Capítulo 1

Sniffer i^2c

El cuadricóptero adquirido resuelve la comunicación entre los controladores de los motores (**ESCs**) y el microprocesador mediante el protocolo **i²c**.

Para el presente proyecto resulta de vital importancia conocer dicho protocolo, ya que se utilizará otro microprocesador que deberá comandar a esos mismos ESCs, supliendo el trabajo del anterior. Es entonces imprescindible conocer al detalle el funcionamiento de este protocolo, para luego poder reproducirlo.

Dado que no se cuenta con la colaboración de los fabricantes, y toda la información parece ser privativa, no pudiendo conseguir dato alguno de su implementación, es necesario realizar un proceso de ingeniería inversa para poder analizar, decodificar, entender y reproducir el protocolo existente. Dicho proceso de ingeniería inversa se realiza utilizando el hardware existente del cuadricóptero comercial adquirido y un analizador lógico[1] que es capaz de leer e interpretar las líneas del bus i^2c sin intervenir en las mismas.

Antes de presentar los resultados obtenidos en el proceso, se realiza una breve introducción al protocolo i^2c y se presenta en la figura 1.1 la definición de los ejes a utilizar, lo cual será de utilidad más adelante.

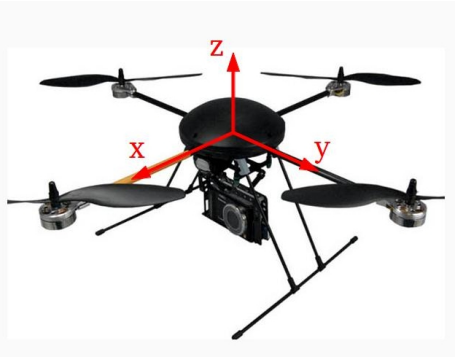


Figura 1.1: Definición de ejes

[1] ChronoVu

1.1. Introducción al protocolo i^2c

El bus i^2c es un bus de comunicaciones serie. Su nombre viene de *Inter-Integrated Circuit* (Circuitos Inter-Integrados).
Utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. Además será necesaria una tercera línea de tierra, como referencia.
En la imagen [2] se muestra un diagrama de un circuito equivalente simplificado de la conexión i^2c entre 2 dispositivos.

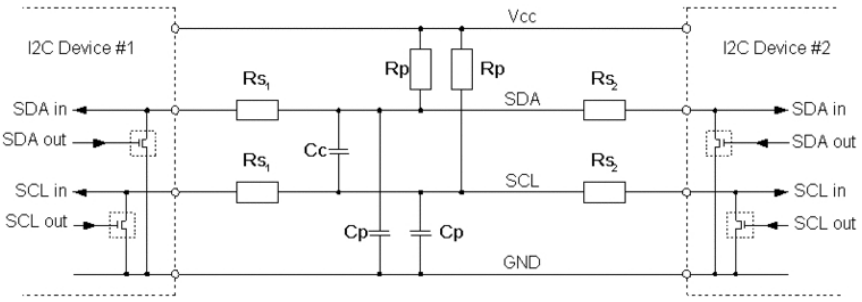


Figura 1.2: Conexión i^2c.

donde:

```
Vcc  Voltaje de entrada, típicamente varía entre 1.2 V y 5.5 V
GND  Tierra común
SDA  Línea serial de datos
SCL  Línea serial de reloj
Rp   Resistencia de "Pull-up"
Rs   Resistencia serie
Cp   Capacitancia del cable
Cc   Capacitancia de canal cruzado
```

Las líneas **SDA** y **SCL** son de drenador abierto, lo que significa que tanto el maestro como los esclavos solamente pueden conducir a nivel bajo estas líneas o dejarlos abiertos. Si ningún dispositivo i^2c está conduciendo hacia abajo la línea, la resistencia de *pull-up* R_p se encarga de conducir la línea a V_cc.

Cada dispositivo tiene asignada una dirección que lo identifica. Para establecer una comunicación la secuencia típica empieza por el maestro enviando una secuencia de comienzo de conexión, seguida de la dirección del esclavo con el cual desea comunicarse. Seguidamente el maestro envía un bit que determina si la acción que desea realizar es escritura o lectura, a lo que el esclavo correspondiente responde con un bit de *acknowledge* (**Ack**). Luego el maestro envía la dirección de memoria interna del esclavo donde debe ser almacenada la información enviada, y por último envía los datos. Para finalizar la conexión, el maestro envía una secuencia de fin de conexión.

[2] Imagen tomada de www.i2c-bus.org

1.2. Pruebas en régimen

Como se dijo anteriormente, para realizar el proceso de ingeniería inversa se utilizó el cuadricóptero y un analizador lógico. La forma de operar es enviar comandos conocidos con el control remoto y analizar los datos que se cursan en las líneas del bus. En la figura 1.3 se muestra una foto del proceso descripto.

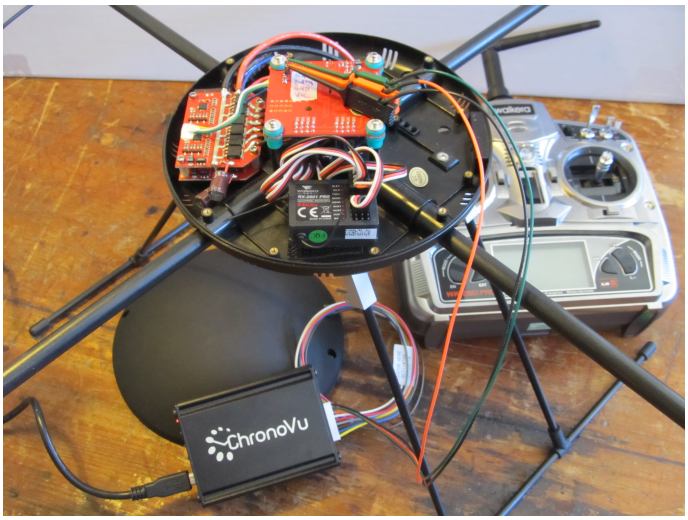


Figura 1.3: Proceso de lectura del bus.

Al realizar este proceso se puede observar que se repiten cada $2ms$ bloques similares al mostrado en la figura 1.4.

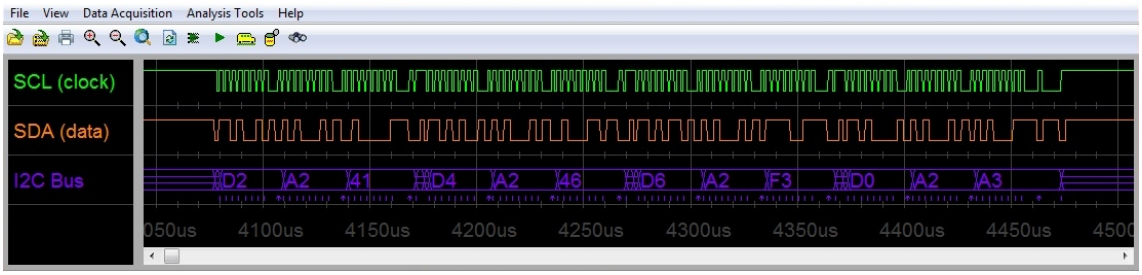


Figura 1.4: Bloque de transmisión i^2c

Rápidamente, visto lo anterior, se puede deducir que hay 4 esclavos correspondientes a los 4 ESC's de los 4 motores, cuyas direcciones en hexadecimal son **D0**, **D2**, **D4** y **D6**. La dirección de la memoria interna de los esclavos donde se almacenan los datos que envía el maestro es, en todos los casos **A2**. Además el maestro envía un tercer conjunto de datos que refiere, de alguna manera, a la velocidad con la que debe girar cada motor. Este conjunto de órdenes, agrupadas por bloques como el que se muestra en la figura 1.4, se repite periódicamente, indicando la velocidad con la que debe girar cada motor.

Para comprender las pruebas realizadas es importante dejar en claro algunos aspectos previos. En la figura 1.5 se muestra el transmisor utilizado para enviar comandos al cuadricóptero, un **Walkera WK-2801** y se indican los nombres de los mandos más importantes del mismo.

Figura 1.5: Transmisor

Al mover el mando de la izquierda (**Elev/Rudder**) en la dirección vertical (**Elev**) se logra que el cuadricóptero se eleve verticalmente, dando igual potencia a todos los motores, mientras que al moverlo en la dirección horizontal, el cuadricóptero presenta un movimiento de rotación según su eje vertical (que pasa por el centro).

Al mover el mando de la derecha (**Throttle/Aile**) en la dirección horizontal (**Aile**) y vertical (**Throttle**), se logran movimientos de rotación según los ejes horizontales x e y del cuadricóptero. Las definiciones de los ejes se pueden ver en la figura 1.1.

Se analizan las siguientes situaciones:

Id	Elev	Ruddle	Aile	Throttle	Movimiento
0	atrás[3]	medio	medio	medio	idle
1	medio	medio	medio	medio	vertical hacia arriba con aceleración constante
2	adelante	medio	medio	medio	vertical hacia arriba con aceleración constante
3	medio	izquierda	medio	medio	giro según eje z
4	medio	derecha	medio	medio	giro según eje $-z$
5	medio	medio	izquierda	medio	giro según eje $-x$
6	medio	medio	derecha	medio	giro según eje x
7	medio	medio	medio	atrás	giro según eje $-y$
8	medio	medio	medio	adelante	giro según eje y

Cuadro 1.1: Pruebas realizadas

Se realiza un análisis de los resultados obtenidos para cada motor, graficando el contenido del byte de datos que se le transmite a cada motor. Se obtienen representaciones como la mostrada en la figura 1.6

En la figura 1.6 se puede observar que a los 4 motores les llega un byte con el valor promedio en 50, el cual corresponde a la mínima potencia entregada a los motores para encenderlos.

Haciendo un análisis similar con el resto de las pruebas detalladas en la tabla 1.1 se construye la tabla 1.2 donde se muestran los valores enviados a cada motor en promedio en todas las pruebas.

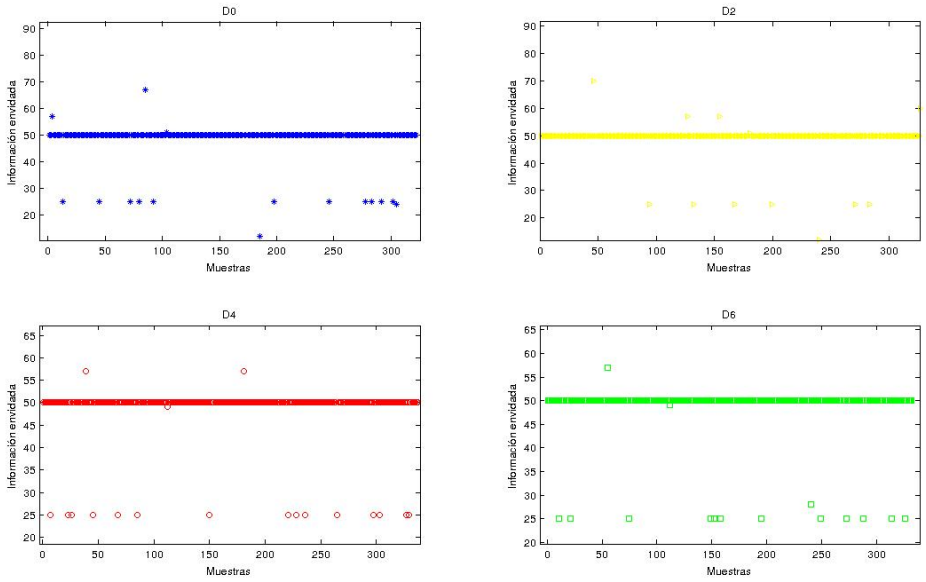


Figura 1.6: Prueba N^o 0

	Valor promedio				
Prueba	D0	D2	D4	D6	Movimiento
0	50	50	50	50	Idle
1	80	140	180	140	$a_z = cte \neq 0$
2	180	220	250	240	$a_z = cte \neq 0$
3	160	70	60	250	giro según eje z
4	50	200	200	100	giro según eje $-z$
5	250	160	140	50	giro según eje $-x$
6	50	160	150	250	giro según eje x
7	90	50	250	160	giro según eje $-y$
8	110	250	50	100	giro según eje y

Cuadro 1.2: Resumen de los resultados obtenidos

De dicha tabla se pueden sacar algunas conclusiones que se analizarán en la sección 1.5;

1.3. Prueba de arranque

En esta sección se analiza la secuencia de arranque del cuadricóptero, para obtener conclusiones sobre la misma. Se parte con la palanca de elevación al mínimo y se procede a moverla para hacer arrancar los motores.

Mientras la palanca de elevación se mantiene al mínimo, se le envía el valor 0 a los cuatro motores. Al mover la palanca, luego de un *tiempo muerto* donde las líneas quedan inactivas, se les empieza a mandar el valor correspondiente a cada motor, como se puede ver en la figura 1.7.

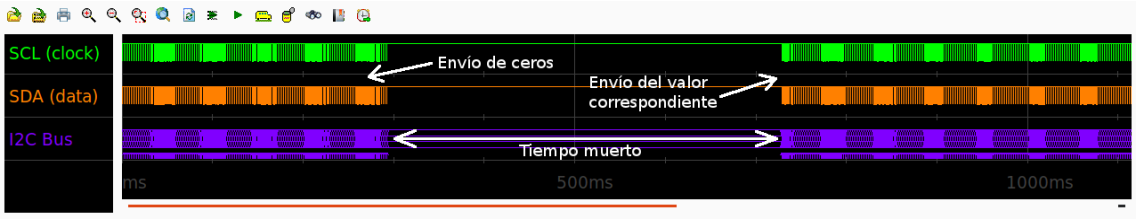
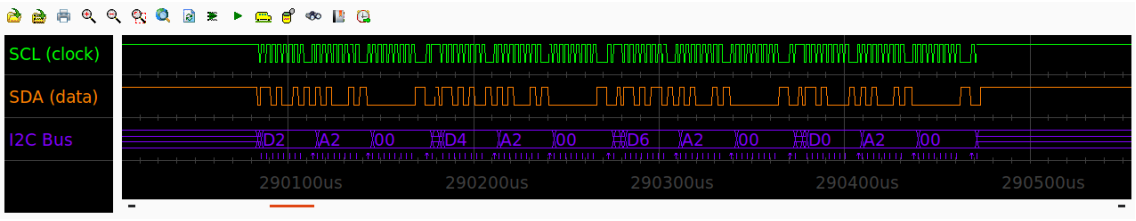
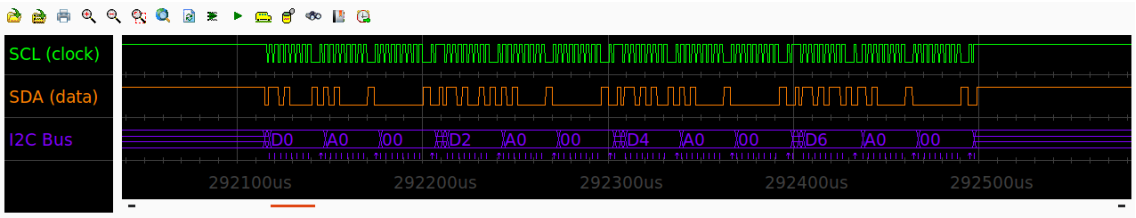


Figura 1.7: Arranque

Observando más detenidamente los comandos enviados se pueden sacar algunas conclusiones importantes. En la gran mayoría de los casos se envían los comandos en el orden $D2 \rightarrow D4 \rightarrow D6 \rightarrow D0$ y se guardan en el registro 0xA2 del esclavo. Pero se puede observar una importante diferencia en la el último comando que se manda a cada motor antes de arrancar, es decir, el último comando en la tanda de ceros mostrada en la figura 1.7. En la figura 1.8a se muestra una tanda regular de ceros y en la figura 1.8b se muestra la última tanda de ceros mencionada.



(a) Tanda normal de ceros



(b) Última tanda de ceros

Figura 1.8: Envío de ceros

Se observan 2 claras diferencias:

- **El orden en el que se envían datos a los motores no es mismo** que el anterior. En este caso el orden es: $D0 \rightarrow D2 \rightarrow D4 \rightarrow D6$.
- **El registro en el que se escriben los datos es 0xA0**

1.4. Prueba de frenado

Se procede del mismo modo que en la prueba anterior (sección 1.3), analizando los comandos enviados a los motores a la hora de apagarlos. Se realiza una prueba análoga a la anterior partiendo de los motores funcionando y bajando al mínimo la palanca de elevación, causando que estos se detengan.

Una vez que se lleva la palanca de elevación al mínimo, los motores se apagan y se permanece enviando ceros a los motores en el orden $D2 \rightarrow D4 \rightarrow D6 \rightarrow D0$. Se pueden sacar algunas conclusiones importantes de la última tanda de valores distintos de cero enviados:

- **El orden en el que se envían datos es** $D0 \rightarrow D2 \rightarrow D4 \rightarrow D6$
- **El registro en el que se escriben los datos es 0xA1**

1.5. Conclusiones

- La comunicación entre el amo y los esclavos por medio del protocolo i^2c se lleva a cabo mediante un formato del tipo

  ```
  Dirección esclavo - Lugar de memoria donde guardar dato - dato
  ```

- Dicho formato se repite para todos los esclavos
- Cada esclavo recibe una actualización de estado (un dato nuevo) cada $2ms$
- La velocidad mínima de funcionamiento se logra enviando el valor 50
- La velocidad máxima de funcionamiento se logra enviando el valor 250
- Cuando el mando de Elevación del control se encuentra al mínimo, se les envía el valor 0 a los cuatro motores periódicamente con el formato mencionado.
- La dirección de memoria interna de todos los esclavos donde se guardan los datos recibidos por el maestro es siempre 0x$A2$ (ó 162), a excepción del arranque y el frenado.
- El último comando enviado antes de arrancar se guarda en el registro 0xA0.
- El último comando enviado antes de frenar se guarda en el registro 0xA1.
- El orden normal en el que se envían los comandos es $D2 \rightarrow D4 \rightarrow D6 \rightarrow D0$, menos en los casos descriptos en los 2 puntos anteriores donde el orden es $D0 \rightarrow D2 \rightarrow D4 \rightarrow D6$.
- La correspondencia entre las direcciones y los motores se muestra en la figura 1.9. Mirando el cuadricóptero desde arriba, el motor con dirección 0xD0 es el de la derecha, el motor con dirección 0xD6 el de la izquierda, el 0xD4 el de adelante y el 0xD2 el de atrás.
- Las direcciones 0xD0, 0xD2, 0xD4 y 0xD6 (de 8 bits) en realidad no refieren únicamente a direccionamiento, sino que incluyen además el bit de lectura/escritura, por lo cual cada esclavo tendrá una única dirección de 7 bits. Notar que las direcciones 0xD0 a 0xD6 al pasarlas a binario todas terminan en 0, lo cual implica que se tratan de comandos de escritura. Al omitir el último bit se

obtienen las direcciones (sin incluir el bit de lectura/escritura) como se puede ver a continuación:

$$0xD0(11010000) \longrightarrow 0x68(1101000) \quad (1.1)$$
$$0xD2(11010010) \longrightarrow 0x69(1101001) \quad (1.2)$$
$$0xD4(11010100) \longrightarrow 0x6A(1101010) \quad (1.3)$$
$$0xD6(11010110) \longrightarrow 0x6B(1101011) \quad (1.4)$$

Las direcciones de los motores son entonces: **0x68**, **0x69**, **0x6A**, **0x6B**.

- En las pruebas 0, 1, 2, 5, 6 y 7 se observa que la suma del valor promedio enviado al motor D0 más el enviado a la dirección D2 es similar a la suma de los valores entregados a los motores con dirección D4 y D6, lo cual implica un equilibrio en los pares realizados en ambos sentidos. Se infiere entonces que el movimiento según el eje "z"se logra desequilibrando los pares en ambos sentidos (según "z" y según "-z")

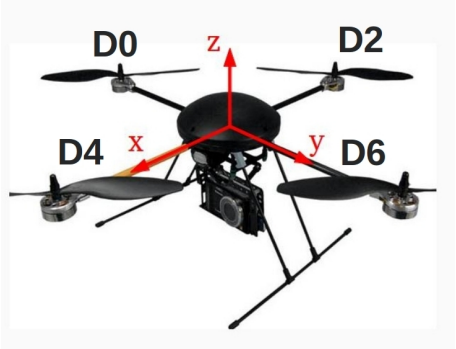


Figura 1.9: correspondencias

1.6. Verificación

TODO: verificaciones - arreglar fotos que tienen un marco feo - cambiar D0 con D2 en la ultima figura - grafica en el arranque - snifeada al frenar

Comando i2cdetect de la librería nosequé... Resultado en terminal

```
root@beagleboard:~# i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n] y

     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 69 6a 6b -- -- -- --
70: -- -- -- -- -- -- -- --
```