1) **Limitaciones del sensor**:
   Giroscopio: el giroscopio tiene la limitación de un bias que se va sumando en cada paso de integración por lo que su la estimación de ángulo está sesgada por la misma

   acelerómetro: la limitación del acelerómetro viene dada por mucho ruido en la medición de ángulos por lo que se desconoce el verdadero valor a priori.

2) **Retardo en la comunicación I2C:**
   En la hoja de datos del sensor se especifica una frecuencia máxima para la comunicación I2C entre los registros de 400 kHz. Una primera aproximación para estimar el retardo de la comunicación es ver a qué frecuencia define la comunicación la clase wire el arduino.

```
// TWI clock frequency
static const uint32_t TWI_CLOCK = 100000;
uint32_t twiClock;
```

   Se ve que la “Two Wire Clock Frequency” está definida como 100 kHz, entonces dado a que le pedimos en cada ciclo los datos de la aceleración en los tres ángulos y los datos del giroscopio en los 3 ángulos (apuntamos a las direcciones de memoria de dichos registros de 16 bit) y cada uno de estos datos está asociado a la conversión de un ADC de 16 bit, entonces se podría decir que cada comunicación tarda aproximadamente:
   $t_{com} = 2 * 16 * 2 * (1/100.000)\, s = 640\, us$
   Que es un retardo que quizás se deba considerar al momento de realizar el ciclo de control cuando se trabaja con frecuencias más altas.

   ancho de banda:
   Según el ancho de banda del filtro pasabajos que se use se debe considerar el delay para obtener los datos del giroscopio y del acelerómetro.

DLPF_CFG	Accelerometer (F_s = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

   en donde según el ancho de banda que se elija se tiene un mayor delay.

3) Mínima unidad discernible:

   Acelerómetro:
   La mínima unidad discernible viene dada por la sensitividad en donde para nuestro se tiene seteado en ±8g que es decir tengo 16g para repartir en 2^{16} quedando $16g\ /\ 2^{16} = 0.000244\ g/LSB$, la mínima unidad discernible con configuración ±8g.
   La cual es coherente con la inversa de la sensitividad que muestra en la siguiente tabla de la hoja de dato.

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Giroscopio:
Análogamente para el giroscopio se lo tiene configurado en ±500°/s dándonos una mínima unidad discernible de 1/65.6 °/S/LBS = 0.0153 °/s /LSB

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

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

#define SEC2MILLIS 1000
#define SENSOR_UPDATE_INTERVAL_MS 10 //10ms
Adafruit_MPU6050 mpu;
sensors_event_t a, g, temp;

float theta;

void set_timer1();

void setup(void) {
  Serial.begin(115200);

  // Try to initialize!
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) {
      delay(10);
    }
  }
```

```
  Serial.println("MPU6050 Found!");

  // set accelerometer range to +-8G
  mpu.setAccelerometerRange(MPU6050_RANGE_8_G);

  // set gyro range to +- 500 deg/s
  mpu.setGyroRange(MPU6050_RANGE_500_DEG);

  // set filter bandwidth to 5-10-21-44-94-184-260 Hz
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  //Config frequency= 100Hz with timer 1
  set_timer1();
  Serial.println("Timer 1 OK");

}

void loop2()
{
  static unsigned long lastUpdateTime = 0;
  unsigned long currentTime = millis();

  // Verificar si ha pasado el tiempo suficiente para actualizar el sensor
  if (currentTime - lastUpdateTime >= SENSOR_UPDATE_INTERVAL_MS) {
    lastUpdateTime = currentTime;
    mpu.getEvent(&a, &g, &temp);
  }

  float grados_g = complementary_filter(0.01);
  Serial.println(grados_g);

  delay(100);

}
void loop() {
  //unsigned int t0, t1;
  //t0 = millis();
  //'Serial.println("Angulo x [grad/s]:");

  float grados_g = angle_g(0.01);
  float grados_a = angle_a();
  //float grados_filt = complementary_filter(0.01);
  //Serial.println(theta);
  //t1 = millis();
  //delay(100);
  //delay(0.04 - t1 + t0);
  //Matlab send
```

```
  matlab_send(grados_g, grados_a, theta);
}

void matlab_send(float dato1, float dato2, float dato3)
{
  Serial.write("abcd");
  byte * b = (byte *) &dato1;
  Serial.write(b,4);
  b = (byte *) &dato2;
  Serial.write(b,4);
  b = (byte *) &dato3;
  Serial.write(b,4);
}

void print_aceleration()
{
  /* Print out the values */
  Serial.print("Acceleration X: ");
  Serial.print(a.acceleration.x);
  Serial.print(", Y: ");
  Serial.print(a.acceleration.y);
  Serial.print(", Z: ");
  Serial.print(a.acceleration.z);
  Serial.println(" m/s^2");

  Serial.print("Rotation X: ");
  Serial.print(g.gyro.x);
  Serial.print(", Y: ");
  Serial.print(g.gyro.y);
  Serial.print(", Z: ");
  Serial.print(g.gyro.z);
  Serial.println(" rad/s");

  //Serial.print("Temperature: ");
  //Serial.print(temp.temperature);
  //Serial.println(" degC");

  Serial.println("");

}

float angle_g(float dt)
{
  /**
  g_x = g.gyro.x [rad/seg] (global)
  dt  = 0.01 [seg] , 0.01s = 1/100hz
```

```
  return [grados]
  **/

  // el giroscopio tiene el error del bias.
  static float theta_x_rad = 0.0;
  theta_x_rad= theta_x_rad + g.gyro.x*dt;

  return theta_x_rad*(180.0/M_PI);
}

float angle_a()
{
  /**
  acc_x = m/s^2
  **/
  float a_x, a_y2, a_z2, theta_x_rad;
  a_x  = a.acceleration.x;
  a_y2 = a.acceleration.y*a.acceleration.y;
  a_z2 = a.acceleration.z*a.acceleration.z;

  theta_x_rad = atan2( a_x , ( sqrt( a_y2 + a_z2 ) ) );

  return theta_x_rad*(180.0/M_PI);
}

float complementary_filter(float dt)
{
  static float theta_x_rad = 0;
  float a_x, a_y2, a_z2, lambda; // theta_x_rad_acc, theta_x_rad_gyro;
  float t1, t2, dt2;
  lambda = 0.98;

  t1 = millis();
  a_x  = a.acceleration.x;
  a_y2 = a.acceleration.y*a.acceleration.y;
  a_z2 = a.acceleration.z*a.acceleration.z;
  t2 = millis();
  dt2 = dt - t2 + t1;
  //acc_x = m/s^2
  //theta_x_rad_acc = ;
  //g_x = g.gyro.x [rad/seg] (global) ,   dt  = 0.01 [seg] , 0.01s = 1/100hz
  //theta_x_rad_gyro= ;
  theta_x_rad = lambda*(theta_x_rad + g.gyro.x*dt2) + (1-lambda)*atan2(
a.acceleration.y , a.acceleration.z ) ;

  return  theta_x_rad*(180.0/M_PI);
}
```

```
void set_timer1()
{
  // Configuración del TIMER1
  TCCR1A = 0;                                // Limpiar el registro de control A
  TCCR1B = 0;                                // Limpiar el registro de control B
  TCNT1  = 0;                                // Inicializar el temporizador
  OCR1A  = 625;                              // Valor para generar una
frecuencia de 100Hz, 16Mhz/(100*256) = 625
  TCCR1B |= (1 << WGM12);                    // Modo CTC (Clear Timer on Compare
Match)
  TCCR1B |= (1 << CS12 ) ;                    // Prescaler de 256: CS12 = 1,
CS11 = 0 y CS10 = 0
  TIMSK1 |= (1 << OCIE1A);                   // Habilitar interrupción por
coincidencia de comparación en OCR1A
}

ISR(TIMER1_COMPA_vect) {
  interrupts();
  mpu.getEvent(&a, &g, &temp);
  theta = complementary_filter(0.01);
  noInterrupts();
}
```

Con este código pudimos corroborar que funciona el filtro complementario a la frecuencia de trabajo requerida.