CAPÍTULO 1

TESTEO DEL BEAGLEBOARD Y EL *IMU*

Durante la elección del hardware se buscó información sobre el *Beagleboard* (de ahora en más "*BB*") y sobre el *Atomic IMU* (de ahora en más "*IMU*"), y se concluyó que serían suficientes para los requerimientos del proyecto. En este capítulo se explican las pruebas realizadas sobre ambas placas. Los objetivos de dichas pruebas fueron:

- Verificar el correcto funcionamiento de las placas.
- Confirmar que cumplen con los requerimientos del proyecto.

1.1. Testeo del Beagleboard

El microcontrolador a bordo del *BB* (de ahora en más "*omap*") es suficiente para manejar el loop de control, y tiene un *DSP*, por lo que la capacidad de procesamiento no debería ser un problema. La evaluación de dicha capacidad se hará más adelante, cuando se cuente con software adecuado. Por ahora solamente se analizó la disponibilidad de señales *PWM*.

1.1.1. *PWM*

El *omap* cuenta con 4 módulos para *PWM* implementados en hardware. Para poder utilizarlos es necesario deshabilitar una función de ahorro de energía en el kernel. Se utilizó OpenEmbedded para compilar un kernel adecuado.

Se utilizó un driver hecho por Scottellis para manejar el acceso a registros en el *omap*. Puede que no sea necesario manejar los *PWM* desde el kernel, ya que la implementación está en hardware, por lo que se está considerando pasa a usar un programa en *user space*, evitando los riesgos de trabajar a bajo nivel. Un error a nivel de kernel podría dejar deshabilitar el *BB*, lo cual llevaría inevitablemente a perder el control del cuadricóptero.

Solamente 3 de los 4 *PWM* están mapeados a pines accesibles en el *BB*. Para tener acceso al cuarto *PWM* es necesario modificar la configuración de la multiplexación en el *omap*.

Habilitar el cuarto *PWM* trae al menos dos problemas:

1. Queda inutilizable el integrado `U14`, que es un *Hi-Speed USB 2.0 Transceiver*.
2. Hace falta conectar un cable a un punto en la mitad del *BB* para ver la señal de *PWM*.

Se encontraron comentarios en foros sobre posibles problemas al utilizar los *PWM* en conjunto con el *DSP*, pero no se encontró información que confirmara esto en las hojas de datos, y todavía no se cuenta con software adecuado para hacer pruebas.

Se determinó que es posible utilizar el *BB* para generar 4 señales *PWM*, pero con un costo que puede ser importante.

Alternativas bajo estudio:

- Utilizar alguno de los 6 *PWM* disponibles en el microcontrolador a bordo del *IMU*:
  - Permitiría utilizar todas las funcionalidades del *BB*.
  - El *IMU* puede quedar sobrecargado, resultando en pérdidas de datos de los sensores.
  - El tiempo que tarda en llegar una orden emitida por el loop de control a su destino puede ser un problema Dicha orden tiene que seguir el siguiente camino:
    1. Sale del *BB* y va al *IMU*.
    2. El *IMU* configura el *PWM* adecuadamente.
    3. La señal de *PWM* va al *ESC* correspondiente.
    4. El *ESC* actúa sobre el motor correspondiente.
- Utilizar una tercer placa para generar los *PWM*.

1.2. Testeo del *IMU*

El microcontrolador a bordo del *IMU* tiene un *ADC* de 10 bits, a cuya entrada están conectadas, mediante multiplexación, las salidas de los sensores: 3 giróscopos y 3 acelerómetros[1].

El software que corre en el el *IMU* permite configurar qué sensores se desean leer, y la frecuencia con la que se debe realizar el muestreo. Las lecturas se transmiten mediante una *UART*. Esta información se puede ver en una PC utilizando un adaptador *USB-serie*, o se pueden recibir en el *BB* conectando el *IMU* a alguna de las *UART*s en el *BB*.

Para verificar el correcto funcionamiento del *IMU* se partió de [**?**], un software escrito para el Razor 9DOF, y se lo modificó para adecuarlo al *IMU*.

El código que corre en el *IMU* permite elegir entre un formate *ASCII*, o formato binario.

La información transmitida por el *IMU* es de la forma:

```
A count accx accy accz pitch roll yaw Z
```

[1] En realidad es 1 acelerómetro de 3 ejes.

Donde:

1. `A`: Indica comienzo de datos - 2 bytes.
2. `count`: Cada tira de datos tiene un número asociado - 4 bytes.
3. `acc*`: Lectura del *ADC* para los acelerómetros - 2 bytes cada uno.
4. `pitch roll yaw`: Lectura del *ADC* para los giróscopos - 2 bytes cada uno.
5. `Z`: Indica fin de datos - 2 bytes.

En modo binario solamente se transmiten los campos previamente mecionados, mientras que en modo *ASCII*:

1. Cada dígito se transmite como un `char`, ocupando 1 byte. Esto implica que un valor de `count` de 24662, cuya representación en hexadecimal es $0x6056$ ocupará 5 bytes, en lugar de 4.
2. Se utilizan tabulaciones para separar cada campo.

El modo *ASCII* es más sencillo, pero el modo binario es más eficiente. Se optó por utilizar el modo *ASCII* para pruebas, y el modo binario para la implementación que correrá en el *BB*.

1.2.1. Software de prueba

Pseudocódigo:

1. Conectar a al puerto serie.
2. Inicializar un display que representa la estimación de la posición en la cual se encuentra el *IMU*.
3. Correr dos threads:
   a) Thread 1: Espera input del usuario. Se usa para cambiar la frecuencia de muestreo o la sensibilidad de los sensores.
   b) Thread 2: Lee datos del puerto serie y luego:
      - Imprime a la consola.
      - Escribe a un log.
      - Actualiza el display.

El programa permite elegir tres formas de manipular los datos de los sensores:

- Solamente giróscopos → *Drift*
- Solamente acelerómetros → Información ruidosa
- Combinar giróscopos y acelerómetros → *Kalman*

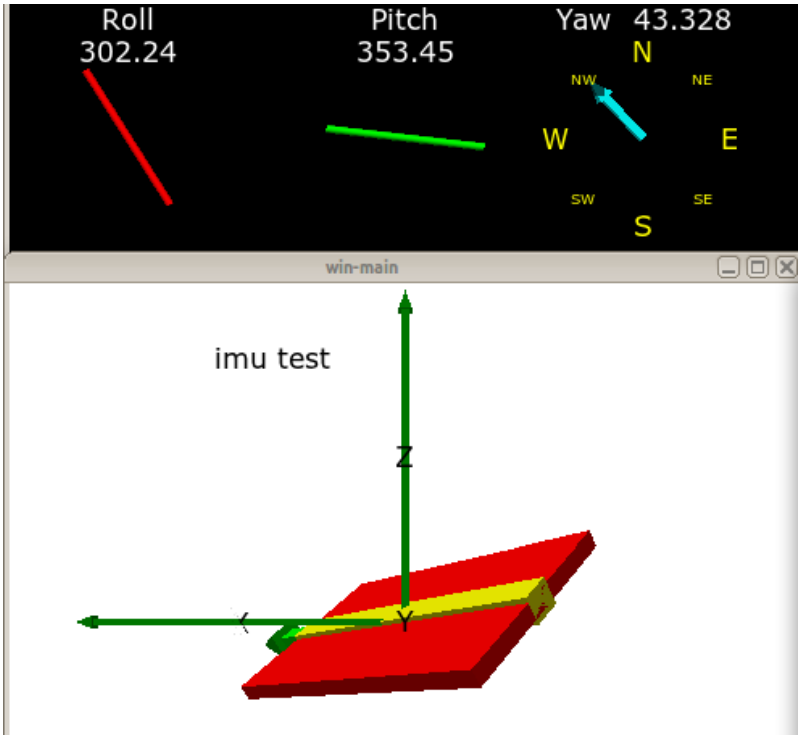

Figura 1.1: Programa para ver datos del *IMU*.

Se hace una calibración sencilla para tomar en cuenta el que el cero ("*null*") de los sensores no se corresponde exactamente con lectura cero.

El programa está hecho en python, su única función es mostrar que el *IMU* funciona correctamente. Un software similar correrá en el BB, pero será escrito en *C*.

1.2.2. Tareas pendientes

Si se desea obtener información más "limpia" de los sensores es necesario mejorar la calibración.

Otro punto donde se podría mejorar es en el *ADC*. La respuesta del *ADC* se asume lineal, pero no lo es.

Si la performance de los sensores no resulta adecuada, se podría:

- Hacer una calibración que tome en cuenta no solamente la estimación del *null*, sino también la ganancia y las no linealidades del sensor.
- Caracterizar la no linealidad del *ADC* y compensarla.