Informe Crítico de Diseño

Equipo Team Ramiro

Mentor: Manuel Blazquez Perez Merino
Participantes: Esteban Lorite Arauzo,David Tarín Alcaide, Iker González-Casallo Castro, Gael Álvarez Monés, Clara Jimeno Redondo, Marco Padillo

IES Ramiro de Maeztu, Madrid, Madrid

Índice

Resumen: Cansat es un programa dirigido hacia institutos, universidades y grados superiores. Esta competición está iniciada por la Agencia Espacial Europea en la que cada equipo debe diseñar, programar y construir un satélite del tamaño de una lata capaz de cumplir una misión determinada, recoger datos y efectuar retornos controlados.

Este documento recoge toda la información del proyecto del Team Ramiro para el concurso CanSat España 2023-2024. En él se describe el trabajo realizado para el desarrollo del proyecto. El documento sigue las bases especificadas por la organización del concurso. Estos datos pueden consultarse en su página web. https://esero.es/cansat-2/

1. Introducción

1.1 Equipo, organización y roles

El equipo Team Ramiro está formado por 6 estudiantes de primer curso del bachillerato tecnológico y artes en IES Ramiro de Maeztu con una gran inquietud por la adquisición de conocimiento, en Madrid.

	David	Marco	Gael	Clara	Iker	Esteban
Diseño mecánico						
Diseño electrónico						
Sistema de frenado						
Divulgación y redes sociales						
Contabilidad						
Planificación						
Desarrollo del informe						
Comunicaciones						
Telemetría avanzada						
Soldado						

Alta implicación
Media implicación
Baja implicación

También forma parte del equipo como educador y mentor Manuel Pedro Blázquez Merino, profesor de TIC en el IES Ramiro de Maeztu.

1.2 Objetivos del proyecto

El CanSat llevará a cabo dos misiones: una misión primaria, impuesta por las bases del concurso y común a todos los equipos, y una misión secundaria, la cual elige cada equipo.

La misión primaria consistirá en la toma de una serie de datos de la atmósfera durante el descenso y la transmisión de estos por telemetría a la estación de tierra a al menos 1 transmisión por segundo.

La misión secundaria de nuestro equipo es la telemetría avanzada y el registro visual del descenso. Tras la liberación y durante el descenso el cansat medirá y transmite telemetría adicional a la exigida para la misión primaria. Estas son: Aceleración, posicionamiento GPS y el nivel de la calidad del aire. También por otra parte durante el descenso se realizará una grabación del trayecto del satélite desde el propio satélite.

Para mantener una velocidad constante y un descenso con una velocidad controlada se empleara un paracaídas.

2. Descripción del proyecto

2.1 Fases de la misión

Hemos dividido la misión en 4 fases, la fase 0, de la cual carecemos control y en la que no podemos realizar nada, que sería la fase de lanzamiento y alcance de altitud máxima, la cual se aproxima a los 1000 metros. Después le siguen las siguientes 3 fases las cuales son:

Fase 1: En esta fase, después de ser liberado, el cansat desplegará su paracaídas gracias a la implementación del acelerómetro y el gps, los cuales determinarán si el cansat ha empezado a caer para la liberación del paracaídas.

Fase 2: En esta fase se hará toda la toma de datos de tanto la misión principal como la misión secundaría. Durante esta fase el CanSat se comunicará a la base de tierra mediante radio transmisión.

Fase 3: En esta fase el CanSat aterrizará y gracias a la integración de la unidad de GPS se localizara su ubicación para el rescate del mismo.

2.2 Misión Primaria

La misión principal consiste en la toma de datos de temperatura, presión y humedad y el envío de los mismos mediante un transmisor de radio LoRa (Long Range) a un mínimo de 1 comunicación por segundo con el control de tierra.

Utilizaremos como controlador un Arduino Nano V3. Hemos elegido este módulo por su reducido tamaño y consumo, aparte de la facilidad que proporciona a la hora de programar los sensores y la planificación de los circuitos.

El módulo BMP280 realizará las tres mediciones de los datos correspondientes, los cuales serán recopilados y enviados a través de la antena del modelo RYLR 896.

Esta misión transcurre durante toda la etapa de vuelo del satélite, desde su despliegue a los 1000 metros de altitud hasta el aterrizaje en tierra. Para la recogida de datos se emplea una base en tierra con otra antena que se conectara a un ordenador para recopilar los datos que recoja el Arduino.

2.3 Misión Secundaria

La misión secundaria se conforma de tres puntos principales, siendo estos los siguientes:

2.3.1 GPS

Para obtener los datos del posicionamiento del CanSat en tiempo real se empleara una unidad de GPS NEO-6M de U-Blox. Este módulo es capaz de conseguir las coordenadas con una precisión del 2,5 m en posición y 0,5º en orientación, por lo que se obtendrá una posición casi exacta del mismo.

Dicho módulo desempeña dos funciones, seguimiento del CanSat, lo cual resultará útil a la hora de encontrar la ubicación del aterrizaje y la recopilación de los datos de la altura y la velocidad durante el descenso.

2.3.2 Telemetría avanzada

Esta misión está designada a la recolección de datos adicionales a los que requiere la misión principal, en nuestro caso elegimos recolectar datos relacionados con la calidad del aire y la aceleración del CanSat durante el descenso y el ascenso.

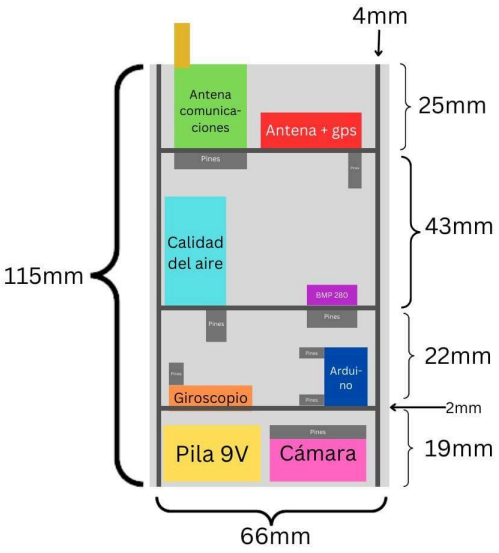
Para la primera misión se utilizará un módulo MQ-135, un sensor de calidad del aire de alta sensibilidad que se utiliza para determinar la presencia de ciertos gases en el aire que está en contacto con el módulo. Este módulo es capaz de medir NH3, NOx, alcohol, Benceno, Humo y CO2 dando las medidas en partes por millón (PPM).

Para la segunda misión se empleara el módulo MPU 6050, un giroscopio y acelerómetro, el cual como su nombre bien indica, sirve para medir inclinaciones y aceleraciones respecto al propio sensor.

2.3.3 Registro visual del descenso

Esta misión está designada para la grabación del trayecto de descenso del CanSat. Para dicha tarea empleamos una cámara Arducam modelo OV2640 mini con una calidad de video de 2 megapixels y con la capacidad de grabar con una resolución de 1600x1200 a 15fps, o en caso de ser necesario una mayor tasa de frames, se podría reducir a 800x600 a 30fps.

2.4 Diseño Mecánico



Este es el diseño mecánico que hemos hecho de la estructura interna del CanSat. Como se puede observar, hemos hecho los sensores y antena para poder estructurarlos dentro de la lata.

El diseño mecánico del cansat es una estructura cilíndrica de 115mm de alto y 66mm de diámetro. Dentro de este cilindro se encuentran los sensores y la unidad de procesamiento.

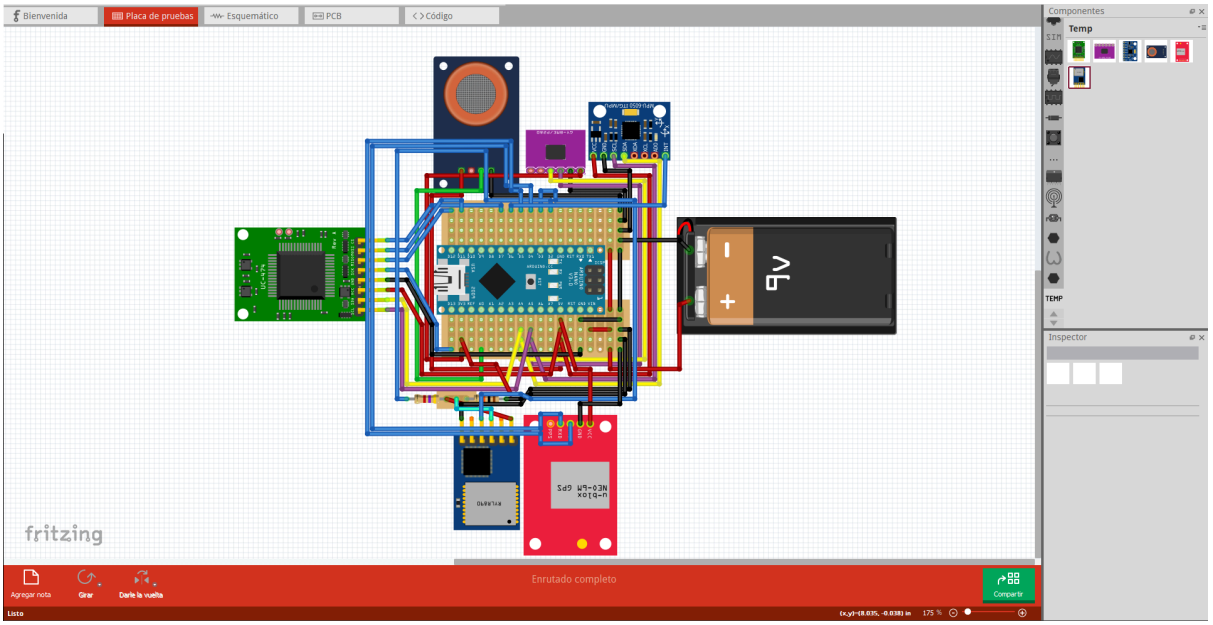
Aquí se puede ver el modelo casi final de la lata, con una compuerta para meter y sacar los sensores. Solamente faltan los agujeros de estos (que no sobresalen) y el enganche para el paracaídas.

Aquí podrán ver la disposición de los sensores (sin las bases de los niveles)

https://www.tinkercad.com/things/gNLMXktC0wG-lata-cansat/edit?returnTo=%2Fdashboard&sharecode=y1gsc81akA-Rrn-SdK9fpfYsltwXEowZ_ou_V2YZOVE

2.5 Diseño eléctrico

Como ordenador central del CanSat se utiliza un Arduino NANO debido a su sencillez al programar y la variedad de sensores y componentes hechos específicamente para dicho microcontrolador



Diseño del circuito en Arduino con todos los componentes que se van a utilizar para la misión

2.6 Paracaídas

Con el fin de tener una caída controlada a una velocidad constante, el CanSat contará con un paracaídas que lo ayudará a frenar la caída durante el desarrollo de la misión.

Para la fabricación del paracaídas del cansat, debe tenerse en cuenta la masa, la velocidad a la que caerá el CanSat, la gravedad de la tierra ($9,8 m/s^2$), la densidad del aire y la forma del paracaídas.

$$F_g = \frac{1}{2} \cdot C_d \cdot \rho \cdot A \cdot v^2$$

$$F_g = m \cdot g$$

Para que el CanSat caiga a velocidad constante, la fuerza de la caída y la de arrastre se deben anular. Tal que:

$$F_g = F_a$$

$$A = \frac{2 \cdot m \cdot g}{C_d \cdot \rho \cdot v^2}$$

Donde la densidad (ρ) del aire es aproximadamente, $1,225 kg/m^3$, la velocidad límite a la que se espera que caiga el CanSat (v) es de $8 m/s$, la masa del CanSat es de xg y C_d es un coeficiente que depende de la forma del paracaídas y se calcula de forma experimental.

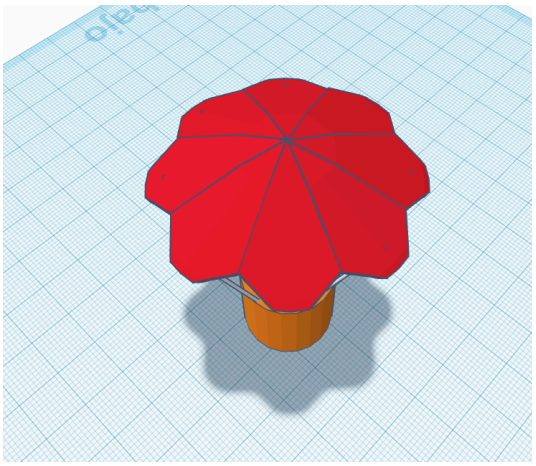
Dicho paracaídas será fabricado a partir de una tela 3.8oz Polainas y unido al cansat con hilo monofilamento y tendrá una superficie de aproximadamente 0,14 m² y seguirá uno de los patrones propuestos a continuación

Paracaídas número 1:
Este paracaídas tendrá un radio de unos 22,5 cm de radio para que caiga a la velocidad requerida (entre 8 y 12 m/s). esto lo sabemos gracias a la siguiente fórmula:

$$r = \sqrt{\frac{2 \cdot m \cdot g}{\pi \cdot C_d \cdot \rho \cdot v^2}}$$

Los triángulos que se formarían en el modelo, estarían con lados redondeados, para obtener una mayor estabilidad.

Según las pruebas realizadas, hubo un cálculo erróneo a la hora de ajustar el radio, ya que era la sombra del paracaídas lo que debía medir 22,5. Este error resultó idóneo para poder ajustar con mayor exactitud esas longitudes.

A la izquierda: Imagen del modelo del paracaídas terminado.
A la derecha: Imagen del diseño del modelo del paracaídas.

Modelo número 2:

Este modelo está aún en proceso. Medirá 25 cm de radio, y tendrá un agujero de 3 cm de radio en el centro. Esto otorgará mayor estabilidad al CanSat.

Modelo número 3:
Este modelo es uno de los más estables de todos. Se trata de un paracaídas cuadrado. Gracias a los huecos que tiene en las esquinas lo hacen superestable.

Gracias a la siguiente fórmula hemos calculado las medidas del mismo.

$$A = \frac{2 \cdot m \cdot g}{C_d \cdot \rho \cdot v^2}$$

m = 350 kg
Cd = 0,80
v = 6 m/s

Después de calcular el área que este abarca, calculamos el lado del paracaídas (recuerden que este será cuadrado).

Como podrán ver en la foto del final del modelo, este tendrá unos "salientes" del 20% del largo del paracaídas.

Entonces el paracaídas tendrá las siguientes medidas:

Lado del paracaídas: 41,89 cm

Longitud del saliente: 8,37 cm

Hemos calculado el lado con una velocidad de 6 m/s ya que hemos tomado en cuenta la densidad del aire de Madrid. Mas el lanzamiento definitivo se ejecutará en Cádiz, con una mayor densidad en el aire que Madrid.

Sin embargo, este paracaídas está todavía en fase de pruebas, ya que el modelo que hicimos tenía algunos defectos en las medidas.

Modelo número 4:

El nuevo paracaídas triangular, utilizado incluso en las fuerzas militares de Colombia (también llamado X-TRIANGLE) es extremadamente estable al péndulo gracias a su CONSTRUCCIÓN EN 3 VÉRTICES, la forma triangular tiene un efecto auto estable.

Este paracaídas tiene una teoría similar al anterior, salvo por la forma triangular de la base. Como el anterior, tiene unos salientes que ayudan a estabilizar el CanSat.

Con el área obtenida anteriormente, calculamos el lado del triángulo, y como antes, hacemos unos salientes del 20%.

Entonces las medidas serían:

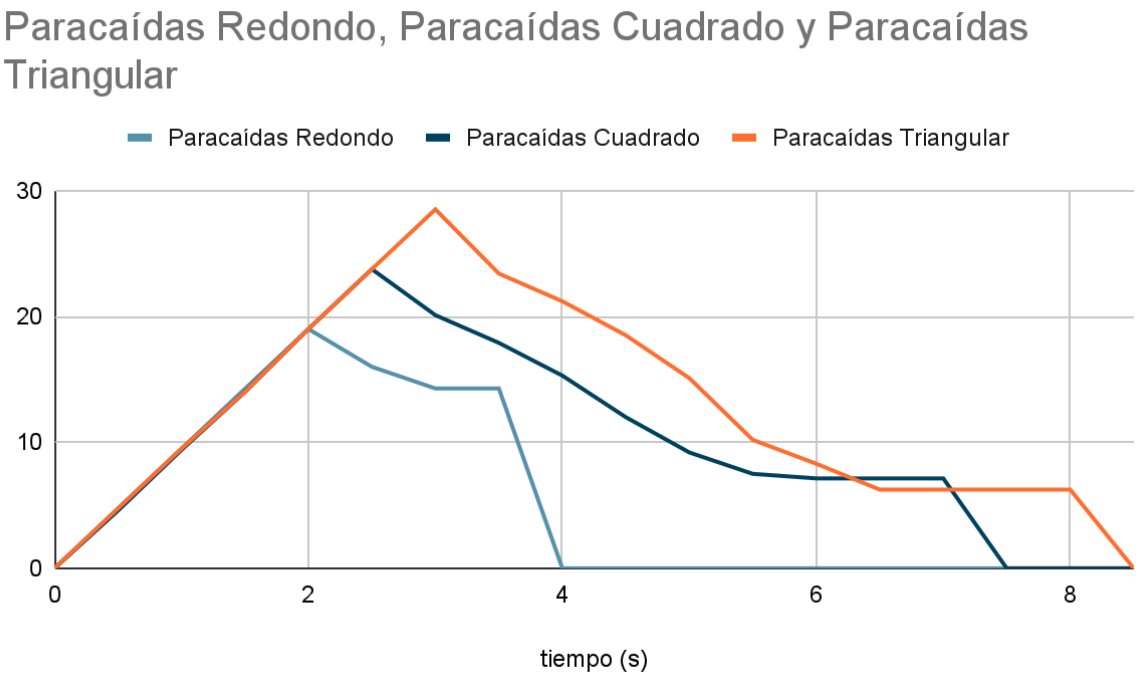
Lado del triángulo: 59,24 cm

Longitud del saliente: 11,85 cm

Este paracaídas está aún en fase de pruebas, pero está dando muy buenos resultados, y posiblemente sea el definitivo.

Paracaídas triangular

Gráficos de los paracaídas



2.7 Estación de tierra

Hemos comenzado las pruebas de comunicaciones entre el Arduino del CanSat y un ordenador que reciba, intérprete y almacene los datos transmitidos, a través de la antena RLYR 896.
https://how2electronics.com/reyax-rylr890-lora-module-with-arduino/#google_vignette

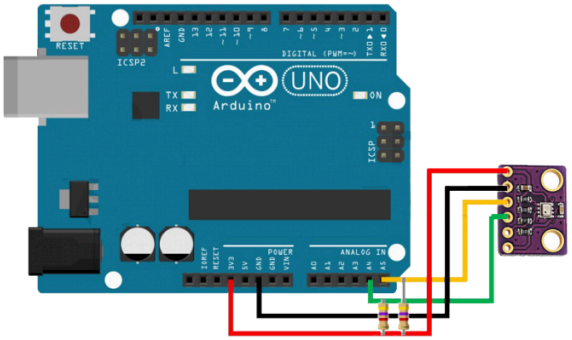
Esta antena posee una rango operativo de hasta 15 km, el cual supera con creces lo que requiere para la misión.

La transmisión de los datos se realizará de manera que haga al menos 1 por cada segundo, la cual es el mínimo impuesto por la competición, aunque la antena tiene la capacidad de emitir las señales cada menor periodo, y se tendrá en cuenta a la hora de crear el código final para la transmisión de datos.

Los dos encargados de la electrónica y programación estamos barajando la opción de crear un programa que te permita ver los datos en tiempo real de manera gráfica, no solo a través del monitor en serie, aparte de almacenarlos para que sea más fácil interpretarlos durante la misión.

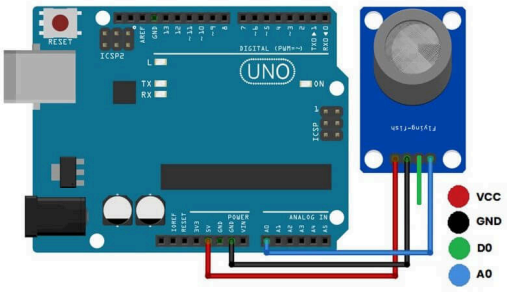
2.8 Software

BMP280



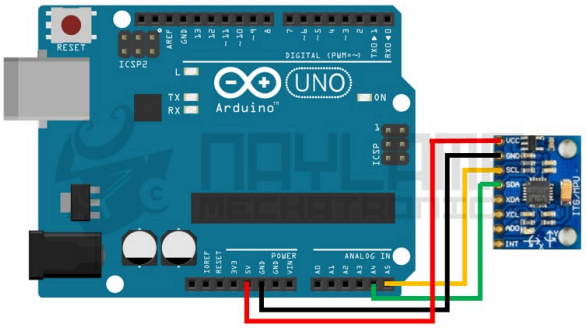
```C/C++
#include <Adafruit_Sensor.h>
#include "Adafruit_BMP280.h"

Adafruit_BMP280 bmp;

float presion;
float temperatura;
int altitud;

void setup() {
   bmp.begin();
}

void loop() {
   presion = bmp.readPressure()/100;
   temperatura = bmp.readTemperature();
   altitud = bmp.readAltitude (1015); // Ajustar con el valor local

   delay(1000);
}
```

MQ-135




```C/C++
#define MQ135pin (0)
float sensorValue; //variable para guardar el valor analógico del sensor

void setup()
{
  Serial.begin(9600); // Inicializamos el puerto serial a 9600
  Serial.println("El sensor de gas se esta pre-calentando");
  delay(20000); // Espera a que el sensor se caliente durante 20 segundos
}

void loop()
{
  sensorValue = analogRead(MQ135pin); // lectura de la entrada analogica
"A0""
  Serial.print("Valor detectado por el sensor: ");
  Serial.print(sensorValue);
  if(sensorValue > 300)
  {
   Serial.print(" | Se ha detectado gas!");
  }
  Serial.println("");
  delay(2000); // espera por 2 segundos para la siguiente lectura
}
```

MPU 6050

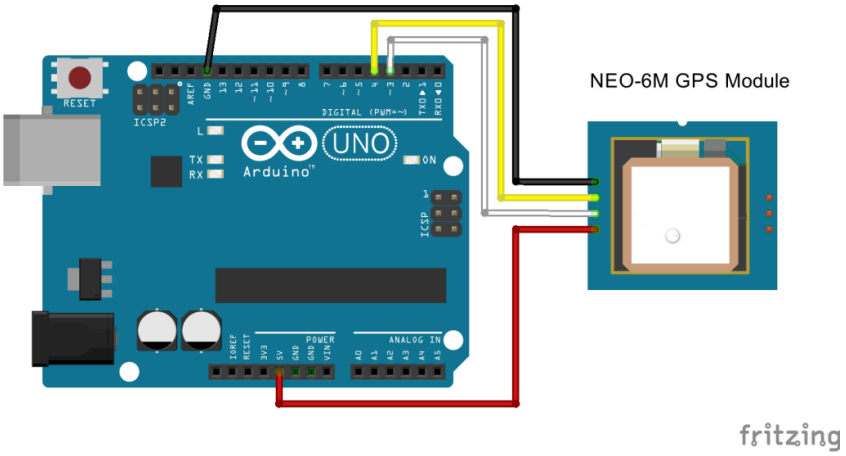


```C/C++
// Librerias I2C para controlar el mpu6050
// la libreria MPU6050.h necesita I2Cdev.h, I2Cdev.h necesita Wire.h
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estará implicito
MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;

void setup() {
  Serial.begin(57600);     //Iniciando puerto serial
  Wire.begin();            //Iniciando I2C
  sensor.initialize();     //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");
  else Serial.println("Error al iniciar el sensor");
}
```

```
void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  //Mostrar las lecturas separadas por un [tab]
  Serial.print("a[x y z] g[x y z]:\t");
  Serial.print(ax); Serial.print("\t");
  Serial.print(ay); Serial.print("\t");
  Serial.print(az); Serial.print("\t");
  Serial.print(gx); Serial.print("\t");
  Serial.print(gy); Serial.print("\t");
  Serial.println(gz);

  delay(100);
}
```

Neo 6M

https://github.com/mikalhart/TinyGPSPlus
https://randomnerdtutorials.com/guide-to-neo-6m-gps-module-with-arduino/




```
#include <TinyGPS++.h>
```

```
#include <SoftwareSerial.h>

static const int RXPin = 4, TXPin = 3;
static const uint32_t GPSBaud = 9600;

// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

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

void loop(){
  // This sketch displays information every time a new sentence is correctly
encoded.
  while (ss.available() > 0){
    gps.encode(ss.read());
    if (gps.location.isUpdated()){
      // Latitude in degrees (double)
      Serial.print("Latitude= ");
      Serial.print(gps.location.lat(), 6);
      // Longitude in degrees (double)
      Serial.print(" Longitude= ");
      Serial.println(gps.location.lng(), 6);

      // Raw latitude in whole degrees
      Serial.print("Raw latitude = ");
      Serial.print(gps.location.rawLat().negative ? "-" : "+");
      Serial.println(gps.location.rawLat().deg);
      // ... and billionths (u16/u32)
      Serial.println(gps.location.rawLat().billionths);
```

```
// Raw longitude in whole degrees
Serial.print("Raw longitude = ");
Serial.print(gps.location.rawLng().negative ? "-" : "+");
Serial.println(gps.location.rawLng().deg);
// ... and billionths (u16/u32)
Serial.println(gps.location.rawLng().billionths);

// Raw date in DDMMYY format (u32)
Serial.print("Raw date DDMMYY = ");
Serial.println(gps.date.value());

// Year (2000+) (u16)
Serial.print("Year = ");
Serial.println(gps.date.year());
// Month (1-12) (u8)
Serial.print("Month = ");
Serial.println(gps.date.month());
// Day (1-31) (u8)
Serial.print("Day = ");
Serial.println(gps.date.day());

// Raw time in HHMMSSCC format (u32)
Serial.print("Raw time in HHMMSSCC = ");
Serial.println(gps.time.value());

// Hour (0-23) (u8)
Serial.print("Hour = ");
Serial.println(gps.time.hour());
// Minute (0-59) (u8)
Serial.print("Minute = ");
Serial.println(gps.time.minute());
// Second (0-59) (u8)
Serial.print("Second = ");
Serial.println(gps.time.second());
// 100ths of a second (0-99) (u8)
Serial.print("Centisecond = ");
```

```
Serial.println(gps.time.centisecond());

// Raw speed in 100ths of a knot (i32)
Serial.print("Raw speed in 100ths/knot = ");
Serial.println(gps.speed.value());
// Speed in knots (double)
Serial.print("Speed in knots/h = ");
Serial.println(gps.speed.knots());
// Speed in miles per hour (double)
Serial.print("Speed in miles/h = ");
Serial.println(gps.speed.mph());
// Speed in meters per second (double)
Serial.print("Speed in m/s = ");
Serial.println(gps.speed.mps());
// Speed in kilometers per hour (double)
Serial.print("Speed in km/h = ");
Serial.println(gps.speed.kmph());

// Raw course in 100ths of a degree (i32)
Serial.print("Raw course in degrees = ");
Serial.println(gps.course.value());
// Course in degrees (double)
Serial.print("Course in degrees = ");
Serial.println(gps.course.deg());

// Raw altitude in centimeters (i32)
Serial.print("Raw altitude in centimeters = ");
Serial.println(gps.altitude.value());
// Altitude in meters (double)
Serial.print("Altitude in meters = ");
Serial.println(gps.altitude.meters());
// Altitude in miles (double)
Serial.print("Altitude in miles = ");
Serial.println(gps.altitude.miles());
// Altitude in kilometers (double)
Serial.print("Altitude in kilometers = ");
```

```
      Serial.println(gps.altitude.kilometers());
      // Altitude in feet (double)
      Serial.print("Altitude in feet = ");
      Serial.println(gps.altitude.feet());

      // Number of satellites in use (u32)
      Serial.print("Number os satellites in use = ");
      Serial.println(gps.satellites.value());

      // Horizontal Dim. of Precision (100ths-i32)
      Serial.print("HDOP = ");
      Serial.println(gps.hdop.value());
    }
  }
}
```

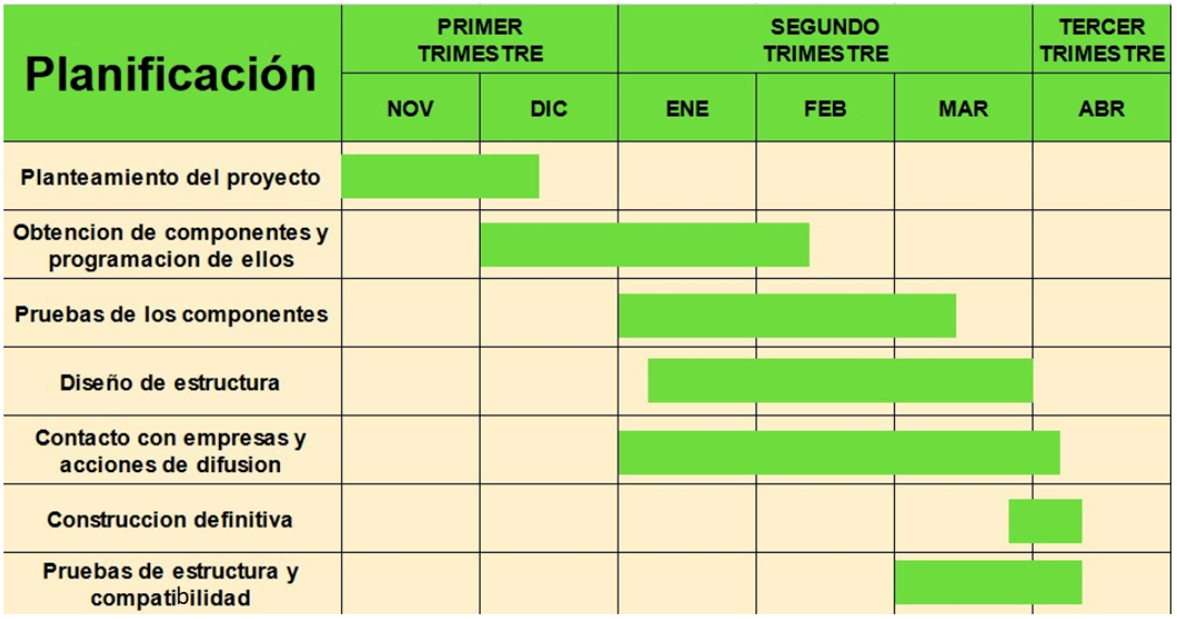
OV2640

Hemos empezado con el código de la cámara OV2640 usando las librerías Arducam y hemos conseguido grabar vídeos. Enlace a la librería utilizada

3. Planificación

3.1 Planificación del proyecto

Al principio del proyecto se planificó las fases del proyecto, la gestión de materiales y la organización de roles. Teniendo en cuenta la difusión del proyecto y apoyo externo.



3.2 Estimación del producto

3.2.1 Presupuesto

Componente	Precio unitario	Cantidad	Precio final
Arduino Nano	9,99€	2	19,98€
BMP 280	1,99€	1	1,99€
NEO 6MW2	9,99€	1	9,99€
Pilas 9V Recargables	11,42€	2	22,84€
Base carga pilas	11,99€	2	11,99€
MPU 6050	3,48€	1	3,48€
RLYR 896	18.00€	2	36.00€

OV2640	25.99€	1	25.99€
Pack de Cables (120)	8€	1	8€
Pack de Resistencias (400)	4,99€	1	4,99€
Tela Paracaídas Ripstop 2.8oz 2 metros	11,25€	1	11,25€
Hilo Monofilamento 0,5mmØ, 200m	5,99€	1	5,99€
Total: **154,49€**			

3.2.2 Financiación y apoyo externo

Estamos financiados por el departamento de tecnología del instituto Ramiro de Maeztu.

4. Programa de difusión

A lo largo del proyecto se han llevado a cabo varias actividades con el propósito de difundir y divulgar sobre el proyecto, el equipo y la competición.

- Comunicación boca a boca: Los miembros del equipo comentaron con sus familias el proyecto para que estos lo contarán en puestos de trabajo etc.

- R.R.S.S: El equipo posee una cuenta de instagram y twitter, en las cuales se publican avances del proyecto. También estamos desarrollando una página web, en la que publicaremos avances del proyecto con menos frecuencia pero más detalladas
    - Enlace a la cuenta de Instagram: @Cansat Ramiro24
    - Enlace a la cuenta de Twitter: @CanSat_Ramiro24

- Carteles: Se emplearon carteles para la difusión de las R.R.S.S del equipo los cuales se colocaron dentro del recinto escolar para dar a conocer el proyecto a otros alumnos.

5. Requisitos para lanzamiento

Características	Unidades
Altura del CanSat	115 mm
Diámetro del CanSat	66 mm
Masa del CanSat	350 g
Velocidad de descenso	8 m/s
Tiempo de vuelo	12´5 s
Frecuencia de transmisión	868 MHz
Autonomía	12 h
Coste total del CanSat	154’49€

6. Observaciones

Programas utilizados para la elaboración de este proyecto:

- Diagramas:
- Diseño Electrónico: Fritzing.
- Informe: Google Docs.
- Diseño mecánico e impresión 3D: Tinkercad.
- Programación CanSat: Arduino IDE.
- Programación estación de tierra:

7. Bibliografía

https://esero.es/wp-content/uploads/2021/09/plantilla-informecdr-cansat.pdf
https://esero.es/wp-content/uploads/2023/12/plantilla-pdr-cansat-2024-v3.pdf