Interfaz para la implementación del sensor MPU-6050

Aplicaciones del sensor MPU-6050

Aguilar Bárcenas Bayrón Esaú
Mendoza Lara Saul Eugenio
Montante Gaytán Luis Armando
Instituto Tecnológico de San Luis Potosí, Soledad de graciano Sánchez, S.L.P, México 02/05/24

***Resumen*—El desarrollo de una GUI para la utilización de un sensor MPU-6050 el cual consiste en un acelerómetro y un giroscopio, mediante el cual se realiza una demostración 3D de algunos de los usos que se le puede dar a este sensor de su funcionamiento, Por lo cual para el uso de la GUI se debe programar ciertos atajos para abrir ventanas en las que se presenta lo necesario para el posterior uso y calibración del sensor.**

Todo esto de modo tal que da pie a mayores proyectos futuros

***Abstract*—The development of a GUI for the use of an MPU-6050 sensor which consists of an accelerometer and a gyroscope, through which a 3D demonstration of some of the uses that can be given to this sensor and its operation is carried out. , Therefore, to use the GUI, certain shortcuts must be programmed to open windows in which what is necessary for the subsequent use and calibration of the sensor is presented.**

All this in such a way that it gives rise to greater future projects

I. INTRODUCCION

Este documento provee el proceso realizado en la elaboración del proyecto, usando el sensor MPU-6050, mediante la implementación de una interfaz grafica de usuario la cual al momento de ejecutarse de acceso a una simulación para posterior demostración del funcionamiento del sensor.

II. MARCO TEORICO

Antecendentes:

¿Qué son los sensores?

Los sensores son herramientas que detectan y responden a algún tipo de información del entorno físico.

Existe una amplia gama de sensores utilizados en la vida diaria, que se clasifican según las cantidades y características que detectan [1].

Algunos ejemplos incluyen:

-sensores de corriente eléctrica
-magnéticos o de radio
-sensores de humedad,
-sensores de velocidad o flujo de fluidos
-sensores de presión
-sensores térmicos o de temperatura
-sensores ópticos, sensores de posición
-sensores ambientales
-sensores químicos

MPU-6050 El sensor MPU-6050 es una unidad de medición inercial o IMU (Inertial Measurment Units) de 6 grados de libertad (DoF) pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc. [2].

Dicho sensor tiene un gran numero de posibilidades al momento de darle una aplicación, puede encontrarse principalmente en Drones (Tanto recreativos como de uso industrial / militar) con los cuales ayudan a medir la altitud, velocidad y distancia que se encuentran de un objetivo, esto al estar conectados en conjunto (en algunos casos) con un equipo de video tal como es una cámara GoPro.

Esto debido a los componentes de este sensor los cuales son:

Giroscopio: son aparatos que miden la velocidad de rotación. Estos son particularmente útiles para estabilizar la conducción del robot, o para medir el camino o la inclinación mediante la integración (suma) de las medidas para obtener la medida del desplazamiento angular total [3].

Acelerómetro: Un acelerómetro es un dispositivo que mide la vibración o la aceleración del movimiento de una estructura. La fuerza generada por la vibración o el

cambio en el movimiento (aceleración) hace que la masa "comprima" el material piezoeléctrico, generando una carga eléctrica que es proporcional a la fuerza ejercida sobre él [4].

Las principales aplicaciones en las que se encuentra involucrado el MPU-6050 son:

-*robótica*: Ayuda a la estabilidad del robot
-*Drones*: En los drones se saca el máximo potencial de este sensor, ya que como anteriormente se mencionó, ayuda con la estabilidad, altura y movimiento del Drone.

Proceso realizado

Para la realización de este proyecto, lo primero a realizar fue conseguir el sensor. Posterior a su adquisición se empezó a trabajar en la Interfaz en el software de Visual studio.

La interfaz cuenta con 6 (seis) Buttons (Botones) en los cuales cambiamos el texto dentro de él dándole una indicación para que el usuario conozca lo que cada uno de ellos realiza. Un menú desplegable en el cual se indicará el puerto COMM en el que se encueta nuestro ARDUINO UNO con el cual se realizó nuestro proyecto.

Dos de los botones están reservados para marcar el estado de si nuestro Arduino UNO, en los cuales se lee conect (Conectado) que como indica nos ayuda a conectar el Arduino, por su parte el segundo botón Disconect que cortara la conexión de la interfaz con el Arduino.

Otro de los botones hace llamar un segundo programa, el cual es el lenguaje de Arduino, con el cual se procede a calibrar el sensor, ya que para su posterior uso primero debe de cargarse un código de calibración para su correcto funcionamiento.

En este lenguaje, para la correcta calibración del sensor se precarga un código de ayuda que es proporcionado por Arduino, en el cual para que funcione se debe des comentar una línea de código, cargar y posteriormente una vez terminado de cargarse, se mostrara en la pantalla de visualización varios números consecutivos, lo que nos indica que la primera parte de la calibración fue concluida con éxito, Volvemos a comentar dicha línea y des comentamos una segunda línea, que se encuentra poco más debajo de la principal.

Una vez terminado, la calibración del MPU-6050 ha terminado con éxito.

Ilustración 1- Interfaz Grafica del Sensor MPU-6050

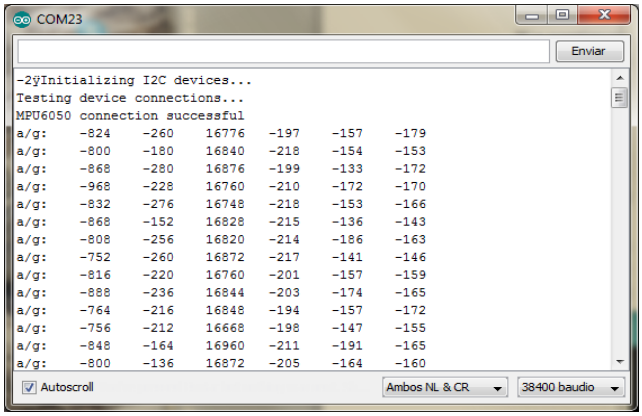

Ilustración 2- Ejemplo de la pantalla de calibración del sensor

Continuando con la descripción de los botones en la interfaz, se eligió colocar dos especiales para proporcionar ayuda al usuario en caso que ocurra un imprevisto y deba consultarse de manera rápida el como se conecta el sensor con el Arduino, por lo que el botón "MPU6050 Connecting to ARDUINO diagram" Despliega o manda a llamar una pestaña nueva en la que se muestra el diagrama de conexión. de Arduino y el sensor

Finalizando, un ultimo botón el cual da las indicaciones a seguir para realizar la calibración correcta del sensor, describiendo las líneas que se han de comentar o des comentar en fin de calibrar el sensor, la librería a utilizar en el ambiente de Arduino, la ruta a utilizar para la calibración, indicaciones de como tener configurado el Arduino. Además de dar a conocer al usuario que es necesario el uso de "Processing" el cual es un ambiente de programación en base a JAVA en el cual se ha de cargar código.

Con todo esto finalmente pasamos al botón restante. Este botón es el encargado de conectar el todo lo anterior a la simulación la cual es ejecutada con "Procesing", el cual es un software utilizado para realizar simulaciones en base a modelos 3D, al mover el sensor en diferentes dirección en la simulación nuestro modelo (en este caso un avión) se moverá en sintonía con el desplazamiento del sensor MPU-6050.

III.MATERIAL UTILIZADO

Para la realización de este proyecto se utilizo el siguiente equipo y material

- Software Visual Studio (2019 / 2022)
- Software "Processing"
- Software "Teapot code"
- Tarjeta de desarrollo Arduino UNO
- Cables Dupont
- Protoboard
- Sensor MPU-5060
- Laptop

Mediante el diagrama utilizado en la interfaz es como se debe realizar la conexión entre el ARDUINO y el sensor MPU-6050:

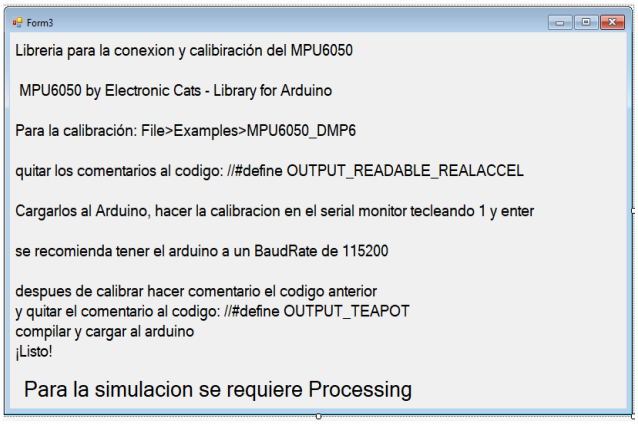

Ilustración 3- Pantalla de información "Help info" detallando pasos a seguir

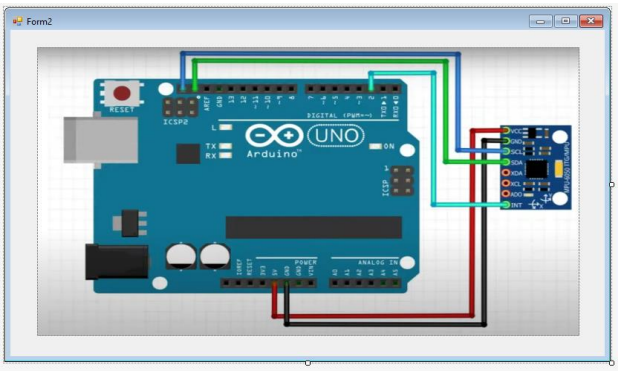

Ilustración 4- Pantalla de información de conexión de Arduino con el sensor MPU-6050

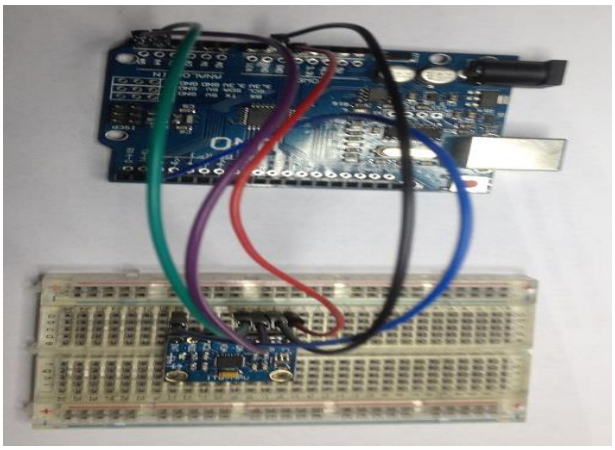

Ilustración 5- Ejemplo de armado del sensor con Arduino, sensor colocado en una protoboard

El ensamblaje del circuito es simple, debido a que solo es necesario conectar dos pares de cables dupont, los cuales van uno a la alimentación, otro a la tierra (GND) y otro par conectado a lo marcado como ISCP2, además de un 5 cable al puerto marcado con el numero 2.

Dentro de la parte de programación e visual BASIC, se declaro el uso del puerto serial para la conexión con la interfaz, mediante la programación del botón 1 el cual es el encargado de la conexión de Arduino a nuestro COM3, mostrando el estado de conexión mediante un label al presionar el botón marcado como "Connect"

Anteriormente se mencionó el botón que inicia la simulación, este botón llama al software "processing" que esta en conjunto con la librería "MPU6050 by Electronic Cats- Library for Arduino" la cual nos permite ejecutar el código "MPUTeapot" con el que se da inicio a la simulación de mover el avión hecho con polígonos sencillos.

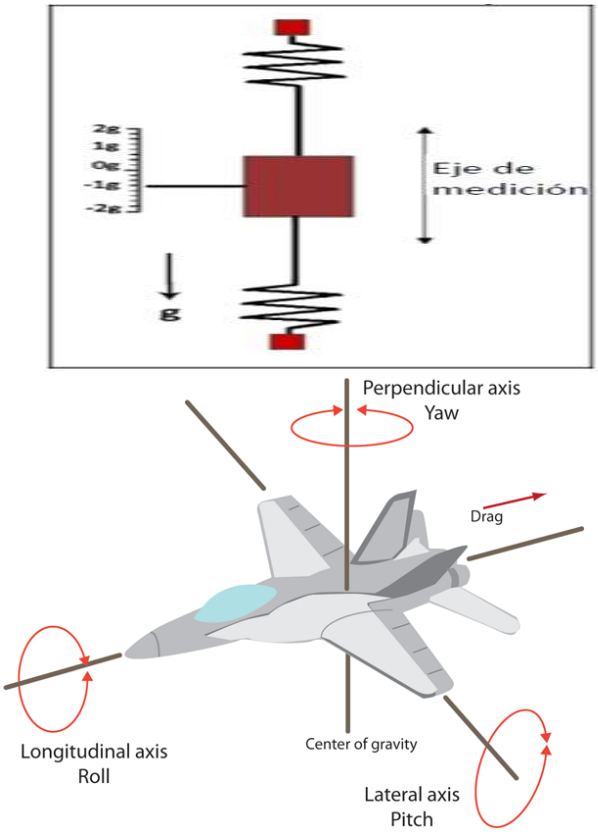

Ilustración 6 (Arriba) , 7 (Abajo). La ilustración 6 muestra el diagrama del sensor MPU-6050, la imagen 7 muestra un modelo de lo que se mostrara en la simulación al momento de dar movimiento al sensor

En el código del programa se le indica que desconecte el Arduino de la interfaz para que Processing pueda trabajar con el Arduino debido a que el Arduino no se puede conectar a distintos softwares a la vez.

```
0 references
Private Sub BtnCalibrar_Click(sender As Object, e As EventArgs) Handles BtnCalibrar.Click
    SerialPort1.Close()
    Try
        Dim exePatch As String = "C:\Users\Asus VivoBook\AppData\Local\Programs\Arduino IDE\Arduino IDE.exe"
        Process.Start(exePatch)
    Catch ex As Exception
        MessageBox.Show("No se pudo abrir el archivo: " & ex.Message)
    End Try
End Sub
```

Ilustración 8- Fragmento del código en Visual Basic el cual es el encargado de llamar al programa Arduino

A continuación, se desglosan los códigos utilizado en el desarrollo del proyecto:

Codigo 1- Visual Studio (Visual Basic)

```
Imports System.IO.Ports
Imports Microsoft.VisualBasic.Devices
Imports System.Diagnostics


Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Dim ports As String() = SerialPort1.GetPortNames()
        Dim port As String


        For Each port In ports
            ComboBox1.Items.Add(port)
        Next port
    End Sub


    Private Sub btnconectar_Click(sender As Object, e As EventArgs)
Handles btnconectar.Click
        Try
            With SerialPort1
                .BaudRate = 115200
                .DataBits = 8
                .StopBits = IO.Ports.StopBits.One
                .Parity = IO.Ports.Parity.None
                .PortName = ComboBox1.Text
                .Open()
                If .IsOpen Then
                    Label1.Text = "Conectado"
                Else
                    MsgBox("Conexión Fallida", MsgBoxStyle.Critical)
                End If
            End With
        Catch ex As Exception
            MsgBox(ex.Message, MsgBoxStyle.Critical)
        End Try



    End Sub

    Private Sub Btndesconectar_Click(sender As Object, e As EventArgs)
Handles Btndesconectar.Click
        SerialPort1.Close()
        Label1.Text = "Desconectado"
    End Sub

    Private Sub ComboBox1_SelectedIndexChanged(sender As Object, e As
EventArgs) Handles ComboBox1.SelectedIndexChanged

    End Sub

    Private Sub BtnAbrirPrograma_Click(sender As Object, e As EventArgs)
Handles BtnAbrirPrograma.Click
        SerialPort1.Close()
        Try
            Dim exePatch As String = "C:\processing-4.3\processing.exe"

            Process.Start(exePatch)
        Catch ex As Exception
            MessageBox.Show("No se pudo abrir el archivo: " & ex.Message)
        End Try
    End Sub

    Private Sub BtnCalibrar_Click(sender As Object, e As EventArgs)
Handles BtnCalibrar.Click
        SerialPort1.Close()
        Try
            Dim exePatch As String = "C:\Users\Asus
VivoBook\AppData\Local\Programs\Arduino IDE\Arduino IDE.exe"
            Process.Start(exePatch)
        Catch ex As Exception
            MessageBox.Show("No se pudo abrir el archivo: " & ex.Message)
        End Try
    End Sub
```

```
    Private Sub BtnDiagrama_Click(sender As Object, e As EventArgs)
Handles BtnDiagrama.Click
        Form2.Show()
    End Sub

    Private Sub BtnHelpInfo_Click(sender As Object, e As EventArgs)
Handles BtnHelpInfo.Click
        Form3.Show()
    End Sub
End Class
```

Codigo 2- Calibración del MPU-6050

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050
class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
//     Updates     should     (hopefully)     always     be     available     at
https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2019-07-08 - Added Auto Calibration and offset generator
//          - and altered FIFO retrieval sequence to avoid using blocking code
//      2016-04-18 - Eliminated a potential infinite loop
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility
error
//      2012-06-20 - improved FIFO overflow handling and simplified read
process
//      2012-06-19 - completely rearranged DMP initialization code and
simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of
reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt
detection to RISING
//          2012-06-05 - add gravity-compensated initial reference frame
acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks
Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h
files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense
evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/*
=========================================================
===================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 *
=========================================================
=================== */

/*
=========================================================
===================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 *
=========================================================
=================== */



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see
the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler
angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to
see the yaw/
```

```
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}

// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
    // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
```

```
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

       // set our DMP Ready flag so the main loop() function knows it's okay
to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



//
============================================================
==========
// ===                  MAIN PROGRAM LOOP                   ===
//
============================================================
==========

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;
    // read a packet from FIFO
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
            Serial.print(euler[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(euler[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(euler[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            Serial.print("areal\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

Codigo 3- “Teapot para Arduino”

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050
class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
//  Updates should (hopefully) always be available at
https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//      2019-07-08 - Added Auto Calibration and offset generator
//         - and altered FIFO retrieval sequence to avoid using blocking code
//      2016-04-18 - Eliminated a potential infinite loop
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility
error
//      2012-06-20 - improved FIFO overflow handling and simplified read
process
//      2012-06-19 - completely rearranged DMP initialization code and
simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of
reading directly
```

```
//     2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//     2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                - add 3D math helper file to DMP6 example sketch
//                - add Euler output and Yaw/Pitch/Roll output formats
//     2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//     2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//     2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 *
 * ========================================================================= */

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 *
 * ========================================================================= */


// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
#define OUTPUT_TEAPOT


#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
```

```
VectorInt16 aaReal;      // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;     // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;     // [x, y, z]            gravity vector
float euler[3];          // [psi, theta, phi]    Euler angle container
float ypr[3];            // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };


// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}


// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3V or Arduino
    // Pro Mini running at 3.3V, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // Calibration Time: generate offsets and calibrate our MPU6050
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.print(F("Enabling interrupt detection (Arduino external interrupt "));
        Serial.print(digitalPinToInterrupt(INTERRUPT_PIN));
        Serial.println(F(")..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}


// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return;
    // read a packet from FIFO
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Get the Latest packet
        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
```

```
    Serial.print(q.y);
    Serial.print("\t");
    Serial.println(q.z);
#endif

#ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(ypr[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t");
    Serial.print(aaWorld.x);
    Serial.print("\t");
    Serial.print(aaWorld.y);
    Serial.print("\t");
    Serial.println(aaWorld.z);
#endif

#ifdef OUTPUT_TEAPOT
    // display quaternion values in InvenSense Teapot demo format:
    teapotPacket[2] = fifoBuffer[0];
    teapotPacket[3] = fifoBuffer[1];
    teapotPacket[4] = fifoBuffer[4];
    teapotPacket[5] = fifoBuffer[5];
    teapotPacket[6] = fifoBuffer[8];
    teapotPacket[7] = fifoBuffer[9];
    teapotPacket[8] = fifoBuffer[12];
    teapotPacket[9] = fifoBuffer[13];
    Serial.write(teapotPacket, 14);
    teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
#endif

// blink LED to indicate activity
blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
  }
}
```

Codigo 4- Sketch Teapot Processing

```
// I2C device class (I2Cdev) demonstration Processing sketch for MPU6050 DMP output
// 6/20/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     2012-06-20 - initial release

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
============================================
*/

import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;

// NOTE: requires ToxicLibs to be installed in order to run properly.
// 1. Download from https://github.com/postspectacular/toxiclibs/releases
// 2. Extract into [userdir]/Processing/libraries
//     (location may be different on Mac/Linux)
// 3. Run and bask in awesomeness

ToxiclibsSupport gfx;

Serial port;                         // The serial port
char[] teapotPacket = new char[14];  // InvenSense Teapot packet
int serialCount = 0;                 // current packet byte position
int synced = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];

void setup() {
    // 300px square viewport using OpenGL rendering
    size(300, 300, OPENGL);
    gfx = new ToxiclibsSupport(this);
```

```
    // setup lights and antialiasing
    lights();
    smooth();

    // display serial port list for debugging/clarity
    println(Serial.list());

    // get the first available port (use EITHER this OR the specific port code below)
    String portName = Serial.list()[0];

    // get a specific serial port (use EITHER this OR the first-available code above)
    //String portName = "COM4";

    // open the serial port
    port = new Serial(this, portName, 115200);

    // send single character to trigger DMP init/start
    // (expected by MPU6050_DMP6 example Arduino sketch)
    port.write('r');
}

void draw() {
    if (millis() - interval > 1000) {
        // resend single character to trigger DMP init/start
        // in case the MPU is halted/reset while applet is running
        port.write('r');
        interval = millis();
    }

    // black background
    background(0);

    // translate everything to the middle of the viewport
    pushMatrix();
    translate(width / 2, height / 2);

    // 3-step rotation from yaw/pitch/roll angles (gimbal lock!)
    // ...and other weirdness I haven't figured out yet
    //rotateY(-ypr[0]);
    //rotateZ(-ypr[1]);
    //rotateX(-ypr[2]);

    // toxiclibs direct angle/axis rotation from quaternion (NO gimbal lock!)
    // (axis order [1, 3, 2] and inversion [-1, +1, +1] is a consequence of
    // different coordinate system orientation assumptions between Processing
    // and InvenSense DMP)
    float[] axis = quat.toAxisAngle();
    rotate(axis[0], -axis[1], axis[3], axis[2]);

    // draw main body in red
    fill(255, 0, 0, 200);
    box(10, 10, 200);

    // draw front-facing tip in blue
    fill(0, 0, 255, 200);
    pushMatrix();
    translate(0, 0, -120);
    rotateX(PI/2);
    drawCylinder(0, 20, 20, 8);
    popMatrix();

    // draw wings and tail fin in green
    fill(0, 255, 0, 200);
    beginShape(TRIANGLES);
    vertex(-100, 2, 30); vertex(0, 2, -80); vertex(100, 2, 30);  // wing top layer
    vertex(-100, -2, 30); vertex(0, -2, -80); vertex(100, -2, 30);  // wing bottom layer
    vertex(-2, 0, 98); vertex(-2, -30, 98); vertex(-2, 0, 70);  // tail left layer
    vertex( 2, 0, 98); vertex( 2, -30, 98); vertex( 2, 0, 70);  // tail right layer
    endShape();
    beginShape(QUADS);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex( 0, -2, -80); vertex( 0, 2, -80);
    vertex( 100, 2, 30); vertex( 100, -2, 30); vertex( 0, -2, -80); vertex( 0, 2, -80);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(100, -2, 30); vertex(100, 2, 30);
    vertex(-2, 0, 98); vertex(2, 0, 98); vertex(2, -30, 98); vertex(-2, -30, 98);
    vertex(-2, 0, 98); vertex(2, 0, 98); vertex(2, 0, 70); vertex(-2, 0, 70);
    vertex(-2, -30, 98); vertex(2, -30, 98); vertex(2, 0, 70); vertex(-2, 0, 70);
    endShape();

    popMatrix();
}

void serialEvent(Serial port) {
    interval = millis();
    while (port.available() > 0) {
        int ch = port.read();

        if (synced == 0 && ch != '$') return;  // initial synchronization - also used to resync/realign if needed
        synced = 1;
        print ((char)ch);

        if ((serialCount == 1 && ch != 2)
            || (serialCount == 12 && ch != '\r')
            || (serialCount == 13 && ch != '\n'))  {
            serialCount = 0;
            synced = 0;
            return;
        }

        if (serialCount > 0 || ch == '$') {
            teapotPacket[serialCount++] = (char)ch;
            if (serialCount == 14) {
                serialCount = 0; // restart packet byte position

                // get quaternion from data packet
                q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
                q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
                q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
                q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
                for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];

                // set our toxilibs quaternion to new data
                quat.set(q[0], q[1], q[2], q[3]);

                /*
                // below calculations unnecessary for orientation only using toxilibs

                // calculate gravity vector
                gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
                gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
                gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];

                // calculate Euler angles
                euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
                euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1], 2*q[0]*q[0] + 2*q[3]*q[3] - 1);

                // calculate yaw/pitch/roll angles
                ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
                ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] + gravity[2]*gravity[2]));
                ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] + gravity[2]*gravity[2]));

                // output various components for debugging
```

```
            //println("q:\t" + round(q[0]*100.0f)/100.0f + "\t" + round(q[1]*100.0f)/100.0f + "\t" + round(q[2]*100.0f)/100.0f + "\t" + round(q[3]*100.0f)/100.0f);
            //println("euler:\t" + euler[0]*180.0f/PI + "\t" + euler[1]*180.0f/PI + "\t" + euler[2]*180.0f/PI);
            //println("ypr:\t" + ypr[0]*180.0f/PI + "\t" + ypr[1]*180.0f/PI + "\t" + ypr[2]*180.0f/PI);
             */
          }
        }
    }
}

void drawCylinder(float topRadius, float bottomRadius, float tall, int sides)
{
    float angle = 0;
    float angleIncrement = TWO_PI / sides;
    beginShape(QUAD_STRIP);
    for (int i = 0; i < sides + 1; ++i) {
        vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
        vertex(bottomRadius*cos(angle), tall, bottomRadius*sin(angle));
        angle += angleIncrement;
    }
    endShape();

    // If it is not a cone, draw the circular top cap
    if (topRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, 0, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(topRadius * cos(angle), 0, topRadius * sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }

    // If it is not a cone, draw the circular bottom cap
    if (bottomRadius != 0) {
        angle = 0;
        beginShape(TRIANGLE_FAN);

        // Center point
        vertex(0, tall, 0);
        for (int i = 0; i < sides + 1; i++) {
            vertex(bottomRadius * cos(angle), tall, bottomRadius * sin(angle));
            angle += angleIncrement;
        }
        endShape();
    }
}
```

IV- RESULTADOS OBTENIDAS

Los resultados obtenidos fueron mejor de lo esperado ya que la finalidad principal era de solo poner en practica lo visto a lo largo del curso, sin embargo y a base de encontrar problemas durante la elaboración del proyecto. No obstante gracias a la asesoría de terceros se pudo obtener diversos puntos de vista con los que se superaron dichos problemas y concluyendo así el proyecto.

V- COMO CONCLUSIÓN

Aguilar Bárcenas Bayrón Esaú; La elaboración de este proyecto fue complicada debido a al nulo conocimiento del componente MPU6050, y la dificultad para hacer que en visual studio mostrara las coordenadas del MPU-6050. Se hizo uso de varias herramientas de programación como IDE de Arduino, Visual studio y otros. Además, para el correcto funcionamiento del MPU-6050 primero se debió calibrar con ayuda de una librería específica, y se coloco el componente sobre una superficie plana para obtener las coordenadas precisas de lo contrario no funcionaria.

Mendoza Lara Saúl Eugenio: Este proyecto fue un reto, no sabíamos que hacer con el sensor MPU6050, solo se nos mencionó que era y que hace, con el corto tiempo que se tenía se encontró un uso que nos hizo aprender más sobre el uso del sensor, más funciones en el visual studio y el lenguaje visual basic, así como también existen softwares que puedes hacer funcionar los sensores, Arduino y más.

Montante Gaytán Luis Armando: Durante el desarrollo del proyecto se topo con un par de problemas debido a complicaciones en el código, el cual impedía llamar la simulación de manera rápida, pero después de una asesoría por parte de un profesor se consiguió llegar a una solución, la cual fue jugar con el programa para que este llame a un segundo ambiente de programación en el cual se llama a la simulación, todo esto mientras se “engaña” al ambiente principal (Visual Basic).

REFERENCIAS

[1] Sensores. (s.f.). National Institute of Biomedical Imaging and Bioengineering. [Online] Available https://www.nibib.nih.gov/espanol/temas-cientificos/sensores

[2] Tutorial MPU6050, Acelerómetro y Giroscopio. (s.f.). Naylamp Mechatronics - Perú. [Online] Available https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html

[3] Giroscopio - Hardware. (s.f.). FIRST Robotics Competition Documentation. [Online] Available https://docs.wpilib.org/es/stable/docs/hardware/sensors/gyros-hardware.html

[4] Acelerómetro | Omega engineering. (s.f.). Termopares, cÃ©lulas de carga, RTDs, transductores de presiÃ³n, medidores de tensiÃ³n, Wireless y automatización. [Online] Available https://es.omega.com/prodinfo/acelerometro.html#:~:text=Un%20acelerómetro%20es%20un%20dispositivo,del%20movimiento%20de%20una%20estructura.

Aguilar Barcenas Bayron Esau egresado del Centro de Bachillerato Tecnologico Industrial y de Servisios No. 121 actualmente estudiando y cursando la carrera de Ingeniería Electrónica en 4to semestre.

Mendoza Lara Saúl Eugenio. Técnico en mecánica automotriz y técnico en sistemas eléctricos de maquinaria pesada que actualmente permanece cursando la carrera de Ingeniería Electrónica en el Instituto Tecnológico de San Luis Potosí.

Montante Gaytán Luis Armando. Técnico en radiología e imagen, certificado en epidemiologia ambiental y Prevención del suicidio y autolesiones; actualmente permanece estudiando y cursando la carrera de Ingeniería Electrónica.