Universidad de Buenos Aires
Facultad de Ingeniería
Año 2025 - 2º cuatrimestre

Taller de automatización y control (TA135)

Trabajo Práctico 1 - Introducción al control digital

ESTUDIANTES: Grupo 6

Del Rio, Francisco Agustín	110761
fadelrio@fi.uba.ar	
Monti, Martina	110574
mmonti@fi.uba.ar	

Índice

1. Introducción **2**

2. Limitaciones de los sensores **2**
2.1. Potenciómetro 2
2.2. Sensor de distancia 2
2.3. IMU 3

3. Limitaciones de los actuadores **3**

4. Limitaciones de las tareas adicionales **4**

5. Conclusión **5**

6. Anexo-Códigos **6**
6.1. Código para medir tiempo de lectura ADC 6
6.2. Código para medir tiempo de medición del sensor de distancia 6
6.3. Código para medir tiempo de lectura de la IMU 7
6.4. Código para medir tiempo de envío por puerto serie 7

1. Introducción

En este informe se especifican tanto los cálculos teóricos con datos proporcionados por fabricantes como las mediciones prácticas de los tiempos de procesamiento de tareas de censado, actuación y envío de datos. Se detallan los tiempos de una medición del conversor analógico digital del microprocesador, del sensor de distancia HC-SR04, de la unidad de medición inercial, de un servomotor y del envío de datos a través de puerto serie.

2. Limitaciones de los sensores

2.1. Potenciómetro

El conversor analógico-digital de la placa arduino utilizada tiene una resolución de 10 bits, lo que se convierte en $2^{10} = 1024$ valores distintos que puede tomar la medición. Los cuales si se tiene en cuenta la tensión de referencia, permite calcular la resolución por incremento en volts:

$$\Delta V = \frac{V_{ref}}{2^{10}} = \frac{5V}{1024} = 4{,}88mV$$

Considerando que el potenciómetro puede realizar un giro de 270°, se mide la resolución angular

$$\Delta\theta = \frac{270}{1024} = 0°15'49'' = 4{,}6e^{-3}rad$$

Según la documentación un Arduino Nano usa un clock de 16 MHz, y con el prescaler default de 128, tiene una frecuencia de 125 kHz. A su vez, para realizar una conversión se requieren 13 ciclos de reloj. Por lo tanto una medición tardara:

$$t = \frac{13}{125000} = 104\mu s$$

Utilizando el código de la Subsección 6.1 se midió empíricamente el tiempo que se tarda en medir un valor del ADC y se obtuvo un tiempo promedio de 116 µs. Si se considera que la función *analogRead* indica un tiempo de lectura aproximado de 110 µs y el uso de funciones adicionales para obtener el resultado en consola, el valor obtenido es cercano al esperado.

2.2. Sensor de distancia

El sensor de distancia realiza la medición emitiendo un pulso de sonido y tomando el tiempo que tarda en rebotar en un objeto y volver al sensor. Por lo tanto la distancia se calcula de la siguiente manera:

$$d = \frac{v_{sonido}t}{2}$$

Se toma la velocidad del sonido como 343 $\frac{\text{m}}{\text{s}}$, y se considera la mitad del tiempo ya el total es lo que el pulso tarda en ir y volver.

Esto se realiza utilizando la función **Ping()**, la cual internamente utiliza la función **PulseIn()**. Está ultima toma el tiempo desde que un pin pasa de **LOW** a **HIGH** hasta que vuelve a **LOW**, con una resolución de 4 µs. Usando este valor se puede calcular la mínima resolución en la medición de la distancia.

$$\Delta d = \frac{v_{sonido}\Delta t}{2} = 0{,}686mm$$

Sin embargo, esta cuenta no considera las limitaciones físicas del sensor. Según la hoja de datos la mínima lectura posible es de $\Delta d = 3mm$. Por lo tanto, se toma en cuenta el de mayor valor.

El tiempo requerido para hacer una medición va a variar con la distancia - a mayor distancia, mayor tiempo. Se fijo que la máxima distancia sea de 50 mm ya que es apenas mayor al largo de la barra. Utilizando el código de la Subsección 6.2 se obtuvo que para esa distancia se tardan 5 ms en realizar una medición.

2.3. IMU

La unidad de medición inercial permite estimar el ángulo de la barra mediante la medición de fuerzas especificas y aceleraciones angulares. El dispositivo utilizado es el MPU6050 mediante comunicación I²C. Para estimar el ángulo, utilizando solamente los datos del acelerómetro se estableció el rango de medición en $\pm 8\,\mathrm{g}$, lo que implica que los 16 bits se utilizarán para codificar $16\,\mathrm{g}$, obteniendo una mínima resolución de

$$\Delta a = \frac{16\,\mathrm{g}}{2^{16}} = 244{,}14\,\mu\mathrm{g} \tag{1}$$

Además, sabiendo que el ángulo de la IMU se puede obtener como

$$\theta = arctan(\frac{a_y}{a_z}) \tag{2}$$

Para obtener el error de θ se utiliza el método de derivadas parciales, entonces

$$\Delta\theta = |\frac{\partial}{\partial a_y} arctan(\frac{a_y}{a_z})| \cdot \Delta a_y + |\frac{\partial}{\partial a_z} arctan(\frac{a_y}{a_z})| \cdot \Delta a_z \tag{3}$$

$$\Delta\theta = |\frac{1}{a_z(1 + (a_y/a_z)^2)}| \cdot \Delta a_y + |\frac{-a_y}{a_z^2(1 + (a_y/a_z)^2)}| \cdot \Delta a_z \tag{4}$$

Evaluando para el sensor perpendicular al radio terrestre, $a_y = 0$ y $a_z = 1\,\mathrm{g}$

$$\Delta\theta = |\frac{1}{1\,\mathrm{g}(1 + (0/1\,\mathrm{g})^2)}| \cdot 244{,}14\,\mu\mathrm{g} = 244{,}14 \times 10^{-6}\texttt{rad} \approxeq 0{,}014^\circ \tag{5}$$

Sin embargo, debido a la presencia de ruido en las mediciones, el error real será mayor.

Según la hoja de datos proporcionada por el fabricante, los datos del giroscopo se pueden obtener a una frecuencia máxima de 8 kHz, y el acelerómetro a una frecuencia máxima de 1 kHz. Como se obtienen y utilizan los datos de ambos sensores al mismo tiempo, se toma la más baja de las frecuencias de adquisición, entonces como máximo se podrán obtener los datos de la IMU cada $\frac{1}{1\,\mathrm{kHz}} = 1\,\mathrm{ms}$.

Utilizando el código de la Subsección 6.3 se midió el tiempo que se tarda en obtener todos los datos de los sensores, y se obtuvo un tiempo promedio de 3060 µs. La discrepancia entre los cálculos teóricos y simulados se puede atribuir a el tiempo que toma el envío de datos a través del protocolo I²C, y además los valores utilizados para los tiempos calculados en base a la hoja de datos son proporcionados como valores máximos, lo que indica que no necesariamente son los que se van a obtener en la práctica.

3. Limitaciones de los actuadores

Según la datasheet el servo toma señales PWM de entre 500 µs y 2400 µs, sin embargo para realizar las pruebas se uso 560 µs a 2380 µs, ya que de forma empírica se encontró que el servo llegaba hasta sus limites de rango de movimiento entre estos valores. La función **writeMicroseconds()** permite generar pulsos de 1 µs de precisión. Considerando esto se puede establecer el tamaño del rango de valores que se le puede comandar al servo

$$range = \frac{2380\mu - 560\mu}{1\mu} = 1820$$

Considerando que según lo evaluado empíricamente el servo puede realizar un movimiento de 180°, se calcula la resolución angular

$$\Delta\theta = \frac{180}{1820} = 0{,}0989$$

Sin embargo en la practica la máxima precisión es mas cercana a 1° debido a factores físicos como el juego mecánico y la carga.

El tiempo de respuesta del servo a la tension en que se esta operando, es de 0,2 µs/60°.

$$t_{30} = \frac{t_{60}}{2} = 0{,}1\mu s$$

Lamentablemente, ese valor es valido para el servo sin carga. Para obtener un resultado mas cercano a la realidad es necesario realizar una medición real. Para ello se aprovechó de la planta ya armada y de

la integración del sistema barra-servo-IMU, ya que se pudo comandar una referencia de 30° al servo y luego medir el tiempo que se tarda en llegar a esa referencia en el sistema. Como se puede apreciar en la Figura 1, el servo tarda aproximadamente 400 ms en llegar a la referencia.

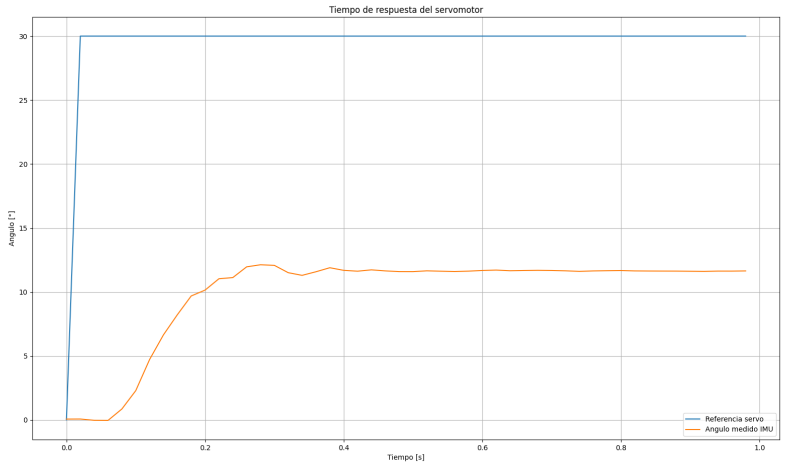

Figura 1: Respuesta medida por la IMU ante una referencia de 30°

Cuando se usa una frecuencia de 50 Hz el servo recibe pulsos cada 20 ms, al disminuir la frecuencia a 1 Hz los pulsos serán recibidos cada 1 s.

En principio, la precisión angular no se vería afectada ya que el cambio en la frecuencia no afecta la precisión en el ancho del pulso enviado. No obstante, en caso de haber ruido que afecte la precisión del pulso, la posición tardara mas en ser corregida debido a que el servo solo recibe información cada un segundo.

El tiempo de respuesta del servo sera mayor ya que los movimientos del servo serán mas lentos, además sera mas probable que el servo entre en modo reposo generando que se tarde mas en iniciar el movimiento. Otra consecuencia de entrar en modo reposo es que se deja de aplicar fuerza, y por lo tanto el angulo puede cambiar sin ser deseado.

4. Limitaciones de las tareas adicionales

Mediante los sensores se pueden adquirir datos del estado de la planta, y mediante los actuadores se puede hacer variar el estado de la planta. Sin embargo, hay otro sistema que se debe utilizar para el control, que permite monitorear los datos adquiridos a tiempo real para realizar análisis de estos. El sistema en cuestión es, en este caso, el puerto serie de arduino. Mediante este puerto se pueden enviar bytes correspondientes a los datos adquiridos por los sensores, para luego recibirlos desde una computadora en un programa (en nuestro caso MATLAB) que permite graficar los mismos y realizar cálculos.

Toda la información necesaria sobre el estado de la planta se puede traducir en datos de tipo float, por lo que resulta relevante conocer el tiempo de procesamiento necesario para enviar este tipo de dato a través del puerto serie. Para esta comunicación se utiliza un Baudrate de 115200, ya que permite enviar datos a una velocidad considerable, y recibirlos en la computadora sin perdidas.

Teniendo en cuenta las consideraciones ya mencionadas, se utilizó el código de la Subsección 6.4 para medir el tiempo que se tarda en enviar 100 datos de tipo float y se obtuvo un tiempo de 34,000 µs. Lo que implica que para enviar un solo float se tardará en promedio 340 µs. El tiempo se obtuvo de esta forma ya que el microcontrolador utiliza registros para enviar datos por serial, entonces el tiempo que se tarda en enviar el dato depende tanto del largo del dato como del estado del registro en el momento que se quiere enviar. Al enviar 100 datos juntos se evalúa el peor caso, en el que el registro se encuentra cargado, y el tiempo de envío es máximo.

5. Conclusión

Se pudieron calcular y medir los tiempos de procesamiento necesarios, como el error obtenido a la hora de utilizar los sensores y actuadores. Los resultados obtenidos fueron, en la mayoría de los casos, coherentes entre lo estimado y lo simulado. El conocimiento de los errores tanto como los tiempos resultan cruciales para tener en cuenta a la hora de controlar el sistema, ya que cambiar alguno de los valores obtenidos en este informe implicaría reemplazar o modificar el hardware involucrado en el sistema.

6. Anexo-Códigos

6.1. Código para medir tiempo de lectura ADC

```
int pin_pote = A0;

void setup() {

  Serial.begin(9600);

}

void loop() {
  int t_start = micros();

  int medida = analogRead(pin_pote);

  int t_finish = micros();

  Serial.println(medida);

  Serial.print("Tardo : ");
  Serial.print(t_finish - t_start);
  Serial.print(" us");
  Serial.print("\n");
}
```

6.2. Código para medir tiempo de medición del sensor de distancia

```
#define TRIGGER_PIN 6
#define ECHO_PIN 7
#define MAX_DISTANCE 50
#define PERIODO_MS 20


NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);


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

void loop() {

  unsigned long t_start = micros();

  int round_time_us = sonar.ping(MAX_DISTANCE);
  float distance = round_time_us/59.974;
  Serial.print(distance);

  unsigned long t_finish = micros();

  int delay_us = PERIODO_MS*1000 - (t_finish - t_start);

  delayMicroseconds(delay_us);

  t_finish = micros();

  Serial.print("\nTardo : ");
  Serial.print(t_finish - t_start);
```

```
  Serial.print(" us");
  Serial.print("\n");
}
```

6.3. Código para medir tiempo de lectura de la IMU

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

#define PERIODO_MS 20
#define MAX_DELAY_US 16383

Adafruit_MPU6050 mpu;

void setup() {
  Serial.begin(115200);
  if (!mpu.begin()) {
    Serial.println("No se encontro el 6050");
    while (1) {
      delay(10);
    }
  }
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_44_HZ);
}

void loop() {
  unsigned long t_start = micros();
  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  float data[6] =
      {a.acceleration.x,a.acceleration.y,a.acceleration.z,g.gyro.x,g.gyro.y,g.gyro.z};

  unsigned long t_finish = micros();

  Serial.print("\n");
  Serial.print("Tardo : ");
  Serial.print(t_finish - t_start);
  Serial.print(" us");
  Serial.print("\n");
}
```

6.4. Código para medir tiempo de envío por puerto serie

```
void setup(void) {
  Serial.begin(115200);
  delay(100);
}

void loop() {

  unsigned long t_start = micros();
```

```
  sendn_floats(100);

  unsigned long t_finish = micros();

  Serial.print("\n");
  Serial.print("Tardo : ");
  Serial.print(t_finish - t_start);
  Serial.print(" us");
  Serial.print("\n");
}

//envia n floats por serial
void sendn_floats(int n){
  float f = 1.5;
  byte * b;
  for (int i = 0; i < n;i++){
    f = f+i;
    b = (byte *) &f;
    Serial.write(b,4);
  }
}
```