A Project Report on

DELTA MODULATION & ADAPTIVE DELTA MODULATION USING ARDUINO UNO

By

NAME	Prithwiraj Roy
ROLL NUMBER	2004195
SECTION	ETC- 3
SEMESTER	Fifth Semester
DATE OF SUBMISSION	26/11/2022

School of Electronics Engineering
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
(Deemed to be University)
BHUBANESWAR

Introduction:

Digital Communication is one of the most important aspects of the present era. Physical World works on analog communication. To be brief- whatever is received by our sensor organs are analog data or signals. So to use various advantages of digital communication we have to convert the analog data into digital format and transmit it. Converting analog signals into digital and to transmit it- is called Digital Modulation.**Delta Modulation (DM)** is one of these Digital Modulation Techniques. In this project report an easy to understand demonstration of DM is implemented using Arduino UNO Micro controller and MPU6050 (this sensor gives real time accelerometer reading which is used as the input signal here). Further a more advanced version of Delta Modulation that is called Adaptive Delta Modulation (ADM) is implemented to remove some of the limitations of DM.

Theory:

To convert an analog signal into digital, samples at a particular time interval (Ts) is taken from the Analog Data. Then after following processes like Quantization and Encoding we get the Digital Form of the input Analog Data. Now this digital data needs to be transmitted. Instead of transmitting the entire data, the difference between two samples can be transmitted. In the receiver side those samples can be added to predict the transmitted signal. This process is called Differential Pulse Code Modulation (DPCM). Delta Modulation (DM) is the simplest form of DPCM where the difference between successive samples is encoded into n-bit data streams (during Encoding Process). In delta modulation, the transmitted data are reduced to a 1-bit data stream. So basically DM is 1-bit DPCM. In Delta Modulation there is a fixed step size which adds to the previous sample when the amplitude of the input signal rises and gets subtracted when the amplitude decreases- thus giving us a prediction of the input signal.

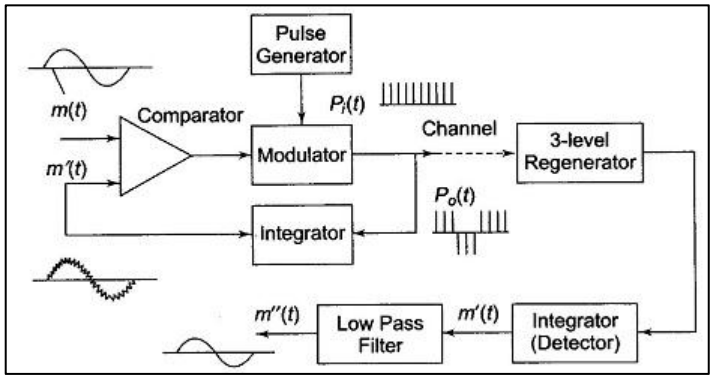

Figure: 1

Block Diagram of Delta Modulation Transmission System

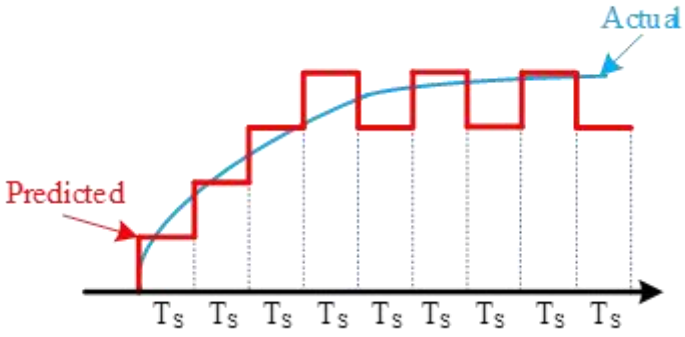

Figure: 2

Visualization of Delta Modulation

In this transmission method there are few problems too. If the input message signal (analog) values changes too fast then the prediction cannot follow the rise as the step size is limited in this modulation technique. So an error occurs named as **Slope Overload.** There is another error present in this modulation system which is referred as **Granular Noise.** The slope overload noise can be reduced by increasing the step size but it increases the granular noise.

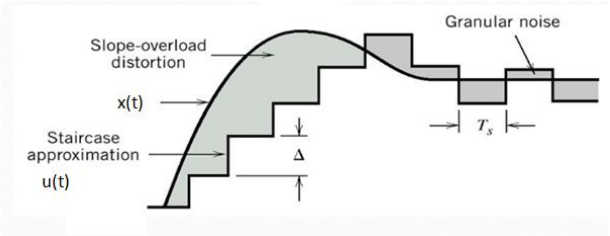

Figure: 3

To reduce these errors we can vary the step size accordingly in the advanced version of Delta Modulation which is called **Adaptive Delta Modulation.**

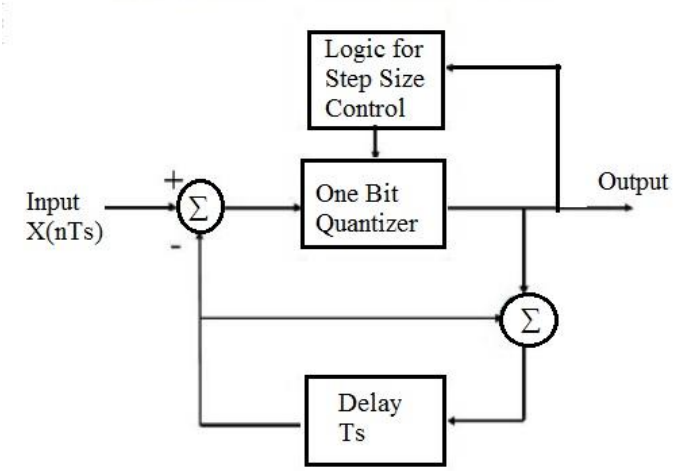

Figure: 4

Block Diagram Of Adaptive Delta Modulation

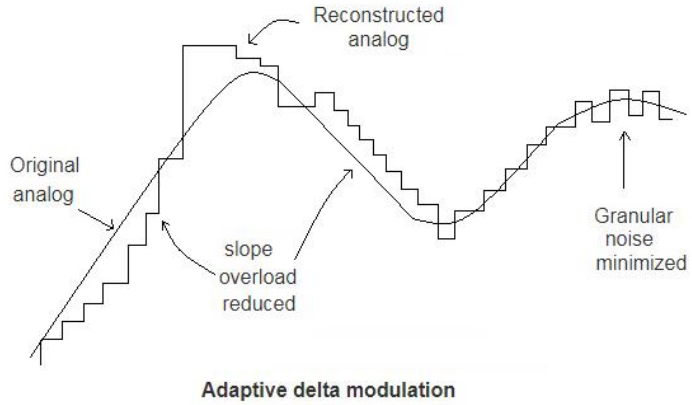

Figure: 5

Connection Diagram:

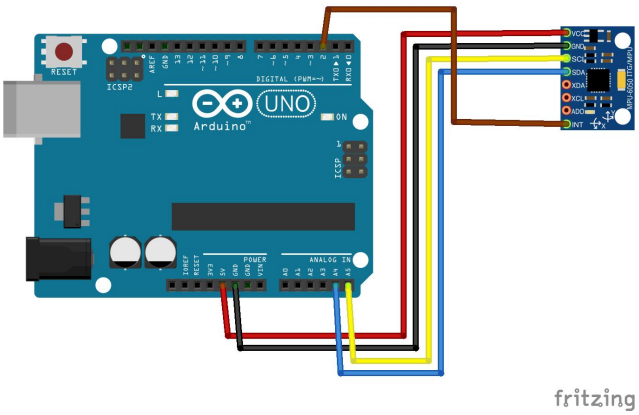

Figure: 6

Connection Diagram for the Setup

Codes:

(Note- Raw Accelerometer values (-17000 to 17000) received from MPU6050 has been mapped to 0 to 8000)

For Delta Modulation:

```
#include <MPU6050.h>
#include <Wire.h>
#include <I2Cdev.h>
int delta=0,count_outside=0, count_inside=0;
int prev_val=0,curr_val=0;

MPU6050 mpu;
int16_t ax,ay,az;
int16_t gx,gy,gz;
int val1=0,val2=0,val3=2;
int preVal1,preVal2,gyro_error=0;
float Acceleration_angle[2];
float Gyro_angle[2];
float Total_angle[2];
float Gyro_raw_error_x, Gyro_raw_error_y;
int16_t Gyr_rawX, Gyr_rawY, Gyr_rawZ;
float Gyro_angle_x, Gyro_angle_y;
float elapsedTime,timePrev;
int i;
float rad_to_deg = 180/3.141592654;

void setup() {
   Wire.begin();
   Serial.begin(9600);
   mpu.initialize();
   Serial.println(mpu.testConnection() ? "Connected" : "Not Connected");

   if(gyro_error==0)
   {
      for(int i=0; i<200; i++)
      {
         mpu.getMotion6(&ax , &ay, &az, &gx, &gy, &gz);

         /*---X---*/
         Gyro_raw_error_x = Gyro_raw_error_x + (gx/131.0);
         /*---Y---*/
```

```
        Gyro_raw_error_y = Gyro_raw_error_y + (gy/131.0);
        if(i==199)
        {
          Gyro_raw_error_x = Gyro_raw_error_x/200;
          Gyro_raw_error_y = Gyro_raw_error_y/200;
          gyro_error=1;
        }
      }
    }
}

void loop() {
   mpu.getMotion6(&ax , &ay, &az, &gx, &gy, &gz);
   curr_val=map(ax,-17000,17000,0,8000);
   Serial.println("Current_Value: ");
   Serial.println(curr_val);

   if(delta<curr_val&&(count_outside-count_inside)>5){
      delta=delta+100;
      count_inside=count_outside;
      prev_val=curr_val;
      }

   if(delta>curr_val&&(count_outside-count_inside)>5){
      delta=delta-100;
      count_inside=count_outside;
      prev_val=curr_val;
      }
      Serial.print("Delta: ");
      Serial.println(delta);

   count_outside++;
}
```

For Adaptive Delta Modulation:

```
#include <MPU6050.h>
#include <Wire.h>
#include <I2Cdev.h>
int delta=0,count_outside=0, count_inside=0;
int prev_val=0,curr_val=0;
int step_size=100;


MPU6050 mpu;
int16_t ax,ay,az;
int16_t gx,gy,gz;
int val1=0,val2=0,val3=2;
int preVal1,preVal2,gyro_error=0;
float Acceleration_angle[2];
float Gyro_angle[2];
float Total_angle[2];
float Gyro_raw_error_x, Gyro_raw_error_y;
int16_t Gyr_rawX, Gyr_rawY, Gyr_rawZ;
float Gyro_angle_x, Gyro_angle_y;
float elapsedTime,timePrev;
int i;
float rad_to_deg = 180/3.141592654;

void setup() {
   Wire.begin();
   Serial.begin(9600);
   mpu.initialize();
   Serial.println(mpu.testConnection() ? "Connected" : "Not Connected");

   if(gyro_error==0)
   {
```

```
      for(int i=0; i<200; i++)
      {
        mpu.getMotion6(&ax , &ay, &az, &gx, &gy, &gz);

        /*---X---*/
        Gyro_raw_error_x = Gyro_raw_error_x + (gx/131.0);
        /*---Y---*/
        Gyro_raw_error_y = Gyro_raw_error_y + (gy/131.0);
        if(i==199)
        {
          Gyro_raw_error_x = Gyro_raw_error_x/200;
          Gyro_raw_error_y = Gyro_raw_error_y/200;
          gyro_error=1;
        }
      }
    }
}

void loop() {
   mpu.getMotion6(&ax , &ay, &az, &gx, &gy, &gz);
   curr_val=map(ax,-17000,17000,0,8000);
   Serial.println("Current_Value: ");
   Serial.println(curr_val);
      if((curr_val-delta)>step_size){
      step_size=step_size+100;
      if(step_size==0){
        step_size=50;
      }
      }
      if((curr_val-delta)<step_size){
      step_size=step_size-100;
      if(step_size==0){
        step_size=-50;
      }
      }

   if(delta!=curr_val&&(count_outside-count_inside)>5){
      count_inside=0;
      count_outside=0;

      delta=delta+step_size;
      }
      Serial.print("Delta: ");
      Serial.println(delta);

   count_outside++;
}
```

Observations:

Arduino's official Code Editor has been used for writing code. Serial Plotter of the IDE is used to get live plotting of graph (Input signal and predicted signal (delta modulated)). Below are the observed results-

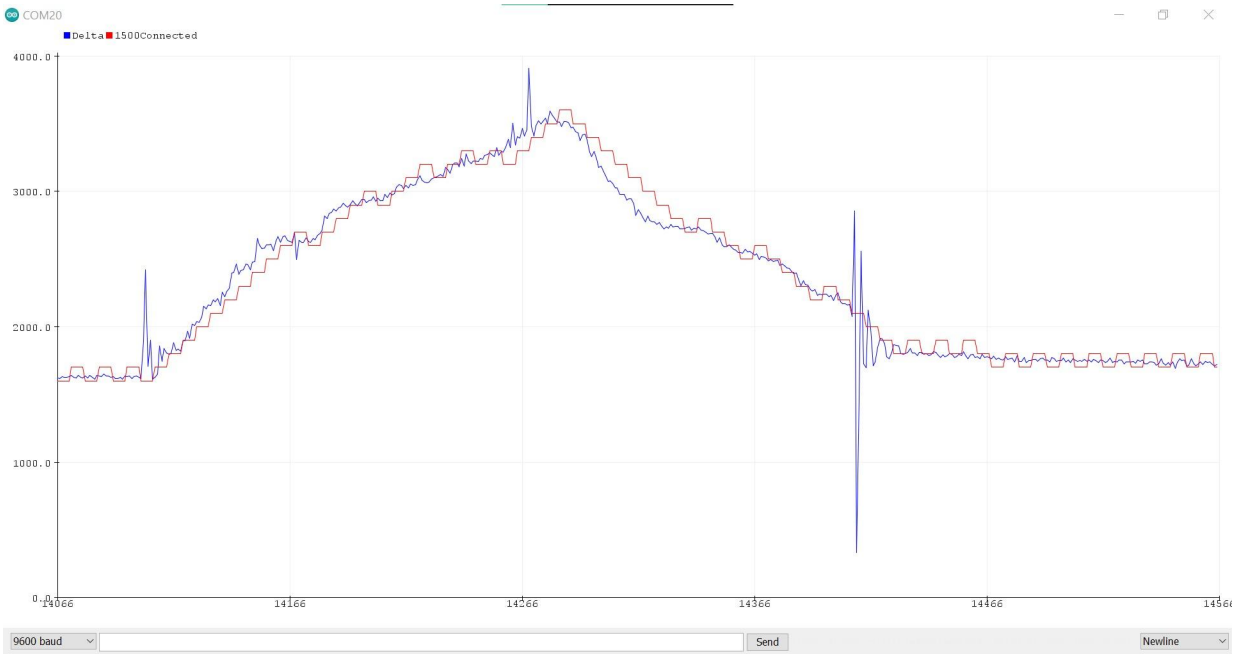

Figure: 7

Delta Modulation

Blue Coloured plot is of the input signal
Red Coloured plot is of the predicted signal

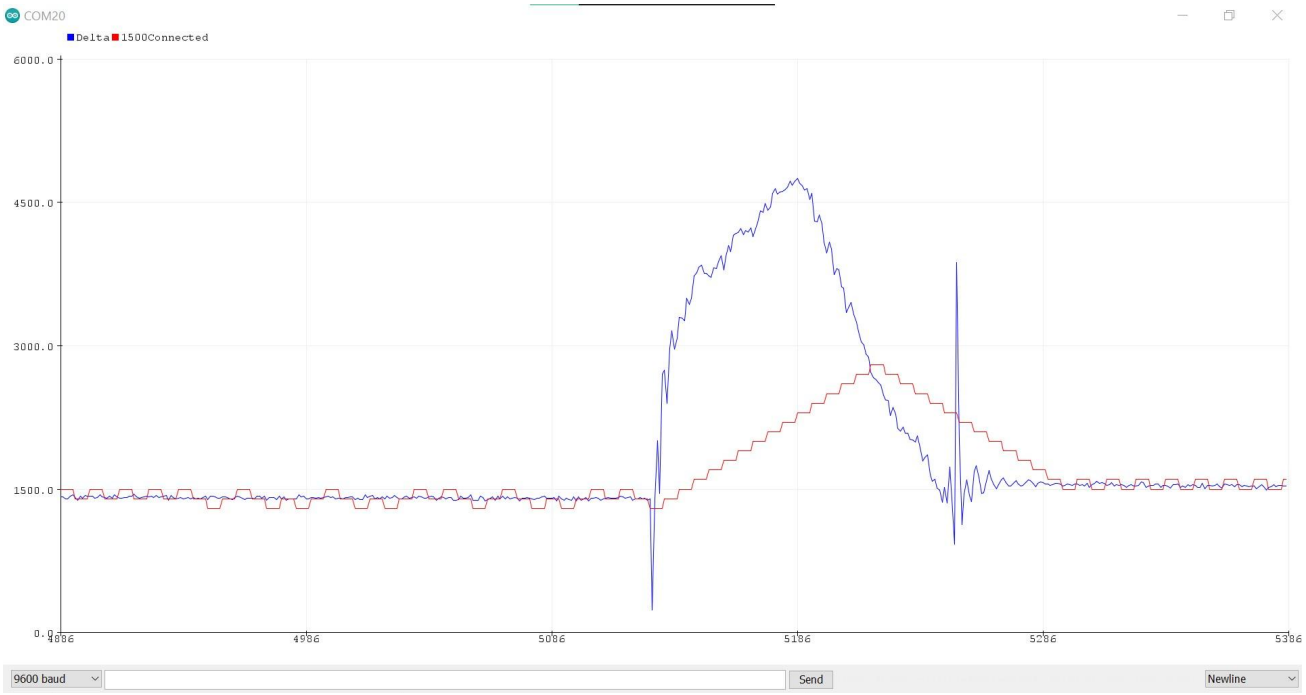

Figure: 8

Slope Overload and Granular noise in Delta Modulation
(when input signal amplitide changes too fast**)**

Blue Coloured plot is of the input signal
Red Coloured plot is of the predicted signal

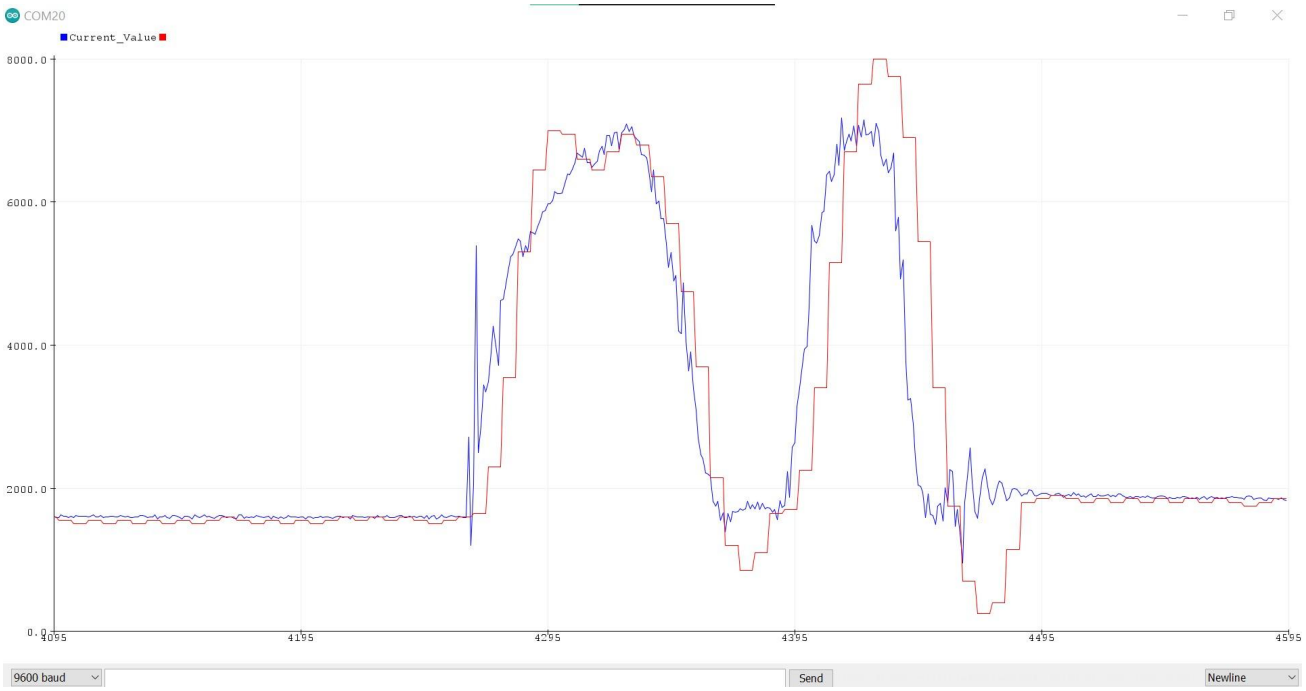

Figure: 9

Slope Overload and Granular Noise is reduced in
ADAPTIVE DELTA MODULATION
(Step size changes according to the input message amplitude change**)**

Blue Coloured plot is of the input signal
Red Coloured plot is of the predicted signal

Conclusion:

Delta Modulation (DM) and Adaptive Delta Modulation (ADM) have been successfully implemented through this project. It's also observed that in case of ADM the two mentioned noise (Slope Overload and Granular Noise) is significantly less as compared to DM.

Signature of the Student

Prithwiraj Roy

Prithwiraj Roy