilizar señales provenientes del GPS sabiendo que estarán sincronizadas cada cierto tiempo.

Para comprobar la fiabilidad, se diseñará un mismo sistema, acondicionado a cada sensor con el que se realice la prueba, y mediante la señal PPS que emitan controlar su sincronización.

Este sistema podrá generar diferentes señales de sincronismo con una referencia única global permitiendo tener sistema de sincronización Wireless.

Abstract

In this project we will test the reliability of a system to achieve that all the signals of an autonomous vehicle belonging to INVETT group to be synchronized regardless of their location or type.

To achieve this, we will rely on the atomic clock signal captured by the GPS satellites. In this way, we will be able to use GPS signals knowing that they will be synchronized.

To check the reliability, a system will be designed, conditioned to each sensor, in order to control their synchronizations through the PPS signal emitted.

This system will be able to generate different synchronism signals with a single global reference allowing to have a wireless synchronization system.

Palabras clave

- Sincronización
- GPS
- GNSS
- Telemetría
- Arduino

1. Introducción

1.1. Motivación del Trabajo de Fin de Grado

El objetivo de este Trabajo de Fin de Grado es diseñar, implementar e integrar un circuito electrónico que permita el control de muestreo de varias señales procedentes de distintos sensores sincronizándolas para su posterior visualización mediante la señal de reloj atómico que emiten los satélites.

En las Ilustraciones 1 y 2 se ven ejemplos del uso de este circuito en diferentes sensores emisores y receptores.

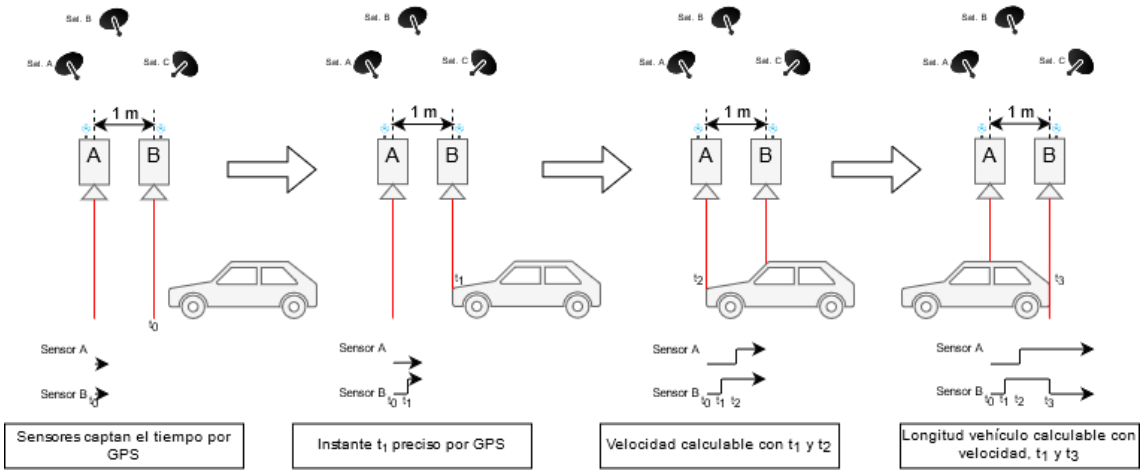


Ilustración 1. Ejemplo de sistema de cálculo preciso de velocidad y longitud de un vehículo con 2 sensores láser.

En este primer ejemplo podemos calcular la velocidad de un vehículo con 2 sensores láser si sabemos la distancia entre los sensores y el tiempo que pasa entre que se dispara el sensor A y el sensor B. Se registra la marca de tiempo en la que se disparan los sensores gracias al tiempo recibido por señal satélite y se obtiene la diferencia entre los disparos.

En el ejemplo de la Ilustración 2 se tienen diferentes sensores, tanto emisores como receptores, todos ellos recibiendo el tiempo por el circuito electrónico a desarrollar y funcionando a la velocidad establecida y sincronizada. Se ven en la gráfica las señales de las cámaras sincronizadas con los cambios correspondientes a los iluminadores y láseres. Según el instante representado, la cámara A, en movimiento, capta los 2 iluminadores encendidos, la cámara B, estática, capta los 2 iluminadores encendidos también, la cámara C no capta el láser A porque el coche pasa por delante y la cámara D capta el láser B porque no hay ningún obstáculo en medio.

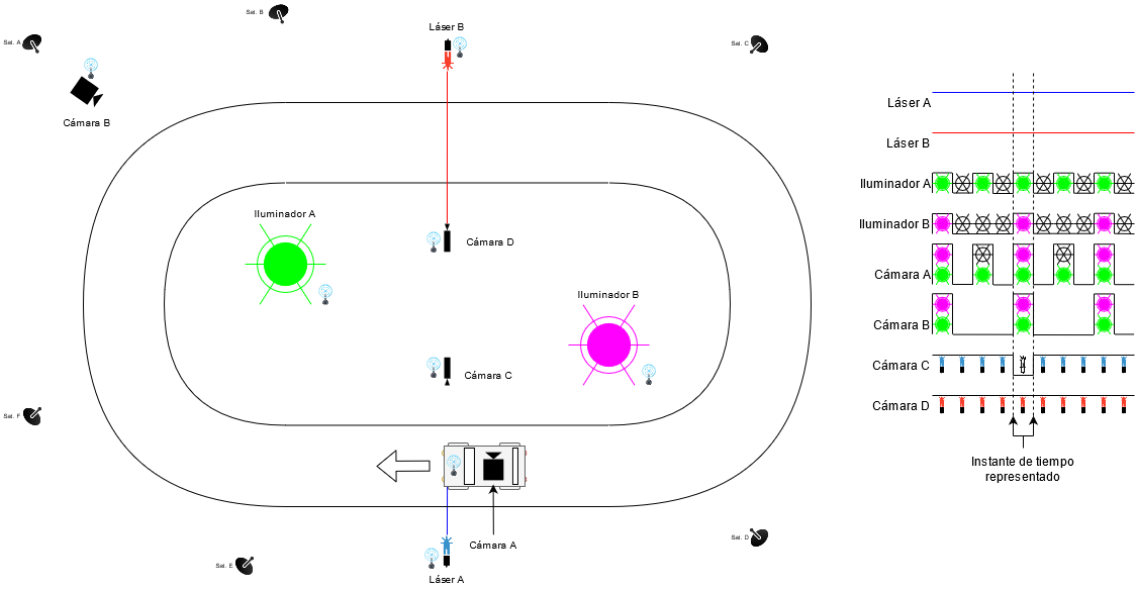
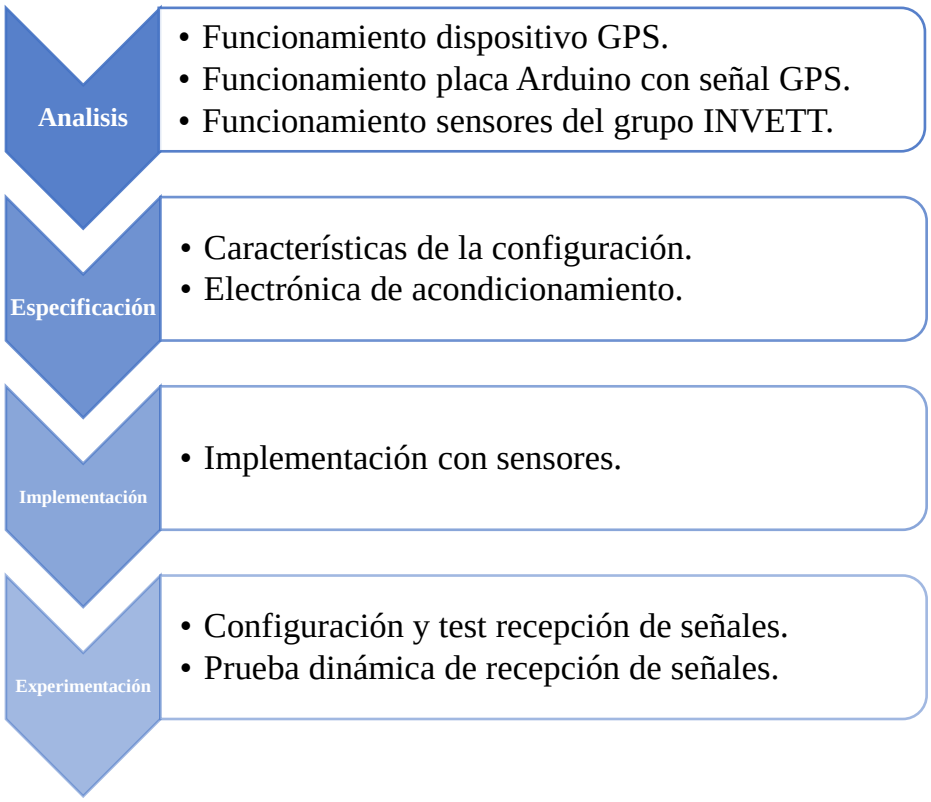


Ilustración 2. Ejemplo de sincronización de diferentes sensores en un circuito. Sensores alejados entre ellos y en movimiento.

Para llevar a cabo el Trabajo de Fin de Grado se tratarán los siguientes objetivos:

- Estudio y análisis de la señal que recibe un dispositivo GPS de las señales satélite.
- Desarrollo de habilidades en la programación de una tarjeta de desarrollo de Arduino utilizando C++ como lenguaje de programación base.
- Desarrollo de habilidades para diseñar los sistemas de acondicionamiento de requeridos para trabajar con los diferentes sensores.

Para ello, se seguirá un proceso que dividirá el trabajo en 4 fases. Estas fases dividen los principales bloques de trabajo.



En la primera fase, análisis, se estudiará el funcionamiento de la electrónica. Se utilizará un módulo GPS de la marca u-Blox. Este módulo recibirá una cadena GNSS de la que se extraerá información del tiempo para poder sincronizar las señales de los distintos sensores. Además, se estudiará el funcionamiento de una placa Arduino que utilizaremos para configurar la recepción de señales y los sensores disponibles del grupo INVETT.

En la segunda fase, especificación, se desarrollarán las configuraciones deseadas para poder muestrear las señales de los sensores como se especifique. Además, se estudiará la electrónica de acondicionamiento necesaria para que todos los sensores se implementen de manera correcta con nuestra señal.

En la tercera fase, implementación, se diseñará el programa de comunicación entre la Placa Arduino y el Módulo GPS, además de establecer el comportamiento estándar del Módulo GPS.

En la cuarta fase, experimentación, primero se realizarán pruebas sencillas con distintas configuraciones, una vez que el programa cargado en la placa Arduino funcione de manera correcta e interaccione como deseamos con el módulo GPS y los sensores se realizará una prueba dinámica en la que se sincronizarán todos los sensores.

1.2. Estado del arte

En la actualidad, las señales GNSS se utilizan normalmente con propósitos de geolocalización a pesar de que sus señales contienen mucha más información. La más extendida es la señal GPS.

Sin embargo, a pesar de que se puede obtener una marca de tiempo fiable de estas señales, lo más utilizado es el protocolo de internet NTP, cuyo error es de hasta varios milisegundos, superior al error que podemos obtener utilizando las señales GNSS.

El protocolo de internet NTP permite una sincronización entre dispositivos con un error máximo de 10 milisegundos a través de internet y de hasta 200 microsegundos en condiciones óptimas. Esto se debe a que el algoritmo que utiliza usa aproximaciones para calcular el instante de tiempo y puede fluctuar en función de lo que tarden los mensajes en enviarse y recibirse de cada dispositivo.

Este error que puede oscilar entre una escala de milisegundos a microsegundos no hace factible el protocolo NTP como medida de sincronización de señales, pero es el más usado cuando una alta fiabilidad no es necesaria.

Existen otros protocolos como SNTP, que omite algunos pasos y hace ajustes periódicamente para reducir el consumo o la memoria a cambio de una menor precisión que el protocolo NTP, o el protocolo PTP, utilizado en redes industriales que aporta una gran precisión siempre y cuando el retraso en la recepción y procesado del mensaje de cada uno de los dispositivos sea el mismo, es decir, se asume una simetría, en caso contrario no tiene de corregir los desfases.

Al final, estructuras críticas como las redes de telecomunicación móviles, redes de transmisión eléctrica o centros de transacciones financieras utilizan las señales GNSS para obtener una marca de tiempo de gran precisión, pero no se contempla su uso de manera cotidiana como pueda ser en móviles o interfaces con acceso a estas señales. [1] [2] [3] [4]

2. Proyecto

2.1. Análisis

El estudio preliminar para determinar qué dispositivo GPS se utiliza nos lleva a seleccionar el Módulo GPS u-blox NEO-6M, de ahora en adelante Módulo GPS. Los criterios utilizados durante el estudio preliminar fueron la disponibilidad, precio y rangos de trabajo que ofrecía el mercado, además de las necesidades requeridas para el proyecto.

Este módulo ofrece una disponibilidad real debido a su extenso uso en aplicaciones de geolocalización. Cuenta con las interfaces UART y USB, necesarias para realizar las pruebas y la integración final junto con una placa Arduino UNO, de ahora en adelante Placa Arduino.

Además, dado su extenso uso viene premontado en una placa que facilita la conexión con otros sistemas electrónicos. Se pueden encontrar una gran variedad de combinaciones en el mercado, lo que facilita escoger el montaje adecuado para cada situación.

Se elige el uso de una Placa Arduino para el proyecto por la disponibilidad y versatilidad que ofrecen. Se hacen uso de los pines Tx, Rx, 5V y GND para comunicarse por el puerto UART al Módulo GPS, Ilustración 3.

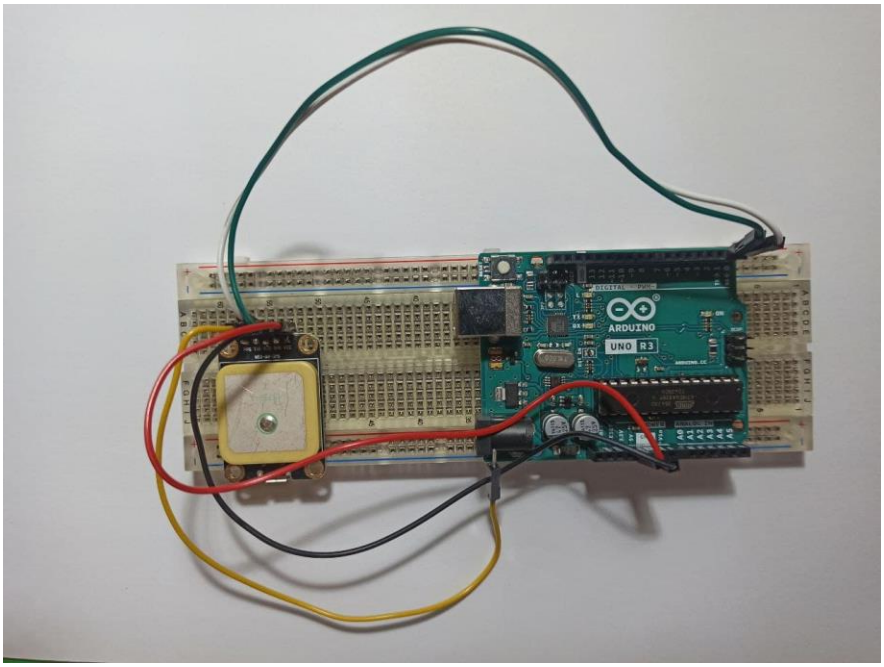


Ilustración 3. Conexión entre la Placa Arduino y el Módulo GPS.

La Placa Arduino será la que comande al Módulo GPS enviando las órdenes programadas mientras recibe constantemente una señal de tiempo recibida por satélite.

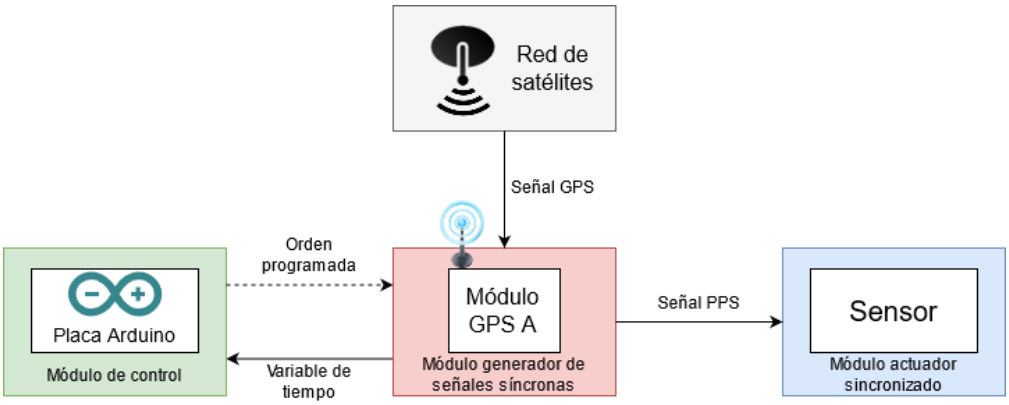


Ilustración 4. Diagrama de bloques según su función.

Con la señal de reloj atómica recibida de los satélites se enviarán en los instantes o franjas precisos los pulsos programados en el Módulo GPS por la Placa Arduino al emisor/receptor pertinente, permitiendo una sincronización entre diferentes emisores/receptores con la adecuada configuración para cada conjunto de Módulo GPS y Placa Arduino.

2.1.1. Módulo GPS u-blox NEO-6M

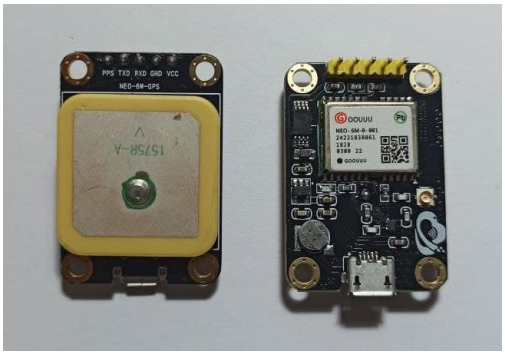


Ilustración 5. Módulos GPS u-Blox NEO-6M premontados en placa.

Las características principales por las que se elige este módulo GPS podemos verlas en la Ilustración 6:

Model	Type					Supply		Interfaces				Features						
	GPS	PPP	Timing	Raw Data	Dead Reckoning	1.75 V - 2.0 V	2.7 V - 3.6 V	UART	USB	SPI	DDC (I²C compliant)	Programmable (Flash) FW update	TCXO	RTC crystal	Antenna supply and supervisor	Configuration pins	Timepulse	External interrupt/ Wakeup
NEO-6G	•					•		•	•	•	•		•	•	o	3	1	•
NEO-6Q	•						•	•	•	•	•		•	•	o	3	1	•
NEO-6M	•						•	•	•	•	•			•	o	3	1	•
NEO-6P	•	•		•			•	•	•	•	•			•	o	3	1	•
NEO-6V	•				•		•	•	•	•	•			•	o	3	1	•
NEO-6T	•		•	•			•	•	•	•	•		•	•	o	3	1	•

O = Requires external components and integration on application processor

Ilustración 6. Características de los módulos GPS u-Blox de la serie NEO-6. [8].

Se observa que el módulo GPS se alimenta entre 2.7V y 3.6V haciendo viable la alimentación a través de una placa Arduino UNO. Además, cuenta con la interfaz UART para conectarse a la Placa Arduino e interfaz USB para conectarla a un puerto USB de un ordenador para experimentar con el programa dedicado U-Center.

Dispone también del pin/interfaz *Timepulse* que permitirá configurar la señal que se quiere enviar a los sensores.

Las especificaciones de funcionamiento de este Módulo GPS son muy aceptables para el propósito con el que será utilizado, siendo el desempeño principal la precisión de la señal *Timepulse* con un valor de error inferior a 60ns en el 99% de los casos. Otros desempeños interesantes para las aplicaciones en los que se podría usar son los límites operaciones, permitiendo el uso de este módulo GPS incluso en aplicaciones de aviónica.

Por último, este módulo GPS cuenta con los protocolos necesarios siendo capaz de leer la información recibida por satélite y procesarla, protocolo NMEA, para luego transmitir la información y posteriormente ser comandado por la Placa Arduino, protocolo UBX.

Protocolo NMEA

El protocolo NMEA o Standard NMEA es un protocolo de intercambio de datos entre dispositivos electrónicos marinos. Su origen se remonta a 1957 cuando un grupo de fabricantes de electrónica se ponen de acuerdo para conseguir un sistema común de comunicaciones entre diferentes marcas de electrónica naval.

Este protocolo permite recibir a través de GPS una serie de cadenas en formato ASCII con diferentes valores como latitud, altitud, velocidad y tiempo, siendo de interés para el proyecto el tiempo en centésimas de segundo.

Estas cadenas tienen el siguiente formato:

$GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

El inicio de cada cadena está determinado por el carácter $, seguido de un identificador que indica los datos aportados en la cadena. Después siguen los datos divididos por comas y el último valor es un checksum de seguridad

En el caso anterior, se tendría el siguiente formato:

$GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,ddmmyy,x.x,a*hh

hhmmss.ss	Hora UTC
A	Estado receptor (A = OK, V=Warning)
llll.l,a	Latitud (a = N o S)
yyyyy.y,a	Longitud (a = E o W)
x.x	Velocidad en nudos in knots
x.x	Curso en grados
ddmmyy	Fecha UT
x.x,a	Variación magnética en grados (a = E o W)
*hh	Checksum

Tabla 1. Descripción de valores de una secuencia NMEA. [5]

Protocolo UBX

La estructura de mensajes que utiliza UBX para todos sus mensajes es la siguiente:

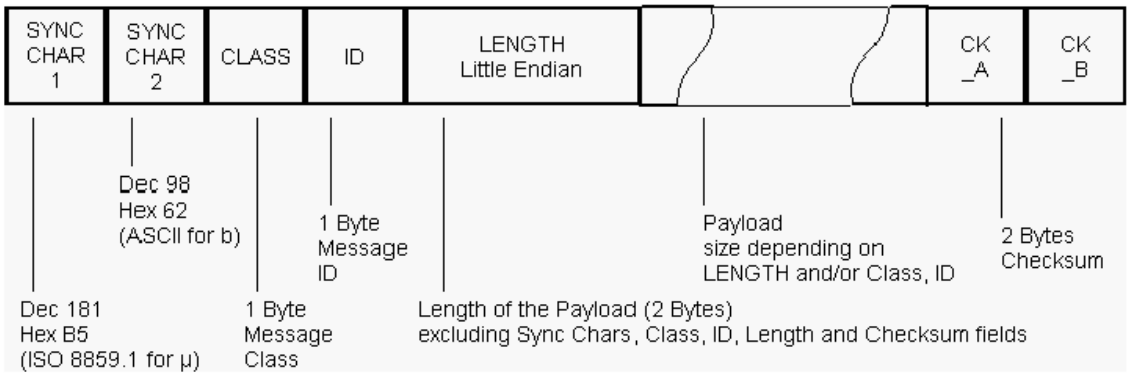


Ilustración 7. Descripción de estructura de mensajes según protocolo UBX. [9].

Cada mensaje empieza con dos bytes, 0xB5, 0x62. Le sigue un byte que indica el grupo al que pertenece el mensaje y luego otro byte que indica el mensaje en concreto dentro de ese grupo. Después dos bytes más indican el número de bytes que vienen a continuación, estos bytes son los que le indicarán la configuración correspondiente al Módulo GPS. Por último, dos bytes finales forman dos *checksum* distintos para comprobar que todo ha llegado correctamente.

2.1.2. Placa Arduino UNO

Arduino es una plataforma electrónica de código abierto basada en hardware y software fácil de usar.

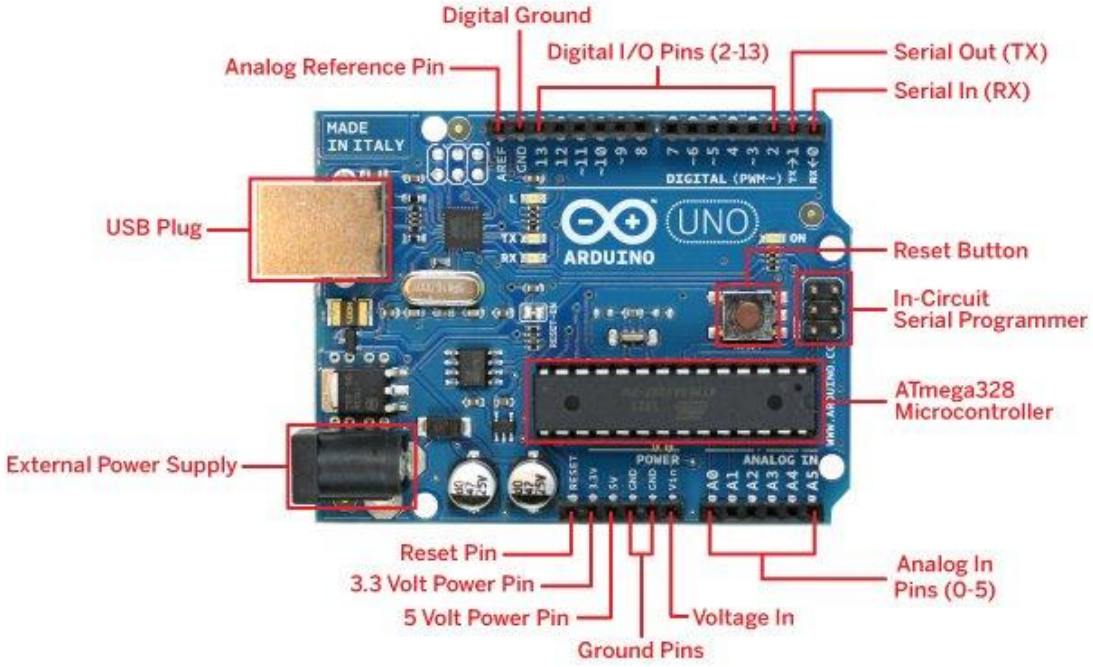


Ilustración 8. Vista superior de una placa Arduino UNO y descripción de componentes. [6]

Podemos alimentar la Placa Arduino desde el *USB Plug*, el *External Power Supply* o conectarla a una fuente de alimentación de 5 voltios a través de *5 Volt Power Pin* y *Ground Pin.* Esto facilita la independencia de una fuente de alimentación estática, permitiendo su transporte o traslado, mientras funciona, si está conectada a una batería portátil.

Para nuestro proyecto, en la primera parte de la Fase 4, experimentación, se alimentará a través de USB para poder ver por el puerto serie la información que recibe la Placa Arduino. Posteriormente, una vez que se haya desarrollado el software de control se alimentará con una batería portátil.

Además, cuenta con los pines *Serial Out (Tx)* y *Serian In (Rx)* que harán posible el intercambio de información entre el Módulo GPS y la Placa Arduino a una velocidad de 9600 Baudios gracias al protocolo UBX.

La versatilidad que nos ofrece una placa Arduino para diseñar software gracias al lenguaje base de programación que usa, C++, permite el desarrollo del programa necesario para leer la información recibida del GPS e interpretarla gracias a la librería TinyGPS [7].

Esta interpretación permite aislar el valor de tiempo hhmmss.ss para poder usarlo como elemento principal de sincronía entre los diferentes sensores usando un mismo valor para iniciar la emisión o recepción de datos.

Librería TinyGPS

Esta librería ha sido desarrollada para parsear las cadenas NMEA que se reciben por GPS y trabajar con los valores individualmente.

```
#include <TinyGPS.h>
TinyGPS gps;
unsigned long age, date, time; //Date(ddmmyy); Time(hhmmsscc)
bool newdata = false;


void setup() {
  Serial.begin(9600);
}

void loop() {
     if (Serial.available()) {
      char c = Serial.read();
      if (gps.encode(c)) {
        newdata = true;
      }
     }
     if (newdata) {
        gps.get_datetime(&date, &time, &age);
        newdata = false;
      }
}
```

Código 1. Ejemplo Librería TinyGPS.

Vemos en el ejemplo del Código 1 cómo cada vez que se recibe una cadena por el puerto serie se almacena en la variable "*c"* y se parsea con la función *"gps.encode(c)"*. Si este parseo ha tenido éxito, guardamos el valor de tiempo en la variable "*time"* con la función "*gps.getdatetime(&date, &time, &age)"* y reiniciamos el loop a la espera de una nueva cadena.

En siguientes programas se usará este código para obtener la variable "*time"* y usarla como disparador de la señal configurada.

2.1.3. Sensores del grupo INVETT

Los sensores del grupo INVETT que se utilizan para comprobar la fiabilidad de sincronismo son un iluminador infrarrojo y 2 cámaras GrassHopper3 de la marca PointGrey.

Iluminador Infrarrojo

El iluminador infrarrojo utilizado consta de dos emisores con LEDs infrarrojos, alimentados a 24V. Debido a la falta de electrónica para controlar el encendido y apagado del iluminador, será necesario diseñar un acondicionamiento previo que permita alcanzar tiempos de encendido suficientes para demostrar la fiabilidad de la sincronización con las cámaras.

Ilustración 9. Iluminador Infrarrojo del grupo INVETT.

Cámaras GrassHopper3 de PointGrey

Estas cámaras captan hasta 162 imágenes por segundo. Además de contar con un conector USB 3.0 con el que enviar rápidamente la información captada, cuentan con un conector de 8 pines que se usará para disparar las cámaras en los instantes precisos.

Ilustración 10. Cámara Grasshopper3 marca PointGrey del grupo INVETT.

Ilustración 11. Vista trasera de la cámara del grupo INVETT donde se aprecia el conector de 8 pines.

Diagram	Color	Pin	Function	Description
	Black	1	I0	Opto-isolated input (default Trigger in)
	White	2	O1	Opto-isolated output
	Red	3	IO2	Input/Output/serial transmit (TX)
	Green	4	IO3	Input/Output/serial receive (RX)
	Brown	5	GND	Ground for bi-directional IO, V_{EXT}, +3.3 V pins
	Blue	6	OPTO_GND	Ground for opto-isolated IO pins
	Orange	7	V_{EXT}	Allows the camera to be powered externally
	Yellow	8	+3.3 V	Power external circuitry up to 150 mA

Ilustración 12. Asignación de pines GPIO. [10].

Se usarán los pines 1 como *Trigger in* y 6 como GND según la Ilustración 12 para enviar desde el Módulo GPS la señal deseada. Con cada pulso, la cámara tomará una captura y la enviará al PC conectado como podemos ver en el siguiente diagrama:

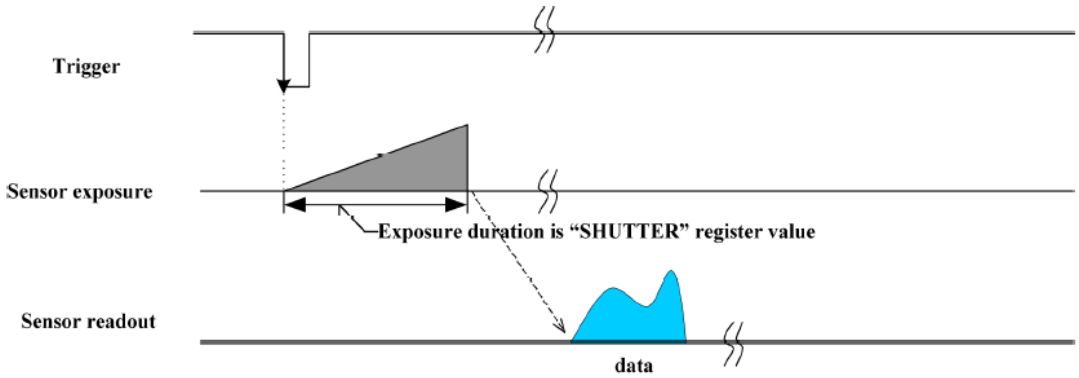


Ilustración 13. Modo de disparo externo estándar (Modo 0). [10].

Durante la configuración de la señal que se desea enviar a través del Módulo GPS se debe tener en cuenta que el tiempo de exposición de la cámara no sea superior al del periodo fijado en la señal del iluminador para evitar que se acople información de diferentes periodos.

2.2. Especificación

La Placa Arduino es la que se encargará de comandar el Módulo GPS para activar, desactivar y configurar la señal que se enviará por el pin PPS.

Para que la Placa Arduino comande el Módulo GPS se simularán los mismos comandos que manda la aplicación U-Center.

Haciendo uso del protocolo UBX, se puede simular el comando que nos interesa y configurarlo cada vez para cada situación. Este comando es UBX CFG-TP5 que permite configurar los siguientes parámetros de la señal:

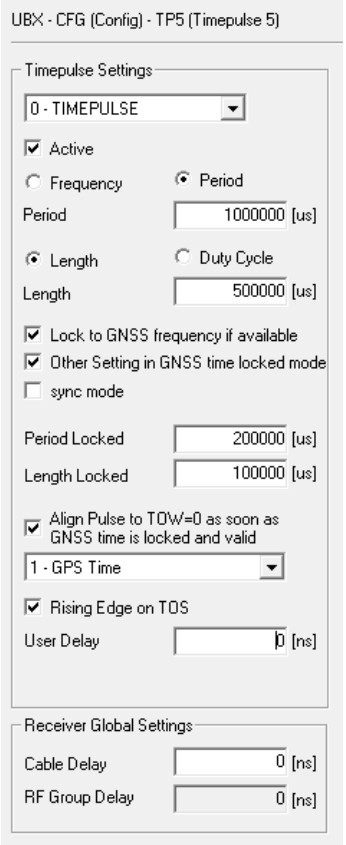


Ilustración 14. Configuración Timepulse5 en la aplicación U-Center.

El desplegable de inicio permite seleccionar entre la configuración 0-TIMEPULSE o la configuración 1-TIMEPULSE2. En este caso, sólo se trabajará con 0-TIMEPULSE.

La casilla *Active* permite la salida de la señal configurada si está marcada.

El periodo se puede representar en frecuencia [hercios] o en tiempo [microsegundos]. Si se configura en tiempo, se debe asegurar que el resto de la división por un segundo sea 0.

La longitud del pulso, al igual que el periodo, puede ser representarlo en frecuencia o tiempo.

Si se marca la casilla *Lock to GNSS frequency if available* se sincronizará la señal en cuanto reciba el valor de tiempo por GPS.

Si se marca la casilla *Other Setting in GNSS time locked mode* se puede emitir 2 señales diferentes. *Period* y *Length* sería la señal que se emite por el pin PPS cuando el GPS todavía no tiene

cobertura vía satélite y *Period Locked* y *Length Locked* sería la señal que se emite una vez que hay cobertura satélite suficiente.

Si se marca la casilla *sync mode*, una vez que hay cobertura satélite y emitimos la señal con los valores de *Period Locked* y *Length Locked*, la señal no volverá a los valores anteriores incluso si se pierde la cobertura. En caso contrario, volvería a emitir la señal anterior.

Si se marca la casilla *Align Pulse to TOW=0 as soon as GNSS time is locked and valid* el inicio de una cadena de periodos que completen un segundo estará sincronizado en cada segundo. Esta característica es la más importante para el proyecto, ya que se comprobará su fiabilidad para aplicarla a los sensores del grupo INVETT.

El siguiente desplegable permite seleccionar de qué GNSS se obtiene el tiempo con el que se sincronizará en cada segundo. En nuestro caso utilizaremos *1- GPS Time*.

Si se marca la casilla *Rising Edge on TOS* se tendrá un flanco de subida al inicio de cada periodo, en caso contrario será un flanco de bajada.

El comando o estructura del mensaje que se enviaría al Módulo GPS sigue la distribución indicada en la Ilustración 15.

Message	**CFG-TP5**				
Description	**Get/Set TimePulse Parameters**				
Firmware	Supported on u-blox 6 from firmware version 6.00 up to version 7.03.				
Type	Get/Set				
Comment	-				
Message Structure	*Header*	*ID*	*Length (Bytes)*	*Payload*	*Checksum*
	0xB5 0x62	0x06 0x31	32	*see below*	CK_A CK_B

Payload Contents:

Byte Offset	*Number Format*	*Scaling*	*Name*	*Unit*	*Description*
0	U1	-	`tpIdx`	-	Timepulse selection (0 = TIMEPULSE, 1 = TIMEPULSE2)
1	U1	-	`reserved0`	-	Reserved
2	U2	-	`reserved1`	-	Reserved
4	I2	-	`antCableDelay`	ns	Antenna cable delay
6	I2	-	`rfGroupDelay`	ns	RF group delay
8	U4	-	`freqPeriod`	Hz/us	Frequency or period time, depending on setting of bit 'isFreq'
12	U4	-	`freqPeriodLock`	Hz/us	Frequency or period time when locked to GPS time, only used if 'lockedOtherSet' is set
16	U4	1/2^-32	`pulseLenRatio`	us/-	Pulse length or duty cycle, depending on 'isLength'
20	U4	1/2^-32	`pulseLenRatioLock`	us/-	Pulse length or duty cycle when locked to GPS time, only used if 'lockedOtherSet' is set
24	I4	-	`userConfigDelay`	ns	User configurable timepulse delay
28	X4	-	`flags`	-	Configuration flags (see graphic below)

Bitfield flags

This Graphic explains the bits of `flags`

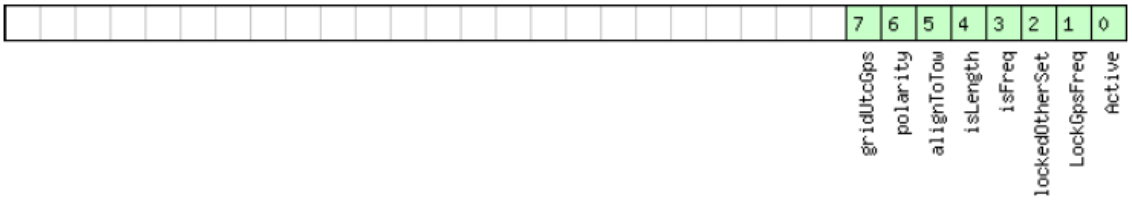


Ilustración 15. Estructura del mensaje CFG-TP5 según el protocolo UBX. [9].

El mensaje correspondiente a la Ilustración 14, sería el siguiente:

[0xB5, 0x62], [0x06, 0x31], [0x20, 0x00], [0x00], [0x01], [0x00, 0x00], [0x00, 0x00], [0x00, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x40, 0x0D, 0x03, 0x00], [0x20, 0xA1, 0x07, 0x00], [0xA0, 0x86, 0x01, 0x00], [0x00, 0x00, 0x00, 0x00], [0xF7, 0x00, 0x00, 0x00], [0x1F, 0x48]

Así, según corresponde como se indican en las ilustraciones 14 y 15:

[0xB5, 0x62]	Encabezado, siempre es el mismo.
[0x06, 0x31]	Grupo e ID, indican grupo CFG, ID TP5
[0x20, 0x00]	Número de bytes siguientes sin contar el checksum, 32 bytes
[0x00]	tpIdx, hemos seleccionado 0 - TIMEPULSE
[0x01]	Reservado
[0x00, 0x00]	Reservado

[0x00, 0x00]	El retraso por cable de antena es 0
[0x00, 0x00]	El retraso por grupo RF es 0
[0x40, 0x42, 0x0F, 0x00]	El periodo es de 1000 ms.
[0x40, 0x0D, 0x03, 0x00]	El pulso es de 500 ms.
[0x20, 0xA1, 0x07, 0x00]	El periodo con cobertura GPS es de 200 ms.
[0xA0, 0x86, 0x01, 0x00]	El pulso con cobertura GPS es de 100 ms.
[0x00, 0x00, 0x00, 0x00]	El retraso configurado por el usuario es 0
[0xF7, 0x00, 0x00, 0x00]	Estos 4 bytes contienen la siguiente información en función de la posición del bit: 0- Señal activa o no, en este caso bit0=1 1- *Lock to GNSS frequency if available*, bit1=1 2- *Other Setting in GNSS time locked mode, bit2=1* 3- Trabajar con periodo o con frecuencia, bit3=0 4- Trabajar con duración de pulso o ciclo de trabajo, bit4=1 5- *Align Pulse to TOW=0 as soon as GNSS time is locked and valid, bit5=1* 6- *Rising Edge on TOS, bit6=1* 7- Indica de dónde se obtiene el tiempo, en este caso *1- GPS Time*, bit7=1 bit8=0 bit9=0 b11110111=0xF7, correspondiente al primer byte.

[0x1F, 0x48]	Checksum A y Checksum B, se calculan como se indica en la ilustración 16.
	`CK_A = 0, CK_B = 0`
	`For(I=0;I<N;I++)`
	`{`
	`CK_A = CK_A + Buffer[I]`
	`CK_B = CK_B + CK_A`
	`}`
	Ilustración 16. Cálculo de Checksum A y Checksum B. [9].

Tabla 2. Significado y valores de la cadena UBX correspondiente a la Ilustración 13.

2.2.1. Electrónica de acondicionamiento

La finalidad de utilizar una electrónica de acondicionamiento es garantizar que la señal que emite el Módulo GPS por el pin PPS sea leía correctamente.

La tensión de salida del pin PPS es de 3.3V. Si el disparo del sensor puede funcionar directamente con 3.3V, la electrónica de acondicionamiento no es necesaria, como en el caso de las cámaras GrassHopper3.

En caso de no funcionar a 3.3V, se deberá diseñar una electrónica de acondicionamiento para que el sensor funcione correctamente con nuestra señal, como en el caso del iluminador infrarrojo.

Acondicionamiento para Iluminador infrarrojo de 24V

Antes de diseñar una electrónica de acondicionamiento para el iluminador, se realizará una pequeña prueba con un único LED conectado a una resistencia de 1 kΩ y conectarlo directamente entre el pin PPS y el pin GND del Módulo GPS.

Suponiendo una señal preestablecida de salida continua a nivel bajo, el LED permanece apagado. En cuanto configuramos una señal con periodo y pulso cualquiera con la aplicación U-Center, observamos que el LED se enciende durante el tiempo establecido de pulso y repite el proceso cada tiempo establecido de periodo.

Realizar la prueba anterior con el iluminador infrarrojo del grupo INVETT no es tan sencillo debido a la fuente de alimentación necesaria que se utiliza. Por ello, es necesario una electrónica de acondicionamiento que permita una rápida transición entre los estados ON y OFF acorde a la señal deseada. Además, debido a las limitaciones de captura de las cámaras a 162 FPS y un mínimo de 2 ms de tiempo de exposición, se debe de mantener encendido el iluminador al menos 2 ms cada periodo de 10 ms si quisiéramos conseguir 100 FPS y que las cámaras tomaran buenas capturas.

Este requerimiento hace que las pérdidas por calor sean muy altas debidas a los picos de intensidad que se producen continuamente con cada encendido y apagado del iluminador, además de las pérdidas por calor que conlleva tener tantos LEDs encendidos durante 2ms con una alta intensidad.

Para solventar este problema, se propone una electrónica de acondicionamiento que consiste en utilizar transistores MOSFET como interruptor de encendido y apagado permitiendo el paso de la corriente proveniente de la batería de 24 voltios conectada al iluminador infrarrojo.

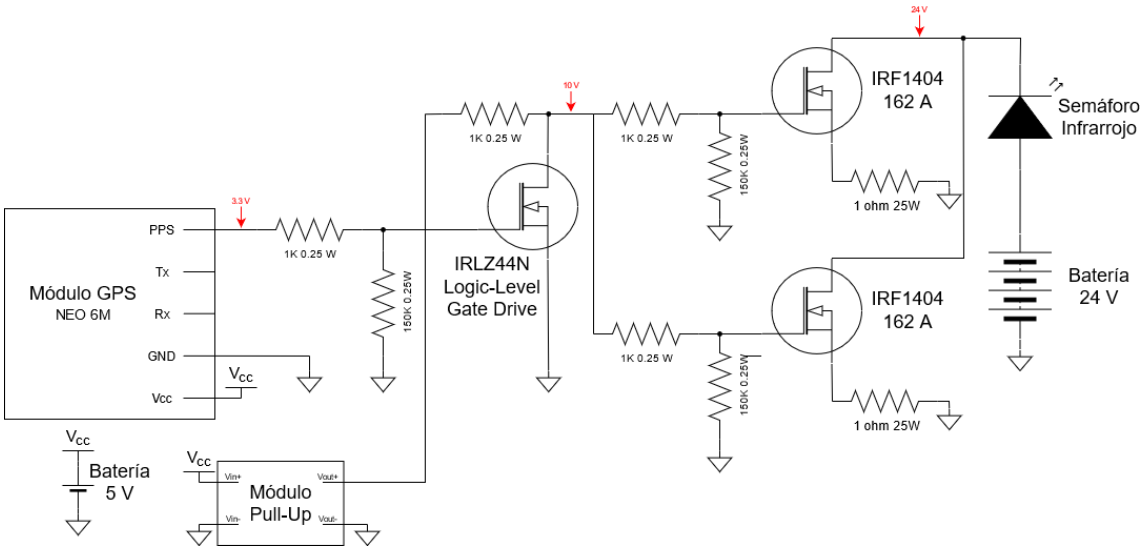


Ilustración 17. Electrónica de acondicionamiento Módulo GPS – Iluminador. [11] [12]

La electrónica de acondicionamiento para el iluminador, Ilustración 17, consta de 3 transistores utilizados como interruptores. Debido a que dos de ellos son de alta potencia, IRF1404 [12], el nivel de alimentación de la puerta debe ser de 10 V según especificaciones para funcionar correctamente como interruptor. Además, se utilizan dos para dividir la carga y una resistencia en serie para cada uno de 25 W para evitar posibles desbalances en la carga que soportan.

La señal que emite el Módulo GPS es de 3.3 V, lejos de los 10 V necesarios. Por eso, utilizamos un transistor que funciona a niveles lógicos como interruptor que deja pasar 10 V. Estos 10 V se consiguen gracias a un módulo pull-up que emite una señal constante de 10 V.

Al utilizar 2 bloques de transistores, la señal que sacamos del Módulo GPS llega invertida al iluminador como se ve en la Ilustración 18.

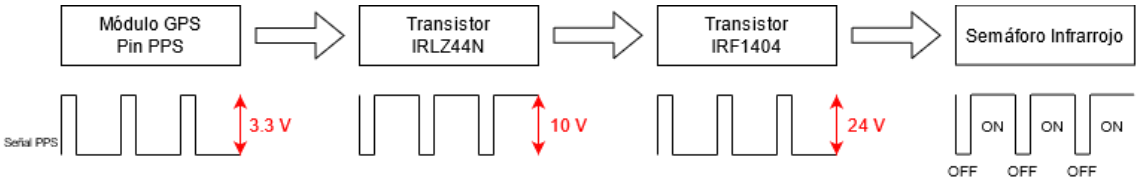


Ilustración 18. Evolución de la señal PPS.

Como se puede ver, cuando la señal PPS está en un nivel bajo, el iluminador se encendería. Por eso, es importante configurar el Módulo GPS para que en estado de reposo, es decir, mientras no emite ninguna señal, emita una señal a nivel alto constante para mantener el iluminador apagado.

2.3. Implementación

2.3.1. Programación de señales con Arduino a través del puerto serie

El software diseñado para programar una cadena UBX que le indique al Módulo GPS la señal que queremos que envíe empieza por una serie de preguntas al usuario a través del monitor serie de la aplicación de Arduino para determinar el periodo y pulso de la señal, en qué instante de tiempo quiere que se active y si lo desea en qué instante quiere que se apague. También puede enviar la señal sin necesidad de esperar a un instante de tiempo.

El programa utiliza un valor que actúa de contador para ir avanzando por las preguntas saltándose aquellas que no sean de interés en función de los valores previos marcados por el usuario.

Tras pasar por todos los apartados, se habrá generado una cadena de nombre *"message"* que cumple todos los requisitos del protocolo UBX y que enviaremos al Módulo GPS con la señal deseada. La cadena *"message"* tiene campos predefinidos, sólo se modificarán aquellos que afecten a la forma de la señal, el encabezado, el grupo, el ID y aquellos valores reservados no se modificarán.

[0xB5, 0x62], [0x06, 0x31], [0x20, 0x00], [0x00], [0x01], [0x00, 0x00], [0x00, 0x00], [0x00, 0x00], [14, 14, 14, 14], [18, 18, 18, 18], [22, 22, 22, 22], [26, 26, 26, 26], [0x00, 0x00, 0x00, 0x00], [34, 0x00, 0x00, 0x00], [CK_A, CK_B]

[0xB5, 0x62]	Valor predefinido por defecto. No se modifica.
[0x06, 0x31]	Valor predefinido por defecto. No se modifica.
[0x20, 0x00]	Valor predefinido por defecto. No se modifica.
[0x00]	Valor predefinido por defecto. No se modifica.
[0x01]	Valor predefinido por defecto. No se modifica.
[0x00, 0x00]	Valor predefinido por defecto. No se modifica.
[0x00, 0x00]	Valor predefinido por defecto. No se modifica.
[0x00, 0x00]	Valor predefinido por defecto. No se modifica.
[14, 14, 14, 14]	Valor correspondiente al periodo, modificado por el usuario.
[18, 18, 18, 18]	Valor correspondiente al periodo con cobertura GPS, modificado por el usuario.
[22, 22, 22, 22]	Valor correspondiente al pulso, modificado por el usuario.
[26, 26, 26, 26]	Valor correspondiente al pulso con cobertura GPS, modificado por el usuario.
[0x00, 0x00, 0x00, 0x00]	Valor predefinido por defecto. No se modifica.
[34, 0x00, 0x00, 0x00]	Valor predefinido por defecto. Se modifica según las preferencias de polaridad del usuario o si quiere activar o desactivar la señal.
[CK_A, CK_B]	Checksum A y B calculados por el programa una vez que tenemos todos los valores.

Tabla 3. Valores predefinidos y modificables por el usuario de la cadena UBX "message".

Librerías y variables

Con el fin de entender el significado de cada valor utilizado durante la programación, se pueden ver a continuación qué representa cada variable que aparece en el Código 2:

```
#include <TinyGPS.h>
TinyGPS gps;
unsigned long age, date, time; //Date(ddmmyy); Time(hhmmsscc)
unsigned long tiempo_encendido = (20355900 - 2000000);
unsigned long tiempo_apagado =    (20370000 - 2000000);
byte APAGADO_BAJO[] = {0xB5, 0x62, 0x06, 0x31, 0x20, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x40, 0x42, 0x0F, 0x00, 0x40, 0x42, 0x0F, 0x00, 0x40, 0x0D, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xF2, 0x00, 0x00, 0x00, 0xBC, 0xCB}; //Señal estandar, por PPS sacamos 0
byte APAGADO_ALTO[] = {0xB5, 0x62, 0x06, 0x31, 0x20, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x40, 0x42, 0x0F, 0x00, 0x40, 0x42, 0x0F, 0x00, 0x40, 0x42, 0x0F, 0x00, 0x40, 0x42, 0x0F, 0x00, 0x00, 0x00,
0x00, 0x00, 0xF7, 0x00, 0x00, 0x00, 0x93, 0x0E};// Señal estandar invertida, por PPS sacamos 1
byte protocol_OUT_ON[]  = {0xB5, 0x62, 0x06, 0x00, 0x14, 0x00, 0x01, 0x00, 0x00, 0x00, 0xD0, 0x08, 0x00,
0x00, 0x80, 0x25, 0x00, 0x00, 0x07, 0x00, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0xA2, 0xB5}; //Indicamos al
Módulo GPS que envíe las cadenas NMEA por el puerto serie de la Placa Arduino
byte protocol_OUT_OFF[] = {0xB5, 0x62, 0x06, 0x00, 0x14, 0x00, 0x01, 0x00, 0x00, 0x00, 0xD0, 0x08, 0x00,
0x00, 0x80, 0x25, 0x00, 0x00, 0x23, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xBB, 0x83}; //Indicamos al
Módulo GPS que no envíe las cadenas NMEA por el puerto serie de la Placa Arduino
bool newdata = false;
String periodo_S;
String pulso_S;
String tiempo_encendido_S;
String tiempo_apagado_S;
unsigned long periodo_UL; //us
unsigned long pulso_UL; //us
unsigned long periodo_UL_ms; //ms
unsigned long pulso_UL_ms; //ms
byte flag = "Y";
byte CK_A = 0;
byte CK_B = 0;
byte message[] = {0xB5, 0x62, 0x06, 0x31, 0x20, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
14, 14, 14, 14, 18, 18, 18, 18, 22, 22, 22, 22, 26, 26, 26, 26, 0x00, 0x00, 0x00, 0x00, 34, 0x00, 0x00, 0x00,
CK_A, CK_B}; //Delay y Reserved por defecto
//byte message[] = {0xB5, 0x62, 0x06, 0x31, 0x20, 0x00, 0x00, 0x07, 0x40, 0x00, 0x32, 0x00, 0x40, 0x00,
14, 14, 14, 14, 0x40, 0x42, 0x0F, 0x00, 22, 22, 22, 22, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
34, 0x00, 0x00, 0x00, CK_A, CK_B}; //Con delays y Reserved0 y 1 cambiados
int cont1 = 0;
int flag_encendido = 0;
int flag_apagado = 0;
int flag_verano = 0;
int flag_flanco = 0;
```

Código 2. Librerías y Variables del programa para control del Módulo GPS.

Sólo se incluye la librería TinyGPS puesto que no requerimos de ninguna más.

Las variables podemos verlas en la Tabla 4:

unsigned long age, date, time	Valor donde se guardarán los datos recibidos de las cadenas GNSS que envía el Módulo GPS y son interpretadas gracias a la librería TinyGPS
unsigned long tiempo_encendido	Valor ya transformado del tiempo de encendido de la señal
unsigned long tiempo_apagado	Valor ya transformado del tiempo de apagado de la señal
byte APAGADO_BAJO[]	Cadena UBX que asegura un nivel bajo constante, 0V, por el pin PPS
byte APAGADO_ALTO[]	Cadena UBX que asegura un nivel alto constante, 3.3V, por el pin PPS
byte protocol_OUT_ON[]	Cadena UBX que permite al Módulo GPS enviar datos.
byte protocol_OUT_OFF[]	Cadena UBX que impide al Módulo GPS enviar datos
bool newdata	Valor queindica si se ha recibido algún dato del Módulo GPS, false por defecto, true si se reciben valores
String period_S	Valor leído por el puerto serie para el periodo
String pulso_S	Valor leído por el puerto serie para el pulso
String tiempo_encendido_S	Valor leído por el puerto serie como tiempo de encendido de la señal

String tiempo_apagado_S	Valor leído por el puerto serie como tiempo de apagado de la señal
unsigned long periodo_UL	Valor ya transformado del periodo
unsigned long pulso_UL	Valor ya transformado del pulso
unsigned long periodo_UL_ms	Valor ya transformado del pulso en ms
unsigned long pulso_UL_ms	Valor ya transformado del pulso en ms
byte flag	Valor donde se va guardando la respuesta Yes/No por parte del usuario.
byte CK_A	Byte correspondiente al checksum A
byte CK_B	Byte correspondiente al checksum B
byte message []	Cadena UBX con valores predefinidos para enviar al Módulo GPS una vez que esté completamente configurada
int cont1	Contador de avance por el programa
int flag_encendido	Accionador de tiempo de encendido, 0=NO 1=SI
int flag_apagado	Accionador de tiempo de apagado, 0=NO 1=SI
int flag_verano	Horario de Verano, 0=NO 1=SI
int flag_flanco	Polaridad, 0=flanco subida 1= flanco bajada 2=apagado

Tabla 4. Descripción de las variables del programa.

Setup inicial

La configuración inicial primero permitirá una correcta comunicación con el Módulo GPS a 9600 baudios, además, desactiva el protocolo NMEA y UBX de salida para que deje el puerto serie de la Placa Arduino libre para programar la cadena UBX como podemos ver en la Ilustración 19.

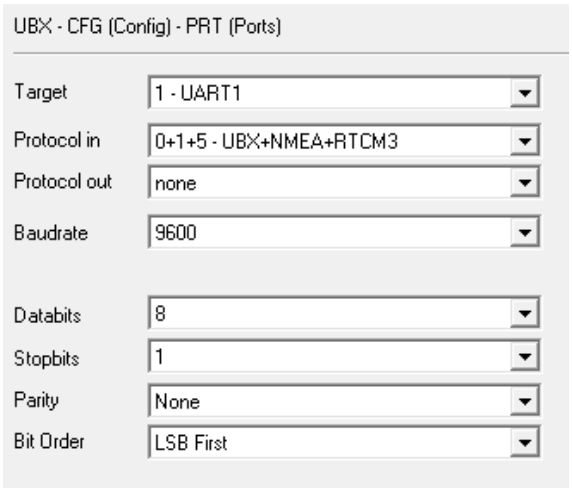


Ilustración 19. Configuración Ports en la aplicación U-Center.

Podemos observar cómo el apartado *Protocol out* no muestra ningún protocolo de salida, por lo que de esta manera nos aseguramos de que el puerto serie no quede ocupado por el Módulo GPS.

A continuación, podemos ver el Código 3 correspondiente a esta configuración inicial:

```
void setup() {
  Serial.begin(9600);
  delay(200);
  Serial.write(protocol_OUT_OFF, sizeof(protocol_OUT_OFF));
  delay(200);
  Serial.write(APAGADO_BAJO, sizeof(APAGADO_BAJO));
  delay(200);
}
```

Código 3. Setup Inicial con señal de apagado a nivel bajo.

La cadena UBX *"protocol_OUT_OFF"* es la siguiente:

[0xB5, 0x62], [0x06, 0x00], [0x14, 0x00], [0x01], [0x00], [0x00, 0x00], [0xD0, 0x08, 0x00, 0x00], [0x80, 0x25, 0x00, 0x00], [0x23, 0x00], [0x00, 0x00], [0x00, 0x00], [0x00, 0x00], [0xBB, 0x83]

Como se vio en la Ilustración7, observamos los campos correspondientes al protocolo UBX, en este caso UBX CFG-PRT. Se puede comprobar el significado de cada campo en la página 129 del Manual donde se detalla su funcionamiento [9].

Al enviar esta cadena, estaremos indicando al Módulo GPS que cese de enviar información y sólo reciba.

La cadena UBX *"APAGADO_BAJO"* es la siguiente:

[0xB5, 0x62], [0x06, 0x31], [0x20, 0x00], [0x00], [0x01], [0x00, 0x00], [0x00, 0x00], [0x00, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x40, 0x0D, 0x03, 0x00], [0x00, 0x00, 0x00, 0x00], [0x00, 0x00, 0x00, 0x00], [0xF2, 0x00, 0x00, 0x00], [0xBC, 0xCB]

En este caso, la estructura es la misma que la de la Ilustración 14, UBX CFG-TP5. Se puede comprobar el significado de cada campo en la página 146 del Manual donde se detalla su funcionamiento [9].

Al enviar esta cadena estamos desactivando la señal, lo que permite enviar un valor constante de salida a nivel bajo, 0 voltios.

No obstante, para aquellos sensores cuyo nivel de entrada debe ser inverso, enviar un nivel bajo no sirve a no ser que una electrónica de acondicionamiento invierta la señal. Para estos casos y evitar los costos de esta electrónica de acondicionamiento, se genera un segundo programa que difiere del original en la cadena UBX que desactiva la señal.

```
void setup() {
  Serial.begin(9600);
  delay(200);
  Serial.write(protocol_OUT_OFF, sizeof(protocol_OUT_OFF));
  delay(200);
  Serial.write(APAGADO_ALTO, sizeof(APAGADO_ALTO));
  delay(200);
}
```

Código 4. Setup Inicial con señal de apagado a nivel alto.

En este caso, la cadena UBX *"APAGADO_ALTO"* es la siguiente:

[0xB5, 0x62], [0x06, 0x31], [0x20, 0x00], [0x00], [0x01], [0x00, 0x00], [0x00, 0x00], [0x00, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x40, 0x42, 0x0F, 0x00], [0x00, 0x00, 0x00, 0x00], [0xF7, 0x00, 0x00, 0x00], [0xBC, 0xCB]

De esta manera, en la señal que estamos generando tanto el periodo y como el pulso tienen la misma longitud, por lo que siempre enviará un nivel alto, 3.3 voltios, por la salida del pin PPS del Módulo GPS.

Obtener el periodo de la señal

Aunque la configuración en la aplicación U-Center nos permite introducir este valor en hercios o en microsegundos, nosotros trabajaremos con milisegundos exclusivamente. Por eso, lo primero que se le pide al usuario cuando abre el monitor serie de la aplicación de Arduino es introducir un valor de periodo en milisegundos como vemos en el Código 5:

```
while (cont1 == 0) {
  Serial.println();
  Serial.println("Introduzca el periodo del pulso en ms.");
  while (Serial.available() == 0) {} //Comprobamos cada medio segundo si hay algún valor introducido por el usuario
  delay(500);

  while (Serial.available() > 0) {
    periodo_S = Serial.readString(); //Guardamos el valor introducido por el usuario en ms en un valor tipo String
    periodo_UL = periodo_S.toInt() * 1000; //Transformamos el valor guardado a microsegundos al tipo unsigned long
    periodo_UL_ms = periodo_S.toInt(); //Transformamos el valor guardado al tipo unsigned long
  }

  if (1000000 % periodo_UL == 0) { //Comprobamos si al dividir un segundo por el valor introducido el resto es 0.
    Serial.print("Periodo : "); //Mostramos el valor introducido por el usuario
    Serial.print(periodo_UL_ms, DEC);
    Serial.println(" ms.");
    Serial.println();
    //Guardamos el valor introducido por el usuario en el campo correspondiente al periodo cuando no tenemos cobertura GPS
    *((unsigned long *)(message + 14)) = periodo_UL;
    //Guardamos el valor introducido por el usuario en el campo correspondiente al periodo cuando tenemos cobertura GPS
    *((unsigned long *)(message + 18)) = periodo_UL;
    cont1 = 1;
  }

  else { //En caso de que el resto no sea 0.
    Serial.println("El periodo del pulso debe ser una división entera de 1 segundo, introduzca otro valor.");
  }
}
```

Código 5. Código para obtener el periodo de la señal por parte del usuario y comprobaciones.

Una vez que tenemos el valor introducido por el usuario, se debe transformar la variable al tipo Unsigned Long para poder trabajar con ella según el protocolo UBX.

Se comprueba después si el valor cumple los requisitos, en este caso, que al dividir 1 segundo el resto sea 0, asegurando así que el valor que recibe el Módulo GPS es correcto y no es rechazado.

Si la comprobación es exitosa, se empieza a configurar la cadena *"message"* guardando el valor introducido por el usuario en la posición 14 y 18, correspondientes al periodo y al periodo con cobertura GPS respectivamente.

Por último, se actualiza el contador *"cont1"* a 1 para continuar con el proceso.

Obtener la longitud del pulso del periodo

Igual que en el apartado anterior, trabajaremos con milisegundos exclusivamente. Ahora se le pide al usuario introducir el valor del pulso deseado en milisegundos como vemos en el Código 6:

```
while (cont1 == 1) {
  Serial.println("Introduzca la longitud del pulso en ms.");
  while (Serial.available() == 0) {} //Comprobamos cada medio segundo si hay algún valor introducido por el usuario
  delay(500);

  while (Serial.available() > 0) {
    pulso_S = Serial.readString(); //Guardamos el valor introducido por el usuario en ms en un valor tipo String
    pulso_UL = pulso_S.toInt() * 1000; //Transformamos el valor guardado a microsegundos al tipo unsigned long
    pulso_UL_ms = pulso_S.toInt(); //Transformamos el valor guardado al tipo unsigned long
  }

  if (pulso_UL <= (periodo_UL)) { //Comprobamos que el pulso no es mayor que el periodo
    Serial.print("Longitud del pulso : "); //Mostramos el valor introducido por el usuario
    Serial.print(pulso_UL_ms, DEC);
    Serial.println(" ms.");
    Serial.println();
    //Guardamos el valor introducido por el usuario en el campo correspondiente al pulso cuando no tenemos cobertura GPS
    *((unsigned long *)(message + 22)) = pulso_UL;
    //Guardamos el valor introducido por el usuario en el campo correspondiente al pulso cuando tenemos cobertura GPS
    *((unsigned long *)(message + 26)) = pulso_UL;
    cont1 = 2;
  }

  else { //En caso de que el pulso sea mayor que el periodo establecido
    Serial.println("La longitud del pulso no es menor que el periodo, introduzca otro valor.");
  }
}
```

Código 6. Código para obtener el pulso de la señal por parte del usuario y comprobaciones.

Una vez que el usuario introduce un valor, se transforma la variable al tipo Unsigned Long para poder trabajar con ella según el protocolo UBX.

Se comprueba después si el valor cumple los requisitos, en este caso, que el valor del pulso sea igual o inferior al del periodo fijado.

Si la comprobación es exitosa, configuramos la cadena *"message"* guardando el valor introducido por el usuario en la posición 22 y 26, correspondientes al pulso y al pulso con cobertura GPS respectivamente.

Por último, se actualiza el contador *"cont1"* a 2 para continuar con el proceso.

Determinar la polaridad/flanco de la señal

Se preguntará al usuario si desea flanco de subida o bajada, aunque se ha incorporado una tercera opción, oculta, que determinaría que la señal está desactivada, accesible introduciendo el valor "d" o "D", tal como se ve en el Código 7.

```
while (cont1 == 2) {
  Serial.println("¿Desea flanco de subida? (Y/N) ");
  while (Serial.available() == 0) {}//Comprobamos cada medio segundo si hay algún valor introducido por el usuario
  delay(500);
  while (Serial.available() > 0) {
    flag = Serial.read();
  }
  //Para desactivado
  if (flag == 100 or flag == 68) { //Opción oculta al usuario, accesible con el valor "d" o "D"
    Serial.println("Flanco desactivado.");
    *((byte *)(message + 34)) = 0xF2; //Guardamos la configuración del usuario en el campo correspondiente a señal desactivada
    flag_flanco = 2; //Valor que nos indica que la señal tiene que estar desactivada
    cont1 = 3;
  }

  //Para YES
  else if (flag == 121 or flag == 89) { //Si el usuario ingresa "y" o "Y", se configurará flanco de subida
    Serial.println("Flanco de subida seleccionado.");
    *((byte *)(message + 34)) = 0xF7; //Guardamos la configuración del usuario en el campo correspondiente a flanco de subida
    flag_flanco = 0; //Valor que nos indica que el flanco es de subida
    cont1 = 3;
  }

  //Para NO
  else if (flag == 110 or flag == 78) { //Si el usuario ingresa "n" o "N", se configurará flanco de bajada
    Serial.println("Flanco de bajada seleccionado.");
    *((byte *)(message + 34)) = 0xB7; //Guardamos la configuración del usuario en el campo correspondiente a flanco de bajada
    flag_flanco = 1; //Valor que nos indica que el flanco es de bajada
    cont1 = 3;
  }

  else {
    Serial.println("Opción no contemplada, pruebe con y ó n.");
  }
}
```

Código 7. Código para determinar el flanco de la señal o si ésta debe estar activada.

Una vez que el usuario haya dado una respuesta válida, se determinará si se quiere flanco de subida, de bajada o se quiere una señal desactivada y se guardará el valor correspondiente a esta configuración en la posición 34 de la cadena UBX *"message"*.

Por último, se incrementará el contador *"cont1"* a 3 para avanzar al siguiente apartado.

Calcular el Checksum

Este tramo del programa no requiere intervención por parte del usuario, puesto que es automático una vez que el usuario introduce todos los datos oportunos. Igual que vimos en la Ilustración 16, hemos desarrollado el cálculo del Checksum según el Código 8 para que guarde los valores correspondientes en las posiciones 38 y 39 de nuestra cadena UBX *"message"*.

```
while (cont1 == 3) {
  for (int I = 2; I < (sizeof(message) - 2); I++) {
    CK_A = CK_A + message[I];
    CK_B = CK_B + CK_A;
  }
  message[38] = CK_A;
  message[39] = CK_B;
  CK_A = 0;
  CK_B = 0;
  cont1 = 4;
}
```

Código 8. Cálculo del checksum correspondiente a la cadena UBX "message".

```
while (cont1 == 3) {
  for (int I = 2; I < (sizeof(message) - 2); I++) {
    CK_A = CK_A + message[I];
    CK_B = CK_B + CK_A;
  }
  message[38] = CK_A;
  message[39] = CK_B;
  CK_A = 0;
  CK_B = 0;
  cont1 = 4;
}
```

```
CK_A = 0, CK_B = 0
For(I=0;I<N;I++)
{
    CK_A = CK_A + Buffer[I]
    CK_B = CK_B + CK_A
}
```

Ilustración 20. Comparación Checksum.

Con esta última interacción la cadena UBX *"message"* está completamente configurada.

Accionador para un tiempo de encendido

Para comprobar la fiabilidad de la sincronización de las señales en distintos sensores cada uno con su propio conjunto de Placa Arduino, Módulo GPS y electrónica de acondicionamiento correspondiente, debemos establecer un mismo tiempo de encendido para que todos los sensores empiecen a funcionar en el mismo instante. Para lograr este objetivo se desarrolla el Código 9.

```
while (cont1 == 4) {
  Serial.println("¿Desea establecer un tiempo de encendido? (Y/N) ");
  while (Serial.available() == 0) {} //Comprobamos cada medio segundo si hay algún valor introducido por el usuario
  delay(500);
  while (Serial.available() > 0) {
    flag = Serial.read();
  }
  //Para YES
  if (flag == 121 or flag == 89) { //Si el usuario ingresa "y" o "Y", se pedirá al usuario un tiempo para el encendido
    Serial.println("Establezca la hora de encendido en formato hhmmss:");
    while (Serial.available() == 0) {}
    delay(500);
    while (Serial.available() > 0) {
      tiempo_encendido_S = Serial.readString();
    }
    Serial.println("¿Horario de verano? (Y/N) "); //Debemos establecer correctamente la hora para UTC
    while (Serial.available() == 0) {}
    delay(500);
    while (Serial.available() > 0) {
      flag = Serial.read();
    }
    //Para YES
    if (flag == 121 or flag == 89) { //Si horario de verano afirmativo, son 2 horas menos para UTC
      flag_verano = 1;
      flag_encendido = 1;
      tiempo_encendido = (tiempo_encendido_S.toInt() * 100) - 2000000;
      cont1 = 5;
    }
    //Para NO
    else if (flag == 110 or flag == 78) { //Si horario de verano negativo, es 1 hora menos para UTC
      flag_verano = 0;
      flag_encendido = 1;
      tiempo_encendido = (tiempo_encendido_S.toInt() * 100) - 1000000;
      cont1 = 5;
    }
    else {
      Serial.println("Opción no contemplada, pruebe con y ó n.");
    }
  }

  //Para NO
  else if (flag == 110 or flag == 78) { //Si no se quiere un tiempo para el encendido, se envía la cadena UBX al Módulo GPS
    Serial.write(message, sizeof(message));
    delay(100);
    cont1 = 6;
  }

  else {
    Serial.println("Opción no contemplada, pruebe con y ó n.");
  }
}
```

Código 9. Código para determinar el tiempo de encendido en UTC.

En caso de que no sea necesario un tiempo de encendido, se envía la cadena UBX completamente configurada con las especificaciones del usuario al Módulo GPS y se evita la opción donde nos preguntan por un tiempo de apagado, saltando directamente al resumen de la señal configurada.

En caso contrario, se pide al usuario introducir la hora en formato hhmmss en la que desea empezar a enviar la señal configurada en la cadena UBX *"message"*. Se debe tener en cuenta que el programa contempla su uso dentro de territorio cuya zona horaria sea GMT +1:00h con posibilidad de horario de verano.

Si no fuese el caso, habría que modificar el programa para su correcto funcionamiento fuera de esta zona horaria, puesto que el Módulo GPS trabaja con la hora UTC (Tiempo universal coordinado).

Una vez indicado el tiempo de encendido y si estamos en horario de verano, se establece el valor de "*tiempo_encendido"* para posteriormente enviar la señal en el instante deseado y se establece *"flag_encendido"* como 1 para indicar más adelante que hay un tiempo de encendido fijado.

Accionador para un tiempo de apagado

Si se ha establecido un tiempo de encendido, el valor de *"cont1"* será 5, permitiendo la configuración de un tiempo de apagado, en caso contrario, su valor será 6 y no podremos configurar un tiempo de apagado. Aunque se establezca un tiempo de encendido, no se tiene que configurar uno de apagado obligatoriamente.

```
while (cont1 == 5) {
  Serial.println("¿Desea establecer un tiempo de apagado? (Y/N) ");
  while (Serial.available() == 0) {} //Comprobamos cada medio segundo si hay algún valor introducido por el usuario
  delay(500);
  while (Serial.available() > 0) {
    flag = Serial.read();
  }
  //Para YES
  if (flag == 121 or flag == 89) { //Si el usuario ingresa "y" o "Y", se pedirá al usuario un tiempo para el apagado
    Serial.println("Establezca la hora de apagado en formato hhmmss:");
    while (Serial.available() == 0) {}
    delay(500);
    while (Serial.available() > 0) {
      tiempo_apagado_S = Serial.readString();
    }
    if (flag_verano == 0) {
      tiempo_apagado = (tiempo_apagado_S.toInt() * 100) - 1000000;
      flag_apagado = 1;
      cont1 = 6;
    }
    //Para NO
    else if (flag_verano == 1) {
      tiempo_apagado = (tiempo_apagado_S.toInt() * 100) - 2000000;
      flag_apagado = 1;
      cont1 = 6;
    }
  }

  //Para NO
  else if (flag == 110 or flag == 78) { //Si el usuario ingresa "n" o "N", no se configurará un tiempo para el apagado
    flag_apagado = 0;
    cont1 = 6;
  }

  else {
    Serial.println("Opción no contemplada, pruebe con y ó n.");
  }
}
```

Código 10. Código para determinar el tiempo de apagado en UTC.

Al igual que en el Código 9, se calcula el correspondiente tiempo en UTC en función de si es horario de verano.

En caso de que no se quiera un tiempo de apagado, se determina el valor de "*flag_apagado"* como 0 para indicar más adelante que no se desea tiempo de apagado.

Información de la señal obtenida

Una vez que se han establecido todas las propiedades de la señal deseada y los tiempos de encendido y apagado, se muestra un resumen de los datos introducidos.

```
while (cont1 == 6) {
  Serial.println();
  Serial.println("-----------------------------------------------------------------------------------------------");
  Serial.print("Periodo: ");
  Serial.print(periodo_UL_ms, DEC);
  Serial.println(" ms.");
  Serial.print("Pulso: ");
  Serial.print(pulso_UL_ms, DEC);
  Serial.println(" ms.");
  if (flag_flanco == 0) {
    Serial.println("Flanco de subida.");
  }
  else if (flag_flanco == 1) {
    Serial.println("Flanco de bajada.");
  }
  else if (flag_flanco == 2) {
    Serial.println("Flanco desactivado.");
  }

  if (flag_encendido == 1) {
    Serial.print("Accionador de encendido activado para las ");
    Serial.print(tiempo_encendido_S.toInt(), DEC);
    Serial.println(" horas.");
  }
  else  {
    Serial.println("Accionador de encendido desactivado.");
  }
  if (flag_apagado == 1) {
    Serial.print("Accionador de apagado activado para las ");
    Serial.print(tiempo_apagado_S.toInt(), DEC);
    Serial.println(" horas.");
  }
  else  {
    Serial.println("Accionador de apagado desactivado.");
  }
  if (flag_verano == 1 and flag_encendido == 1) {
    Serial.println("Horario de verano.");
  }
  if (flag_verano == 0 and flag_encendido == 1) {
    Serial.println("Horario de invierno.");
  }
```

Código 11. Código que resume todos los datos introducidos.

De esta manera el usuario puede comprobar que los datos que ha introducido son acordes a la señal y los tiempos correspondientes que se requerían.

En caso contrario, como se indicará más adelante al usuario, se debe cerrar y abrir de nuevo el monitor serie de la aplicación de Arduino si se desea volver a programar la señal y sus condiciones de disparo.

Cadena UBX

Lo último que se muestra como resumen es la cadena UBX *"message"* correspondiente para poder comprobar, en caso de que así se quiera, que la señal corresponde con los valores establecidos durante la programación.

```
48   Serial.println();
49   Serial.println("Cadena UBX en código hexadecimal correspondiente a la señal deseada:");
50   Serial.print("{");
51   for (int i = 0; i < 40; i++) {
52     Serial.print("0x");
53     if (message[i] < 16) { //si el byte es menor de 16 ó 0x10 (del 0x0 al 0xF) pone un 0 delante.
54       Serial.print("0");
55     }
56     Serial.print(message[i], HEX);
57     if (i < 39) {
58       Serial.print(", ");
59     }
60     else {
61       Serial.println("}");
62       Serial.println("----------------------------------------------------------------------------------------------");
63     }
64   }
65   cont1 = 7;
66 }
```

Código 12. Código que muestra la cadena UBX "message" tras su configuración.

Después del resumen, se pasará a la última estructura de código correspondiente al control del envío de la señal incrementando el valor de *"cont1"* a 7.

Enviar la cadena UBX al Módulo GPS

Primero se comprueban las especificaciones del usuario para los tiempos de encendido y de apagado. Puede haber 3 casos, el primero que se determine tiempo de encendido y de apagado, el segundo caso que sólo se determine tiempo de encendido y por último que no se determine ningún tiempo.

En función del caso, podemos ver que el valor de "*cont1"* varía según corresponda ejecutar una estructura final u otra.

```
1  //Si hemos terminado la configuración de la señal y se ha determinado un tiempo de encendido
2  if (cont1 == 7 and flag_encendido == 1) {
3    Serial.write(APAGADO_BAJO, sizeof(APAGADO_BAJO));
4    delay(100);
5    Serial.write(protocol_OUT_ON, sizeof(protocol_OUT_ON));
6    delay(100);
7    //Actualizamos la variable cont1 a 8 para ejecutar el bucle correspondiente con tiempo de apagado
8    if (flag_apagado == 1) {
9      cont1 = 8;
10     Serial.println("Para configurar una nueva señal cierre y vuelva a abrir el monitor serie.");
11   }
12   //Actualizamos la variable cont1 a 9 para ejecutar el bucle correspondiente sin tiempo de apagado
13   else {
14     cont1 = 9;
15     Serial.println("Para configurar una nueva señal cierre y vuelva a abrir el monitor serie.");
16   }
17 }
18 //Si hemos terminado la configuración de la señal pero no hemos determinado ningún tiempo de encendido
19 else {
20   Serial.println("Para configurar una nueva señal cierre y vuelva a abrir el monitor serie.");
21   while (1) {}
22 }
```

Código 13. Código de selección de estructura para el control de los tiempos de encendido y apagado.

Según el Código 13, si se determina que queremos un tiempo de encendido y uno de apagado, primero se activa el protocolo NMEA de salida para que el Módulo GPS pueda enviar información a la Placa Arduino. Y en función de si hemos configurado un tiempo de apagado, la variable *"cont1"* toma el valor de 8 o 9 para activar el bucle correspondiente como se ve en el Código 14.

En caso de que no se hubiera configurado ningún tiempo de encendido, la cadena UBX *"message"* ya se habría enviado y se entraría en un bucle infinito permitiendo al Módulo GPS emitir la señal deseada hasta que se volviera a abrir el monitor serie de la aplicación de Arduino para empezar una nueva configuración.

```
//Bucle de control para tiempo de encendido y tiempo de apagado
while (cont1 == 8) {
  if (Serial.available()) {
    char c = Serial.read();
    if (gps.encode(c)) {
      newdata = true;
    }
  }
  if (newdata) {
    gps.get_datetime(&date, &time, &age);
    if (time == tiempo_encendido) { //Se envía la cadena UBX "message" para tiempo de encendido
      Serial.write(message, sizeof(message));
      delay(100);
    }
    if (time == tiempo_apagado) { //Se envía la cadena UBX "APAGADO" correspondiente para tiempo de apagado
      Serial.write(APAGADO_BAJO, sizeof(APAGADO_BAJO));
      delay(100);
      Serial.write(protocol_OUT_OFF, sizeof(protocol_OUT_OFF)); //Se cesa el envío de datos desde el Módulo GPS
      delay(100);

    }
    newdata = false;
  }
}
//Bucle de control sólo para tiempo de encendido.
while (cont1 == 9) {
  if (Serial.available()) {
    char c = Serial.read();
    if (gps.encode(c)) {
      newdata = true;
    }
  }
  if (newdata) {
    gps.get_datetime(&date, &time, &age);
    if (time == tiempo_encendido) { //Se envía la cadena UBX "message" para tiempo de encendido
      Serial.write(message, sizeof(message));
      delay(100);
    }
  }
  newdata = false;
}
```

Código 14. Estructura de control de los tiempos de encendido y apagado.

Si no fuera ese caso, se utiliza la variable *"time"* como comparador para enviar la cadena UBX *"message"* cuando coincida con el tiempo de encendido programado según muestra el Código 14. En función de si se programó un tiempo de apagado, se sigue comparando la variable *"time"* hasta que coincida con el tiempo de apagado para enviar la cadena UBX APAGADO correspondiente.

2.3.2. Comportamiento del Módulo GPS en función del sensor

Ya que será la Placa Arduino y el software programado en ella quien se encargue de comandar el Módulo GPS enviando las correspondientes cadenas UBX tal y como se ha visto en el apartado anterior "Programación de señales con Arduino a través del del puerto serie", la configuración se centrará en cómo se quiere que se encienda por primera vez y cómo funcionará mientras espera instrucciones por parte de la Placa Arduino.

Para ello, se hará uso del comando UBX CFG-CFG que permite guardar en la memoria no volátil del Módulo GPS una configuración predeterminada para cada vez que se enciende. Haciendo uso

de la aplicación U-Center como vemos en la Ilustración 21, podemos guardar el estado actual del módulo con todas las configuraciones que nos interesan.

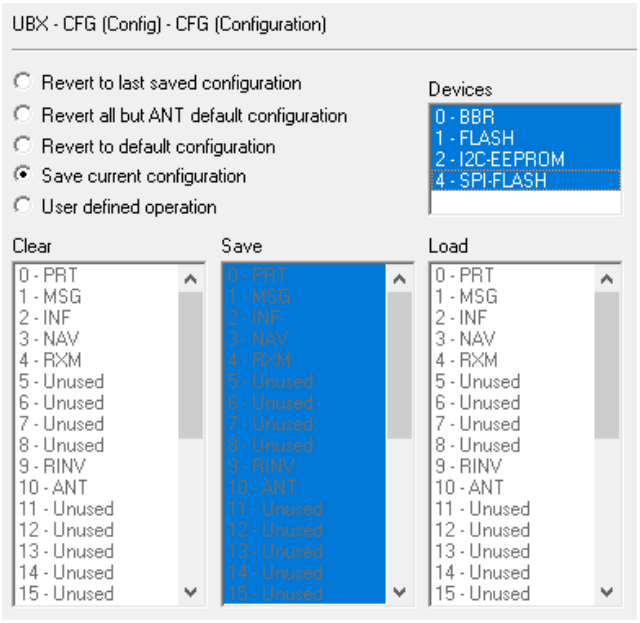


Ilustración 21. Configuración "Configuración" en la aplicación U-Center.

UBX CFG-CFG

Este comando permite guardar la configuración actual del Módulo GPS en la memoria no volátil, lo que permite establecer las condiciones de arranque del módulo desde el primer momento que reciba alimentación por el puerto USB o UART.

Para ello, según la Ilustración 21, debemos seleccionar la opción *Save current configuration* marcar todos los dispositivos y en la columna *Save* seleccionar todos los comandos, aunque muchos de ellos no se tocarán.

UBX CFG-PRT

Este comando establece los protocolos de entrada y salida del Módulo GPS, el puerto escogido para la comunicación y la velocidad a la que se comunicará.

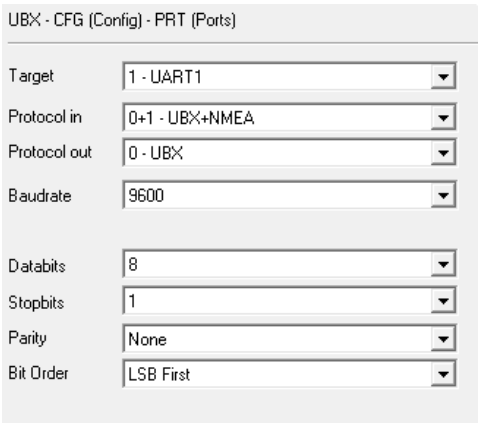


Ilustración 22. Configuración Ports predeterminada para el encendido del Módulo GPS.

En este caso se configurará el puerto UART1 para la comunicación, los protocolos de entrada UBX y NMEA activos y el protocolo de salida UBX activo, aunque este último no aportará nada para el programa final, sí puede resultar útil durante la fase de experimentación para ver las cadenas UBX que manda el Módulo GPS de confirmación.

Se establece la velocidad de comunicación a 9600 baudios y los paquetes de datos que enviaremos constarán de 8 bits/1byte, con un bit de stop y se seguirá la regla del bit menos significativo primero (LSB). De esta manera, se podrá establecer una comunicación entre la Placa Arduino y el Módulo GPS cuando generemos las cadenas UBX correspondientes.

UBX CFG-TP5

En este comando se determina la configuración del pin PPS que queremos tener precargado.

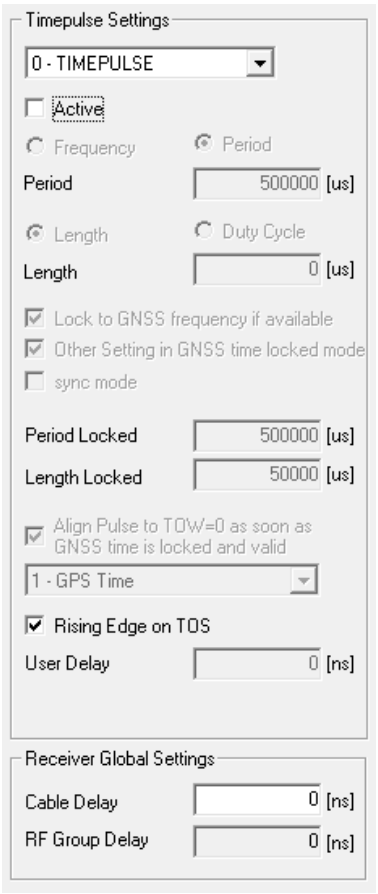


Ilustración 23. Configuración TP5 para el encendido del Módulo GPS con señal constante a nivel bajo.

Se usará la señal *0- TIMEPULSE* y estará desactivada por defecto, de esta manera se evita enviar pulsos hasta que realmente el usuario quiera configurar la señal deseada a través del monitor serie de la aplicación de Arduino.

Este comando es importante configurarlo en función del sensor que se vaya a utilizar. En el caso de la Ilustración 23 vemos que la señal que se enviará en cuanto el Módulo GPS se encienda es una señal nula, continua a nivel bajo. Si se quisiese enviar una señal constante a nivel alto desde el inicio porque la entrada del sensor está invertida, se debe preconfigurar el comando UBX CFG-TP5 como alguna de las maneras mostradas en la ilustración 24.

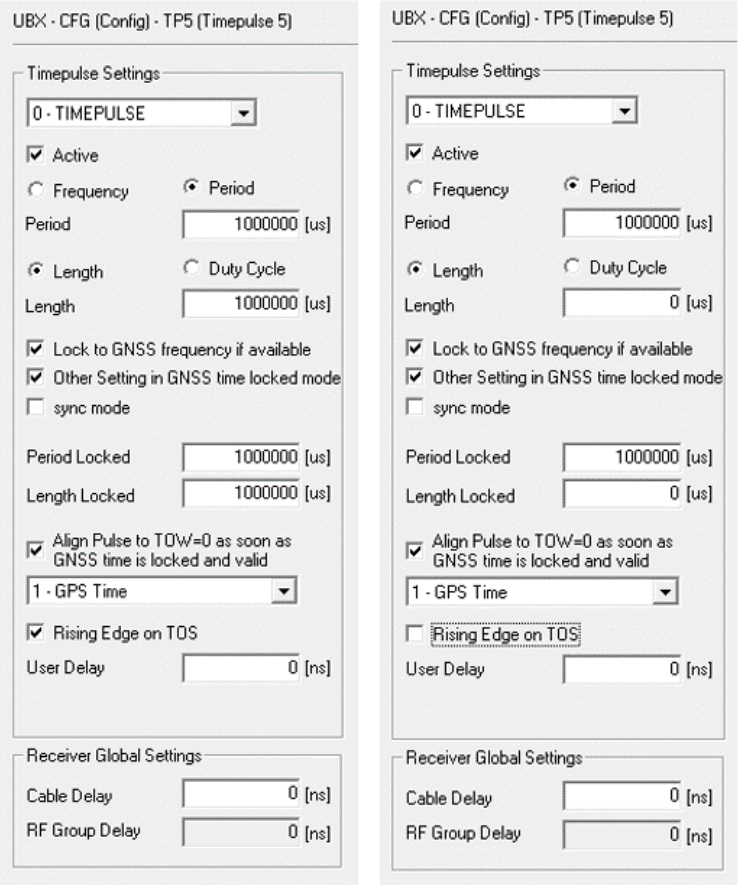


Ilustración 24. Configuración TP5 para el encendido del Módulo GPS con señal constante a nivel alto.

Con los sensores del grupo INVETT de los que se disponen, las cámaras Grasshopper3 deberían de tener una entrada constante a nivel bajo, ya que se configuran para que se activen por flanco de subida. En cambio, con el iluminador infrarrojo, aunque los LEDs emiten a nivel alto, debido a la electrónica de acondicionamiento, la señal llega invertida, por lo que un nivel alto a la salida del Módulo GPS supone que el LED está apagado, evitando daños en el iluminador y en la electrónica de acondicionamiento por sobrecaliento.

2.4. Experimentación

La prueba que definirá la fiabilidad del proyecto será sincronizar la captura de las cámaras con la emisión del iluminador infrarrojo, estando cada uno de los sensores separados cierta distancia y con su propio conjunto de Módulo GPS y Placa Arduino con sus respectivas señales.

Para llegar a realizar esta prueba se realizan pruebas preliminares que permitirán asegurar que todas las configuraciones y bases explicadas anteriormente funcionan correctamente, dando paso a la última prueba, la cual verificará la fiabilidad o no de este sistema.

2.4.1. Funcionamiento de las cámaras Grasshopper3 con la señal PPS

Como se ha visto anteriormente, la cámara Grasshopper3 de la marca PointGrey tiene un modo de funcionamiento que le permite capturar una imagen cuando un pulso le llega a través de los pines 1 y 6 del puerto de 8 pines de la Ilustración 12.

Esta prueba consiste en generar una señal cualquiera con la aplicación U-Center para enviarla a través del pin PPS del Módulo GPS y comprobar con un programa como FlyCapture2 o cualquiera que permita visionar las capturas de la cámara en tiempo real.

En este caso, las señales para realizar capturas con la cámara son a 5 FPS, a 10 FPS a 30 FPS y a 100 FPS y se comprueban con el programa utilizado si con cada una de las señales se capturan imágenes a esa velocidad.

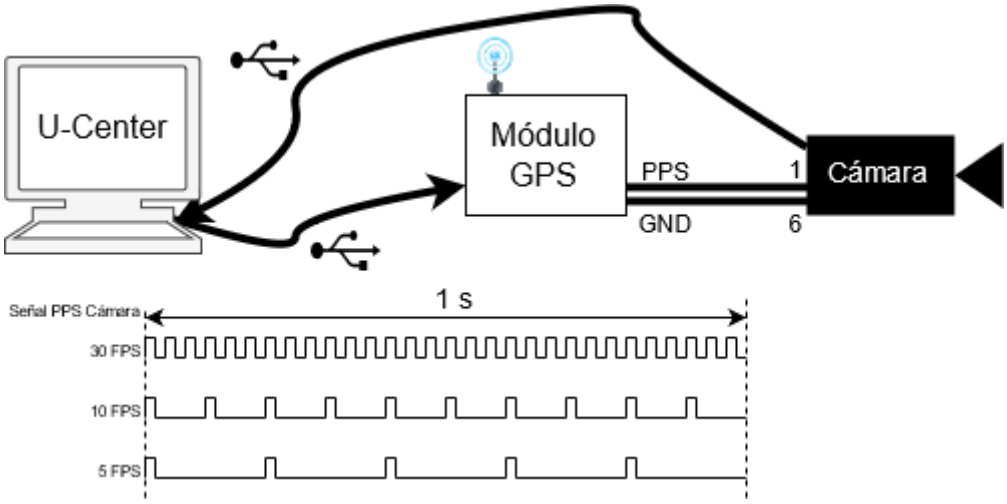


Ilustración 25. Montaje prueba 1.

FlyCapture2 tiene un campo que indica la velocidad de captura de la cámara en tiempo real, por lo que es sencillo comprobar que efectivamente captura a las velocidades deseadas.

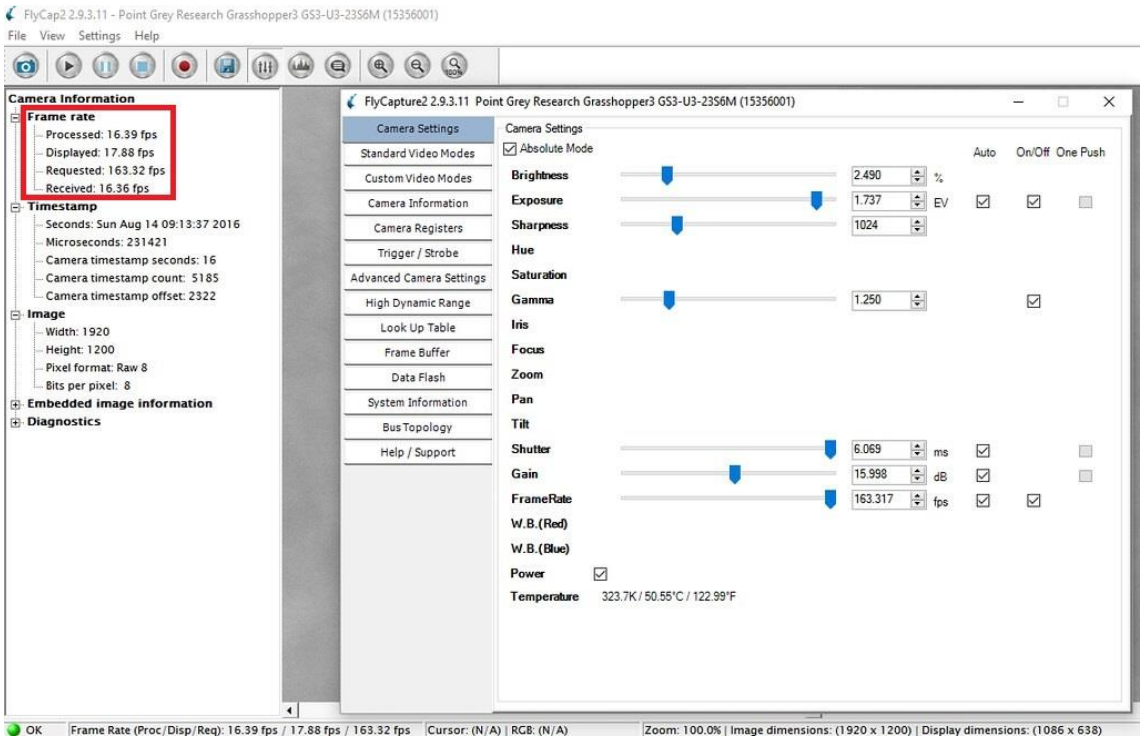


Ilustración 26. Pantalla programa FlyCapture2 donde se muestra el campo que comprueba los FPS actuales de la cámara.

2.4.2. Funcionamiento del iluminador infrarrojo con la señal PPS

Una vez diseñada y montada la electrónica de acondicionamiento para el iluminador infrarrojo, podemos realizar la misma prueba que se realizó en el apartado anterior, teniendo en cuenta que la señal está invertida, por lo que la señal PPS que emita el Módulo GPS se configurará para flanco de bajada además de emitir una señal constante a nivel alto durante el estado de reposo.

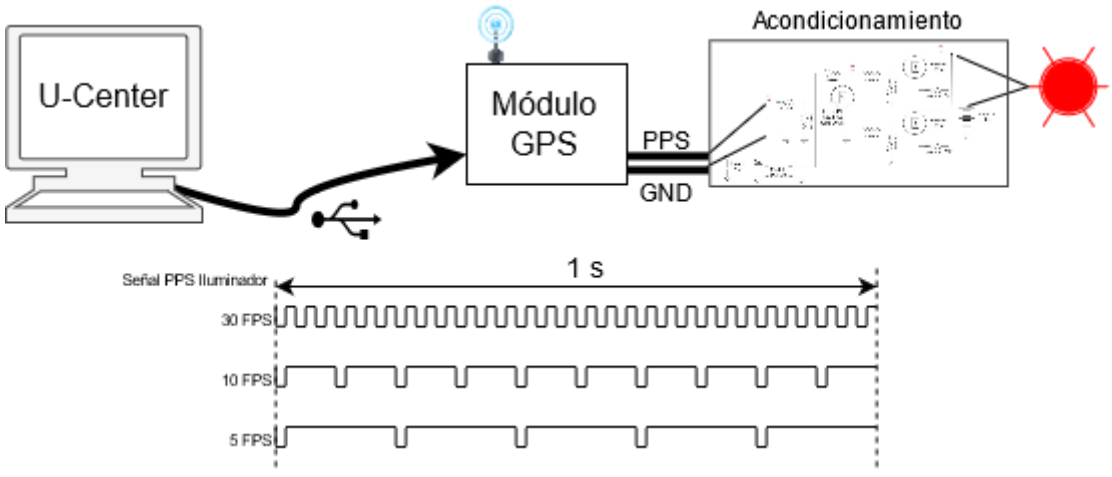


Ilustración 27. Montaje prueba 2.

Se puede comprobar por observación directa el funcionamiento de diferentes señales como en el caso anterior, confirmando que la electrónica de acondicionamiento cumple su función.

Durante el desarrollo de esta prueba, se observa que, si el Módulo GPS se queda sin alimentación, o lo que es lo mismo, no se emite una señal constante de 3.3 voltios por el pin PPS, los transistores

IRF1404 permiten el paso de la corriente al iluminador de manera continua, provocando daños irreversibles en los transistores por calor incluso llegando a afectar al iluminador.

2.4.3. Funcionamiento de cámara e iluminador con la misma señal PPS (1 Módulo GPS)

Habiendo realizado las dos pruebas anteriores, ahora se debe cerciorar que la cámara es capaz de captar adecuadamente los instantes en los que el iluminador está encendido.

Para ello, se prepara una señal cuyo pulso sea de 2ms o superior para ambos sensores. El periodo puede ser elevado, ya que nos interesa poder observar en las capturas de la cámara cambios en el ambiente además del iluminador encendido mientras que observamos de forma directa si el iluminador se enciende y se apaga.

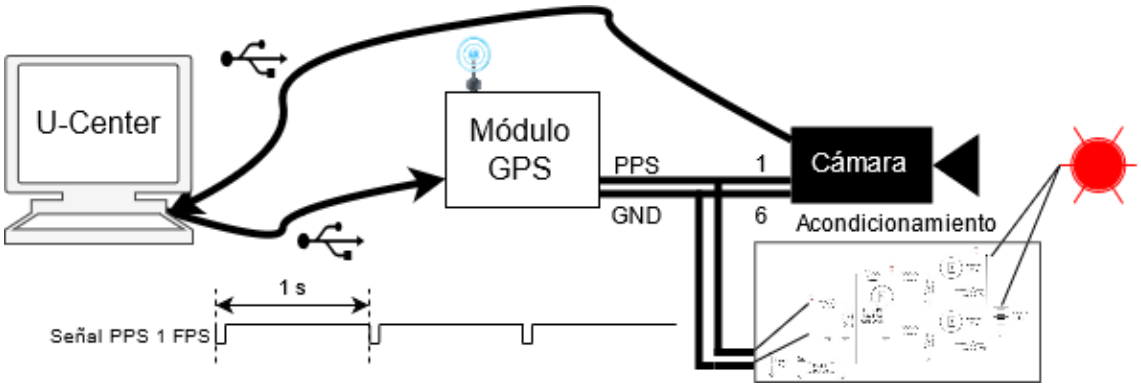


Ilustración 28. Montaje prueba 3.

Para esta prueba, la cámara se configura para que se accione el disparador por flanco de bajada, ya que necesitamos que la señal empiece por flanco de bajada para que el iluminador funcione correctamente.

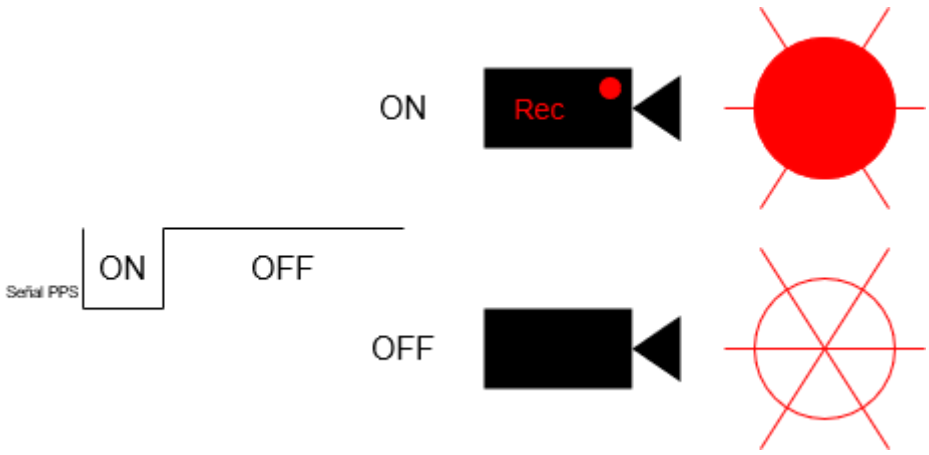


Ilustración 29. Prueba captura de iluminador encendido con una señal PPS.

Es de suponer que siempre que la cámara capture, el iluminador estará encendido, tal y como se observa en las pruebas.

Esta prueba principalmente nos asegura que la cámara es capaz de captar de manera correcta el iluminador encendido.

2.4.4. Funcionamiento de cámara e iluminador con señales PPS distintas (2 Módulos GPS)

El paso siguiente a la prueba anterior es conectar una señal PPS de diferentes Módulos GPS a cada sensor para realizar la primera prueba de sincronización real.

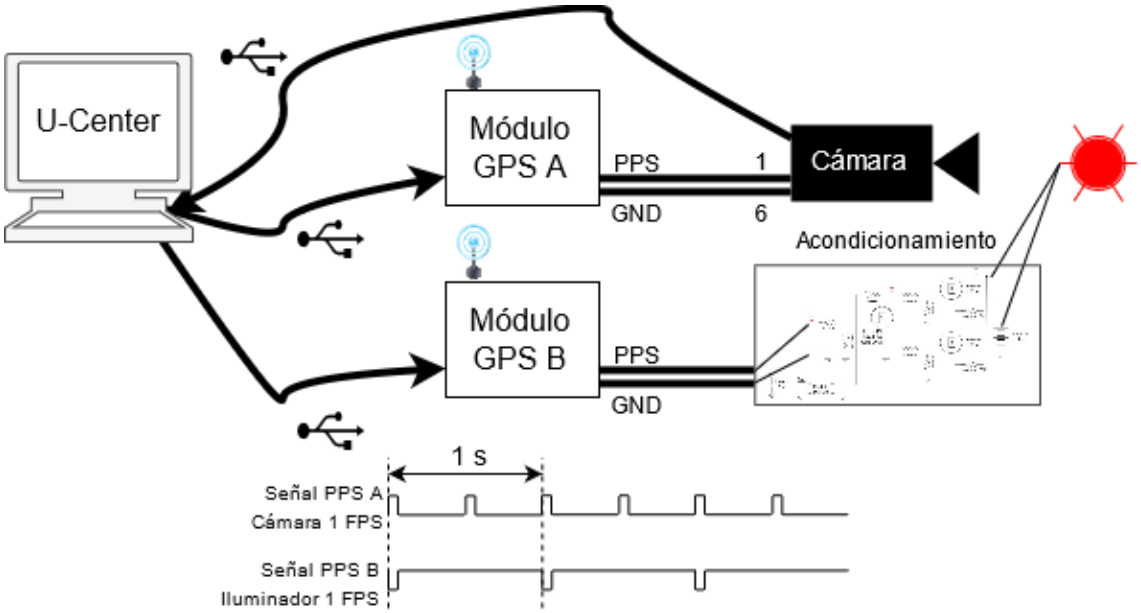


Ilustración 30. Montaje prueba 4.

La señal PPS de la cámara en este caso será por flanco de subida y será el doble de rápida que la señal PPS del iluminador, de esta manera se observará un patrón en las capturas de la imagen.

Si la señal de la cámara es el doble de rápido, se observará el iluminador encendido en un fotograma y en el siguiente aparecerá apagado.

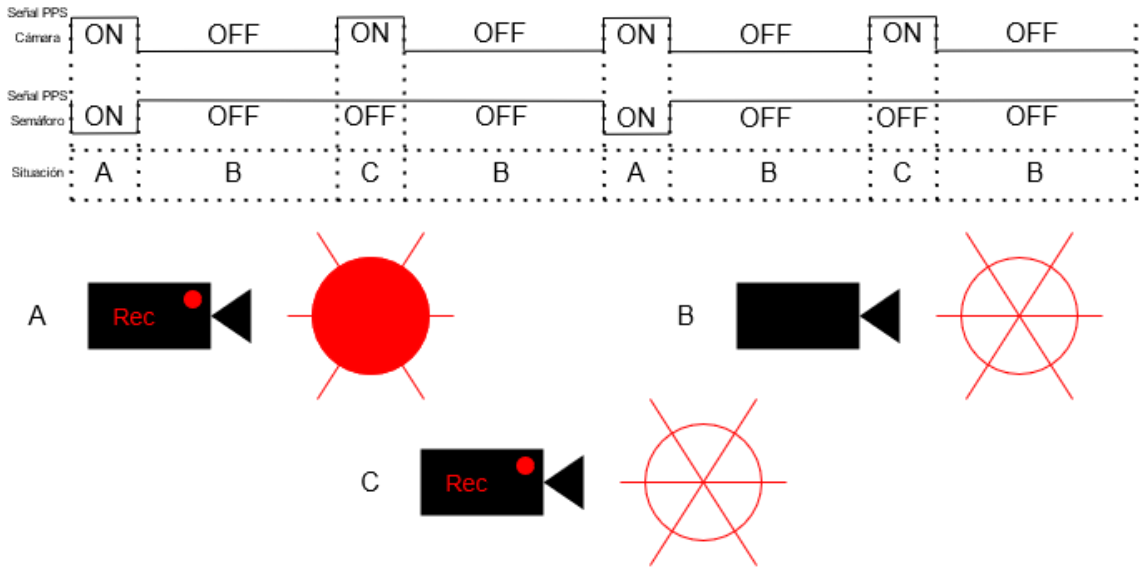


Ilustración 31. Prueba captura de iluminador encendido con señales PPS diferentes.

Con esta prueba queda demostrado que la sincronización es posible a pesar de haberse realizado en un espacio cerrado y con los Módulos GPS próximos entre si, por lo que todavía no se ha demostrado si captando diferentes señales de satélites y estando a cierta distancia la sincronización es fiable.

2.4.5. Envío de cadenas UBX prediseñadas al Módulo GPS a través de la Placa Arduino

A partir de esta prueba entra en juego el software de control diseñado para la Placa Arduino.

Para poder enviar las cadenas UBX se conecta el Módulo GPS a la Placa Arduino de tal manera que los pines Tx y Rx del Módulo GPS estén conectados a los pines Rx y Tx de la Placa Arduino respectivamente como en la Ilustración 3.

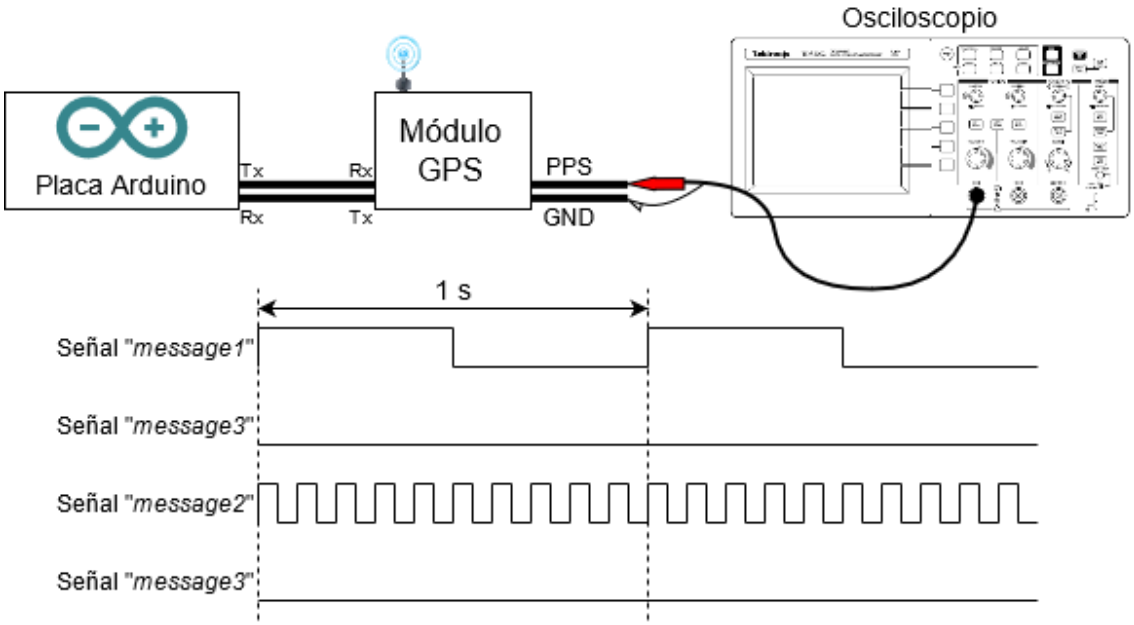


Ilustración 32. Montaje prueba 5.

Se usará el Código 15 para enviar la cadena UBX con la señal prediseñada a través del puerto serie cada vez que se encienda la Placa Arduino y se observará cómo la señal PPS del Módulo GPS cambian en función de la señal prediseñada.

```
1  //message1 Señal con periodo 1s y pulso 500ms
2  //message2 Señal con periodo 100ms y pulso 50ms
3  //message3 Señal desactivada
4
5  void setup()
6  {
7    Serial.begin(9600);
8    delay(200);
9    Serial.write(protocol_OUT_OFF, sizeof(protocol_OUT_OFF));
10   delay(200);
11   Serial.write(message3, sizeof(message3));
12   delay(200);
13 }
14
15
16 void loop() {
17
18   while (1) {
19     delay(10000);
20     Serial.write(message1, sizeof(message1));
21     delay(10000);
22     Serial.write(message3, sizeof(message3));
23     delay(10000);
24     Serial.write(message2, sizeof(message2));
25     delay(10000);
26     Serial.write(message3, sizeof(message3));
27   }
28 }
```

Código 15. Código para probar en envío de Cadenas UBX por la Placa Arduino.

Una vez que el programa está compilado en la Placa Arduino y ésta se conecta al Módulo GPS, cuando recibe alimentación se observa las siguientes señales por el pin PPS:

- Durante 10 segundos, señal con periodo de 1 segundo y pulso de 500 milisegundos, correspondiente a la cadena UBX *"message1"*.
- Durante 10 segundos, señal desactivada, correspondiente a la cadena UBX *"message3"*.
- Durante 10 segundos, señal con periodo de 100 milisegundos y pulso de 50 milisegundos, correspondiente a la cadena UBX *"message2"*.
- Durante 10 segundos, señal desactivada, correspondiente a la cadena UBX *"message3"*.

Esta secuencia se repite cada 40 segundos. Se demuestra así que las cadenas UBX que se envían a través de la Placa Arduino, funcionan igual que si son mandadas a través de la aplicación U-Center.

2.4.6. Funcionamiento de las cámaras Grashooper3 comandadas por la Placa Arduino

De igual manera que se hizo directamente conectando el Módulo GPS a la cámara y controlando la señal PPS con la aplicación U-Center, en este caso será la Placa Arduino quien, en un determinado instante de tiempo, comande al Módulo GPS para que emita una señal PPS deseada para que capture a una determinada velocidad y cese de capturar en otro instante.

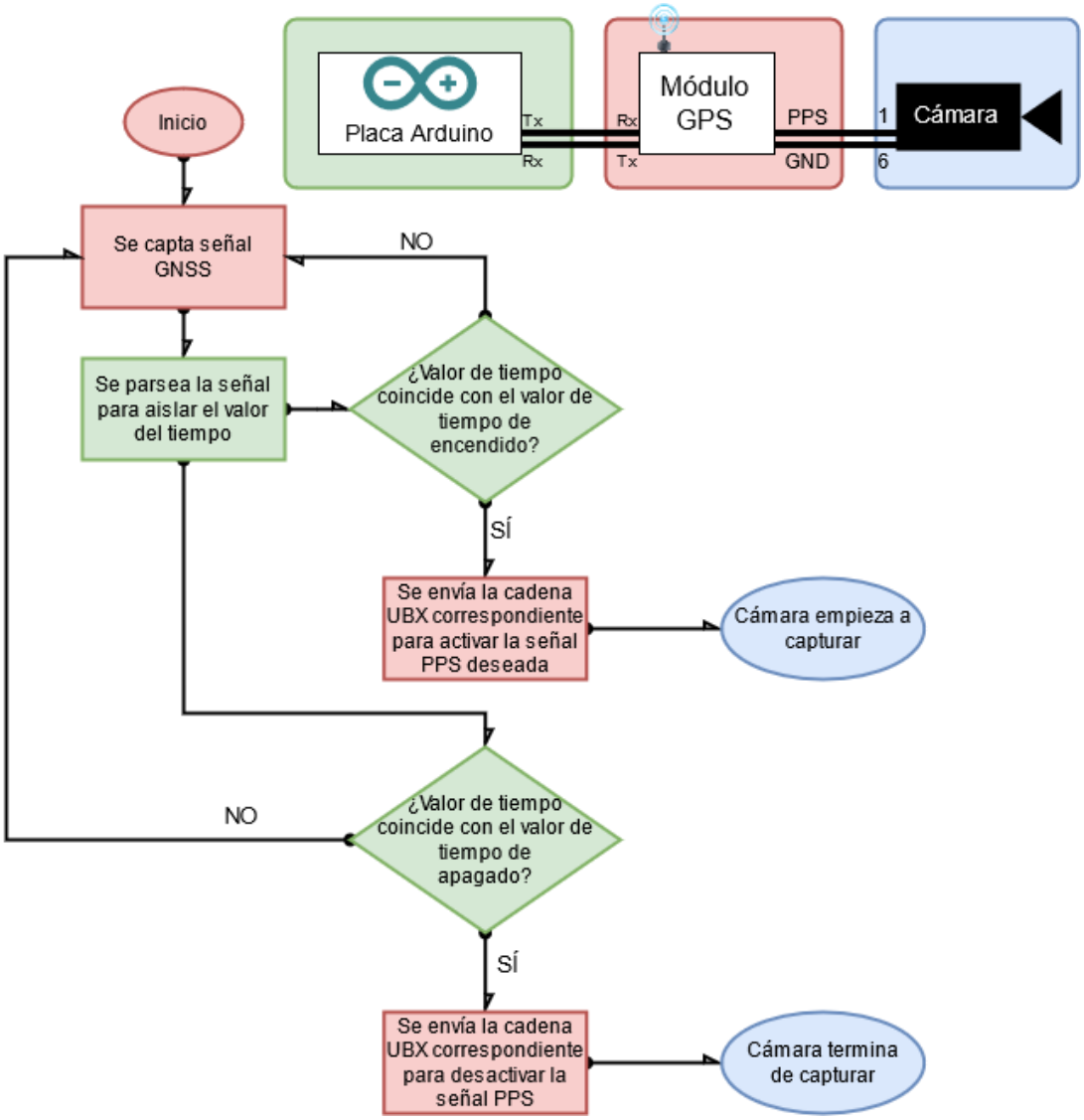


Ilustración 33. Montaje prueba 6. Flujograma de funcionamiento con Placa Arduino y Módulo GPS.

Se hace uso del software desarrollado y se configura la señal deseada para capturar a 25 FPS, esto es una señal de 40 milisegundos de periodo con un pulso de 2 milisegundos al menos, por flanco de subida y se determina el tiempo de encendido y el tiempo de apagado.

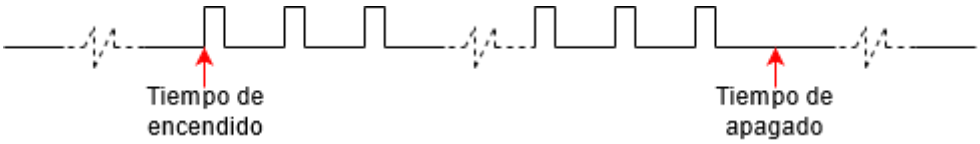


Ilustración 34. Ejemplo de señal enviada a la cámara con tiempo de encendido y apagado programada en la Placa Arduino.

Se comprueba visualmente con el programa FlyCapture2 que se está capturando a la velocidad deseada entre el instante de tiempo de encendido y el de apagado exclusivamente.

Es necesario comprobar que se pueden gestionar las diferentes señales deseadas para cada sensor del grupo INVETT y tener la opción de utilizar un tiempo de encendido y tiempo de apagado para aquellos sensores que interese activar en una determinada franja de tiempo.

2.4.7. Funcionamiento del iluminador infrarrojo comandado por la Placa Arduino

Ahora se realiza la misma prueba que antes, pero en este caso en el iluminador infrarrojo. La señal debe ser inversa, configurada para flanco de bajada, y el Módulo GPS tiene que estar previamente configurado para ser usado con la electrónica de acondicionamiento y no provoque daños por calor.

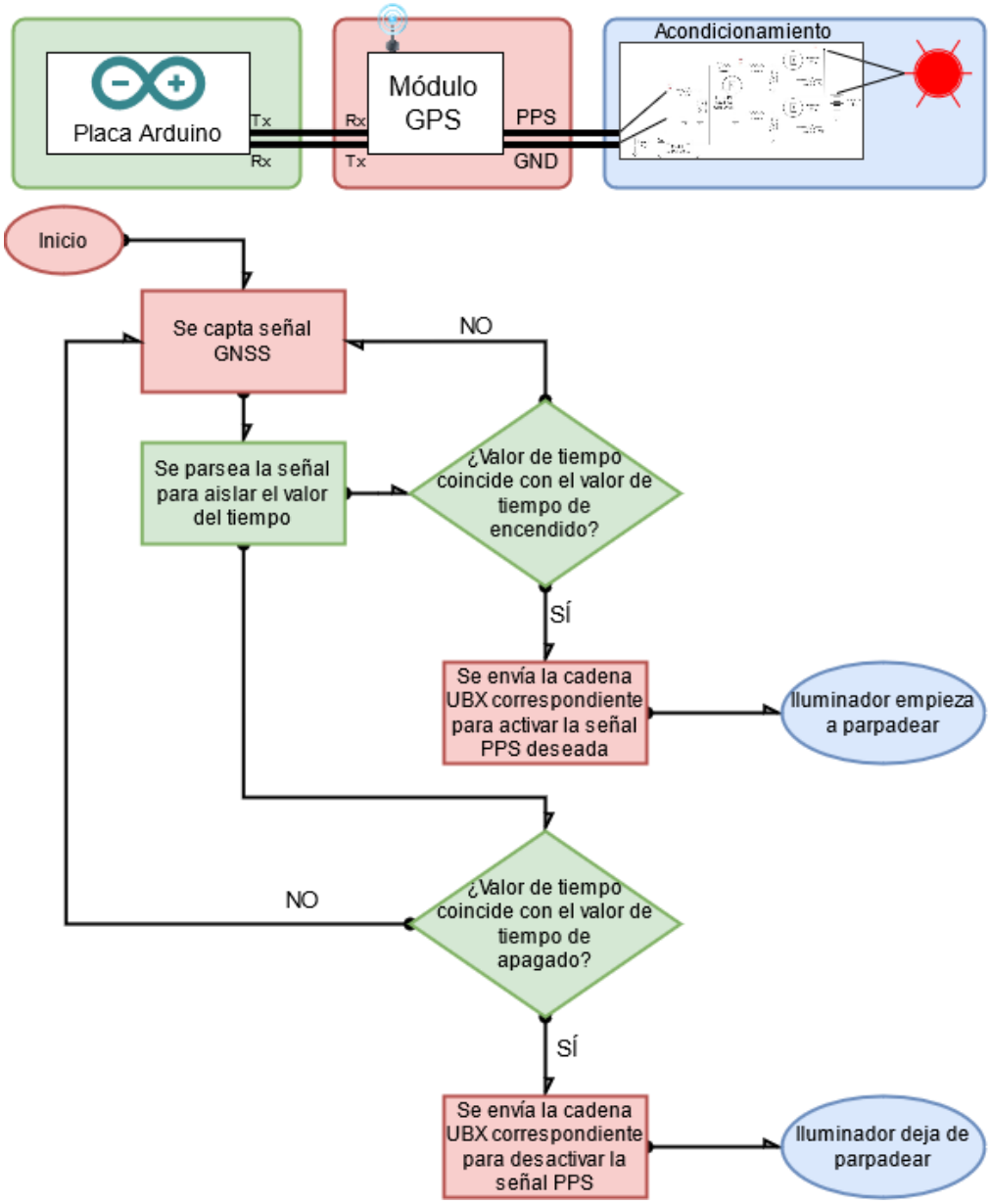


Ilustración 35. Montaje prueba 7. Flujograma de funcionamiento con Placa Arduino y Módulo GPS.

Se hace uso del software desarrollado y se configura la señal deseada para encender el iluminador infrarrojo 25 veces por segundo, esto es una señal de 40 milisegundos de periodo con un pulso de 2 milisegundos al menos, por flanco de subida y se determina el tiempo de encendido y el tiempo de apagado.

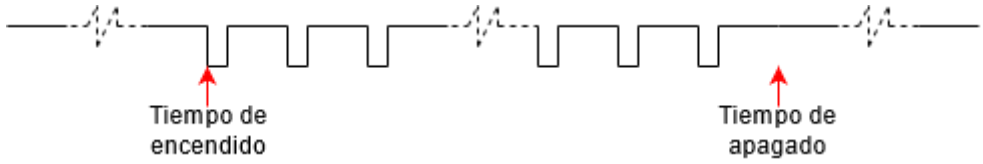


Ilustración 36. Ejemplo de señal enviada al iluminador con tiempo de encendido y apagado programada en la Placa Arduino.

Por observación se confirma que el iluminador se enciende acorde a la señal programada.

Gracias a esta prueba se puede validar que al usar de una electrónica de acondicionamiento no se perturba el funcionamiento deseado utilizando la Placa Arduino.

2.4.8. Comprobación de fiabilidad de sincronización entre los sensores *in-door*

Una vez comprobado que tanto el iluminador infrarrojo como la cámara funcionan correctamente con las señales PPS que emiten los diferentes Módulos GPS comandados por sus respectivas Placas Arduino, se va a repetir la prueba en la que se tenía una señal PPS para la cámara el doble de rápida que la del iluminador, pero en este caso las señales se configurarán para un tiempo de encendido y un tiempo de apagado a través del programa compilado en la Placa Arduino.

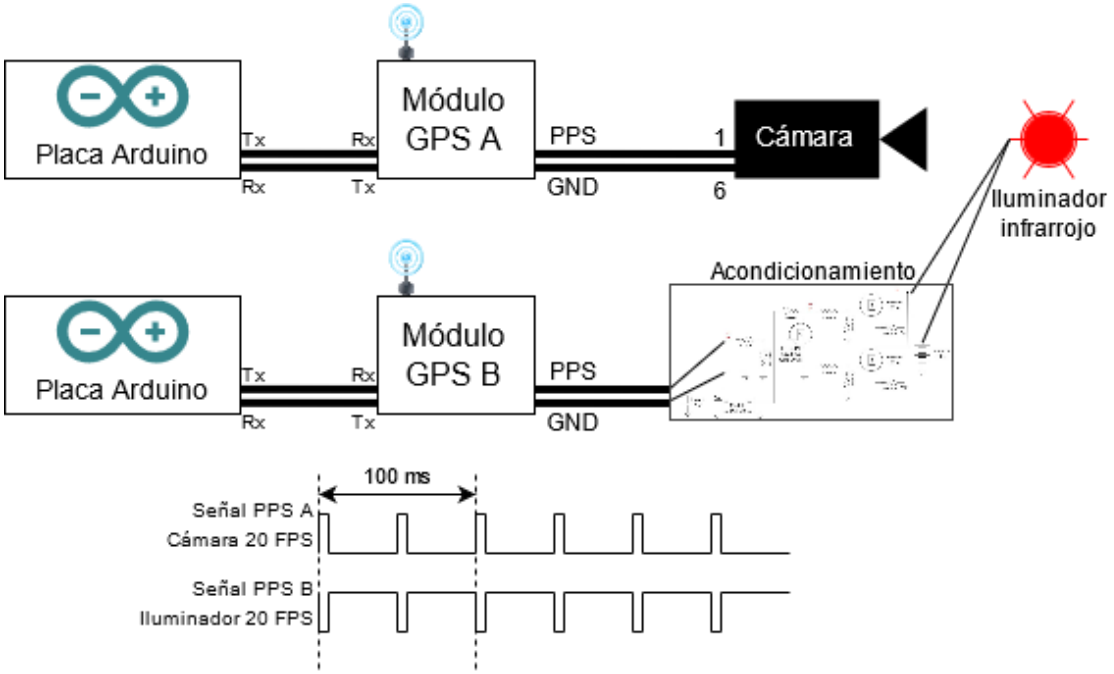


Ilustración 37. Montaje prueba 8.

Primero se configurará paso a paso la señal PPS para la cámara, se configurará un tiempo de encendido y uno de apagado. Para esta prueba, la cámara guardará los fotogramas que capture para poder comprobarlos después.

Después se configurará la misma señal, pero con flanco de bajada para el iluminador y se definirán los mismos tiempos de encendido y apagado.

Ilustración 38. Capturas prueba in-door.

La señal PPS configurada, independientemente del flanco, tiene un periodo de 50 milisegundos y un pulso de 5 milisegundos. Se observa cómo todas las capturas que realiza la cámara, Ilustración 38, captan el iluminador infrarrojo encendido como se esperaba. Visualmente durante la prueba se aprecia el parpadeo del iluminador indicándonos que efectivamente está capturando exclusivamente cuando el iluminador está encendido.

2.4.9. Comprobación de fiabilidad de sincronización entre los sensores en el exterior

La última prueba que comprobará la fiabilidad de sincronización entre los diferentes sensores se realiza en la azotea de la Escuela Politécnica Superior colocando cada sensor separados aproximadamente 50 metros formando un triángulo y pegados a una pared para que cada uno capte, por triangulación, diferentes señales satélite.

Al separar los sensores cierta distancia se asemeja el uso de los sensores y el vehículo autónomo del grupo INVETT, puesto que éstos estarán situados en el propio vehículo o en el exterior a varios metros de distancias unos de otros.

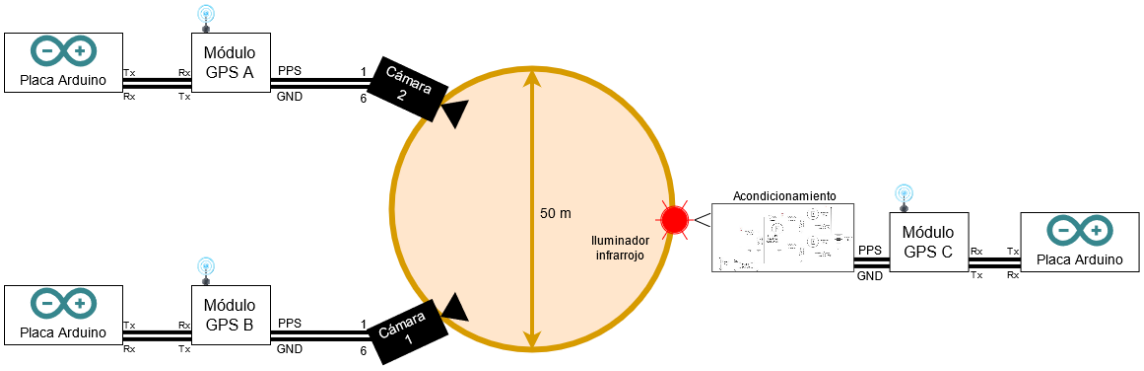


Ilustración 39. Montaje prueba final.

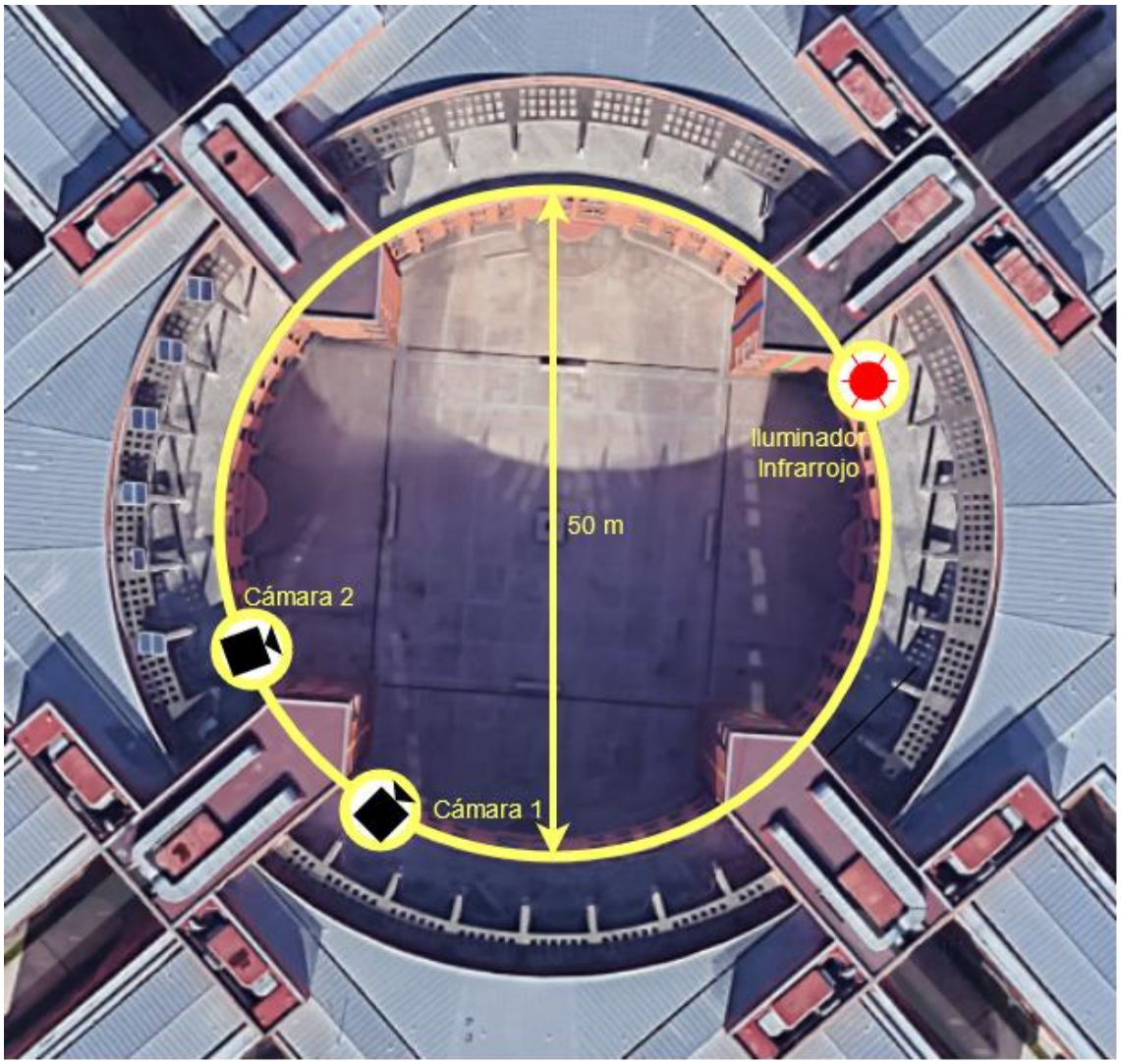


Ilustración 40. Posición sensores en azotea de la EPS UAH.

Para esta prueba, la cámara 1 se configurará con un periodo de 10 milisegundos y un pulso de 5 milisegundos. La cámara 2 se configurará con un periodo de 20 milisegundos y un pulso de 5 milisegundos. Finalmente, el iluminador se configurará con un periodo de 40 milisegundos y un pulso de 5 milisegundos.

Tras configurar las 3 señales a través del monitor serie de la aplicación de Arduino y establecer el mismo tiempo de encendido y el mismo tiempo de apagado se deberá observar lo siguiente:

- Para la cámara 1, deberá capturar primero el iluminador encendido y luego 3 fotogramas apagado.
- Para la cámara 2, deberá capturar primero el iluminador encendido y luego 1 fotograma apagado.

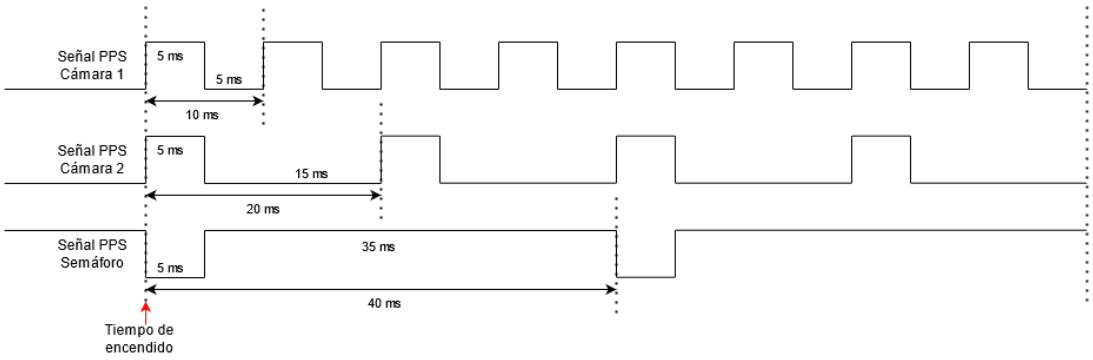


Ilustración 41. Señales prueba final de fiabilidad con los 3 sensores.

Las capturas correspondientes a esta prueba son las siguientes:

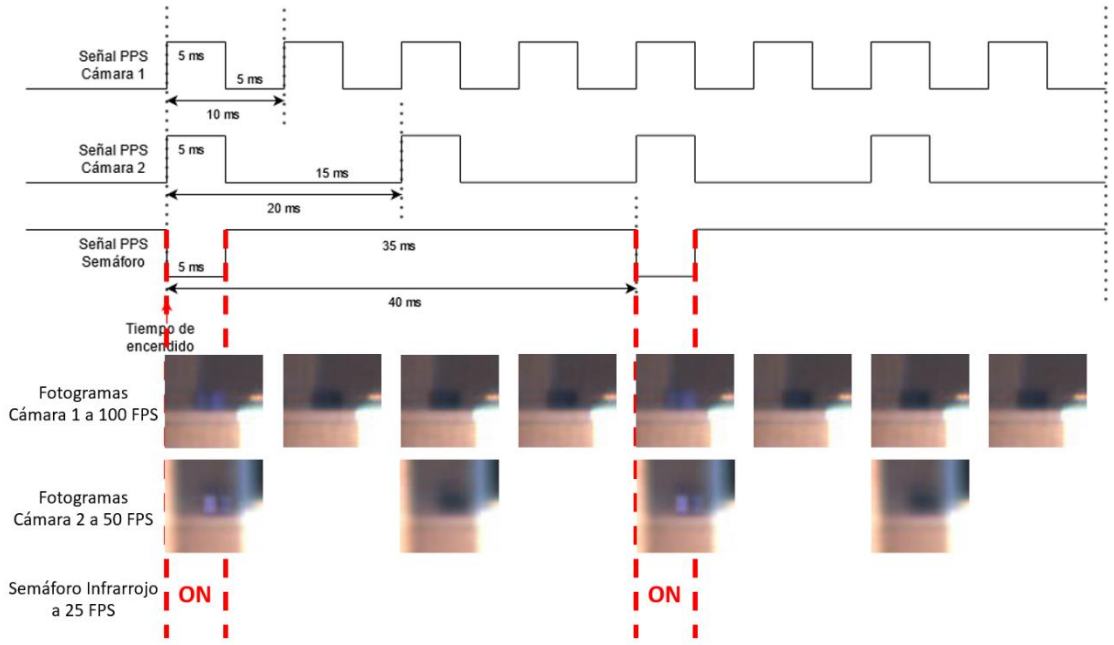


Ilustración 42. Fotogramas capturados de la prueba final de fiabilidad con los 3 sensores.

Se puede apreciar en la Ilustración 42 cómo los fotogramas de la cámara 1 y la cámara 2 correspondientes al estado ON del iluminador muestran un brillo en el iluminador, mientras que en el resto de los fotogramas no se aprecia.

Los fotogramas mostrados se han sacado de los videos en bruto que han grabado los sensores y están representado en el mismo orden en el que se han capturado.

Por último, para comprobar las señales satélite que recibía cada Módulo GPS se procede a capturar una imagen del programa U-Center que muestra en ese momento los satélites de los que proviene la señal.

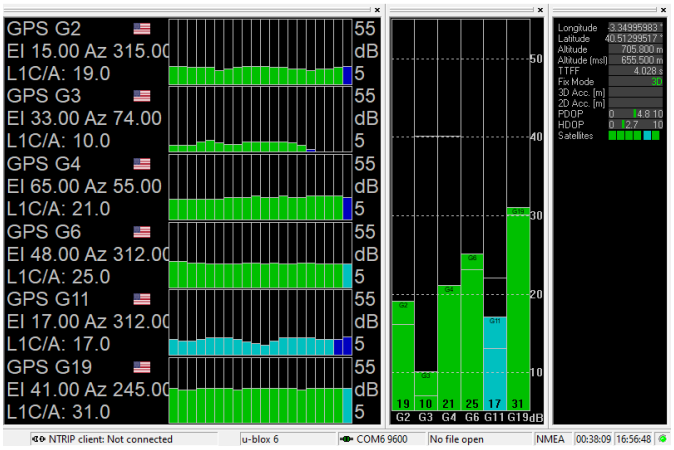


Ilustración 43. Señales satélite de la cámara 1 durante la prueba.

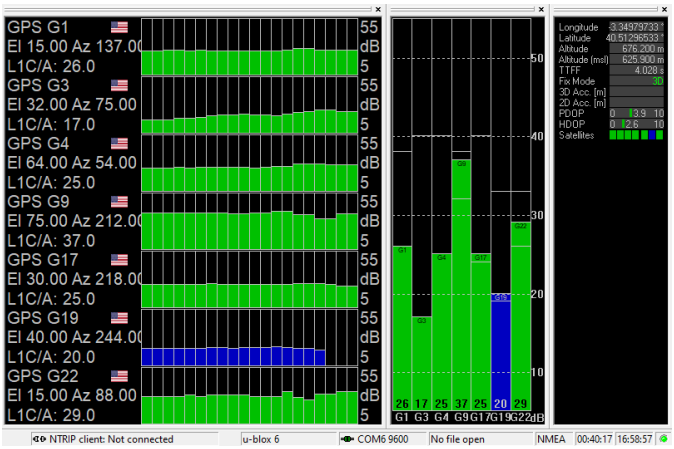


Ilustración 44. Señales satélite de la cámara 2 durante la prueba.

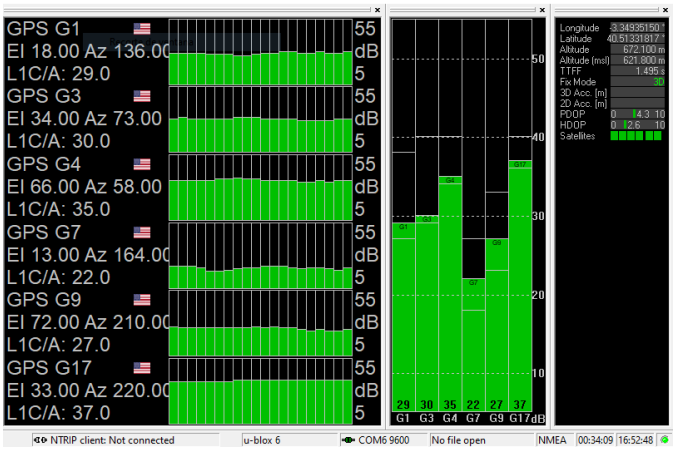


Ilustración 45. Señales satélite del iluminador infrarrojo durante la prueba.

Podemos apreciar en las Ilustraciones 43, 44 y 45 que las señales proceden de diferentes satélites, por lo que independientemente de la señal satélite que se reciba, la variable de tiempo que obtenemos es la misma.

Se determina con esta última prueba que es posible sincronizar diferentes sensores utilizando la variable de tiempo recibida por satélite independientemente de la localización del sensor.

3. Conclusiones

Concluimos que la sincronización de señales utilizando GPSs como referencia global es fiable tal y como se ha demostrado con las pruebas experimentales descritas en el apartado 3 de este proyecto. El margen de error que ocurre cada segundo, siendo éste inferior a 60ns por periodo, hace viable la utilización del Módulo GPS escogido para aplicaciones que requieren una precisión de milisegundos, dejando margen hasta décimas de milisegundo.

Esta limitación viene dada por el error captado por el Módulo GPS, siendo posible otras aplicaciones más precisas si las especificaciones lo permiten.

3.1. Trabajo futuro

Se puede implementar, utilizando los pines de la Placa Arduino un sistema que reconozca si el sensor objetivo funciona por flanco de subida o flanco de bajada. Este sistema puede comprobar el sensor de manera directa. De forma más sencilla, se podría utilizar uno de los pines de la Placa Arduino para que en función de si recibe un valor a nivel alto o no, utilice la cadena UBX "*APAGADO_ALTO*" o "*APAGADO_BAJO"* en el *Setup* inicial correspondiente a los Códigos 3 y 4.

Se podría implementar una mejora para calcular la posición con el uso de otras señales GPS y así determinar la zona horaria en la que está situado el sensor, estableciendo automáticamente la hora UTC y el horario de verano si corresponde.

4. Listado de referencias

[1] La sincronización utilizando satélites de navegación, https://www.gmv.com/blog_gmv/language/es/la-sincronizacion-utilizando-satelites-de-navegacion/

[2] NTP, SNTP y PTP: ¿Qué sincronización de tiempo necesito?, https://www.incibe-cert.es/blog/ntp-sntp-y-ptp-sincronizacion-tiempo-necesito

[3] PTP – Protocolo de tiempo de precisión, https://www.perlesystems.es/supportfiles/precision-time-protocol.shtml

[4] Network Time Protocol, https://en.wikipedia.org/wiki/Network_Time_Protocol

[5] LOCALIZACIÓN GPS CON ARDUINO Y LOS MÓDULOS GPS NEO-6, https://www.luisllamas.es/localizacion-gps-con-arduino-y-los-modulos-gps-neo-6/

[6] Vista superior de una placa Arduino UNO y descripción de componentes, https://aprendiendoarduino.wordpress.com/2016/06/27/arduino-uno-a-fondo-mapa-de-pines-2/

[7] Librería TinyGPS para Arduino, https://www.arduino.cc/reference/en/libraries/tinygps/

[8] Manual NEO-6 u-Blox 6 GPS Modules Data Sheet.

[9] Manual u-blox6 Receiver Description Including Protocol Specification.

[10] Manual GrassHopper 3 U3 Technical Reference.

[11] DataSheet IRLZ44N

[12] DataSheet IRF1404

5. Bibliografía

- Tutorial Arduino #0009 - Módulo GPS & Display i2C, https://www.tr3sdland.com/2012/03/tutorial-arduino-0009-modulo-gps-display-i2c/
- Massimo Banzi y Michael Shiloh, Introducción a Arduino Edición 2016. EDICIONES ANAYA MULTIMEDIA. ISBN: 978-84-415-3744-6
- Arduino Home, https://www.arduino.cc/
- uBlox Home, https://www.u-blox.com/en
- FlyCapture2, https://www.flir.es/
- Materiales y búsqueda de transistores, https://www.electronicaalcala.net/
- Equipamiento, productos y soluciones GPS y GNSS, https://novatel.com/
- Lenguaje de programación C++ para Arduino, https://aprendiendoarduino.wordpress.com/2015/03/26/lenguaje-de-programacion-c/

Universidad de Alcalá