ISLAMIC UNIVERSITY OF TECHNOLOGY, (IUT)

Accelerometer based gesture recognition system

By

Jisan Mahmud (082471)

Murad Mahbub Abrar (082461)

Zakaria Rabbi Toha (082475)

Bachelor of Science in Electrical and Electronic Engineering

Academic Year: 2011-2012

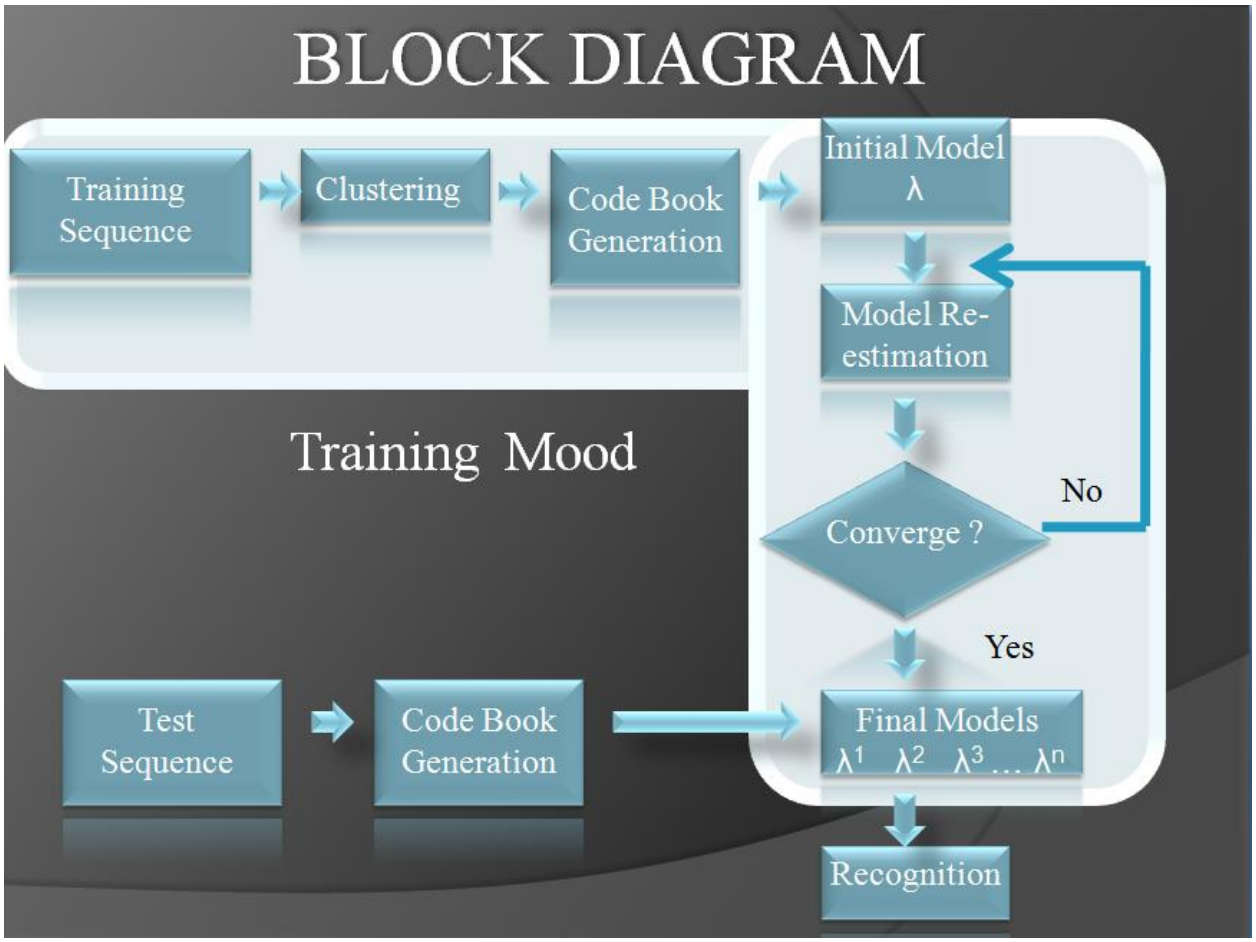
Department of Electrical and Electronics Engineering

Dr. Md. Shahid Ullah

Professor & Head of the

Department of EEE, IUT.

System Block Diagram:



Accelerometer

What is an accelerometer

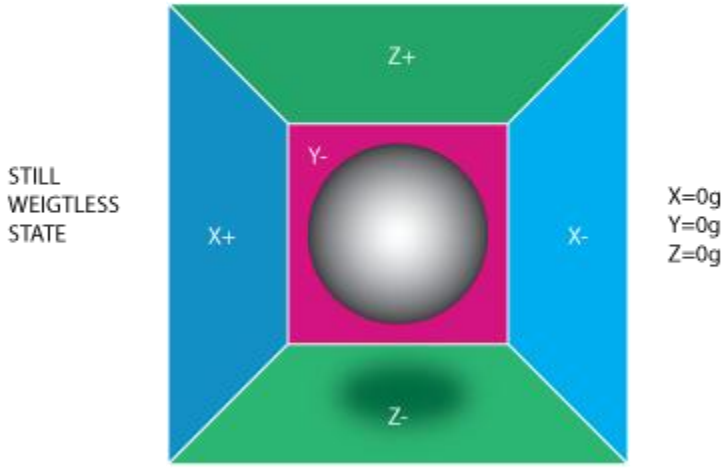
An accelerometer is an electromechanical device that will measure acceleration forces. These forces may be static, like the constant force of gravity pulling at your feet, or they could be dynamic - caused by moving or vibrating the accelerometer.

How do accelerometers work

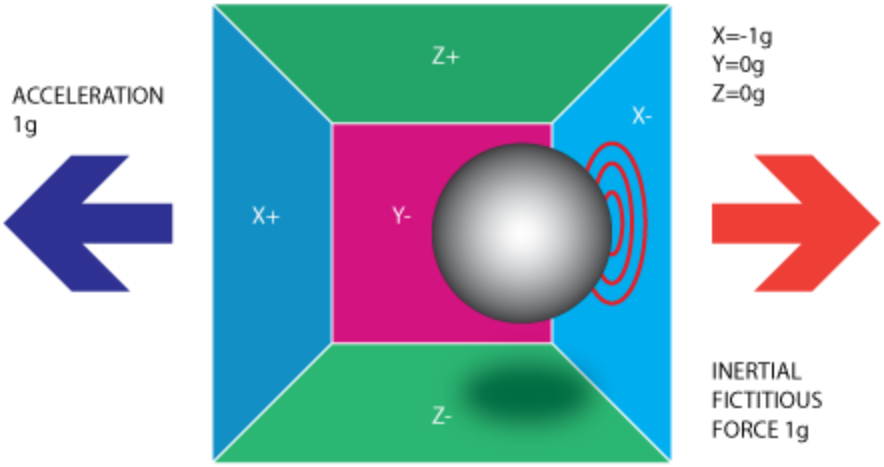
There are many different ways to make an accelerometer! Some accelerometers use the piezoelectric effect - they contain microscopic crystal structures that get stressed by accelerative forces, which causes a voltage to be generated. Another way to do it is by sensing changes in capacitance. If you have two microstructures next to each other, they have a certain capacitance between them. If an accelerative force moves one of the structures, then the capacitance will change. Add some circuitry to convert from capacitance to voltage, and you will get an

accelerometer. There are even more methods, including use of the piezoresistive effect, hot air bubbles, and light.

To understand this unit we'll start with the accelerometer. When thinking about accelerometers it is often useful to image a box in shape of a cube with a ball inside it. You may imagine something else like a cookie or a donut , but I'll imagine a ball:

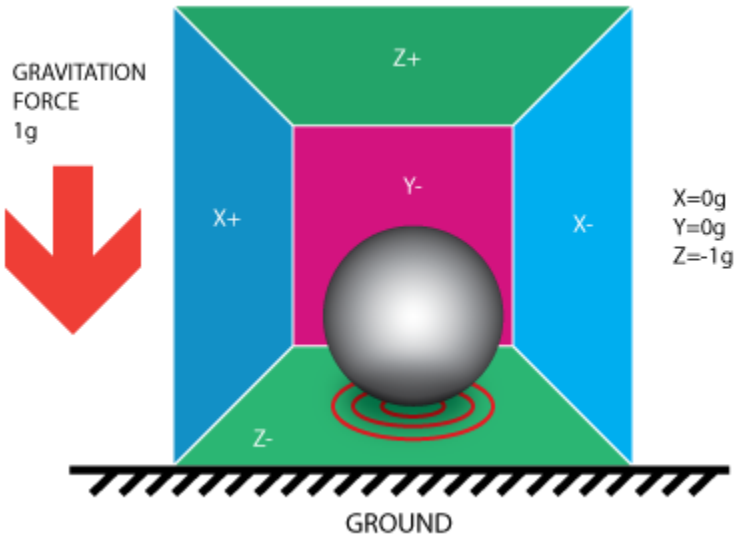


If we take this box in a place with no gravitation fields or for that matter with no other fields that might affect the ball's position - the ball will simply float in the middle of the box. You can imagine the box is in outer-space far-far away from any cosmic bodies, or if such a place is hard to find imagine at least a space craft orbiting around the planet where everything is in weightless state . From the picture above you can see that we assign to each axis a pair of walls (we removed the wall Y+ so we can look inside the box). Imagine that each wall is pressure sensitive. If we move suddenly the box to the left (we accelerate it with acceleration 1g = 9.8m/s^2), the ball will hit the wall X-. We then measure the pressure force that the ball applies to the wall and output a value of -1g on the X axis.



Please note that the accelerometer will actually detect a force that is directed in the opposite direction from the acceleration vector. This force is often called Inertial Force or Fictitious Force . One thing you should learn from this is that an accelerometer measures acceleration indirectly through a force that is applied to one of it's walls (according to our model, it might be a spring or something else in real life accelerometers). This force can be caused by the acceleration , but as we'll see in the next example it is not always caused by acceleration.

If we take our model and put it on Earth the ball will fall on the Z- wall and will apply a force of 1g on the bottom wall, as shown in the picture below:

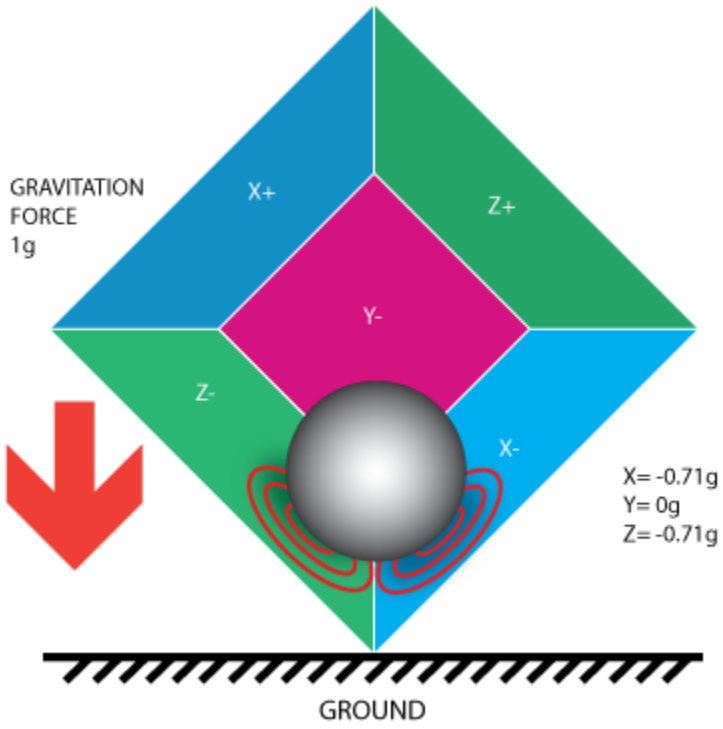


In this case the box isn't moving but we still get a reading of -1g on the Z axis. The pressure that the ball has applied on the wall was caused by a gravitation force. In theory it could be a different type of force - for example, if you imagine that our ball is metallic, placing a magnet

next to the box could move the ball so it hits another wall. This was said just to prove that in essence accelerometer measures force not acceleration. It just happens that acceleration causes an inertial force that is captured by the force detection mechanism of the accelerometer.

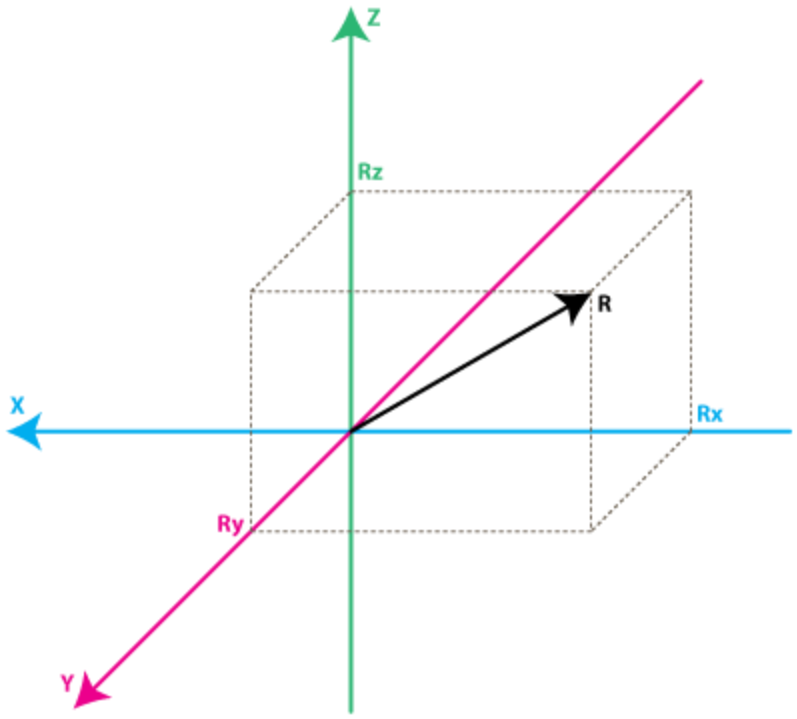
While this model is not exactly how a MEMS sensor is constructed it is often useful in solving accelerometer related problems. There are actually similar sensors that have metallic balls inside, they are called tilt switches, however they are more primitive and usually they can only tell if the device is inclined within some range or not, not the extent of inclination.

So far we have analyzed the accelerometer output on a single axis and this is all you'll get with a single axis accelerometers. The real value of triaxial accelerometers comes from the fact that they can detect inertial forces on all three axes. Let's go back to our box model, and let's rotate the box 45 degrees to the right. The ball will touch 2 walls now: Z- and X- as shown in the picture below:



The values of 0.71 are not arbitrary, they are actually an approximation for SQRT(1/2). This will become more clear as we introduce our next model for the accelerometer.

In the previous model we have fixed the gravitation force and rotated our imaginary box. In last 2 examples we have analyzed the output in 2 different box positions, while the force vector remained constant. While this was useful in understanding how the accelerometer interacts with outside forces, it is more practical to perform calculations if we fix the coordinate system to the axes of the accelerometer and imagine that the force vector rotates around us.



Please have a look at the model above, I preserved the colors of the axes so you can make a mental transition from the previous model to the new one. Just imagine that each axis in the new model is perpendicular to the respective faces of the box in the previous model. The vector R is the force vector that the accelerometer is measuring (it could be either the gravitation force or the inertial force from the examples above or a combination of both). Rx, Ry, Rz are projection of the R vector on the X,Y,Z axes. Please notice the following relation:

R^2 = Rx^2 + Ry^2 + Rz^2 **(Eq. 1)**

which is basically the equivalent of the Pythagorean theorem in 3D.

Remember that a little bit earlier I told you that the values of SQRT(1/2) ~ 0.71 are not random. If you plug them in the formula above, after recalling that our gravitation force was 1 g we can verify that:

1^2 = (-SQRT(1/2))^2 + 0 ^2 + (-SQRT(1/2))^2

simply by substituting R=1, Rx = -SQRT(1/2), Ry = 0 , Rz = -SQRT(1/2) in **Eq.1**

After a long preamble of theory we're getting closer to real life accelerometers. The values Rx, Ry, Rz are actually linearly related to the values that your real-life accelerometer will output and that you can use for performing various calculations.

Before we get there let's talk a little about the way accelerometers will deliver this information to us. Most accelerometers will fall in two categories: digital and analog. Digital accelerometers

will give you information using a serial protocol like I2C , SPI or USART, while analog accelerometers will output a voltage level within a predefined range that you have to convert to a digital value using an ADC (analog to digital converter) module. I will not go into much detail about how ADC works, partly because it is such an extensive topic and partly because it is different from one platform to another. Some microcontroller will have a built-in ADC modules some of them will need external components in order to perform the ADC conversions. No matter what type of ADC module you use you'll end up with a value in a certain range. For example a 10-bit ADC module will output a value in the range of 0..1023, note that 1023 = 2^10 -1. A 12-bit ADC module will output a value in the range of 0..4095, note that 4095 = 2^12-1.

Let's move on by considering a simple example, suppose our 10bit ADC module gave us the following values for the three accelerometer channels (axes):

AdcRx = 586
AdcRy = 630
AdcRz = 561

Each ADC module will have a reference voltage, let's assume in our example it is 3.3V. To convert a 10bit adc value to voltage we use the following formula:

VoltsRx = AdcRx * Vref / 1023

A quick note here: that for 8bit ADC the last divider would be 255 = 2 ^ 8 -1 , and for 12bit ADC last divider would be 4095 = 2^12 -1.

Applying this formula to all 3 channels we get:

VoltsRx = 586 * 3.3V / 1023 =~ 1.89V (we round all results to 2 decimal points)
VoltsRy = 630 * 3.3V / 1023 =~ 2.03V
VoltsRz = 561 * 3.3V / 1023 =~ 1.81V

Each accelerometer has a zero-g voltage level, you can find it in specs, this is the voltage that corresponds to 0g. To get a signed voltage value we need to calculate the shift from this level. Let's say our 0g voltage level is VzeroG = 1.65V. We calculate the voltage shifts from zero-g voltage as follows::

DeltaVoltsRx = 1.89V - 1.65V = 0.24V
DeltaVoltsRy = 2.03V - 1.65V = 0.38V
DeltaVoltsRz = 1.81V - 1.65V = 0.16V

We now have our accelerometer readings in Volts , it's still not in g (9.8 m/s^2), to do the final conversion we apply the accelerometer sensitivity, usually expressed in mV/g. Lets say our Sensitivity = 478.5mV/g = 0.4785V/g. Sensitivity values can be found in accelerometer specifications. To get the final force values expressed in g we use the following formula:

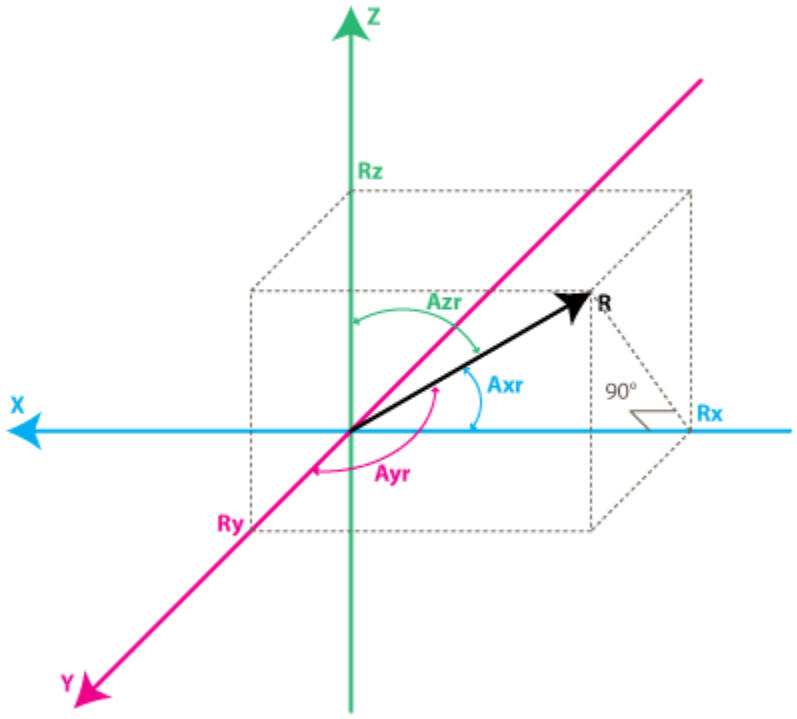
Rx = DeltaVoltsRx / Sensitivity

Rx = 0.24V / 0.4785V/g =~ 0.5g
Ry = 0.38V / 0.4785V/g =~ 0.79g
Rz = 0.16V / 0.4785V/g =~ 0.33g

We could of course combine all steps in one formula, but I went through all the steps to make it clear how you go from ADC readings to a force vector component expressed in g.

Rx = (AdcRx * Vref / 1023 - VzeroG) / Sensitivity **(Eq.2**)
Ry = (AdcRy * Vref / 1023 - VzeroG) / Sensitivity
Rz = (AdcRz * Vref / 1023 - VzeroG) / Sensitivity

We now have all 3 components that define our inertial force vector, if the device is not subject to other forces other than gravitation, we can assume this is the direction of our gravitation force vector. If you want to calculate inclination of device relative to the ground you can calculate the angle between this vector and Z axis. If you are also interested in per-axis direction of inclination you can split this result into 2 components: inclination on the X and Y axis that can be calculated as the angle between gravitation vector and X / Y axes. Calculating these angles is more simple than you might think, now that we have calculated the values for Rx,Ry and Rz. Let's go back to our last accelerometer model and do some additional notations:



The angles that we are interested in are the angles between X,Y,Z axes and the force vector R. We'll define these angles as Axr, Ayr, Azr. You can notice from the right-angle triangle formed by R and Rx that:

cos(Axr) = Rx / R , and similarly :
cos(Ayr) = Ry / R
cos(Azr) = Rz / R

We can deduct from **Eq.1** that R = SQRT(Rx^2 + Ry^2 + Rz^2).

We can find now our angles by using arccos() function (the inverse cos() function):

Axr = arccos(Rx/R)
Ayr = arccos(Ry/R)
Azr = arccos(Rz/R)

We've gone a long way to explain the accelerometer model, just to come up to these formulas. Depending on your applications you might want to use any intermediate formulas that we have derived. We'll also introduce the gyroscope model soon, and we'll see how accelerometer and gyroscope data can be combined to provide even more accurate inclination estimations.

But before we do that let's do some more useful notations:

cosX = cos(Axr) = Rx / R
cosY = cos(Ayr) = Ry / R
cosZ = cos(Azr) = Rz / R

This triplet is often called Direction Cosine , and it basically represents the unit vector (vector with length 1) that has same direction as our R vector. You can easily verify that:

SQRT(cosX^2 + cosY^2 + cosZ^2) = 1

This is a nice property since it absolve us from monitoring the modulus(length) of R vector. Often times if we're just interested in direction of our inertial vector, it makes sense to normalize it's modulus in order to simplify other calculations.

Accelerometer MMA7361

MMA7361L is a Three axis Low-G accelerometer Module which gives selectable acceleration range from ±1.5g or ±6g. Board has all the necessary components required for the chip. Board made up of high quality silver plated double sided PCB for giving extra strength to the connectors.
MMA7361L accelerometer has building signal conditioning, a 1-pole low pass filter, temperature compensation, self test, 0g-Detect which detects linear freefall, and g Select which allows for the selection between 2 sensitivities. It also includes a Sleep Mode that makes it ideal for handheld battery powered electronics.
Possible applications of this board includes 3D-Gaming: Tilt and Motion Sensing, Event

recorder, HDD MP3 Player: Freefall Detection, Laptop PC: Freefall Detection, Anti-Theft, Cell Phone: Image Stability, Text Scroll, Motion Dialing, E-Compass, Pedometer: Motion Sensing, PDA: Text Scroll, Navigation and Dead Reckoning: E Compass Tilt Compensation, Robotics: Motion Sensing etc.

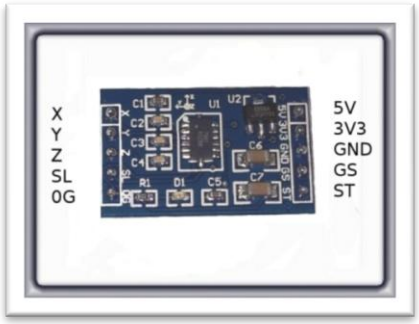
Features

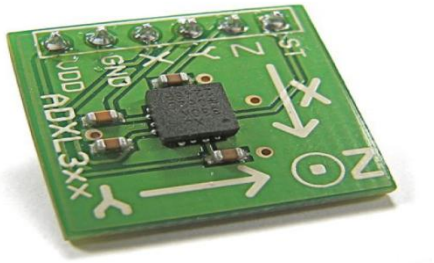
• 3mm x 5mm x 1.0mm LGA-14 Package
• Low Current Consumption: 400 μA
• Sleep Mode: 3 μA
• Low Voltage Operation: 2.2 V – 3.6 V
• High Sensitivity (800 mV/g @ 1.5g)
• Selectable Sensitivity (±1.5g, ±6g)
• Fast Turn On Time (0.5 ms Enable Response Time)
• Self Test for Freefall Detect Diagnosis
• 0g-Detect for Freefall Protection
• Signal Conditioning with Low Pass Filter
• Robust Design, High Shocks Survivability
• RoHS Compliant
• Environmentally Preferred Product
• Low Cost

Typical Applications

• 3D Gaming: Tilt and Motion Sensing, Event Recorder
• HDD MP3 Player: Freefall Detection
• Laptop PC: Freefall Detection, Anti-Theft
• Cell Phone: Image Stability, Text Scroll, Motion Dialing, E-Compass
• Pedometer: Motion Sensing
• PDA: Text Scroll
• Navigation and Dead Reckoning: E-Compass Tilt Compensation
• Robotics: Motion Sensing

Picture of Accelerometer





Pin Configurations

Pin#	Pin name	Description
1	XOUT	X direction output Voltage
2	YOUT	Y direction output Voltage
3	ZOUT	Z direction output Voltage
4	Vss	Supply Ground
5	Vdd	3.3V supply voltage
6	Sleep (Active Low)	Logic input pin for sleep mode
7	0g-Detect	Freefall digital logic output signal
8	g-select	Logic input pin to select G level
9	ST	Self-test (logic 0: normal mode; logic 1: selftest)
10	NC	Leave Unconnected

Pin Descriptions:

XOUT: This provides the change in acceleration in X direction in terms of analog voltage
YOUT: This provides the change in acceleration in Y direction in terms of analog voltage
ZOUT: This provides the change in acceleration in Z direction in terms of analog voltage
Vss: Supply Ground: This is the supply ground pin for system.
Vdd: Supply voltage: apply the 3.3 v supply to this pin.

Sleep(Active Low): This is control input pin, logic High for normal mode where system consume only 400µA current and logic low for Sleep mode where system consume only 10µA current.

0g-Detect: The sensor offers a 0g-Detect feature that provides a logic high signal when all three axes are at 0g. This feature enables the application of Linear Freefall protection if the signal is connected to an interrupt pin or a poled I/O pin on a microcontroller.

G-select: The G-Select feature allows for the selection between two sensitivities. Depending on the logic input placed on this pin, the device internal gain will be changed allowing it to function with a 1.5g or 6g sensitivity.

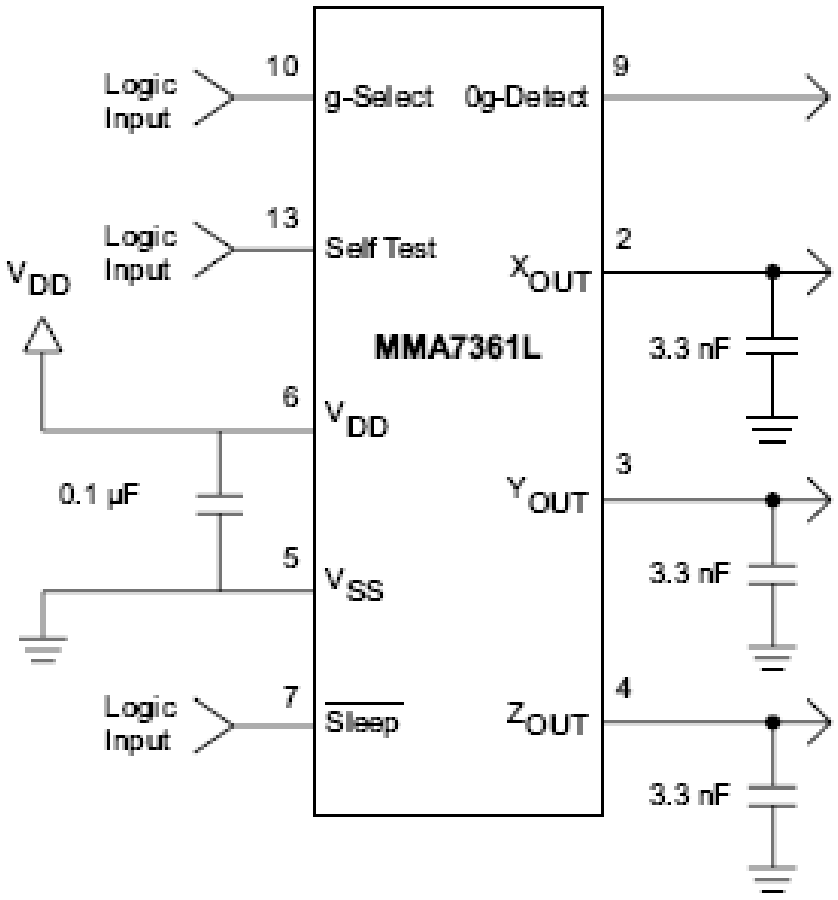
G-Select (logic level) G-Range Sensitivity

0 1.5G 800mV/G

1 6G 206mV/G

ST(Self-test): Self Test allows the verification of the mechanical and electrical integrity of the accelerometer at any time before or after installation.

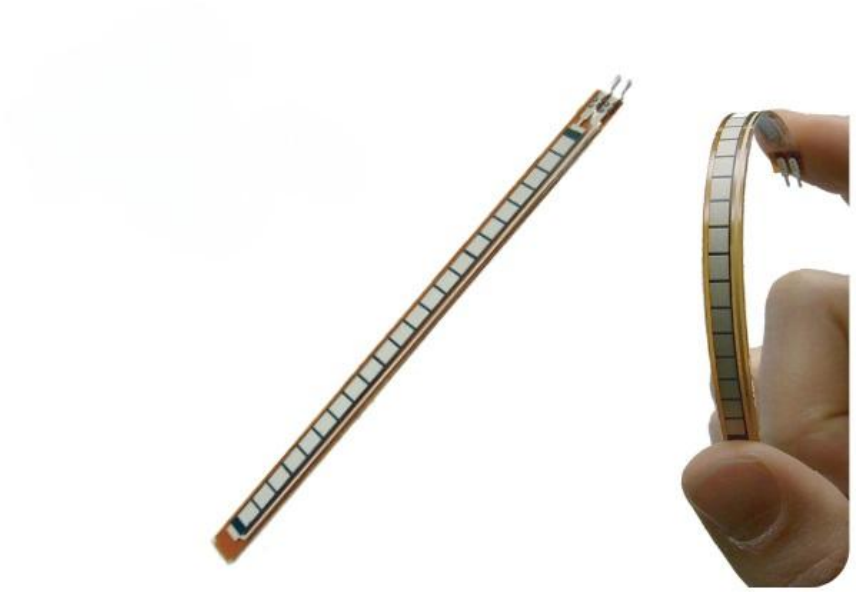
Circuit Diagram:



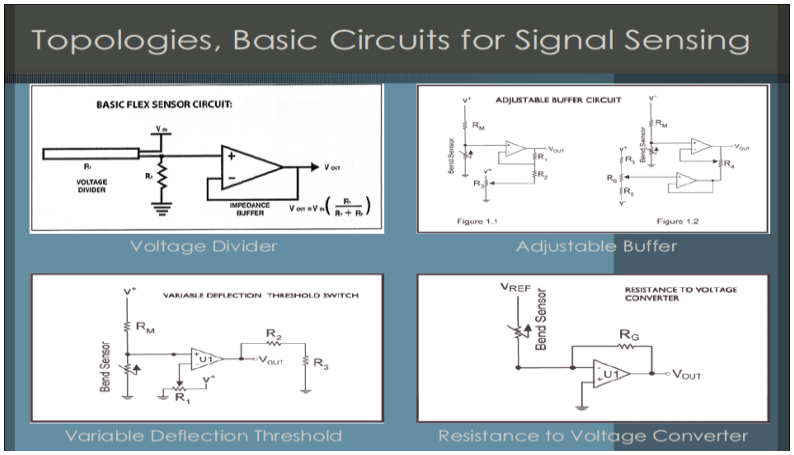
Accelerometer with recommended connections

0.1µF power supply decoupling capacitors placed near the supply pins. Use a 3.3nF capacitor on the outputs of the accelerometer to minimize clock noise (from the switched capacitor filter circuit). A/D sampling rate and any external power supply switching frequency should be selected such that they do not interfere with the internal accelerometer sampling frequency (11 kHz for the sampling frequency). This will prevent aliasing errors.
The compact board with few mounted components gives you the analog output voltage on the connector pins.

Flex Sensors



Flex Sensor Connection Diagram:



Hidden Marcov Model:

A **hidden Markov model** (**HMM**) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (*hidden*) states. An HMM can be considered as the simplest dynamic Bayesian network. The mathematics behind the HMM was developed by L. E. Baum and coworkers. It is closely related to an earlier work on optimal nonlinear filtering problem (stochastic processes) by Ruslan L. Stratonovich, who was the first to describe the forward-backward procedure.

In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Note that the adjective 'hidden' refers to the state sequence through which the model passes, not to the parameters of the model; even if the model parameters are known exactly, the model is still 'hidden'.

Hidden Markov models are especially known for their application in temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bioinformatics.

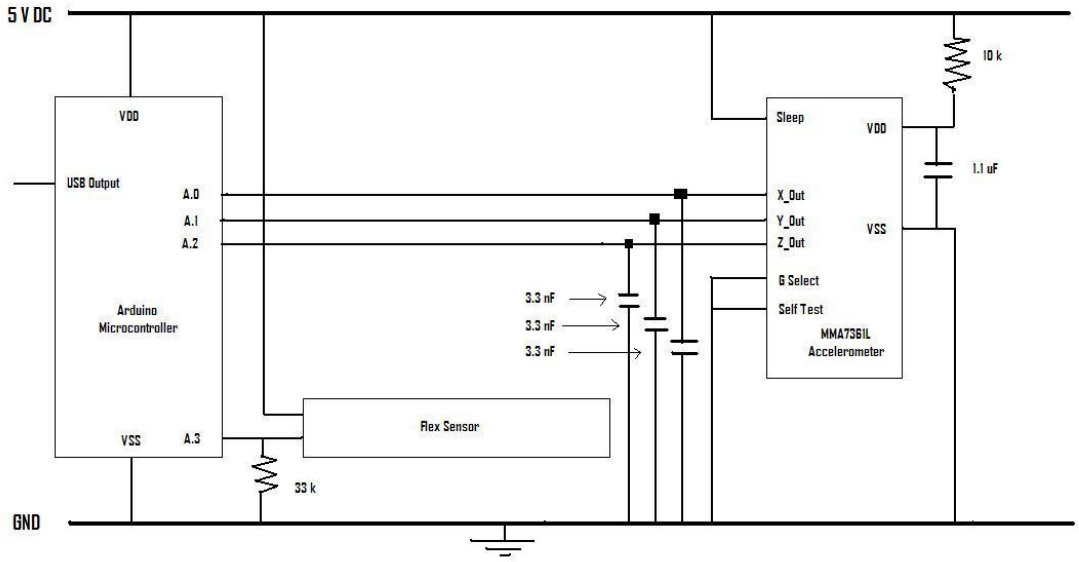
A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

In its discrete form, a hidden Markov process can be visualized as a generalization of the Urn problem In a room that is not visible to an observer there is a genie. The room contains urns X1, X2, X3, ... each of which contains a known mix of balls, each ball labeled y1, y2, y3, ... The genie chooses an urn in that room and randomly draws a ball from that urn. It then puts the ball onto a conveyor belt, where the observer can observe the sequence of the balls but not the sequence of urns from which they were drawn. The genie has some procedure to choose urns; the choice of the urn for the n-th ball depends only upon a random number and the choice of the urn for the $(n-1)$-th ball. The choice of urn does not directly depend on the urns chosen before this single previous urn; therefore, this is called a Markov process. It can be described by the upper part of Figure 1.

The Markov process itself cannot be observed, and only the sequence of labeled balls can be observed, thus this arrangement is called a "hidden Markov process". This is illustrated by the lower part of the diagram shown in Figure 1, where one can see that balls y1, y2, y3, y4 can be drawn at each state. Even if the observer knows the composition of the urns and has just observed a sequence of three balls, *e.g.* y1, y2 and y3 on the conveyor belt, the observer still cannot be sure which urn (*i.e.*, at which state) the genie has drawn the third ball from. However,

the observer can work out other details, such as the identity of the urn the genie is most likely to have drawn the third ball from.

Main Circuit Diagram :



Circuit Diagram for Accelerometer & Flex Sensor Based Hand Gesture Recognition

SOFTWARE

This is the code to program the microcontroller:

We have to UPLOAD SRV.PDE (OR ADIOSRV.PDE) (Given to the CD) TO THE ARDUINO BOARD (to be done only once):

The srv.pde (or adiosrv.pde) is the "server" program that will continuously run on the microcontroller. It listens for MATLAB commands arriving from the serial port, executes the commands, and, if needed, returns a result.

The program is

```
/* Analog and Digital Input and Output Server for MATLAB    */
/* Giampiero Campa, Copyright 2009 The MathWorks, Inc       */

/* This file is meant to be used with the MATLAB arduino IO
   package, however, it can be used from the IDE environment
   (or any other serial terminal) by typing commands like:

0e0   : assigns digital pin #4 (e) as input
0f1   : assigns digital pin #5 (f) as output
0n1   : assigns digital pin #13 (n) as output

1c    : reads digital pin #2 (c)
1e    : reads digital pin #4 (e)
2n0   : sets digital pin #13 (n) low
2n1   : sets digital pin #13 (n) high
2f1   : sets digital pin #5 (f) high
2f0   : sets digital pin #5 (f) low
4j2   : sets digital pin #9 (j) to  50=ascii(2) over 255
4jz   : sets digital pin #9 (j) to 122=ascii(z) over 255
3a    : reads analog pin #0 (a)
3f    : reads analog pin #5 (f)

5a    : reads status (attached/detached) of servo #1
5b    : reads status (attached/detached) of servo #2
6a1   : attaches servo #1
8az   : moves servo #1 of 122 degrees (122=ascii(z))
7a    : reads servo #1 angle
6a0   : detaches servo #1

A1z   : sets speed of motor #1 to 122 over 255  (122=ascii(z))
A4A   : sets speed of motor #4 to 65 over 255 (65=ascii(A))
B1f   : runs motor #1 forward (f=forward)
B4b   : runs motor #1 backward (b=backward)
B1r   : releases motor #1 (r=release)

C12   : sets speed of stepper motor #1 to 50 rpm  (50=ascii(2))
C2Z   : sets speed of stepper motor #2 to 90 rpm  (90=ascii(Z))
D1fsz : do 122 steps on motor #1 forward in single (s) mode
D1biA : does 65 steps on motor #1 backward in interleave (i) mode
D2fdz : does 122 steps on motor #1 forward in double (d) mode
D2bmA : does 65 steps on motor #2 backward in microstep (m) mode
D1r   : releases motor #1 (r=release)
D2r   : releases motor #2 (r=release)

R0    : sets analog reference to DEFAULT
```

```
R1    : sets analog reference to INTERNAL
R2    : sets analog reference to EXTERNAL

99    : returns script type (1 basic, 2 motor, 3 general)

#include <AFMotor.h>
#include <Servo.h>

/* define internal for the MEGA as 1.1V (as as for the 328)  */
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define INTERNAL INTERNAL1V1
#endif

/* create and initialize servos                              */
Servo servo1;
Servo servo2;

/* create and initialize motors                              */
AF_Stepper stm1(200, 1);
AF_Stepper stm2(200, 2);
AF_DCMotor dcm1(1, MOTOR12_64KHZ); // create motor #1, 64KHz pwm
AF_DCMotor dcm2(2, MOTOR12_64KHZ); // create motor #2, 64KHz pwm
AF_DCMotor dcm3(3, MOTOR12_64KHZ); // create motor #3, 64KHz pwm
AF_DCMotor dcm4(4, MOTOR12_64KHZ); // create motor #4, 64KHz pwm

void setup() {
  /* initialize serial                                       */
  Serial.begin(115200);
}


void loop() {

  /* variables declaration and initialization                */

  static int  s   = -1;    /* state                          */
  static int  pin = 13;    /* generic pin number             */
  static int  srv =  2;    /* generic servo number           */
  static int  dcm =  4;    /* generic dc motor number        */

  static int  stm =  2;    /* generic stepper motor number   */
  static int  dir =  0;    /* direction (stepper)            */
  static int  sty =  0;    /* style (stepper)                */

  int  val =  0;           /* generic value read from serial */
  int  agv =  0;           /* generic analog value           */
  int  dgv =  0;           /* generic digital value          */

  /* The following instruction constantly checks if anything
```

```
   is available on the serial port. Nothing gets executed in
   the loop if nothing is available to be read, but as soon
   as anything becomes available, then the part coded after
   the if statement (that is the real stuff) gets executed */

if (Serial.available() >0) {

  /* whatever is available from the serial is read here    */
  val = Serial.read();

  /* This part basically implements a state machine that
     reads the serial port and makes just one transition
     to a new state, depending on both the previous state
     and the command that is read from the serial port.
     Some commands need additional inputs from the serial
     port, so they need 2 or 3 state transitions (each one
     happening as soon as anything new is available from
     the serial port) to be fully executed. After a command
     is fully executed the state returns to its initial
     value s=-1                                            */


  switch (s) {

    /* s=-1 means NOTHING RECEIVED YET ****************** */
    case -1:

    /* calculate next state when s=-1                      */
    if (val>47 && val<90) {
         /* the first received value indicates the mode
         49 is ascii for 1, ... 90 is ascii for Z
         s=0 is change-pin mode
         s=10 is DI;  s=20 is DO;  s=30 is AI;  s=40 is AO;
         s=50 is servo status; s=60 is aervo attach/detach;
         s=70 is servo read;   s=80 is servo write
         s=90 is query script type (1 basic, 2 motor, 3 general)
         s=170 is dc motor set speed
         s=180 is dc motor run/release
         s=190 is stepper motor set speed
         s=200 is stepper motor run/release
         s=340 is change analog reference
                                                          */
      s=10*(val-48);
    }

    /* the following statements are needed to handle
       unexpected first values coming from the serial (if
       the value is unrecognized then it defaults to s=-1) */
    if ((s>90 && s<170) || (s>200 && s!=340)) {
      s=-1;
```

```
}

/* the break statements gets out of the switch-case, so
/* we go back to line 97 and wait for new serial data  */
break; /* s=-1 (initial state) taken care of           */



/* s=0 or 1 means CHANGE PIN MODE                       */

case 0:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<167) {
  pin=val-97;                /* calculate pin          */
  s=1; /* next we will need to get 0 or 1 from serial  */
}
else {
  s=-1; /* if value is not a pin then return to -1     */
}
break; /* s=0 taken care of                            */


case 1:
/* the third received value indicates the value 0 or 1 */
if (val>47 && val<50) {
  /* set pin mode                                      */
  if (val==48) {
    pinMode(pin,INPUT);
  }
  else {
    pinMode(pin,OUTPUT);
  }
}
s=-1;  /* we are done with CHANGE PIN so go to -1      */
break; /* s=1 taken care of                            */



/* s=10 means DIGITAL INPUT ************************** */

case 10:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<167) {
  pin=val-97;                /* calculate pin          */
  dgv=digitalRead(pin);      /* perform Digital Input  */
  Serial.println(dgv);       /* send value via serial  */
}
s=-1;  /* we are done with DI so next state is -1      */
```

```
break; /* s=10 taken care of                              */



/* s=20 or 21 means DIGITAL OUTPUT ****************** */

case 20:
/* the second received value indicates the pin
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<167) {
  pin=val-97;                /* calculate pin          */
  s=21; /* next we will need to get 0 or 1 from serial */
}
else {
  s=-1; /* if value is not a pin then return to -1     */
}
break; /* s=20 taken care of                            */

case 21:
/* the third received value indicates the value 0 or 1 */
if (val>47 && val<50) {
  dgv=val-48;                /* calculate value        */
    digitalWrite(pin,dgv);     /* perform Digital Output */
}
s=-1;  /* we are done with DO so next state is -1      */
break; /* s=21 taken care of                            */



/* s=30 means ANALOG INPUT ************************** */

case 30:
/* the second received value indicates the pin
   from abs('a')=97, pin 0, to abs('f')=102, pin 6,
   note that these are the digital pins from 14 to 19
   located in the lower right part of the board        */
if (val>96 && val<113) {
  pin=val-97;                /* calculate pin          */
  agv=analogRead(pin);       /* perform Analog Input   */
    Serial.println(agv);       /* send value via serial  */
}
s=-1;  /* we are done with AI so next state is -1      */
break; /* s=30 taken care of                            */



/* s=40 or 41 means ANALOG OUTPUT ******************* */

case 40:
/* the second received value indicates the pin
```

```
   from abs('c')=99, pin 2, to abs('t')=116, pin 19    */
if (val>98 && val<167) {
  pin=val-97;                /* calculate pin          */
  s=41; /* next we will need to get value from serial  */
}
else {
  s=-1; /* if value is not a pin then return to -1     */
}
break; /* s=40 taken care of                           */


case 41:
/* the third received value indicates the analog value */
analogWrite(pin,val);        /* perform Analog Output  */
s=-1;  /* we are done with AO so next state is -1      */
break; /* s=41 taken care of                           */



/* s=50 means SERVO STATUS (ATTACHED/DETACHED) ******* */

case 50:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
  srv=val-96;                /* calculate srv          */
  if (srv==1) dgv=servo1.attached();    /* read status */
  if (srv==2) dgv=servo2.attached();
  Serial.println(dgv);       /* send value via serial  */
}
s=-1;  /* we are done with servo status so return to -1*/
break; /* s=50 taken care of                           */



/* s=60 or 61 means SERVO ATTACH/DETACH ************** */

case 60:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
  srv=val-96;                /* calculate srv          */
  s=61; /* next we will need to get 0 or 1 from serial */
}
else {
  s=-1; /* if value is not a servo then return to -1   */
}
break; /* s=60 taken care of                           */
```

```
case 61:
/* the third received value indicates the value 0 or 1
   0 for detach and 1 for attach                        */
if (val>47 && val<50) {
  dgv=val-48;                /* calculate value        */
  if (srv==1) {
    if (dgv) servo1.attach(10);      /* attach servo 1 */
    else servo1.detach();            /* detach servo 1 */
  }
  if (srv==2) {
    if (dgv) servo2.attach(9);       /* attach servo 2 */
    else servo2.detach();            /* detach servo 2 */
  }
}
s=-1;  /* we are done with servo attach/detach so -1   */
break; /* s=61 taken care of                           */



/* s=70 means SERVO READ ***************************** */

case 70:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
  srv=val-96;                /* calculate servo number */
  if (srv==1) agv=servo1.read();         /* read value */
  if (srv==2) agv=servo2.read();
Serial.println(agv);       /* send value via serial  */
}
s=-1;  /* we are done with servo read so go to -1 next */
break; /* s=70 taken care of                           */



/* s=80 or 81 means SERVO WRITE   ******************** */

case 80:
/* the second received value indicates the servo number
   from abs('a')=97, servo1, on top, uses digital pin 10
   to abs('b')=98, servo2, bottom, uses digital pin 9  */
if (val>96 && val<99) {
  srv=val-96;                /* calculate servo number */
  s=81; /* next we will need to get value from serial  */
}
else {
  s=-1; /* if value is not a servo then return to -1   */
```

```
}
break; /* s=80 taken care of                              */


case 81:
/* the third received value indicates the servo angle  */
if (srv==1) servo1.write(val);          /* write value */
if (srv==2) servo2.write(val);
s=-1;  /* we are done with servo write so go to -1 next*/
break; /* s=81 taken care of                              */



/* s=90 means Query Script Type
   (1 basic, 2 motor, 3 general)                        */
case 90:
if (val==57) {
  /* if string sent is 99  send script type via serial */
  Serial.println(3);
}
s=-1;  /* we are done with this so next state is -1    */
break; /* s=90 taken care of                              */



/* s=170 or 171 means DC MOTOR SET SPEED  ************ */

case 170:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('4')=52, motor4    */
if (val>48 && val<53) {
  dcm=val-48;                /* calculate motor number */
  s=171; /* next we will need to get value from serial */
}
else {
  s=-1; /* if value is not a motor then return to -1   */
}
break; /* s=170 taken care of                             */


case 171:
/* the third received value indicates the motor speed  */
if (dcm==1) dcm1.setSpeed(val);
if (dcm==2) dcm2.setSpeed(val);
if (dcm==3) dcm3.setSpeed(val);
if (dcm==4) dcm4.setSpeed(val);
s=-1;  /* we are done with servo write so go to -1 next*/
break; /* s=171 taken care of                             */
```

```
/* s=180 or 181 means DC MOTOR RUN/RELEASE  ********** */
case 180:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('4')=52, motor4    */
if (val>48 && val<53) {
  dcm=val-48;                /* calculate motor number */
  s=181; /* next we will need to get value from serial */
}
else {
  s=-1; /* if value is not a motor then return to -1   */
}
break; /* s=180 taken care of                          */

case 181:
/* the third received value indicates forward, backward,
   release, with characters 'f', 'b', 'r', respectively,
   that have ascii codes 102, 98 and 114                */
if (dcm==1) {
  if (val==102) dcm1.run(FORWARD);
  if (val==98)  dcm1.run(BACKWARD);
  if (val==114) dcm1.run(RELEASE);
}
if (dcm==2) {
  if (val==102) dcm2.run(FORWARD);
  if (val==98)  dcm2.run(BACKWARD);
  if (val==114) dcm2.run(RELEASE);
}
if (dcm==3) {
  if (val==102) dcm3.run(FORWARD);
  if (val==98)  dcm3.run(BACKWARD);
  if (val==114) dcm3.run(RELEASE);
}
if (dcm==4) {
  if (val==102) dcm4.run(FORWARD);
  if (val==98)  dcm4.run(BACKWARD);
  if (val==114) dcm4.run(RELEASE);
}
s=-1;  /* we are done with motor run so go to -1 next  */
break; /* s=181 taken care of                          */



/* s=190 or 191 means STEPPER MOTOR SET SPEED  ******* */

case 190:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('2')=50, motor4    */
if (val>48 && val<51) {
  stm=val-48;                /* calculate motor number */
```

```
  s=191; /* next we will need to get value from serial */
}
else {
  s=-1; /* if value is not a stepper then return to -1 */
}
break; /* s=190 taken care of                          */


case 191:
/* the third received value indicates the speed in rpm */
if (stm==1) stm1.setSpeed(val);
if (stm==2) stm2.setSpeed(val);

s=-1;  /* we are done with set speed so go to -1 next  */
break; /* s=191 taken care of                          */



/* s=200 or 201 means STEPPER MOTOR STEP/RELEASE  **** */

case 200:
/* the second received value indicates the motor number
   from abs('1')=49, motor1, to abs('2')=50, motor4    */
if (val>48 && val<51) {
  stm=val-48;                /* calculate motor number */
  s=201;            /* we still need stuff from serial */
}
else {
  s=-1; /* if value is not a motor then return to -1   */
}
break; /* s=200 taken care of                          */


case 201:
/* the third received value indicates forward, backward,
   release, with characters 'f', 'b', 'r', respectively,
   that have ascii codes 102, 98 and 114               */
switch (val) {

  case 102:
  dir=FORWARD;
  s=202;
  break;

  case 98:
  dir=BACKWARD;
  s=202;
  break;

  case 114: /* release and return to -1 here           */
```

```
  if (stm==1) stm1.release();
  if (stm==2) stm2.release();
  s=-1;
  break;

  default:
  s=-1;  /* unrecognized  character, go to -1          */
  break;
}
break; /* s=201 taken care of                          */


case 202:
/* the fourth received value indicates the style, single,
   double, interleave, microstep, 's', 'd', 'i', 'm'
   that have ascii codes 115,100,105 and 109           */
switch (val) {

  case 115:
  sty=SINGLE;
  s=203;
  break;

  case 100:
  sty=DOUBLE;
  s=203;
  break;

  case 105:
  sty=INTERLEAVE;
  s=203;
  break;

  case 109:
  sty=MICROSTEP;
  s=203;
  break;

  default:
  s=-1;  /* unrecognized  character, go to -1          */
  break;
}
break; /* s=201 taken care of                          */


case 203:
/* the last received value indicates the number of
   steps,                                              */
if (stm==1) stm1.step(val,dir,sty);  /* do the steps   */
if (stm==2) stm2.step(val,dir,sty);
```

```
    s=-1;        /* we are done with step so go to -1 next */
    break;       /* s=203 taken care of                    */



    /* s=340 or 341 means ANALOG REFERENCE *************** */

    case 340:
    /* the second received value indicates the reference,
       which is encoded as is 0,1,2 for DEFAULT, INTERNAL
       and EXTERNAL, respectively                          */

    switch (val) {

      case 48:
      analogReference(DEFAULT);
      break;

      case 49:
      analogReference(INTERNAL);
      break;

      case 50:
      analogReference(EXTERNAL);
      break;

      default:                /* unrecognized, no action  */
      break;
    }

    s=-1;  /* we are done with this so next state is -1    */
    break; /* s=341 taken care of                          */



    /* ******* UNRECOGNIZED STATE, go back to s=-1 ******* */

    default:
    /* we should never get here but if we do it means we
       are in an unexpected state so whatever is the second
       received value we get out of here and back to s=-1  */

    s=-1;  /* go back to the initial state, break unneeded */



  } /* end switch on state s                               */

} /* end if serial available                               */
```

```
} /* end loop statement                                    */
```

The following program is written for matlab software :

MATLAB SUPPORT PACKAGE FOR ARDUINO (Also Known As ARDUINO IO):

This package allows using an Arduino connected to the computer to perform Analog and Digital Input and Output (and command a motor shield) from MATLAB. The Package is given in the CD. Two m files named “install_arduino.m” and “arduino.m” have to be installed in Matlab by opening Matlab as “Run as Administrator”.

Code for Testing Accelerometer:

```
clear all
close all
clc
a = arduino('COM15');

 t = 1;

 x = 0;
 y = 0;
 z = 0;
interv=300;
hold on
 while (t <interv)

 b = a.analogRead (0);
 c = a.analogRead (1);
 d = a.analogRead (2);

 P(t)= b;
 Y(t)= c;
 Z(t)= d;
```

```
 x = [x, b];
 y = [y, c];
 z = [z, d];

 subplot(3,1,1);
 plot (x,'b');
 axis ([0, interv, 100, 1000]);
 grid
 subplot(3,1,2);
 plot (y,'r');
 axis ([0, interv, 100, 1000]);
 grid
 subplot(3,1,3);
 plot (z,'g');
 axis ([0, interv, 100, 1000]);
 grid
 t = t + 1;
 drawnow;

 end

Result = [P' Y' Z'];


delete(a);
```

Code for Testing Flex Sensor:

```
clear all
close all
clc
a = arduino('COM15');

 t = 1;

 x = 0;
 y = 0;
 z = 0;
interv=300;
hold on
 while (t <interv)

 b = a.analogRead (3);
 c = a.analogRead (4);
```

```
 P(t)= b;
 Y(t)= c;


 x = [x, b];
 y = [y, c];


 subplot(2,1,1);
 plot (x,'b');
 axis ([0, interv, 0, 200]);
 grid
 subplot(2,1,2);
 plot (y,'r');
 axis ([0, interv, 0, 500]);
 grid

 t = t + 1;
 drawnow;

 end

Result = [P' Y'];


delete(a);
```

HMM Code For Recognition Pattern:

We proposed a Model for Pattern Recognition. We assumed the inputs as in the coordinate system (‘X’ axis, ‘Y’ axis, ‘Z’ axis). Then we train our model and find out the pattern recognition probability. The codes are …

Training for Square:

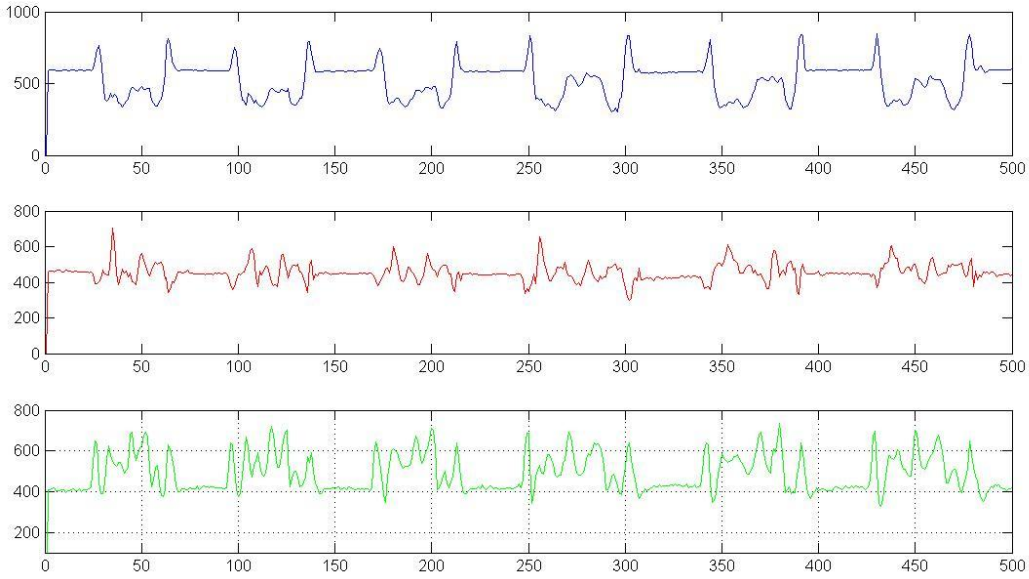
```
clc

clear all

close all
```

```
X=[16 0 44;16 4 44;16 8 44;16 12 44;16 16 44;16 20 44;16 24 44;16 28 44; ...
   16 28 40; 16 28 36; 16 28 32; 16 28 28;16 28 24;16 28 20; ...
   16 24 20;16 20 20;16 16 20;16 12 20;16 8 20;16 4 20;16 0 20; ...
   16 0 24;16 0 28; 16 0 32;16 0 36;16 0 40] % input assumed square
 cluster_no=4;
 [A,C]=kmeans(X,cluster_no)
   plot3(X(A==1,1),X(A==1,2),X(A==1,3),'O','MarkerFaceColor','g','MarkerSize',10)
   box on
   hold on
   grid on
   plot3(X(A==2,1),X(A==2,2),X(A==2,3),'O','MarkerFaceColor','b','MarkerSize',10)
   plot3(X(A==3,1),X(A==3,2),X(A==3,3),'O','MarkerFaceColor','r','MarkerSize',10)
   plot3(X(A==4,1),X(A==4,2),X(A==4,3),'O','MarkerFaceColor','y','MarkerSize',10)
plot3(C(:,1),C(:,2),C(:,3),'kx','MarkerSize',14,'LineWidth',2)
plot3(C(:,1),C(:,2),C(:,3),'kO','MarkerSize',14,'LineWidth',1)
O=A'
S=[1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4];
[T_1,E_1]=hmmestimate(O,S)
[T,E]=hmmtrain(O,T_1,E_1)
[q,ps]=hmmdec(O,T,E)
Pi=[.8 .1 .05 .05 0 0 0 0];
k=size(O);
Time=k(1,2)
N=max(O);
Afw=zeros(Time,N);
```

```
Afw(1,1:N)=Pi(1:N).*E(1:N,O(1))';
if norm(Afw(1,:))<eps
   Afw(1,:)=eps;
end
for l=1:Time-1
   for j=1:N
      s=0;
      for i=1:N
         s=s+Afw(l,i)*T(i,j);
      end
      Afw(l+1,j)=s*E(j,O(l+1));
      if norm(Afw(l+1,j))<eps
         Afw(l+1,j)=eps;
      end
   end
end
Afw
pro=0;
for p=1:N
   pro=pro+Afw(Time,p);
end
Probability=pro
```



Real time graph for Square

Check for Triangle

close all

X=[16 0 20;16 1.67 22.78;16 3.34 25.56;16 5 28.34;16 6.67 31.12;16 8.35 33.9;16 10 36.68;16 11.7 39.46; ...

16 14 43; 16 15.87 39.2; 16 17.5 36.5; 16 19.7 33.5;16 21.2 31;16 23.5 28; ...

16 25.2 25.5;16 26.5 23.78;16 28.2 22;16 30 20;16 26.2 20;16 22.5 20;16 18.8 20; ...

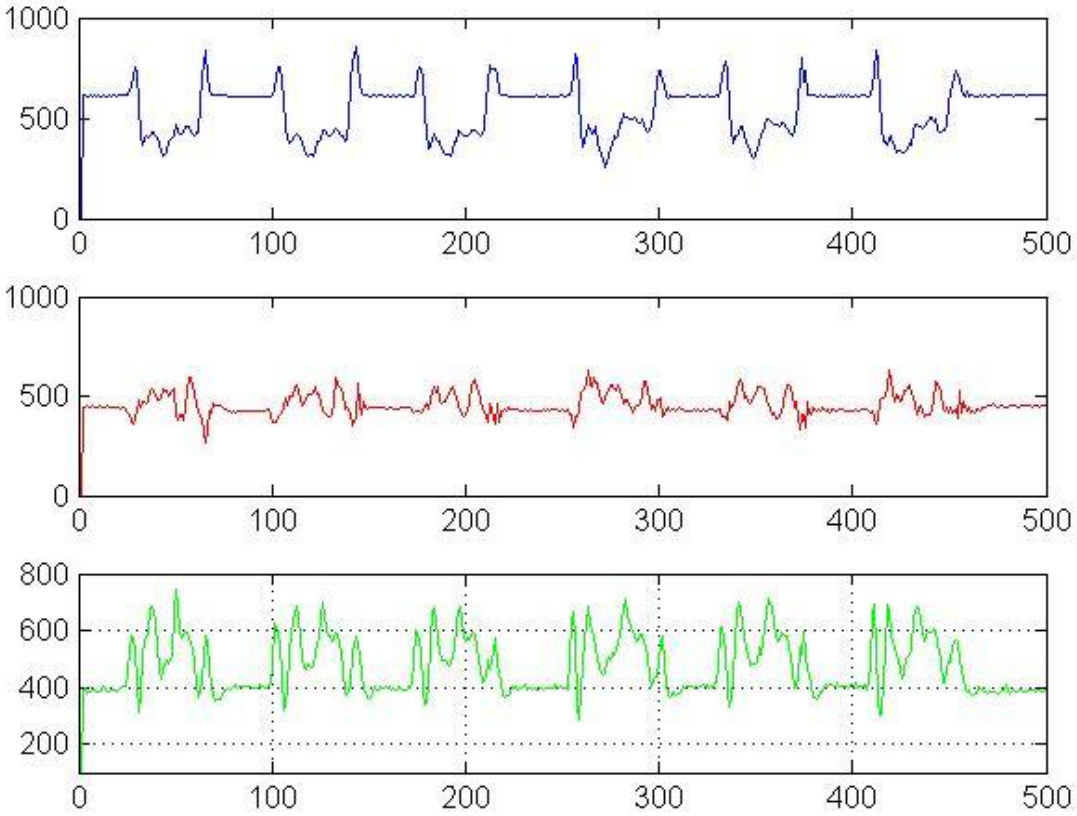
16 14 20;16 10.7 20; 16 7 20;16 3.6 20;16 0 20];

cluster_no=4;

C =[16.0000 24.5714 40.5714;16.0000 24.0000 22.0000; ...

```
  16.0000    4.0000   22.0000;16.0000    3.4286   40.5714];
[A,C]=kmeans(X,cluster_no,'start',C);
   plot3(X(A==1,1),X(A==1,2),X(A==1,3),'O','MarkerFaceColor','g','MarkerSize',10)
   box on
   hold on
   grid on
   plot3(X(A==2,1),X(A==2,2),X(A==2,3),'O','MarkerFaceColor','b','MarkerSize',10)
   plot3(X(A==3,1),X(A==3,2),X(A==3,3),'O','MarkerFaceColor','r','MarkerSize',10)
   plot3(X(A==4,1),X(A==4,2),X(A==4,3),'O','MarkerFaceColor','y','MarkerSize',10)
plot3(C(:,1),C(:,2),C(:,3),'kx','MarkerSize',14,'LineWidth',2)
plot3(C(:,1),C(:,2),C(:,3),'kO','MarkerSize',14,'LineWidth',1)
O=A'
Pi=[.8 .1 .05 .05 0 0 0 0];
k=size(O);
Time=k(1,2)
N=max(O);
Afw=zeros(Time,N);
Afw(1,1:N)=Pi(1:N).*E(1:N,O(1))';
if norm(Afw(1,:))<eps
   Afw(1,:)=eps;
end
for l=1:Time-1
   for j=1:N
      s=0;
      for i=1:N
         s=s+Afw(l,i)*T(i,j);
```

```
        end
        Afw(l+1,j)=s*E(j,O(l+1));
        if norm(Afw(l+1,j))<eps
            Afw(l+1,j)=eps;
        end
    end
end
Afw;
pro=0;
for p=1:N
    pro=pro+Afw(Time,p);
end
Probability=pro
```



Real time graph for triangle

References:

PDF Documents:

1. Introduction to HMM and its application to Classification Problems
2. Jan - Flex Sensor Combined
3. K mean Clustering Basic
4. 53146107-Accelerometer-Based-Mouse Project
5. C Programming for Microcontrollers AVR
6. Basic MATLAB Coding.
7. Arduino 1.0.1 Training manual
8. Interfacing and Data Communication
9. HMM tutorial for speech recognition
10. Clustering Spatial

Websites:

1. www.bubleduck.com
2. www.instructables.com
3. www.scribd.com
4. www.wikipidia.com
5. www.circuitstoday.com
6. www.matlab.com
7. www.arduino.com