Instituto Tecnológico de Buenos Aires

31.99 Mecatrónica Aplicada

Trabajo Práctico: Sensor Fusion

Parra, Rocío 57669

Profesores

Perfumo, Lucas Alberto
Fortunatti, Nelson Ariel

Presentado: 23/10/2020

1. Funcionamiento

1.1. Programa del microcontrolador

El programa del microcontrolador funciona realizando *polling* en el estado de un timer. Este timer, configurado para activarse cada 0.01s, marca el tiempo de muestreo.

Cada vez que se detecta un evento de timeout (a través del cambio de una variable booleana en una interrupción), se lee el estado de los sensores del MPU6500. Tal como se observa en la Figura 1, esto se realiza con protocolo I2C.

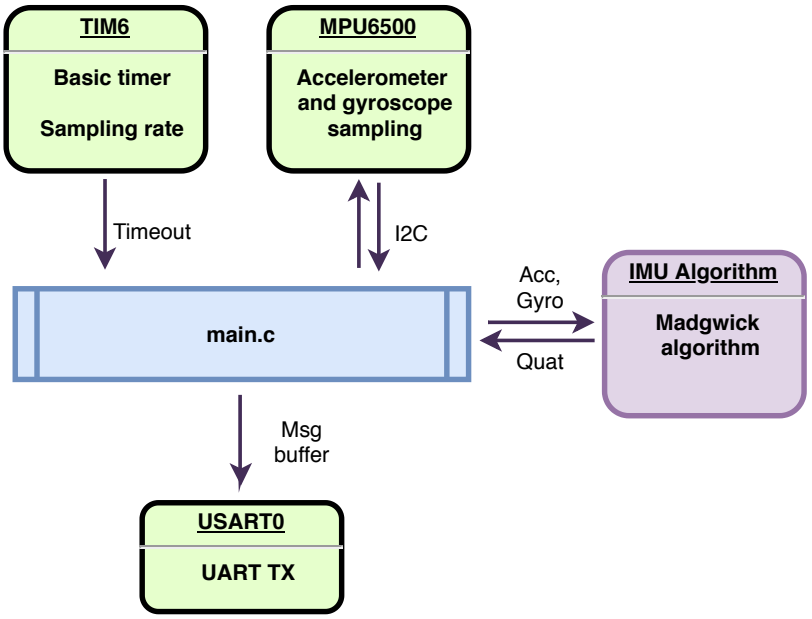

Figura 1: Diagrama de bloques del programa del microcontrolador

Una vez obtenidos los 6 valores de los sensores (las componentes x, y y z del acelerómetro, y del giroscopio), se utilizan para obtener el nuevo valor de las cuatro componentes del cuaternión que describe la orientación del MPU, utilizando el algoritmo IMU de Madgwick. Una vez calculado esto, se comunican la totalidad de los 10 valores obtenidos a través del USART0 (como se explicará en la próxima sección).

1.2. Programa de la computadora

La estructura del programa que corre en la computadora puede observarse en la Figura 2. La sincronización con los mensajes obtenidos a través del puerto serie se realiza con un thread dedicado, para asegurarse de que ningún paquete se pierda. Este thread interactúa con el resto del programa a través de dos objetos:

- una *queue* FIFO, donde se guarda cada paquete con el formato correcto que se recibe (esto se explicará más detalladamente en la próxima sección). A su vez, si se detecta un problema con el puerto serie, también se comunica a través de esta *queue* para que se pueda lidiar con la excepción apropiadamente
- un evento binario (set o cleared), que el programa enciende para indicarle al thread que debe terminar de ejecutar.

El programa principal recibe de la GUI señales que indican qué puerto serie debe utilizarse, cuándo conectarse y desconectarse, y cuándo empezar a graficar los resultados, y si se desea congelar los gráficos. Dichos gráficos se actualizan cada vez que se activa un timer, cuya frecuencia determina el frame rate.

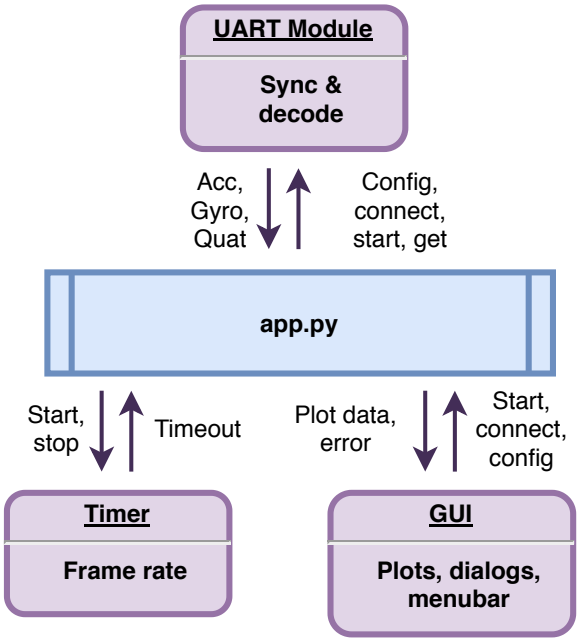

Figura 2: Diagrama de bloques del programa de la computadora

Cada vez que ocurre un evento de timeout, se obtiene toda la información que haya llegado por UART, se calculan los ángulos de pitch y roll, y se muestran los resultados.

A su vez, si se detectase un problema con la conexión al puerto serie, se le avisará al usuario que hubo un error y se desconectará y dejará de graficar automáticamente. El programa queda entonces esperando a que se vuelva a conectar a un puerto disponible.

2. Sincronización

El envío de datos desde el microprocesador a la computadora se realizó vía UART, con un baudrate de 115200, sin paridad, 8 bits por palabra y un stopbit. No se implementó ningún mecanismo de acknowledgement por parte del programa en la computadora, ya que los datos son enviados en tiempo real, y reenviar paquetes comprometería esta característica.

La sincronización se realizó estructurando los mensajes de la siguiente manera:

- 1 byte con la letra 'A' (por aceletrómetro) en ASCII
- 3 floats de 32 bits, indicando las componentes x, y y z del acelerómetro, en ese orden
- 1 byte con la letra 'G' (por giroscopio) en ASCII
- 3 floats de 32 bits, indicando las componentes x, y y z del giroscopio, en ese orden
- 1 byte con la letra 'Q' (por cuaternión en inglés, *quaternion*) en ASCII
- 4 floats de 32 bits, indicando las 4 componentes de la aproximación del cuaternión representando la posición de MPU

De esta manera, cada mensaje cuenta con 40 bytes de datos (3+3+4=10 floats, cada uno de cuatro bytes) y 3 bytes de sincronización. El programa en la computadora detecta la presencia de un mensaje válido a partir de estos tres caracteres en la posición adecuada. Esta tarea se realiza en un *thread* independiente, que constantemente busca nuevos mensajes y los guarda en una *queue*, hasta ser detenido por un evento de *stop*, o detectar un error en el puerto serie.

A continuación se muestra la totalidad del código correspondiente a este *thread*.

```
    def uart_sync(self):
      try:
        # ignore all old msgs
        self.ser.flush()

        # wait for the next whole msg
        buff = self.ser.read(MSG_SIZE)

        # run until told to stop
        while not self.end.is_set():

          # check whether the msg fits the format
          agq = buff[0::FLOAT_SIZE * 3 + 1][:3]
          try:
            agq = agq.decode(encoding='ascii')
          except UnicodeDecodeError:
            pass

          # if the msg fits the format, queue it
          if agq == 'AGQ':
            self.q.put(buff)
            buff = self.ser.read(MSG_SIZE)

          # else drop first byte and read one more
          else:
            buff = buff[1:] + self.ser.read(1)

      except (serial.SerialException, serial.SerialTimeoutException) as e:
        # if there was a problem with the serial port, report it
        self.q.put(e)
      else:
        # if no exceptions were raised, discard all previous msgs
        self.q = Queue(maxsize=QUEUE_SIZE)
      finally:
        # mark event as read
        self.end.clear()

```

3. Videos

Se adjuntan dos videos que demuestran el funcionamiento del programa. El primero de ellos explica cómo se usa la interfaz gráfica. El segundo muestra los gráficos obtenidos para distintas orientaciones del MPU. Este último video también explica cómo distintos movimientos se observan en cada sensor, y en las estimaciones de los ángulos de *pitch* y *roll*.