```
#include <Tone.h>
#include <TinyGPS.h>
#include <EEPROM.h>
#include <Mirf.h>
#include <MirfHardwareSpiDriver.h>
#include <MirfSpiDriver.h>
#include <nRF24L01.h>
#include <SPI.h>
#include <SoftwareSerial.h>



//initializers for GPS serial communication and tones
TinyGPS gps;
SoftwareSerial gpsSerial(4, 3); // RX, TX (TX not used)
Tone toneShock;
Tone toneWarning;

//EEPROM Locations
const int RadLat   = 10;
const int RadLong  = 20;
const int A        = 30;
const int B        = 110;
const int C        = 190;
const int D        = 270;
const int RAD      = 360;

//ID detection variables
byte newDog;
byte dogID = 0x0A;
byte badDogs[1] = {0x0B};

//use in deciding between shock and warning state
int shockCounter = 0;
```

```
bool warningState = false;


//gps data holders
float RADIUS;
float latitude;
float longitude;
unsigned long fix_age;


//pinouts
const int MODE       = 6; //operate or calibrate switch. HIGH = Calibrate, LOW = Operate
const int TYPE       = 7; //HIGH = Corner, LOW = Instant
const int DETECTION  = 5; //ID detection switch active HIGH
const int CONTROL    = 8; //push button for setting radius or corners
const int DIP1       = A5 ; //dipswitch for location or radius
const int DIP2       = A4; //dipswitch for location or radius
const int DIP3       = A3; //dipswitch for location or radius
const int DIP4       = A2; //dipswitch for location or radius
const int SHOCK      = A1; //shock buzzer and red led
const int WARNING    = A0; //warning buzzer and yellow led
const int CE         = 9;
const int CSN        =10;


//feeds serial data into the gps
bool feedgps();
//checks for a bad listed dog in the area
void checkID();
//checks if the dog is in the box
bool inBox();
//gets new gps data
bool getGPS();
//tells the user that a point has been saved
void pointSaved();
```

```
void setup()
{
 pinMode(MODE,     INPUT);
 pinMode(CONTROL,  INPUT);
 pinMode(TYPE,     INPUT);
 pinMode(DIP1,     INPUT);
 pinMode(DIP2,     INPUT);
 pinMode(DIP3,     INPUT);
 pinMode(DIP4,     INPUT);
 pinMode(DETECTION, INPUT);
 pinMode(SHOCK,    OUTPUT);
 pinMode(WARNING,  OUTPUT);

 //initializes shock and warning state
 shockCounter = 0;
 warningState = false;

 //initializes the serial communications and the tones
 Serial.begin(9600);
 gpsSerial.begin(9600);
 toneShock.begin(SHOCK);
 toneWarning.begin(WARNING);

 //initializes the mirf library which controls the transceivers
 Mirf.cePin = CE;
 Mirf.csnPin = CSN;
 Mirf.spi = &MirfHardwareSpi;
 Mirf.init();
 Mirf.setRADDR((byte *)"GPSDS");
 Mirf.setTADDR((byte *)"GPSDS");
 Mirf.payload = 1;
 Mirf.channel = 90;
 Mirf.config();
```

```
}

void loop()
{
 float distance;
 int switchState = 0;
 float instantLat, instantLong;

 Mirf.flushRx();
 switchState = digitalRead(MODE);
 if(switchState == LOW)
 {
  Serial.println("Switch in Operate Mode");

  while(!getGPS())
  {
   Serial.println("Waiting for GPS Data...");
  }
  switchState = digitalRead(TYPE);
  if(switchState == LOW)
  {
   //operate subroutine for radius
   Serial.println("Switch in Type Instant Radius");
   instantLat = Eeprom_Read_Float(RadLat);
   instantLong = Eeprom_Read_Float(RadLong);
   RADIUS = Eeprom_Read_Float(RAD);
   Serial.print("Loaded Radius of: ");
   Serial.print(RADIUS, 1);
   Serial.print(" set at  Lat: ");
   Serial.print(instantLat, 7);
   Serial.print(" Long: ");
   Serial.println(instantLong, 7);
```

```
    distance = gps.distance_between (latitude, longitude, instantLat, instantLong); //calculates the distance between current location and the
radius center
    Serial.print("Dog is ");
    Serial.print(distance, 7 );
    Serial.println(" meters away");
    //checks if dog is inside the radius
    if(distance > RADIUS)
    {
     if(warningState == false)
       warnthem();
     else
       shockthem();
    }
    else
     checkID();
  }
  //suboutine for corner mode operation
  else
  {
   Serial.println("Switch in Type Corner");
   if(inBox() == true)     //checks if the dog is in the box
     checkID();           //if the dog is inside the box then it checks if the dog is near a bad listed dog
   else if(warningState == false)//checks if dog should be warned or shocked
   {
     warnthem();
     warningState = true;
   }
   else
     shockthem();
  }
 }
 else
 {
```

```
//calibrating device
Serial.println("Switch in Calibrate Mode");
switchState = digitalRead(TYPE);
if(switchState == LOW)
{
 Serial.println("Press to Set Radius...");
 while ((digitalRead(CONTROL)) == HIGH){   }//waits for push button to go down
 while ((digitalRead(CONTROL)) == LOW){   }//then back up
 while(!getGPS()) //waits for gps data to come in
 {
  Serial.println("Waiting for GPS Data...");
 }

 //saves radius location in the eeprom for permanent use
 Eeprom_Write_Float(RadLat, latitude);
 Eeprom_Write_Float(RadLong, longitude);

 //decides which radius to use based on the dip switches
 if(digitalRead(DIP2) == HIGH)
 {
  if(digitalRead(DIP3) == HIGH)
    RADIUS = 5.0;
  else
    RADIUS = 10.0;
 }
 else if(digitalRead(DIP3) == HIGH)
    RADIUS = 25.0;
 else
    RADIUS = 50.0;

 Eeprom_Write_Float(RAD, RADIUS);//saves radius in eeprom for permanant use

 Serial.print("Radius of: ");
```

```
 Serial.print(RADIUS, 1);
 Serial.print(" set at  Lat: ");
 Serial.print(latitude, 7);
 Serial.print(" Long: ");
 Serial.println(longitude, 7);
 pointSaved();//displays to the user that the point was saved
}
else
{
 //corner calibration
 Serial.println("Press to Set Corners...");
 int i;
 //checks which location to save to and gives i the location's eeprom location
 if(digitalRead(DIP2) == LOW)
 {
  if(digitalRead(DIP1) == LOW)
    i = A;
  else
    i = B;
 }
 else if(digitalRead(DIP1) == LOW)
  i = C;
 else
  i = D;

 //first corner
 while (digitalRead(CONTROL) == HIGH){  }//repeats button process from earlier
 while (digitalRead(CONTROL) == LOW){  }
 while(!getGPS())
 {
  Serial.println("Waiting for GPS Data...");
 }
 Eeprom_Write_Float(i,     latitude);//saves data in eeprom
```

```
Eeprom_Write_Float(i + 10, longitude);
Serial.println("Corner 1 Set");
pointSaved();

//second corner done exactly like first corner
while (digitalRead(CONTROL) == HIGH){   }
while (digitalRead(CONTROL) == LOW){   }
while(!getGPS())
{
  Serial.println("Waiting for GPS Data...");
}

Eeprom_Write_Float(i + 20,  latitude);
Eeprom_Write_Float(i + 30, longitude);
Serial.println("Corner 2 Set");
pointSaved();

//third corner done exactly like first corner
while (digitalRead(CONTROL) == HIGH){   }
while (digitalRead(CONTROL) == LOW){   }
while(!getGPS())
{
  Serial.println("Waiting for GPS Data...");
}
Eeprom_Write_Float(i + 40,  latitude);
Eeprom_Write_Float(i + 50, longitude);
Serial.println("Corner 3 Set");
pointSaved();

//fourth corner done exactly like first corner
while (digitalRead(CONTROL) == HIGH){   }
while (digitalRead(CONTROL) == LOW){   }
while(!getGPS())
```

```
    {
     Serial.println("Waiting for GPS Data...");
    }
    Eeprom_Write_Float(i + 60, latitude);
    Eeprom_Write_Float(i + 70, longitude);
    Serial.println("Corner 4 Set");
    pointSaved();
   }
 }
}

//turns both lights on for 1 second to show a point has been saved
void pointSaved()
{
 digitalWrite(SHOCK,  HIGH);
 digitalWrite(WARNING, HIGH);
 delay(1000);
 digitalWrite(SHOCK,   LOW);
 digitalWrite(WARNING, LOW);
}

//checks if current location is inside the box
bool inBox()
{
 float long1, long2, long3, long4, lat1, lat2, lat3, lat4;
 float sideA, sideB, sideC, sideD, crossQ, crossP, s1, s2, s3, s4;
 float cross1, cross2, cross3, cross4, area, area1, area2, area3, area4, sum;
 int i = 0;
 //determines location being used

 if(digitalRead(DIP2) == LOW)
 {
  if(digitalRead(DIP1) == LOW)
```

```
  i = A;
 else
  i = B;
}
else if(digitalRead(DIP1) == LOW)
  i = C;
else
  i = D;
//loads the location data from the eeprom
long1 = Eeprom_Read_Float(i + 10);
long2 = Eeprom_Read_Float(i + 30);
long3 = Eeprom_Read_Float(i + 50);
long4 = Eeprom_Read_Float(i + 70);
lat1 = Eeprom_Read_Float(i    );
lat2 = Eeprom_Read_Float(i + 20);
lat3 = Eeprom_Read_Float(i + 40);
lat4 = Eeprom_Read_Float(i + 60);
//calculates side lengths
sideA = gps.distance_between (lat1, long1, lat2, long2);
sideB = gps.distance_between (lat2, long2, lat3, long3);
sideC = gps.distance_between (lat3, long3, lat4, long4);
sideD = gps.distance_between (lat4, long4, lat1, long1);
//calculates the diagonal lengths
crossQ = gps.distance_between (lat1, long1, lat3, long3);
crossP = gps.distance_between (lat2, long2, lat4, long4);
//distance from point to each corner
cross1 = gps.distance_between (lat1, long1, latitude, longitude);
cross2 = gps.distance_between (lat2, long2, latitude, longitude);
cross3 = gps.distance_between (lat3, long3, latitude, longitude);
cross4 = gps.distance_between (lat4, long4, latitude, longitude);
//semi perimeter for each triangle
s1 = (sideA + cross1 + cross2) * .5;
s2 = (sideB + cross2 + cross3) * .5;
```

```
s3 = (sideC + cross3 + cross4) * .5;
s4 = (sideD + cross4 + cross1) * .5;
//area of each triangle
area1 = sqrt(s1 * (s1 - cross1) * (s1 - cross2) * (s1 - sideA));
area2 = sqrt(s2 * (s2 - cross2) * (s2 - cross3) * (s2 - sideB));
area3 = sqrt(s3 * (s3 - cross3) * (s3 - cross4) * (s3 - sideC));
area4 = sqrt(s4 * (s4 - cross4) * (s4 - cross1) * (s4 - sideD));
//area of box
area = .25 * sqrt((4 * crossP * crossP * crossQ * crossQ) - pow((sideA * sideA + sideC * sideC - sideB * sideB -sideD * sideD), 2));
//checks which 3 triangles are the biggest and uses those
if(area1 <= area2)
{
 if(area1 <= area3)
 {
  if(area1 <= area4)
    sum = area2 + area3 + area4;
  else
    sum = area2 + area3 + area1;
 }
 else
 {
  if(area3 <= area4)
    sum = area2 + area1 + area4;
  else
    sum = area2 + area3 + area1;
 }
}
else
{
 if(area2 <= area3)
 {
  if(area2 <= area4)
    sum = area1 + area3 + area4;
```

```
    else
      sum = area2 + area3 + area1;
   }
   else
   {
    if(area3 <= area4)
      sum = area2 + area4 + area1;
    else
      sum = area2 + area3 + area1;
   }
 }
 Serial.print("Sum: ");
 Serial.print(sum, 5);
 Serial.print(" Area: ");
 Serial.println(area, 5);
 //decides if the point is inside or outside of the box
 if(sum > area)
   return false;
 else
   return true;
}

//checks if there are any dog IDs in the area and if they are bad listed. then it transmits its own unique ID
void checkID()
{
 bool switchState;
 bool found = false;
 bool dataFound = false;
 switchState = digitalRead(DETECTION);
 if(switchState == HIGH)
 {
  Serial.println("ID Detection Active");
  Serial.println("Checking for Match...");
```

```
unsigned long start = millis();
while(millis() < start + 100)
{
 if(Mirf.dataReady()) //checks if there is data ready
 {
  dataFound = true;
  Mirf.getData(&newDog);//gets the new data
  for(int i = 0; i < sizeof(badDogs); i++)//checks for each dog in the array
  {
   if(badDogs[i] == newDog) //checks if bad listed
   {
    Serial.print("Found Match: ");
    Serial.println(newDog, HEX);
    if(warningState == false)
     warnthem();
    else
     shockthem();
    found = true;
    break;
   }
  }
  break;
 }
}
if(dataFound == false)//no dogs were found
{
 warningState = false;
 shockCounter = 0;
 Serial.println("No Dogs found");
}
else if(found == false)//dog wasn't a match
{
 warningState = false;
```

```
    shockCounter = 0;
    Serial.print(newDog, HEX);
    Serial.println(" is not a match");
   }
  }
  else
  {
   //ID detection is disabled
   Serial.println("ID Detection Inactive");
   shockCounter = 0;
   warningState = false;
  }
  Serial.println("Sending Unique ID");
  Mirf.send(&dogID);//sends its unique ID
  while(Mirf.isSending()){//waits for it to finish sending
  }
  Serial.println("Finished sending");
}

//gets new gps data for use
bool getGPS()
{
 bool newData = false;
 unsigned long start = millis();
 while(millis() - start < 1000)
 {
  //feeds gps data for a second until its received an entire string of new data
  if(feedgps())
   newData = true;
 }
 if(newData == true)
 {
  //gets the position in floats and fix age
```

```
   gps.f_get_position(&latitude, &longitude, &fix_age);
   unsigned short numSats = gps.satellites();//gets the number of satellites used
   Serial.print("Lat: ");
   Serial.print(latitude, 7);
   Serial.print(" Long: ");
   Serial.print(longitude, 7);
   Serial.print(" Number of Satellites: ");
   Serial.print(numSats);
   Serial.print(" Fix Age: ");
   Serial.println(fix_age);
 }
 return newData;//returns if new data is ready yet or not
}

//takes data from the gps from serial line until a whole line is received
bool feedgps()
{
  while (gpsSerial.available())
   {
    if (gps.encode(gpsSerial.read()))
      return true;
   }
   return false;
}

//warns the dog
void warnthem()
{
 unsigned long start = millis();
 toneWarning.play(2400);
 Serial.println("WARNING DOG");
 while(millis() - start < 1000){
 }
```

```
 toneWarning.stop();//plays tone for 1 second
 delay(2000);//waits 2 seconds more. total of 3 seconds
 warningState = true;//dog has been warned
}

void shockthem()
{
 unsigned long start = millis();
 toneShock.play(4000);
 Serial.println("SHOCKING DOG");
 while(millis() - start < 1000){
 }
 toneShock.stop();//shocks dog for 1 second
 shockCounter++;//increments shock counter
 if(shockCounter == 5)//checks if max shocks has been hit
 {
  Serial.println("Shock Max Hit. Going on Cooldown...");
  warningState = false;
  shockCounter = 0;
  delay(15000); //waits 15 seconds before shocking dog again
 }
 else
  delay(3000);//waits 3 seconds between shocks
}

//subroutine for reading a float from a specific eeprom address
float Eeprom_Read_Float(int addr)
{
    float result;
    byte *ptr= (byte *)&result;
    byte i;

    for (i=0;i<4;i++)
```

```
        *(ptr++)=EEPROM.read(addr++);


    return result;
}


//subroutine for writing a float to a specific eeprom address
void Eeprom_Write_Float(int addr,float data)
{
    byte *ptr= (byte *)&data;
    byte i;


    for (i=0;i<4;i++)
        EEPROM.write(addr++,*(ptr++));
}
```