```
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <AccelStepper.h>

#define DIR_AZ 6 /*PIN for Azimuth Direction*/
#define STEP_AZ 5 /*PIN for Azimuth Steps*/
#define DIR_EL 10 /*PIN for Elevation Direction*/
#define STEP_EL 9 /*PIN for Elevation Steps*/

#define EN 8 /*PIN for Enable or Disable Stepper Motors*/

#define SPR 200 /*Step Per Revolution*/
#define RATIO 54 /*Gear ratio*/
#define T_DELAY 60000 /*Time to disable the motors in millisecond*/

#define HOME_AZ 4 /*Homing switch for Azimuth*/
#define HOME_EL 3 /*Homing switch for Elevation*/

#define MAX_AZ_ANGLE 365 /*Maximum Angle of Azimuth for homing scanning*/
#define MAX_EL_ANGLE 365 /*Maximum Angle of Elevation for homing scanning*/

#define MAX_SPEED 300
#define MAX_ACCELERATION 100

#define MIN_PULSE_WIDTH 20 /*in microsecond*/

#define DEFAULT_HOME_STATE HIGH /*Change to LOW according to Home sensor*/

#define HOME_DELAY 6000 /*Time for homing Decceleration in millisecond*/

#define BufferSize 256
#define BaudRate 19200

/*Global Variables*/
unsigned long t_DIS = 0; /*time to disable the Motors*/
/*Define a stepper and the pins it will use*/
AccelStepper AZstepper(1, STEP_AZ, DIR_AZ);
AccelStepper ELstepper(1, STEP_EL, DIR_EL);

void setup()
{
  /*Change these to suit your stepper if you want*/
  AZstepper.setMaxSpeed(MAX_SPEED);
  AZstepper.setAcceleration(MAX_ACCELERATION);
  /*Change these to suit your stepper if you want*/
  ELstepper.setMaxSpeed(MAX_SPEED);
  ELstepper.setAcceleration(MAX_ACCELERATION);
  /*Set minimum pulse width*/
  AZstepper.setMinPulseWidth(MIN_PULSE_WIDTH);
  ELstepper.setMinPulseWidth(MIN_PULSE_WIDTH);
  /*Enable Motors*/
  pinMode(EN, OUTPUT);
  digitalWrite(EN, LOW);
  /*Homing switch*/
  pinMode(HOME_AZ, INPUT_PULLUP);
  pinMode(HOME_EL, INPUT_PULLUP);
  /*Serial Communication*/
  Serial.begin(BaudRate);
  /*Initial Homing*/
  Homing(deg2step(-MAX_AZ_ANGLE), deg2step(-MAX_EL_ANGLE));
}
```

```
void loop()
{
  /*Define the steps*/
  static int AZstep = 0;
  static int ELstep = 0;
  /*Time Check*/
  if (t_DIS == 0)
    t_DIS = millis();

  /*Disable Motors*/
  if (AZstep == AZstepper.currentPosition() && ELstep == ELstepper.currentPosition()
&& millis()-t_DIS > T_DELAY)
    digitalWrite(EN, HIGH);
  /*Enable Motors*/
  else
    digitalWrite(EN, LOW);

  /*Read the steps from serial*/
  cmd_proc(AZstep, ELstep);
  /*Move the Azimuth & Elevation Motor*/
  stepper_move(AZstep, ELstep);
}

/*Homing Function*/
void Homing(int AZsteps, int ELsteps)
{
  int value_Home_AZ = DEFAULT_HOME_STATE;
  int value_Home_EL = DEFAULT_HOME_STATE;
  boolean isHome_AZ = false;
  boolean isHome_EL = false;

  AZstepper.moveTo(AZsteps);
  ELstepper.moveTo(ELsteps);

  while(isHome_AZ == false || isHome_EL == false)
  {
    value_Home_AZ = digitalRead(HOME_AZ);
    value_Home_EL = digitalRead(HOME_EL);
    /*Change to LOW according to Home sensor*/
    if (value_Home_AZ == DEFAULT_HOME_STATE)
    {
      AZstepper.moveTo(AZstepper.currentPosition());
      isHome_AZ = true;
    }
    /*Change to LOW according to Home sensor*/
    if (value_Home_EL == DEFAULT_HOME_STATE)
    {
      ELstepper.moveTo(ELstepper.currentPosition());
      isHome_EL = true;
    }
    if (AZstepper.distanceToGo() == 0 && !isHome_AZ)
    {
      error(0);
      break;
    }
    if (ELstepper.distanceToGo() == 0 && !isHome_EL)
    {
      error(1);
      break;
    }
    AZstepper.run();
    ELstepper.run();
  }
  /*Delay to Deccelerate*/
  long time = millis();
  while(millis() - time < HOME_DELAY)
  {
    AZstepper.run();
    ELstepper.run();
  }
  /*Reset the steps*/
  AZstepper.setCurrentPosition(0);
  ELstepper.setCurrentPosition(0);
}

/*EasyComm 2 Protocol & Calculate the steps*/
void cmd_proc(int &stepAz, int &stepEl)
{
  /*Serial*/
  char buffer[BufferSize];
  char incomingByte;
  char *Data = buffer;
  char *rawData;
  static int BufferCnt = 0;
  char data[100];

  double angleAz, angleEl;

  /*Read from serial*/
  while (Serial.available() > 0)
  {
    incomingByte = Serial.read();
    /* XXX: Get position using custom and test code */
    if (incomingByte == '!')
    {
      /*Get position*/
      Serial.print("TM");
      Serial.print(1);
      Serial.print(" ");
      Serial.print("AZ");
      Serial.print(10*step2deg(AZstepper.currentPosition()), 1);
      Serial.print(" ");
      Serial.print("EL");
      Serial.println(10*step2deg(ELstepper.currentPosition()), 1);
    }
    /*new data*/
    else if (incomingByte == '\n')
    {
      buffer[BufferCnt] = 0;
      if (buffer[0] == 'A' && buffer[1] == 'Z')
      {
        if (buffer[2] == ' ' && buffer[3] == 'E' && buffer[4] == 'L')
        {
          /*Get position*/
          Serial.print("AZ");
          Serial.print(step2deg(AZstepper.currentPosition()), 1);
          Serial.print(" ");
          Serial.print("EL");
          Serial.print(step2deg(ELstepper.currentPosition()), 1);
          Serial.println(" ");
        }
        else
        {
          /*Get the absolute value of angle*/
          rawData = strtok_r(Data, " " , &Data);
          strncpy(data, rawData+2, 10);
          if (isNumber(data))
          {
            angleAz = atof(data);
            /*Calculate the steps*/
            stepAz = deg2step(angleAz);
          }
          /*Get the absolute value of angle*/
          rawData = strtok_r(Data, " " , &Data);
          if (rawData[0] == 'E' && rawData[1] == 'L')
          {
            strncpy(data, rawData+2, 10);
            if (isNumber(data))
            {
              angleEl = atof(data);
              /*Calculate the steps*/
              stepEl = deg2step(angleEl);
            }
          }
        }
      }
      /*Stop Moving*/
      else if (buffer[0] == 'S' && buffer[1] == 'A' && buffer[2] == ' ' && buffer[3]
== 'S' && buffer[4] == 'E')
      {
        /*Get position*/
        Serial.print("AZ");
        Serial.print(step2deg(AZstepper.currentPosition()), 1);
        Serial.print(" ");
        Serial.print("EL");
        Serial.println(step2deg(ELstepper.currentPosition()), 1);
        stepAz = AZstepper.currentPosition();
        stepEl = ELstepper.currentPosition();
      }
      /*Reset the rotator*/
      else if (buffer[0] == 'R' && buffer[1] == 'E' && buffer[2] == 'S' && buffer[3]
== 'E' && buffer[4] == 'T')
      {
        /*Get position*/
        Serial.print("AZ");
        Serial.print(step2deg(AZstepper.currentPosition()), 1);
        Serial.print(" ");
        Serial.print("EL");
        Serial.println(step2deg(ELstepper.currentPosition()), 1);
        /*Move the steppers to initial position*/
        Homing(deg2step(-MAX_AZ_ANGLE), deg2step(-MAX_EL_ANGLE));
        /*Zero the steps*/
        stepAz = 0;
        stepEl = 0;
      }
      BufferCnt = 0;
      /*Reset the disable motor time*/
      t_DIS = 0;
    }
    /*Fill the buffer with incoming data*/
    else {
      buffer[BufferCnt] = incomingByte;
      BufferCnt++;
    }
  }
```

```
}

/*Error Handling*/
void error(int num_error)
{
  switch (num_error)
  {
    /*Azimuth error*/
    case (0):
      while(1)
      {
        Serial.println("AL001");
        delay(100);
      }
    /*Elevation error*/
    case (1):
      while(1)
      {
        Serial.println("AL002");
        delay(100);
      }
    default:
      while(1)
      {
        Serial.println("AL000");
        delay(100);
      }
  }
}

/*Send pulses to stepper motor drivers*/
void stepper_move(int stepAz, int stepEl)
{
  AZstepper.moveTo(stepAz);
  ELstepper.moveTo(stepEl);

  AZstepper.run();
  ELstepper.run();
}

/*Convert degrees to steps*/
int deg2step(double deg)
{
  return(RATIO*SPR*deg/360);
}

/*Convert steps to degrees*/
double step2deg(int Step)
{
  return(360.00*Step/(SPR*RATIO));
}

/*Check if is argument in number*/
boolean isNumber(char *input)
{
  for (int i = 0; input[i] != '\0'; i++)
  {
    if (isalpha(input[i]))
      return false;
  }
   return true;
}
```