JCAAC (2025), **3**, 53–64

SECTION: PROGRAMACIÓN DE DISPOSITIVOS ASTRONÓMICOS

Diseño, construcción y programación de dispositivos astronómicos: Construcción física, electrónica y *firmware*

Sergio Alonso[1] and Javier Flores[2]

[1] Dpto. de Lenguajes y Sistemas Informáticos, Universidad de Granada ; Sociedad Astronómica Granadina ; España. E-mail: zerjioi@ugr.es.
[2] Observatorio Astronómico de Calar Alto ; Sociedad Astronómica Granadina ; España. E-mail: jflores@caha.es.

Keywords: dispositivos astronómicos, *astronomical devices*, *hardware*, *software*, controlador, *driver*, INDI, INDIGO, panel de *flats*, *flats panel*

Resumen
En esta segunda entrega de la serie dedicada al diseño, construcción y programación de dispositivos astronómicos, se detallan los componentes mecánicos y electrónicos que se emplearán como referencia para la implementación de un panel de *darks/flats*. Asimismo, se presenta el diseño del circuito electrónico, el protocolo de comunicación y el desarrollo del *firmware* correspondiente que será la base para la creación de nuestro driver para INDIGO.

Abstract
In this second articule of the series dedicated to the design, construction and programming of astronomical devices, the mechanical and electronic components that will be used as a reference for the implementation of a darks/flats panel are described in detail. It also presents the design of the electronic circuit, the communication protocol and the development of the corresponding firmware, which will be the basis for the creation of our future INDIGO driver.

Continuación de la serie

En el primer artículo de la serie [1] presentamos los conceptos generales necesarios para poder desarrollar un dispositivo astronómico desde cero. En este artículo nos centraremos en la construcción de la parte física, electrónica y la programación de su *firmware*. En posteriores artículos desarrollaremos un driver INDIGO que nos permita controlar el dispositivo desde cualquier programa compatible con este protocolo, y en el último artículo de la serie programaremos algunos clientes que hagan uso específico del mismo.

El dispositivo que queremos desarrollar consiste en una tapadera para un telescopio (que evitará el polvo mientras no se utiliza la óptica) a la que le hemos añadido un panel luminoso que nos permitirá efectuar tomas de *flats* de manera automática. Además, se le ha añadido un sencillo (y no muy preciso, todo hay que decirlo) sensor de temperatura. Este dispositivo pretende ser didáctico más que plenamente funcional por lo que hemos simplificado su diseño y programación al máximo. Se deja como tarea para el lector interesado el pensar e implementar distintas mejoras como pueden ser: atenuación de la luminosidad del panel *flat*, control de la velocidad de apertura y cierre de la tapa, incorporar sensores más precisos o de otra naturaleza, guardado de los parámetros de configuración en la memoria EEPROM del microcontrolador, etc.

Para el desarrollo del dispositivo, en las siguientes secciones detallaremos los siguientes aspectos:

- Diseño de las piezas físicas del dispositivo.
- Definición del protocolo de comunicación que usará para comunicarse con el ordenador al que se conecte.
- Diseño de la electrónica del dispositivo (basada en la plataforma de microcontroladores Arduino).
- Programación del *firmware* del dispositivo.
- Primeras pruebas de comunicación y control usando el puerto serie de Arduino.

1. Introducción: Creación de dispositivos astronómicos

Los dispositivos astronómicos llevan con nosotros desde que empezamos a mirar hacia las estrellas. Nuestros ojos vieron en el cielo nocturno patrones que servirían para trazar mapas estelares a través de las constelaciones y los calendarios serían usados para controlar los cambios de estación y celebrar rituales. Con ello inevitablemente llegaron los primeros instrumentos astronómicos: dólmenes, gnomon, el astrolabio, etc. A lo largo de la historia ha habido grandes descubrimientos que nos han llevado un poco más lejos en el conocimiento del Universo, desde el telescopio de Galileo y el prisma de Newton hasta la aparición de placas fotográficas y CCDs, que aunque ofrecen una manera distinta de observar el cielo, nos han permitido entender la naturaleza de todos esos puntos brillantes que tanto nos fascinan.

En la actualidad, cuando hablamos de instrumentación astronómica tenemos que distinguir dos grandes grupos. Por un lado nos encontramos con el telescopio y todo lo asociado a él: óptica del propio telescopio, sistemas de movimiento en Alpha (AR) y Delta (DEC), sistemas para controlar el foco, sistemas para controlar la rotación del instrumento, sistemas para controlar cúpula y casetas, etc. Por otro lado tenemos el conjunto de instrumentos encargados de recoger la luz. Dependiendo de como queramos analizar esa luz necesitaremos unos dispositivos u otros. Por lo tanto distinguiremos entre telescopio y todo lo asociado a él y los instrumentos que recolectan la luz.

Desde el punto de vista de la instrumentación astronómica profesional al diseñar un instrumento, en la mayoría de los casos, se diseña totalmente a medida. Esto quiere decir que cada una de las piezas de ese instrumento (cámara, espectrógrafo, unidades de calibración, sistemas ópticos, etc) son cuidadosamente diseñados o, en el caso de detectores, elegidos para ese instrumento.

El diseño de un instrumento normalmente va ligado a un proyecto científico, por lo que tenemos que conocer los objetos que serán fruto de su estudio. También necesitamos conocer cuales son el resto de instrumentos disponibles en el mundo para crear algo novedoso que incremente la importancia y utilidad científica del instrumento diseñado. Por último, necesitamos conocer las limitaciones tecnológicas de la época en la que se construye el instrumento. Por lo tanto, en la construcción de instrumentación, aunque cada proyecto lleva uno o varios Investigadores Principales (PI) asociados, el nivel de participación de personas en distintas áreas de conocimiento es muy grande.

En el caso de la astronomía amateur normalmente no vamos a diseñar un instrumento completo, sino que optaremos por soluciones prácticas adaptadas a las necesidades de nuestro equipo particular; la razón es que, tanto a nivel profesional como aficionado, la creación de dispositivos astronómicos responde a la escasa comercialización o fabricación en serie de muchas unidades. Mientras que en el ámbito profesional el coste se prevé previamente y se integra en el presupuesto, en el amateur el factor económico resulta decisivo, pues aunque a veces existan en el mercado los accesorios que necesitamos, su producción limitada eleva notablemente los precios en comparación con el coste de los materiales. Por eso, aprovechando las nuevas tecnologías —impresión 3D (FDM y resina), corte láser, máquinas CNC, electrónica de bajo coste, etc.- podemos fabricar nosotros mismos las piezas e instrumentos que realmente necesitamos.

Para la creación de dispositivos existen varias alternativas: Por un lado podemos elegir un proyecto que

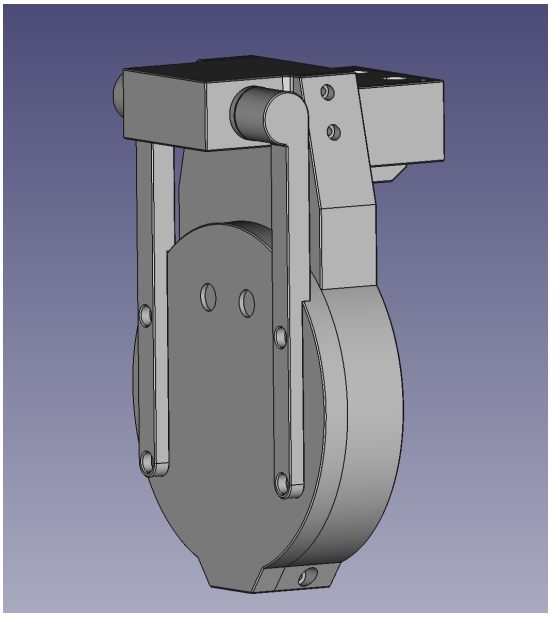


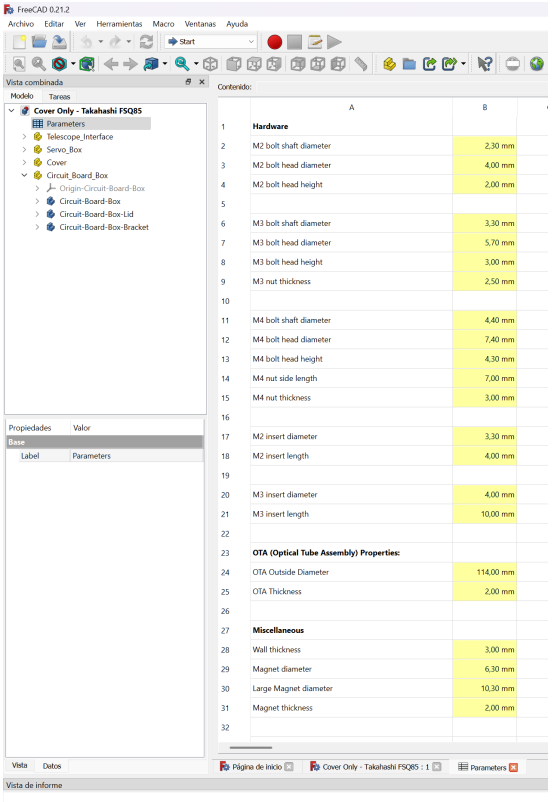
Figura 1. *Modelo 3D del panel de flats ASCOM-Compatible Telescope Cover.*

alguien ha desarrollado previamente e intentar reproducirlo, ya sea con mejoras o adaptado a nuestro caso particular o por otro lado podemos diseñar el dispositivo desde cero y completamente, desde el prototipado y montaje mecánico de las piezas hasta la programación de la electrónica.

2. Diseño de las piezas físicas del dispositivo

Para crear nuestro dispositivo debemos construir una serie de piezas que compongan el armazón donde se incrustarán los componentes mecánicos y electrónicos del mismo (panel LED, motor servo, etc). Para esta tarea, y por no reinventar la rueda, hemos optado por reutilizar un diseño libre existente en la red llamado ASCOM-Compatible Telescope Cover[2], creado por Julien Lecomte. Este diseño puede modificarse utilizando programas libres de diseño 3D paramétrico como es, por ejemplo, FreeCAD [3]. Existen numerosos recursos en línea sobre el uso y manejo de este software[4, 5, 6]. Podemos ver el modelo 3D desde FreeCAD en la Fig. 1. Es importante notar que de dicho proyecto solo vamos a utilizar el diseño de las piezas, ya que la electrónica, el *firmware*, *driver*, etc. será íntegramente desarrollado por nosotros.
Las piezas de dicho diseño pueden ser adaptadas ya que el autor de dicho diseño incorporó algunos parámetros (como la apertura del telescopio) para ajustar el dispositivo a las necesidades particulares de cada usuario, tal y como podemos ver en la pestaña `parameter` (Fig. 2) de FreeCAD.
Una vez ajustado el diseño a nuestras necesidades específicas se pueden imprimir las piezas con una impresora 3D. Para ello hay que exportar cada una de las piezas desde FreeCAD a un formato legible por el programa que usemos para poder hacer el loncheado (*slicing*), como podemos ver en la figura 3 y que permitirá a nuestra impresora producirlas. Lo más común para esta tarea es usar el formato STL, aunque existen otros. El material que hemos usado para la impresión es PLA, pero pueden usarse otros materiales como PETG, ABS, etc.
Una vez impresas, usaremos algunos tornillos, arandelas, tuercas y roscas insertables para ensamblar todas las piezas como podemos ver en la figura 4. En particular, para este diseño, se han utilizado:

- 4 tornillos M3*5mm + 4 tuercas M3



	A	B
1	**Hardware**	
2	M2 bolt shaft diameter	2,30 mm
3	M2 bolt head diameter	4,00 mm
4	M2 bolt head height	2,00 mm
5		
6	M3 bolt shaft diameter	3,30 mm
7	M3 bolt head diameter	5,70 mm
8	M3 bolt head height	3,00 mm
9	M3 nut thickness	2,50 mm
10		
11	M4 bolt shaft diameter	4,40 mm
12	M4 bolt head diameter	7,40 mm
13	M4 bolt head height	4,30 mm
14	M4 nut side length	7,00 mm
15	M4 nut thickness	3,00 mm
16		
17	M2 insert diameter	3,30 mm
18	M2 insert length	4,00 mm
19		
20	M3 insert diameter	4,00 mm
21	M3 insert length	10,00 mm
22		
23	**OTA (Optical Tube Assembly) Properties:**	
24	OTA Outside Diameter	114,00 mm
25	OTA Thickness	2,00 mm
26		
27	**Miscellaneous**	
28	Wall thickness	3,00 mm
29	Magnet diameter	6,30 mm
30	Large Magnet diameter	10,30 mm
31	Magnet thickness	2,00 mm
32		

Figura 2. *Parámetros modificables en FreeCAD del ASCOM-Compatible Telescope Cover.*

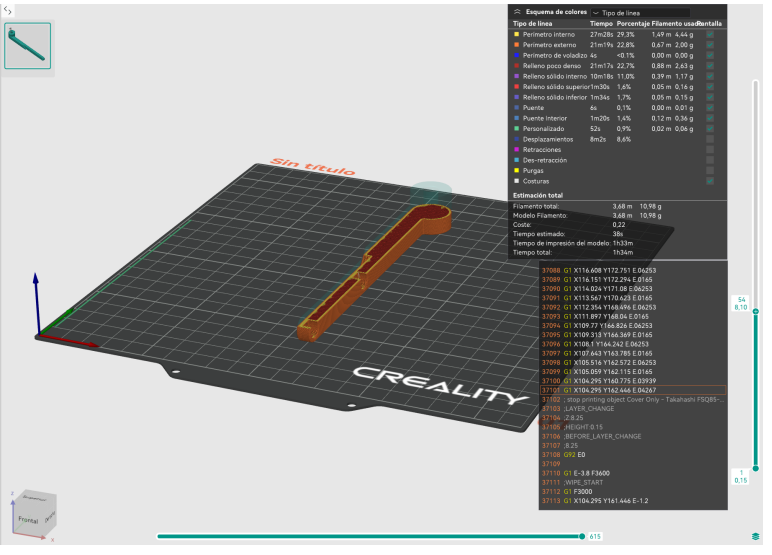


Figura 3. *Programa de slicing (OrcaSlicer) preparando un brazo del panel de flats.*

Figura 4. *Piezas impresas en PLA y montadas.*

- 4 tornillos M3*8mm + 4 roscas insertables M3
- 4 tornillos M3*10mm + 4 roscas insertables M3
- 4 tornillos M3*10mm + 4 tuercas M3
- 3 tornillos M3*10mm + 3 tuercas M3
- 4 tornillos M3*25mm + 4 arandelas M3
- 1 tornillo M4*30mm + 1 tuerca M4

Hemos puesto a disposición del lector dos modelos con los archivos en formato STL que se pueden descargar desde Github: https://github.com/zerjillo/TelescopeFlatsCover/tree/main/STL. Estos modelos son para un refractor Skywatcher Esprit 120ED y para un refractor Askar103APO.

3. Protocolo de comunicación

El protocolo de comunicación consiste en la especificación de los comandos (mensajes) que intercambiarán el ordenador y la electrónica de nuestros dispositivo a través del puerto serie (USB). Cuando se diseña un dispositivo es muy importante definir de manera clara este protocolo para poder luego implementar drivers u otros programas que quieran controlar directamente el dispositivo. Estas instrucciones deben permitir obtener toda la información sobre el dispositivo así como activar cada uno de los actuadores del mismo. Sin embargo, estos comandos deben ser lo más simples posible, dejando que de la lógica de alto nivel se encarguen los drivers o incluso los programas clientes que los usen.

Por ejemplo, para nuestro dispositivo debemos contar con algunos mensajes que permitan abrir o cerrar

Comando	Descripción	Respuesta	Ejemplo
`VERSION`	Pregunta por la versión del *firmware* del dispositivo	Mens. `VERSION`	
`STATUS`	Pide la información de estado del dispositivo (abierto, cerrado, encendido, apagado...)	Mens. `STATUS`	
`OPEN`	Pide que se abra la tapa del dispositivo	Mens. `STATUS`	
`CLOSE`	Pide que se cierre la tapa del dispositivo	Mens. `STATUS`	
`LON`	Pide que se encienda el panel de *flats*	Mens. `STATUS`	
`LOFF`	Pide que se apague el panel de *flats*	Mens. `STATUS`	
`START`	Pide que se comiencen a mandar mensajes de `STATUS` cada cierto intervalo	Mens. `STATUS`	
`STOP`	Pide que se pare de mandar mensajes de `STATUS` cada cierto intervalo	Mens. `STATUS`	
`INTERVAL INT`	Establece el intervalo de los mensajes de `STATUS`. El parámetro es un entero entre 1 y 3600 segundos	Mens. `STATUS`	`INTERVAL 30` Fijará el intervalo de mensajes a 30 segundos.

Cuadro 1. *Comandos hacia el dispositivo del protocolo de comunicación.*

Respuesta	Descripción	Ejemplo
`VERSION`	Devuelve la versión del *firmware* del dispositivo	`VERSION 1.00b`
`STATUS`	Devuelve la información de estado del dispositivo: ▪ cerrado o abierto (`0`/`1`), ▪ apagado o encendido el panel de *flats* (`0`/`1`), ▪ temperatura (grados Celsius), ▪ mandando mensajes de `STATUS` periódicamente o no (`1`/`0`), ▪ intervalo de los mensajes (segundos)	`STATUS 0 1 23.4 1 15` ▪ tapa está cerrada, ▪ panel de *flats* encendido, ▪ el sensor detecta 23.4 °C, ▪ se están mandando mensajes de `STATUS` periódicamente, ▪ cada 15 segundos

Cuadro 2. *Mensajes de respuesta del protocolo de comunicación.*

la tapa, encender o apagar el panel de *flats*, pero no deberíamos incorporar ningún mensaje que, por ejemplo, automatizase el cierre de la tapa, el encendido del panel durante un tiempo y luego volviese a apagarlo y abrirlo. De un control çomplejoçomo el descrito debería hacerse cargo el software del ordenador de control.

En la tabla 1 se muestran los mensajes (comandos) que atenderá nuestro dispositivo (es decir, que puede mandarle el ordenador de control). Cada uno de esos comandos se responderá con otro mensaje de vuelta tal y como se describen en la tabla 2.

Como se deduce de dichas tablas, todos los comandos, exceptuando el comando `VERSION` se responden con un mensaje `STATUS`. Ante un mensaje erróneo o inexistente el dispositivo lo ignorará (no responderá nada). Además, cuando el dispositivo se enciende, enviará una única vez la siguiente cadena de caracteres por el puerto serie hacia el ordenador: `Telescope Flat Cover connected and initialized.`

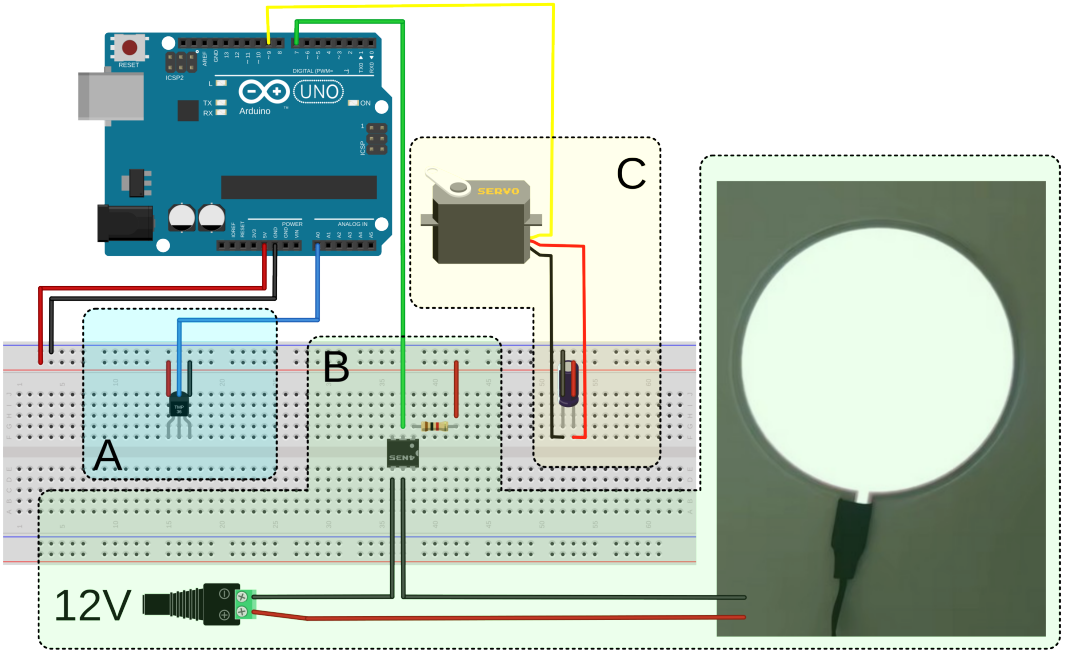


Figura 5. *Esquema de conexión de los componentes electrónicos.*

4. Diseño de la electrónica del dispositivo

Para la electrónica del dispositivo proponemos el uso de la plataforma Arduino [7]. Esta plataforma surgió como un intento de hacer accesible la programación de microcontroladores. Pese al carácter didáctico con el que nació se ha convertido en una plataforma ampliamente utilizada para proyectos de todo tipo. Existen numerosos manuales y tutoriales al respecto en la red [8].

Arduino en realidad se refiere hoy en día a todo un ecosistema de distintas placas electrónicas con un microcontrolador acoplado que permiten interconectar diferentes sensores y actuadores, junto con otros componentes como un puerto USB para programarlo y comunicarlo con el exterior. Para un proyecto como el nuestro se puede usar cualquier placa de la familia de Arduino, pero para los esquemas y prototipos hemos utilizado una placa Arduino UNO. Para el proyecto final podría usarse otra placa más pequeña a la que soldarle los componentes (en vez de montarlos en una placa de prototipado como en las imágenes que acompañan este trabajo). Para programar el *firmware* existe el entorno de programación (IDE) de Arduino, que puede descargarse gratuitamente de la página oficial [7]. Los componentes principales que usaremos en este proyecto (todos bastante baratos y fáciles de conseguir en línea o en tiendas de electrónica) son:

- Placa Arduino UNO
- Sensor de temperatura `TMP36`
- Optoacoplador `4N35`
- Resistencia de 1kΩ.
- Un servomotor
- Un condensador (100µf)
- Una fuente de alimentación de 12V (para el panel de *flats*)
- Panel de *flats* redondo.

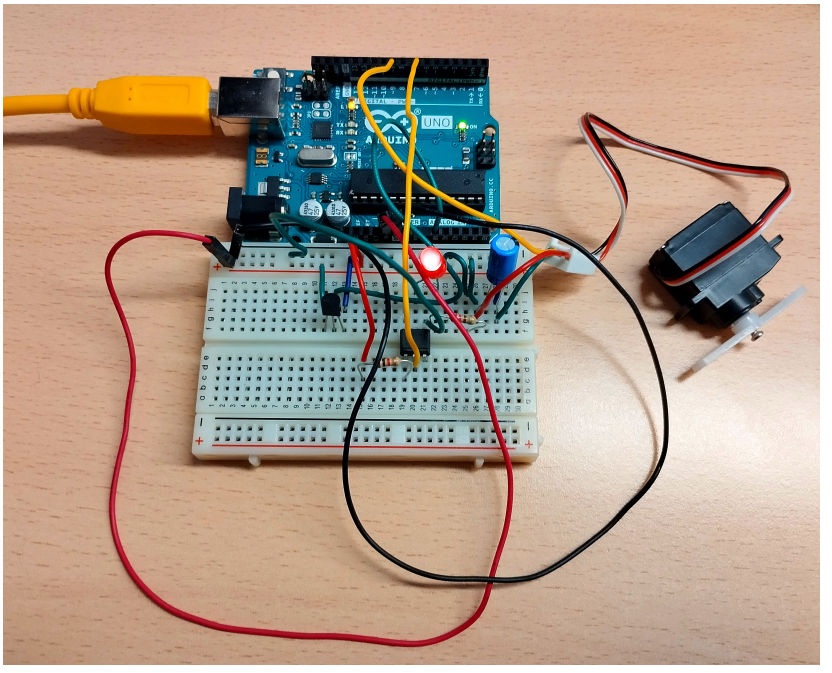


Figura 6. *Montaje de Arduino UNO.*

Dichos componentes se conectan entre sí de acuerdo con el esquema de la figura 5. El circuito tiene 3 bloques principales:

- **Bloque A:** Contiene el sensor de temperatura `TMP36`. Dicho sensor tiene tres patillas: alimentación a `5V`, tierra (`GND`) y la intermedia que conectaremos a la entrada analógica `A0` del Arduino.
- **Bloque B:** Contiene los componentes para controlar el panel de *flats*. Como dicho panel funciona a un voltaje distinto del Arduino (`12V` frente a los `5V` de Arduino) utilizamos un optoacoplador `4N35` que permite aislar el circuito del panel de *flats*. Simplificando mucho, un optoacoplador funciona como un interruptor que puede ser activado por una señal eléctrica. En nuestro caso cierra o abre el circuito de `12V` del panel según sea la señal que le llega desde el pin `7` del Arduino.
- **Bloque C:** Contiene los componentes para controlar el servo motor. Un servomotor es un tipo de motor que puede controlar el ángulo de giro de su vástago según una señal que le proporcionemos desde el microcontrolador (en nuestro caso a través del pin `9` del Arduino). Dado que los motores pueden introducir algunas señales eléctricas en la circuitería de Arduino se recomienda usar un condensador de 100µf entre los dos polos de alimentación del servo para atenuar las posibles interferencias.

En la Fig. 6 se puede ver una foto del prototipo mientras se estaba desarrollando, donde el panel de flats se estaba aún probando como un simple led rojo.

5. Programación del *firmware* del dispositivo

Arduino tiene un lenguaje de programación propio [9] con una sintaxis muy parecida a `C` y `C++`. El lenguaje incorpora además los mecanismos necesarios para activar los distintos pines de entrada/salida de la placa (`I/O`) tanto como para la lectura de señales analógicas (en los pines `A0-A6`), control del puerto serie del microcontrolador (junto con el conversor a puerto USB, que es el que realmente conectaremos al ordenador), así como una gran cantidad de bibliotecas que simplifican el control de muchos motores

(de corriente continua, paso a paso, servos...), actuadores y sensores existentes en el mercado.

A continuación describiremos las secciones de código principales del *firmware* con el que programaremos nuestra placa Arduino para poder controlar los distintos componentes del dispositivo. Dicho código completo puede descargarse del siguiente repositorio de Github: https://github.com/zerjillo/TelescopeFlatsCover.

Para el desarrollo del *firmware* hacemos uso de dos bibliotecas de Arduino. La primera se llama `CommandParser` [10] y tiene como finalidad facilitar el análisis de comandos que se reciban en el microcontrolador (por ejemplo a través del puerto serie). Esta biblioteca no está instalada en el IDE de Arduino por defecto, por lo que es posible que para usarla haga falta instalarla desde `Herramientas ->Gestionar bibliotecas...` del entorno de programación.

Esta biblioteca permite definir una serie de comandos de texto, con parámetros opcionales, de tal manera que cuando se recibe uno de dichos mensajes se llamará a una función *callback* que previamente habremos definido. De esta manera el análisis (*parsing*) de los comandos se simplifica enormemente. En el listado 1 se muestran las secciones de código fundamentales que tienen que ver con el análisis de los comandos recibidos por Arduino. Como se puede apreciar definimos una serie de funciones *callback* (las hemos llamado todas con el prefijo `cmd_`). Cada una de esas funciones incluye el código necesario para responder ante un comando particular recibido. En la función setup() de Arduino, que se ejecuta una única vez cada vez que la placa se enciende y se utiliza para inicializar la placa, activamos e inicializamos el puerto serie y registramos cada uno de los comandos que la placa debe reconocer y la función *callback* a la que tendría que llamar. Así por ejemplo, la instrucción:

```
parser.registerCommand("INTERVAL", "i", &cmd_updateInterval);
```

indica que si recibimos el mensaje `INTERVAL INT`, donde INT es un parámetro entero (`"i"`) debe llamarse a la función *callback* `cmd_updateInterval()` definida anteriormente. Por último, para gestionar la recepción de comandos, en la función `loop()` (que se está ejecutando de manera iterativa durante todo el tiempo que la placa está encendida) comprobamos si existen mensajes por leer en el puerto serie y, en caso afirmativo, le mandamos a la biblioteca `CommandParser` que analice el mensaje y llame a la función *callback* apropiada:

```
parser.processCommand(line, response);
```

```
#include <CommandParser.h>    // Incluimos la biblioteca CommandParser, previamente instalada en el Arduino IDE
...
typedef CommandParser<> MyCommandParser;  // Objeto CommandParser
MyCommandParser parser;
...
// Funciones callback que atienden cada uno de los comandos que se pueden recibir
void cmd_version(MyCommandParser::Argument *args, char *response) { ... }
void cmd_status(MyCommandParser::Argument *args, char *response) { ... }
void cmd_open(MyCommandParser::Argument *args, char *response) { ... }
void cmd_close(MyCommandParser::Argument *args, char *response) { ... }
void cmd_lightOn(MyCommandParser::Argument *args, char *response) { ... }
void cmd_lightOff(MyCommandParser::Argument *args, char *response) { ... }
void cmd_startUpdating(MyCommandParser::Argument *args, char *response) { ... }
void cmd_stopUpdating(MyCommandParser::Argument *args, char *response) { ... }
void cmd_updateInterval(MyCommandParser::Argument *args, char *response) { ... }
...
// La funcion setup() se ejecuta una unica vez cuando la placa Arduino se enciende
void setup() {
  Serial.begin(9600);  // Inicializamos el puerto Serie
  while (!Serial);
...
// Registramos todos los comandos disponibles
  parser.registerCommand("VERSION", "", &cmd_version);
  parser.registerCommand("STATUS", "", &cmd_status);
  parser.registerCommand("OPEN", "", &cmd_open);
  parser.registerCommand("CLOSE", "", &cmd_close);
  parser.registerCommand("LON", "", &cmd_lightOn);
```

```
  parser.registerCommand("LOFF", "", &cmd_lightOff);
  parser.registerCommand("START", "", &cmd_startUpdating);
  parser.registerCommand("STOP", "", &cmd_stopUpdating);
  parser.registerCommand("INTERVAL", "i", &cmd_updateInterval);   // Con un parametro entero entre 1 y 3600
...
}

// La funcion loop() se ejecuta continuamente (en un bucle) mientras que la placa Arduino este encendida
void loop() {
  if (Serial.available()) {  // Si hay mensajes por leer en el puerto serie
    char line[128];
    size_t lineLength = Serial.readBytesUntil('\n', line, 127);  // Read serial port
    line[lineLength] = '\0';

    char response[MyCommandParser::MAX_RESPONSE_SIZE];
    parser.processCommand(line, response); // Analiza el mensaje y llama a la funcion callback correspondiente
  }
...
}
```

Listado 1. *Código del firmware relacionado con el análisis de los comandos.*

Para el control del servo motor hacemos uso de la biblioteca `Servo` (listado 2). Esta biblioteca permite de manera sencilla especificar el ángulo en el que debe posicionarse el vástago del motor llamando a la función:

```
servo.write(angulo);
```

En nuestro caso esa función la llamamos desde la función `setActuators()` que es la que se encarga de comprobar la posición en la que debe estar la tapa (variable `coverOpen` que habrá sido establecida por las funciones *callback* al recibirse los mensajes `OPEN` o `CLOSE`).

```
...
#include <Servo.h>
...
const int SERVO_PIN = 9;
const int CLOSED_ANGLE = 0;
const int OPENED_ANGLE = 90;
...
bool coverOpen = false;  // Esta abierta la tapa?
...
Servo servo;             // El objeto para controlar el motor
...
void setActuators() {
  if (coverOpen) {
    servo.write(OPENED_ANGLE);
  } else {
    servo.write(CLOSED_ANGLE);
  }
...
}
...
void setup() {
...
  servo.attach(SERVO_PIN);   // Initializamos la biblioteca servo con el pin al que lo tenemos conectado
...
}
...
```

Listado 2. *Código del firmware relacionado con el control del motor servo.*

El control del encendido o apagado del panel de *flats* es más sencillo, pues solo tenemos que activar o no el pin donde hemos conectado el optoacoplador (listado 3). Ese pin lo activamos o desactivamos también desde la función `setActuators()`.

```
...
const int OPTO_PIN = 7;
...
```

```
bool lightOn = false;    // Esta el panel encendido?
...
void setActuators() {
...
  if (lightOn) {
    digitalWrite(OPTO_PIN, LOW);
  } else {
    digitalWrite(OPTO_PIN, HIGH);
  }
...
}
...
void setup() {
...
  pinMode(OPTO_PIN, OUTPUT); // Inbicializamos el optoacoplador par aencender y apagar el panel
...
}
...
```

Listado 3. *Código del firmware relacionado con el encendido y apagado del panel de flats.*

La obtención de la temperatura desde el sensor `TMP36` es también simple: solo hay que hacer una lectura del pin analógico donde se ha conectado dicho sensor y hacer algunas operaciones aritméticas para transformar el nivel de la señal eléctrica medida a la temperatura en °C (listado 4).

```
...
const int TEMP_SENSOR = A0;
...
float getTemperature() {
  int read = analogRead(TEMP_SENSOR);
  float temperature = (read * (500.0 / 1023.0)) - 50.0;  // En grados Celsius
  return temperature;
}
...
```

Listado 4. *Código del firmware relacionado con el sensor de temperatura.*

6. Primeras pruebas de comunicación y control (puerto serie)

Para comprobar el correcto funcionamiento de nuestro dispositivo y de su *firmware* podemos utilizar el monitor serie del IDE de Arduino (`Herramientas ->Monitor Serie`) que nos abrirá un panel inferior desde el que podremos mandar comandos al microcontrolador así como ver los mensajes de respuesta del mismo. Podemos por tanto ir haciendo pruebas de cada uno de los comandos definidos y comprobando que los actuadores funcionan correctamente (el motor se mueve y el panel de *flats* se enciende y apaga). En la Fig. 7 se muestra el IDE de Arduino con el monitor serie activado y con algunos comandos mandados a nuestro dispositivo y algunas respuestas recibidas. Queda como ejercicio para el lector descubrir que comandos se le enviaron al dispositivo para obtener los mensajes de respuesta que se observan en la figura.

7. Conclusiones

En este trabajo se ha presentado la parte mecánica, electrónica (basada en la plataforma Arduino) y el *firmware* de un dispositivo astronómico consistente en una tapa para telescopio que incorpora un panel de *flats* y un sensor de temperatura. Además, se ha especificado el protocolo de comunicación que permite mandar comandos desde el ordenador donde se conecte el dispositivo así como las respuestas que el dispositivo realiza ante los comandos recibidos.

En los próximos artículos de la serie construiremos un driver INDIGO que nos permita controlar este instrumento desde cualquier programa de control de dispositivos astronómicos compatible con INDI o

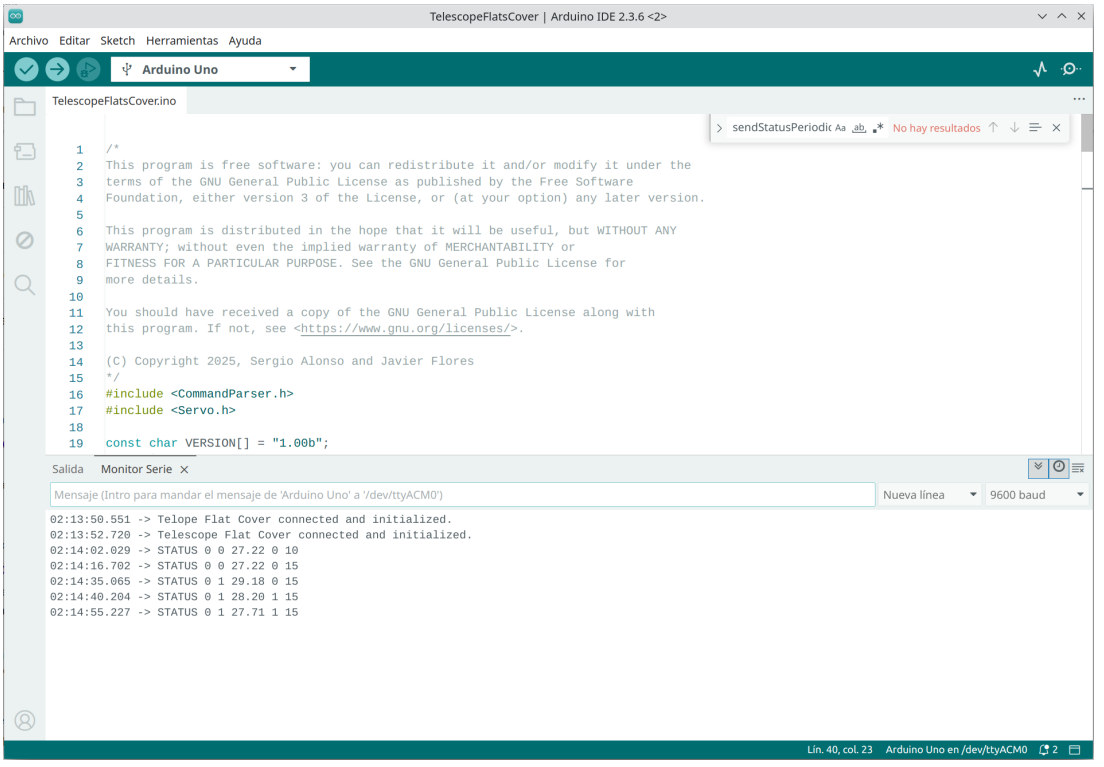


Figura 7. *IDE de Arduino con el monitor serie activo para comprobar la comunicación y funcionamiento del dispositivo.*

INDIGO así como directamente programar un cliente que haga uso del mismo dispositivo de manera programática.

Referencias

[1] S. Alonso y J. Flores, *Diseño, construcción y programación de dispositivos astronómicos: una introducción*, JCAAC, **2**, 75–86 (2025).

[2] Ascom-telescope-cover-v2. Disponible en https://github.com/jlecomte/ascom-telescope-cover-v2

[3] FreeCAD. Tu modelador paramétrico 3D. Accesible y descargable en: https://www.freecad.org/

[4] Impresión 3D. Dispositivos para observatorios en remoto - 1ª parte. Disponible en: https://www.youtube.com/watch?v=IlIANo79QKM&ab_channel=FAAE

[5] Impresión 3D. Dispositivos para observatorios en remoto - 2ª parte. Disponible en: https://www.youtube.com/watch?v=54UVxNeS8cQ&ab_channel=FAAE

[6] Tutorial FreeCAD. J. González Gómez. Disponible en: https://www.youtube.com/watch?v=2_DbFzFV9D4.

[7] Plataforma Arduino. Disponible en: https://www.arduino.cc/

[8] Tutoriales de Arduino y otros microcontroladores por Luis Llamas. Disponibles en: https://www.luisllamas.es/tutoriales-arduino/

[9] Arduino Languege Reference. Disponible en: https://docs.arduino.cc/language-reference/

[10] Biblioteca `CommandParser` para Arduino. Disponible en: https://github.com/Uberi/Arduino-CommandParser