```
% Code description: this is the main code for the fall detection system
    %Code developed by:

%Adham Saleh, Mahmoud AL-Nuimat, and supervised by Dr. Ala' Khalifeh from

%School of Electrical Eng. and IT  German-Jordanian University
%Amman, Jordan. Email: ala.khalifeh@gju.edu.jo
%Last accessed Jul 2016
%The code is freely available to use as far as it follows Creative Commons (CC) licensing model  https://creativecommons.org/

// 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
// The code is taken from:

//In the fall detection system we used the code provided by //i2cdevlib to calibrate the sensor. Here is the link :
//https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050
```

```
#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


// Include the voice recognition module libraries

#include <SoftwareSerial.h>
```

```
#include "VoiceRecognitionV3.h"

// 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:
 * ========================================================================= */



// 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
```

```
int counter = 0;
#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' };



// Define the Pin 6 & 7 as TX and RX
VR myVR(6,7);
uint8_t records[7]; // save record
uint8_t buf[64];

// Define the Trained Records
```

```
#define OkayRecord    (0)
#define FineRecord    (1)
#define EmergencyRecord    (2)

void printSignature(uint8_t *buf, int len)
{
  int i;
  for(i=0; i<len; i++){
    if(buf[i]>0x19 && buf[i]<0x7F){
      Serial.write(buf[i]);
    }
    else{
      Serial.print("[");
      Serial.print(buf[i], HEX);
      Serial.print("]");
    }
  }
}

void printVR(uint8_t *buf)
{
  Serial.println("VR Index\tGroup\tRecordNum\tSignature");

  Serial.print(buf[2], DEC);
  Serial.print("\t\t");

  if(buf[0] == 0xFF){
    Serial.print("NONE");
  }
  else if(buf[0]&0x80){
    Serial.print("UG ");
    Serial.print(buf[0]&(~0x80), DEC);
  }
  else{
    Serial.print("SG ");
    Serial.print(buf[0], DEC);
  }
  Serial.print("\t");

  Serial.print(buf[1], DEC);
  Serial.print("\t\t");
  if(buf[3]>0){
    printSignature(buf+4, buf[3]);
  }
  else{
    Serial.print("NONE");
  }
  Serial.println("\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() {
// Voice recognition module setup
myVR.begin(9600);

Serial.println("Elechouse Voice Recognition V3 Module   ");

if(myVR.clear() == 0){
    Serial.println("Recognizer cleared.");
  }else{
    Serial.println("Not find VoiceRecognitionModule.");
    Serial.println("Please check connection and restart Arduino.");
    while(1);
  }

// Check the Buffer to see which command is sent to the buffer
if(myVR.load((uint8_t)OkayRecord) >= 0){
    Serial.println("OkayRecord loaded");
  }
  if(myVR.load((uint8_t)FineRecord) >= 0){
    Serial.println("FineRecord loaded");
  }
 if(myVR.load((uint8_t)EmergencyRecord) >= 0){
    Serial.println("EmergencyRecord loaded");
  }

  // join I2C bus (I2Cdev library doesn't do this automatically)
```

```
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)
    #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(9600);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // 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();

    // 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) {
  // turn on the DMP, now that it's ready
  Serial.println(F("Enabling DMP..."));
  mpu.setDMPEnabled(true);

  // enable Arduino interrupt detection
  Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
  attachInterrupt(0, 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() {
    // voice module command received
    int ret;
    ret = myVR.recognize(buf, 50);
   int value =0;
  // if programming failed, don't try to do anything
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
    // .
    // .
    // .
    // if you are really paranoid you can frequently test in between other
    // stuff to see if mpuInterrupt is true, and if so, "break;" from the
    // while() loop to immediately process the MPU data
    // .
    // .
    // .
}

// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
    // reset so we can continue cleanly
    mpu.resetFIFO();
    //Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & 0x02) {
    // wait for correct available data length, should be a VERY short wait
    while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
    mpu.getFIFOBytes(fifoBuffer, packetSize);
```

```
// track FIFO count here in case there is > 1 packet available
// (this lets us immediately read more without waiting for an interrupt)
fifoCount -= packetSize;

#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("xyz\t");
       Serial.print(ypr[0] * 180/M_PI);
       Serial.print("\t");
       Serial.print(ypr[1] * 180/M_PI);
       int p = ypr[1] * 180/M_PI;
       Serial.print("\t");
       Serial.println(ypr[2] * 180/M_PI);

        // Threshold range for the fall
     if (p <- 70 && p > - 90) {
     counter++;
     delay(1000);
   }
        // wait ten seconds in order to detect the dall
 if (p <-70 && p>-90 && counter == 10) {
 Serial.println("fall detected");
 Serial.println("Help needed ?");
 //delay(2000);
 }

 // Initialize the Buffer
 int ret;
 ret = myVR.recognize(buf, 50);
 delay(2000);

// if the ret is greater than zero, this means there is a command sent to the buffer
if(ret>0){
```

```
switch(buf[1]){

// patient responded with Okay
case OkayRecord:
Serial.println("patient responded with Okay");
Serial.println("thank god be careful next time");
counter = 0;
break;

// Patient responded with Fine
case FineRecord:
Serial.println("patient responded with Fine");
Serial.println("thank god be careful next time");
counter = 0;
break;

// Patient need help
case EmergencyRecord:
Serial.println("patient responded with Fine");
Serial.println("EMERGENCY!!!!!!!!!");
counter = 0;
break;

// Patient reponded with untrained command to the voice module
default:
Serial.println("Record function undefined");
break;


}
// Print the buffer values
printVR(buf);
```

```
}
}

/* We used this counter in case the patient responded with nothing
which means that the patient is unconscious.
while(p <- 70 && p > - 90 && counter == 3) {
    value ++;
    delay(1000);
Serial.println(value);
if(value ==5){
  Serial.println("No Respond, EMERGECNY!!!");
delay(1000);
counter = 0;
value = 0;
}
}



    #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);
    }
}
```