```
////////////////////////////////////////////////////////////////////////////////
///////
//    Code for Death Ray Capstone
//    Date: 2/23/2023
//    Authors: Elizabeth Wade, Simon Criswell, and Kyle O'Reilly
//
//    Purpose: This code serves as the controls for a solar tracking dish
//    It takes input from:
//          - three sensors: accelerometer, GPS, encoder
//          - one button: loading button
//          - one limit switch:
//    Uses GPS data to determine sun position and informs tilt and pan
//    motor on where to travel to be oriented toward the sun
////////////////////////////////////////////////////////////////////////////////
///////

//////////////////// LIBRARY AND GLOBAL VARIABLE SETUP
//////////////////////////////

// Accelerometer libraries
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(54321);
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);
float accelAngle = 0;  // absolute tilt angle of dish

// GPS Libraries
#include <Adafruit_GPS.h>
#include <SoftwareSerial.h>
SoftwareSerial mySerial(2, 9);
Adafruit_GPS GPS(&mySerial);
#define GPSECHO  true

// Encoder libraries
#include <SPI.h>
#define timoutLimit 100

// Encoder - SPI commands used by the AMT20
#define nop 0x00         //no operation
#define rd_pos 0x10      //read position
#define set_zero_point 0x70 //set zero point
const int CS = 10;
float encDegrees = 0;  // absolute pan angle of dish
float homePanOffset = 0; // Offset set by homing function
float dishDegrees = 0;

// Tilt Motor controller pins
const int ENA1 = 6; // ENA1 on motor controller. PWM to control pan motor speed
```

```
const int IN1 = 7; // IN1 on motor controller. Controls forward/backward of pan
motor
const int IN2 = 8; // IN2 on motor controller. Controls forward/backward of pan
motor
const int actuatorSpeed = 256*.99-1; // 99% speed

//Pan Motor control pins
const int ENA2 = 5; // ENA2 on motor controller. PWM to control speed -
const int IN3 = 4; // IN1 on motor controller. Controls forward/backward
const int IN4 = 3; // IN2 on motor controller. Controls forward/backward
const int panSpeed = 256*.50-1; //50% speed

// GPS global variables
uint8_t month;
uint8_t day;
uint16_t year;
uint8_t hour;
uint8_t minute;
uint8_t second;
uint32_t clockTimer;

// Sun Angle global variables
double elevation = 0; // desired tilt sun angle
double azimuth = 0;   // desired pan sun angle
const int angleTolerance = 4; // +/- 4 degrees is tolerated for
elevation/azimuth
double hrAng = 0;
const float minTiltAngle = 40.0;
const float maxTiltAngle = 70.0;
const double minPanAngle = 20.0;
const float loadPosition = 190.0;
const double maxPanAngle = loadPosition;
int rotationCount = 0.0; // b/c encoder is attached to small gear side, a
complete rotation of the small gear
                         // is equal to 82.12 degrees of the dish
bool homing = 0;

// Load button input assignment
const int PIN_BUTTON = A3; // Pin receiving digital signal from button
const int LIMIT_SWITCH = A0;
///////////////////// END LIBRARY AND GLOBAL VARIABLE SETUP
/////////////////////////////////////

///////////////////// BEGIN PROGRAM SETUP
////////////////////////////////////////////////////////
void setup() {
  // begin serial
  Serial.begin(9600);
```

```
  // motor setup
  pinMode(ENA1,OUTPUT); // ENA1 on motor controller. PWM to control speed
  pinMode(IN1,OUTPUT); // IN1 on motor controller. Controls forward/backward
  pinMode(IN2,OUTPUT); // IN2 on motor controller. Controls forward/backward
  pinMode(ENA2,OUTPUT); // ENA1 on motor controller. PWM to control speed
  pinMode(IN3,OUTPUT); // IN1 on motor controller. Controls forward/backward
  pinMode(IN4,OUTPUT); // IN2 on motor controller. Controls forward/backward

  // Button and current pin declarations
  pinMode(LIMIT_SWITCH,INPUT_PULLUP); // Analog signal from current sensor
  pinMode(PIN_BUTTON,INPUT); // Digital signal from button

  // Set I/O mode of all SPI pins for Encoder
  pinMode(SCK, OUTPUT);
  pinMode(MOSI, OUTPUT);
  pinMode(MISO, INPUT);
  pinMode(CS, OUTPUT);
  SPI.beginTransaction(SPISettings(500000, MSBFIRST, SPI_MODE0));
  digitalWrite(CS, HIGH);

  // Initialize Accelerometer
  if (!mag.begin()) { Serial.println("w: mag"); }
  if (!accel.begin()) { Serial.println("w: accel"); }
  if (!accel.begin() || !mag.begin()) {
      Serial.println("Accelerometer inititializaiton failed.\nReset arduino to
try again.");
      while(1);
  }

  // GPS init
  GPS.begin(9600);

  // uncomment this line to turn on RMC (recommended minimum) and GGA (fix
data) including altitude
  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
   // Set the update rate
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);   // 1 Hz update rate
      // Request updates on antenna status, comment out to keep quiet
  GPS.sendCommand(PGCMD_ANTENNA);

  delay(1000);
  // Ask for firmware version
  mySerial.println(PMTK_Q_RELEASE);


  //read the GPS data a few times to clear out garbage
  unsigned long t = millis();
  while(millis() - t < 5000){
      readGPS();
```

```
  }


  //Once garbage is cleared, read until fix acquired
  while(!GPS.fix){
      readGPS();
  }

  // Set clock timer to millis for keeping time throughout the day
  clockTimer = millis();

  // Read in data from GPS to global variables and adjust for local time
  month = GPS.month;
  day = GPS.day;
  year = GPS.year+2000;
  if(GPS.hour >= 8){
      hour = GPS.hour-8;
  }
  else{
      hour = GPS.hour+16;
      day = day-1;
  }
  minute = GPS.minute;
  second = GPS.seconds;

  // Calculate current solar position
  solarCalc();

  // Home Dish
  goHome();
}
///////////////////// END PROGRAM SETUP
///////////////////////////////////////////////////////

///////////////////// BEGIN PROGRAM LOOP
//////////////////////////////////////////////////////
void loop() {
  // Check button state for Load input
  bool buttonState = digitalRead(PIN_BUTTON);

  // Go to load position if button is pressed
  if(buttonState == HIGH){
      goLoad();
  }

  // Check encoder to determine current pan degree of the dish
  checkEncoder();

  // Check accelerometer to determine current tilt degree of the dish
```

```
  checkAccel();

  // Calculate the elevation and azimuth angles to determine where the dish
should be
  solarCalc();


   // logic for pan movement to orient to the sun
  if ((azimuth < maxPanAngle) && (azimuth > minPanAngle)){
      // while the dish is too far from azimuth angle, move to that angle and
check pan degree
      if ((dishDegrees - azimuth) < -angleTolerance) {
      while (dishDegrees < azimuth){
      turnCW();
      checkEncoder();
      printdata();
      }
      }
      if ((dishDegrees - azimuth) > angleTolerance) {
      while (dishDegrees > azimuth){
      turnCCW();
      checkEncoder();
      printdata();
      }
      }
  }
  // no longer needs to pan, so brake
  brakePanMotor();

  // if solar elevation is less than 40 go to minTiltAngle
  if (elevation < minTiltAngle){
      while(accelAngle > (minTiltAngle - 1.0)){
      contract();
      checkAccel();
      printdata();
      }
      brakeActuator();
  }
  // if solar elevation is above 40 degrees and less than 70 we should decide
if the dish should change its tilt
  if ((elevation > minTiltAngle) && (elevation < maxTiltAngle)) {
      // while the dish is too high, contract and check the tilt degree
      if ((accelAngle-elevation) > angleTolerance) {
      while (accelAngle > elevation){
      contract();
      checkAccel();
      printdata();
      }
      }
```

```
      // while the dish is too low, extend and check the tilt degree
      if((accelAngle-elevation) < -angleTolerance) {
      while (accelAngle < elevation){
      extend();
      checkAccel();
      printdata();
      }
      }
      // no longer needs to tilt, so brake
      brakeActuator();
  }

  //print out current, accel, encoder, dishDegree, elevation, azimuth
  printdata();
  //Serial.println("end loop");
}
///////////////////// END PROGRAM LOOP
////////////////////////////////////////////////////////


//////////////////// PRINT DATA FUNCTION
/////////////////////////////////////////////////////////
//    Purpose: print all pertinent data from sensors and solar angles
//
void printdata(){
  Serial.print(accelAngle);
  Serial.print(",");
  Serial.print(encDegrees);
  Serial.print(",");
  Serial.print(dishDegrees);
  Serial.print(",");
  Serial.print(elevation);
  Serial.print(",");
  Serial.println(azimuth);
}
//////////////////// END PRINT DATA FUNCTION
/////////////////////////////////////////////////////////


//////////////////// GET DAY FUNCTION
/////////////////////////////////////////////////////////
//    Purpose: to get the day of the year from Jan. 1
//
int getDay(unsigned int y, unsigned int m, unsigned int d){
  int days[]={0,31,59,90,120,151,181,212,243,273,304,334};    // Number of days
at the beginning of the month in a not leap year.
  int DN = 0;
  //Start to calculate the number of day
  if (m==1 || m==2){
```

```
      DN = days[(m-1)]+d;                  //for any type of year, it calculate
the number of days for January or february
  }                      // Now, try to calculate for the other months
  else if (y % 4 == 0){  //those are the conditions to have a leap year
      DN = days[(m-1)]+d+1;    // if leap year, calculate in the same way but
increasing one day
  }
  else {                             //if not a leap year, calculate in the
normal way, such as January or February
      DN = days[(m-1)]+d;
  }
  return DN;
}
////////////////////// END GET DAY FUNCTION
//////////////////////////////////////////////////////////

////////////////////// READ GPS FUNCTION
////////////////////////////////////////////////////////////
//    Purpose: to read the GPS sensor and get fix
//
void readGPS()
{
  char c = GPS.read();
  // if you want to debug, this is a good time to do it!
  if ((c) && (GPSECHO))
      Serial.write(c);

  // if a sentence is received, we can check the checksum, parse it...
  if (GPS.newNMEAreceived()) {
      if (!GPS.parse(GPS.lastNMEA()))   // this also sets the newNMEAreceived()
flag to false
      return;  // we can fail to parse a sentence in which case we should just
wait for another
  }
}
////////////////////// END READ GPS FUNCTION
/////////////////////////////////////////////////////////

////////////////////// SPI WRITE FUNCTION
///////////////////////////////////////////////////////////
//    Purpose: to communicate with the encoder through the SPI
//
uint8_t SPIWrite(uint8_t sendByte)
{
  //holder for the received over SPI
  uint8_t data;

  //the AMT20 requires the release of the CS line after each byte
  digitalWrite(CS, LOW);
```

```
  data = SPI.transfer(sendByte);
  digitalWrite(CS, HIGH);

  //we will delay here to prevent the AMT20 from having to prioritize SPI over
obtaining our position
  delayMicroseconds(10);

  return data;
}
///////////////////// END SPI WRITE FUNCTION
////////////////////////////////////////////////////////

///////////////////// PAN MOTOR FUNCTIONS
///////////////////////////////////////////////////////////
//    Purpose: turnCW() turns the dish clockwise
//          turnCCW() turns the dish counter clockwise
//          brakePanMotor() stops the pan motor
//
void turnCW() { // direction dish is moving
  analogWrite(ENA2,panSpeed);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
}

void turnCCW() {
  analogWrite(ENA2,panSpeed);
  digitalWrite(IN3,HIGH);
  digitalWrite(IN4,LOW);
}

void brakePanMotor() {
  digitalWrite(ENA2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,LOW);
}
///////////////////// END PAN MOTOR FUNCTIONS
///////////////////////////////////////////////////////

///////////////////// TILT MOTOR FUNCTIONS
//////////////////////////////////////////////////////////
//    Purpose: extend() tilts dish up
//          contract() tilts dish down
//          brakeActuator() stops the tilt motor
//
void extend() {
  analogWrite(ENA1,actuatorSpeed);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,HIGH);
}
```

```
void contract() {
  analogWrite(ENA1,actuatorSpeed);
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
}

void brakeActuator() {
  digitalWrite(ENA1,LOW);
  digitalWrite(IN1,LOW);
  digitalWrite(IN2,LOW);
}
////////////////////// END TILT MOTOR FUNCTIONS
//////////////////////////////////////////////////////

////////////////////// SOLAR CALC FUNCTION
///////////////////////////////////////////////////////////
//    Purpose: Calculates the azimuth and elevation angles of the sun using the
time and date
//
void solarCalc(){
      int dayOfYear = getDay(year, month, day);
      float currentTime = float(hour) + float(minute)/60.0 +
float(millis()-clockTimer)/3600000.0;

      double dec = -23.45*cos(((double)360 / 365)*(PI/180)*(dayOfYear+10));
      double hrAng = 15*(currentTime-12.25);

      elevation =
asin(sin(dec*(PI/180))*sin(44.623032*(PI/180))+cos(dec*(PI/180))*cos(44.623032*
(PI/180))*cos(hrAng*(PI/180)))*(180/PI);
      if (hrAng >= 0){
      azimuth = 360 -
acos((sin(dec*(PI/180))*cos(44.623032*(PI/180))-cos(dec*(PI/180))*sin(44.623032
*(PI/180))*cos(hrAng*(PI/180)))/cos(elevation*(PI/180)))*(180/PI);
      }
      else{
      azimuth =
acos((sin(dec*(PI/180))*cos(44.623032*(PI/180))-cos(dec*(PI/180))*sin(44.623032
*(PI/180))*cos(hrAng*(PI/180)))/cos(elevation*(PI/180)))*(180/PI);
      }
      azimuth = azimuth - homePanOffset - 30;
      elevation = elevation - 5;
}
////////////////////// END SOLAR CALC FUNCTION
///////////////////////////////////////////////////////

////////////////////// LOAD FUNCTION
/////////////////////////////////////////////////////////////////
```

```
//    Purpose: to instruct the pan motor to move clockwise until it reaches the
load spot
//
void goLoad(){
  bool buttonState = digitalRead(PIN_BUTTON);
  while ((accelAngle < 50) && (buttonState == HIGH)){
      buttonState = digitalRead(PIN_BUTTON);
      extend();
      checkAccel();
      printdata();
  }
  brakeActuator();
  while((dishDegrees < loadPosition) && (buttonState == HIGH)){
      buttonState = digitalRead(PIN_BUTTON);
      turnCW();
      checkEncoder();
  }
  brakePanMotor();
  while (buttonState == HIGH){
      brakePanMotor();
      brakeActuator();
      buttonState = digitalRead(PIN_BUTTON);

  }
}
///////////////////// END LOAD FUNCTION
//////////////////////////////////////////////////////////////

///////////////////// HOME FUNCTION
//////////////////////////////////////////////////////////////////
//    Purpose: to instruct the pan motor to move counter clockwise until it
hits the home
//
void goHome(){
  homing = 1;
  turnCW();
  delay(4000);
  brakePanMotor();
  delay(2000);
  turnCCW();
  bool limitSwitch = digitalRead(LIMIT_SWITCH);
  while (limitSwitch != HIGH){
      limitSwitch = digitalRead(LIMIT_SWITCH);
  }
  brakePanMotor();
  checkEncoder();
  homePanOffset = encDegrees;
  homing = 0;
}
```

```
///////////////////// END HOME FUNCTION
//////////////////////////////////////////////////////////////

///////////////////// CHECK ENCODER
//////////////////////////////////////////////////////////////////
//    Purpose: check encoder degree value to determine current pan degree of
the dish
//
void checkEncoder(){
  uint8_t data;                //this will hold our returned data from the AMT20
  uint8_t timeoutCounter;      //our timeout incrementer
  uint16_t currentPosition;   //this 16 bit variable will hold our 12-bit
position

  timeoutCounter = 0;
  //send the rd_pos command to have the AMT20 begin obtaining the current
position
  data = SPIWrite(rd_pos);
  while (data != rd_pos && timeoutCounter++ < timoutLimit){
      data = SPIWrite(nop);
  }

  if (timeoutCounter < timoutLimit) {  //rd_pos echo received
      currentPosition = (SPIWrite(nop)& 0x0F) << 8;
      currentPosition |= SPIWrite(nop);
  }
  else {  //timeout reached
      Serial.write("Error obtaining position.\nReset Arduino to restart
program.\n");
      //while(true);
  }
  float oldDeg = encDegrees;
  delay(10);
  encDegrees = float(currentPosition)/4096.0*360.0*(8.0/35.0);
  delay(10);
  if (homing == 0){
      if (encDegrees < (oldDeg-50.0)){
      rotationCount++;
      }
      if (encDegrees > (oldDeg + 50.0)){
      rotationCount--;
      }
  }
  dishDegrees = encDegrees + (float(rotationCount)*(82.12)) - homePanOffset;
}
///////////////////// END CHECK ENCODER
//////////////////////////////////////////////////////////////
```

```
///////////////////// CHECK ACCELEROMETER
//////////////////////////////////////////////////////////////////
//    Purpose: to check accelerometer degree to determine current tilt degree
of the dish
//
void checkAccel(){
  sensors_event_t accevent;
  accel.getEvent(&accevent);
  float yAccel = accevent.acceleration.y;
  if (yAccel > 9.81) {yAccel = 9.81;} //Change by possibly sensor calibration
  if (yAccel < -9.81) {yAccel = -9.81;} //Change by possibly sensor calibration
  accelAngle = acos(yAccel/9.81)*180/PI;
  delay(10);
}
///////////////////// END CHECK ACCELEROMETER
//////////////////////////////////////////////////////////////
```