Final Code for Mirror-Tracking System

```
//Solar Still with Concentrated Sunlight Demo on Arduino

//Code written by Javed Akhtar akhtarj@nmsu.edu

//this update: 08/02/2021

//Arduino Mega with clock DS3231

//For eleven mirrors in actual but can be modified for lesser mirrors


#include <Servo.h>

#include <math.h>

#include <DS3231.h>

#include <Wire.h>

DS3231 clock;

/*

  Link for solar information:(1) https://www.esrl.noaa.gov/gmd/grad/solcalc/ ****use this one

                             (2) https://www.esrl.noaa.gov/gmd/grad/solcalc/azel.html

                             (3) https://www.timeanddate.com/sun/usa/las-cruces

  Las Cruces Coorinate: 32.287989° N(+), -106.75385° W (-)

  Azimuth and Elevation - an angular coordinate system for locating positions in the sky. Azimuth is measured clockwise from true north

  to the point on the horizon directly below the object. Elevation is measured vertically from that point on the horizon up to the object.

*/


#define latitude  32.287989 //latitude position (in degree) of the experimental station (Jett Hall Annex)

#define longitude -106.75385 //longitude position (in degree) of the experimental station (Jett Hall Annex)

#define lat  ((3.1415926 / 180) * latitude) //latitude (in radian) for trigonometric functions

#define timezone -6; //timezone in hour from UTC (Mountain Daylight Time)

const unsigned int MirrorStart = 0, MirrorEnd = 11;

//For Clock

bool century = false;
```

```
bool h12Flag;
bool pmFlag;

/*October, 09 2021 - > day_of_year= 282*/

unsigned char DaysofMonth[12] = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334}; //Non leap year
//unsigned char DaysofMonth[12] = {0, 31, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335}; //leap year
unsigned char PTh = 0; //present time in hours*
unsigned char PTm = 0; //present time in minutes*

//unsigned char PTmon = clock.getMonth(century);
//unsigned char PTd = clock.getDate();
unsigned int day_of_year = 0; //day of the year of the experiment*

double h = 35.5; //height(in inch) of convex mirror focal point from the origin
//(centre point of the base on which these mirror units are intalled)
const unsigned char NumOfMirrorUnits = 11; //number of total mirror units

const unsigned char mirror_alt_offset[NumOfMirrorUnits] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //mirror
altitude angle offset
const unsigned char mirror_azi_offset[NumOfMirrorUnits] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //mirror
azimuth angle offset

//pulse range of mirror motors for mirror 1-11
const unsigned int mirror_alt_pulse_0[NumOfMirrorUnits] = {603, 578, 525, 585, 540, 505, 610, 550,
650, 610, 550};
const unsigned int mirror_alt_pulse_180[NumOfMirrorUnits] = {2455, 2380, 2330, 2390, 2410, 2270,
2420, 2345, 2480, 2495, 2355};
const unsigned int mirror_azi_pulse_0[NumOfMirrorUnits] = {490, 490, 605, 550, 490,  500, 470, 470,
520, 710, 675};
const unsigned int mirror_azi_pulse_180[NumOfMirrorUnits] = {2355, 2280, 2470, 2325, 2240, 2385,
2300, 2335, 2355, 2585, 2245};
```

```
double Hour = 0; //initiation of variable "Hour" (hour corresponds to current time)
double gamma = 0; //initiation of variable "gamma" (fractional year (in radian))
double eqtime = 0; //initiation of variable "eqtime" (equation of time (in minute))
double decl = 0; //intiation of variable "decl" (solar declination angle (in radians))
double time_offset = 0; //initiation of variable "time_offset" (time offset (in minutes))
double tst = 0; //initiation of variable "tst" (true solar time (in minutes))
double ha = 0; //initiation of variable "ha" (solar hour angle (in radian))
double phi = 0; //initiation of variable "phi" (solar zenith angle angle (in radian))
double alpha = 0; //initiation of variable "alpha" (solar altitude angle angle (in radian))
double beta = 0; //initiation of variable "beta" (solar azimuth angle angle (in radian))
double snoon = 0; //initiation of variable "snoon" (solar noon (in minute))
double t_CT = 0; //initiation of variable "t_CT" (current time (in minutes))

double sun_vec_x = 0; //initiation of  (x-component of sun_vector)
double sun_vec_y = 0; //initiation of  (y-component of sun_vector)
double sun_vec_z = 0; //initiation of  (z-component of sun_vector)

double pi = 3.1415926; //value of 'pi'
double r2d = 180 / pi; //convesion ratio of radian to degree conversion
double d2r = pi / 180; //convesion ratio of degree to radian conversion

unsigned long CurrTime;
unsigned long LastTime;
//----------------------------------------------------------------------

//---------------UNIT#3-----------------
//------Mirror_Unit_Position--------
```

```
double mirror_pos_x[NumOfMirrorUnits] = { -13.5,    13.5, 0, -0.125, 13.5, -13.5, 19.0625, 19.0625,
28.625, 28.625, 28.625}; //x position (in inch) of center pt. of flat mirrors

double mirror_pos_y[NumOfMirrorUnits] = { -28.6875, -28.6875, -19.125, 19.125, 28.6875, 28.6875,
10.75, -6.75, -17.5, 0, 18.25}; //y position (in inch) of center pt. of flat mirrors

double mirror_pos_z[NumOfMirrorUnits] = { 9.5625,   9.5625, 9.0625, 9.0625, 9.4375, 9.4375, 9.25,
9.25, 9.375, 9.3125, 9.3125}; //z position (in inch) of center pt. of flat mirrors


// x & y position of target point is "0"

double target_pos_z[NumOfMirrorUnits] = {h, h, h, h, h, h, h, h, h, h, h}; //z position of target_point

//target_vector3: unit vector along reflected ray from flat mirror of unit#3

//-------------------------


double mirror_alt[NumOfMirrorUnits] = {0}; //initiation of  (mirror alt angle(in radian) calculated based
on position and sun vector)

double mirror_azi[NumOfMirrorUnits] = {0}; //initiation of  (mirror azi angle(in radian))


//Altitude Yellow wire

//Azimuth Greeen wire

const unsigned char AltServoPins[NumOfMirrorUnits] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}; //initiallzing
servo pins for altitude motors

const unsigned char AziServoPins[NumOfMirrorUnits] = {31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41};
//initiallzing servo pins for azimuth motors


Servo AltServo[NumOfMirrorUnits];

Servo AziServo[NumOfMirrorUnits];

//-------------------------------


void setup()

{

  Wire.begin();

  //keep this section commented unless you need to calibrate the time on clock

  PTh = clock.getHour(h12Flag, pmFlag); //present time in hours*
```

```
  PTm = clock.getMinute(); //present time in minutes*

  day_of_year = DaysofMonth[clock.getMonth(century) - 1] + clock.getDate(); //day of the year of the
experiment*

  for (int i = 0; i < NumOfMirrorUnits; i++)
  {
    AltServo[i].attach(AltServoPins[i], mirror_alt_pulse_0[i], mirror_alt_pulse_180[i]);
    AziServo[i].attach(AziServoPins[i], mirror_azi_pulse_0[i], mirror_azi_pulse_180[i]);
  }
  for (int i = 0; i < NumOfMirrorUnits; i++)
  {
    AltServo[i].write(180);
    AziServo[i].write(180);
  }
  Serial.begin(9600);
}

void loop()
{
  //---------------------------Solar_Tracking--------------------------------
  CurrTime = millis();

  Hour = (((PTh * 60 + PTm) * 60000) + CurrTime) / 3600000; //hour corresponds to current time

  gamma = (2 * pi / 365) * (day_of_year - 1 + ((Hour - 12) / 24)); //fractional year (in radian); [for leap
year, use 366 instead of 365 in the denominator]
  eqtime = 229.18 * (0.000075 + 0.001868 * cos(gamma) - 0.032077 * sin(gamma) - 0.014615 * cos(2 *
gamma) - 0.040849 * sin(2 * gamma)); //equation of time (in minutes)
```

```
  decl = 0.006918 - 0.399912 * cos(gamma) + 0.070257 * sin(gamma) - 0.006758 * cos(2 * gamma) + 0.000907 * sin(2 * gamma) - 0.002697 * cos(3 * gamma) + 0.00148 * sin(3 * gamma);

  //solar declination angle (in radians)

  time_offset = eqtime + 4 * longitude - 60 * timezone; //time offset (in minutes)// because of different time zones


  tst = (PTh * 60 + PTm + time_offset) + (CurrTime / 60000); //true solar time (in minutes) //incorporate time zone offset in solar time


  ha = (d2r) * ((tst / 4) - 180); //solar hour angle(in radian)


  snoon = 360 - 4 * longitude - eqtime; //solar noon for a given location is found from the longitude (in minutes)

  t_CT = (PTh * 60 + PTm) + (CurrTime / 60000); //current time (in minutes)


  phi = acos(sin(lat) * sin(decl) + cos(lat) * cos(decl) * cos(ha)); //solar zenith angle angle (in radian)

  alpha = (pi / 2 - phi); //solar altitude angle (in radian)

  beta = acos((sin(lat) * cos(phi) - sin(decl)) / (cos(lat) * sin(phi))); //solar azimuth angle (in radian)


  if (t_CT <= snoon)

  {

    beta = pi - (beta); //solar azimuth angle (in radian) before solar noon

  }

  if (t_CT > snoon)

  {

    beta = pi + (beta); //solar azimuth angle (in radian) after solar noon

  }


  //------Sun_Vector------

  sun_vec_x = cos(alpha) * cos(beta); //x-component of sun_vector)

  sun_vec_y = -cos(alpha) * sin(beta); //y-component of sun_vector)

  sun_vec_z = sin(alpha); //z-component of sun_vector)
```

```
//----------------------
//---------------------------------------------------------------------------

//---------------------------Mirror_Tracking----------------------------------
int i = MirrorStart;
if (CurrTime - LastTime > 5000)                          //routine to be called after every five seconds
{
  for (; i < MirrorEnd; i++)
  {
    Mirror_angles(sun_vec_x, sun_vec_y, sun_vec_z, i);          //Calculating mirror angles

    //Setting servo motors on calculated angles

    AltServo[i].write(round(mirror_alt[i]));// + mirror_alt_offset[i]);
    AziServo[i].write(round(mirror_azi[i]));// + mirror_azi_offset[i]);
    //      Serial.println("Mirror 3 Azi motor");
    //      Serial.println(mirror_azi[2] + mirror_azi_offset[2]);
    //      Serial.println(AltServo[4].read());
    LastTime = CurrTime;
  }

  //Serial Display for Debugging

  Serial.print(">------------->\nCurrent Time (in minutes) = ");
  Serial.print(t_CT); Serial.print("\t Day of Year = "); Serial.print(day_of_year); Serial.print("\t Solar Noon
= "); Serial.println(snoon);
  Serial.print("Sun Altitude = "); Serial.println(alpha * r2d); Serial.print("Sun Azimuth = ");
Serial.println(beta * r2d);
  Serial.print("Declination = "); Serial.println(decl * r2d);
```

```
  Serial.print("Unit\t"); Serial.print("Altitude\t"); Serial.print("Alt Motor\t"); Serial.print("Azimuth\t\t");
Serial.print("Azi Motor\t\n");

  for (int i = 0; i < NumOfMirrorUnits; i++)

  {

    Serial.print(i + 1); Serial.print("\t"); Serial.print(mirror_alt[i]); Serial.print("\t\t");
Serial.print(AltServo[i].read()); Serial.print("\t\t");

    Serial.print(mirror_azi[i]); Serial.print("\t\t"); Serial.println(AziServo[i].read());

  }

  //  delay(5000);

 }

}

//-----------------------------Mirror_Angles Function---------------------------------


void Mirror_angles(double _sx, double _sy, double _sz, int _mindex)

{

 double _fx = -mirror_pos_x[_mindex] / (sqrt(mirror_pos_x[_mindex] * mirror_pos_x[_mindex] +
mirror_pos_y[_mindex] * mirror_pos_y[_mindex]

                              + (target_pos_z[_mindex] - mirror_pos_z[_mindex]) * (target_pos_z[_mindex] -
mirror_pos_z[_mindex])));

 //x-componenet of target_vector number (_mindex + 1)

 double _fy = -mirror_pos_y[_mindex] / (sqrt(mirror_pos_x[_mindex] * mirror_pos_x[_mindex] +
mirror_pos_y[_mindex] * mirror_pos_y[_mindex]

                              + (target_pos_z[_mindex] - mirror_pos_z[_mindex]) * (target_pos_z[_mindex] -
mirror_pos_z[_mindex])));

 //y-componenet of target_vector number (_mindex + 1)

 double _fz = (target_pos_z[_mindex] - mirror_pos_z[_mindex]) / (sqrt(mirror_pos_x[_mindex] *
mirror_pos_x[_mindex] + mirror_pos_y[_mindex] * mirror_pos_y[_mindex]

           + (target_pos_z[_mindex] - mirror_pos_z[_mindex]) * (target_pos_z[_mindex] -
mirror_pos_z[_mindex])));

 //z-componenet of target_vector number (_mindex + 1)


 double _mx = (((abs(_sz + _fz)) / (_sz + _fz)) * (_sx + _fx)) * (1 / (sqrt((_sx + _fx) * (_sx + _fx) + (_sy + _fy)
* (_sy + _fy) + (_sz + _fz) * (_sz + _fz))));
```

```
//x-component of bisector vector

double _my = (((abs(_sz + _fz)) / (_sz + _fz)) * (_sy + _fy)) * (1 / (sqrt((_sx + _fx) * (_sx + _fx) + (_sy + _fy) * (_sy + _fy) + (_sz + _fz) * (_sz + _fz))));

//y-component of bisector vector

double _mz = (abs(_sz + _fz)) * (1 / (sqrt((_sx + _fx) * (_sx + _fx) + (_sy + _fy) * (_sy + _fy) + (_sz + _fz) * (_sz + _fz)))); //z-component of bisector vector

//-------------calculating mirror angles--------------


if (_mx > 0 && _my >= 0)

{

  mirror_alt[_mindex] = pi / 2 + acos(_mz);

  mirror_azi[_mindex] = pi / 2 - atan(abs(_my) / abs(_mx));

}

else if (_mx <= 0 && _my > 0)

{

  mirror_alt[_mindex] = pi / 2 - acos(_mz);

  mirror_azi[_mindex] = pi / 2 + atan(abs(_my) / abs(_mx));

}

else if (_mx < 0 && _my <= 0)

{

  mirror_alt[_mindex] = pi / 2 - acos(_mz);

  mirror_azi[_mindex] = pi / 2 - atan(abs(_my) / abs(_mx));

}

else if (_mx >= 0 && _my < 0)

{

  mirror_alt[_mindex] = pi / 2 + acos(_mz);

  mirror_azi[_mindex] = pi / 2 + atan(abs(_my) / abs(_mx));

}

else {

  mirror_alt[_mindex] = mirror_alt[_mindex];

  mirror_azi[_mindex] = mirror_azi[_mindex];
```

```
  }


  mirror_alt[_mindex] = (r2d) * mirror_alt[_mindex]; //elevation angle (in degree) of flat mirror of
unit#_mindex

  mirror_azi[_mindex] = (r2d) * mirror_azi[_mindex]; //azimuth angle (in degree) of flat mirror of
unit#_mindex

}

//------------------------------------------------------------------------------------------
```