EHOURA

/ˈɔrə/ (‘aura’)

Interactive device for managing mood changes related to neurovestibular differences in time perception and physical orientation in outer space.

DEA 5210 | Hang Zhao, Jenny Yu, Pika Cai

03

DESIGN PROCESS DESCRIBED

A. Problem & Prompt

"Design an interactive device at a small physical scale responsive to the challenges of those inhabiting the International Space Station."

B. Project Proposal

"Provide psychological assurance to astronauts experiencing neurovestibular challenges with time/spatial perception in the ISS."

C. Ideation I

Created moodboards to generate a representation of the intended before and after experiences of interacting with our design.

D. Ideation II

Created morphological charts to explore the design space, forms & means of achieving our desired effects from our moodboard.

E. Design I

Created concept sketches to visualize the chosen morphological chart system. "The ring shape is easily held and displays lights representing the sun and moon's position to provide psychological assurance pertaining to the astronaut's spatial orientation."

F. Feedback I

Presentation & peer feedback with a gif to describe the interaction. "What is the meaning of this interaction *to the user? What is the* story*?"*

G. Design II

*"*In times of temporal and orientation-related distress, Ehoura helps to assure them *of their current time and spatial orientation, and to better understand their mood patterns* and effectively regulate their emotional well being during their stay in the ISS.*"*

H. Design III

Rapid prototyped with Aluminum to figure out device dimensions & form for optimal grip and component storage. 3D modeling and component purchasing based on decided dimensions and how to electronically express Design II concept.

I. Assembling the Model

Bringing our vision to reality as a functional device. 3D printing, wiring, soldering, and fitting the electronics into the 3D model.

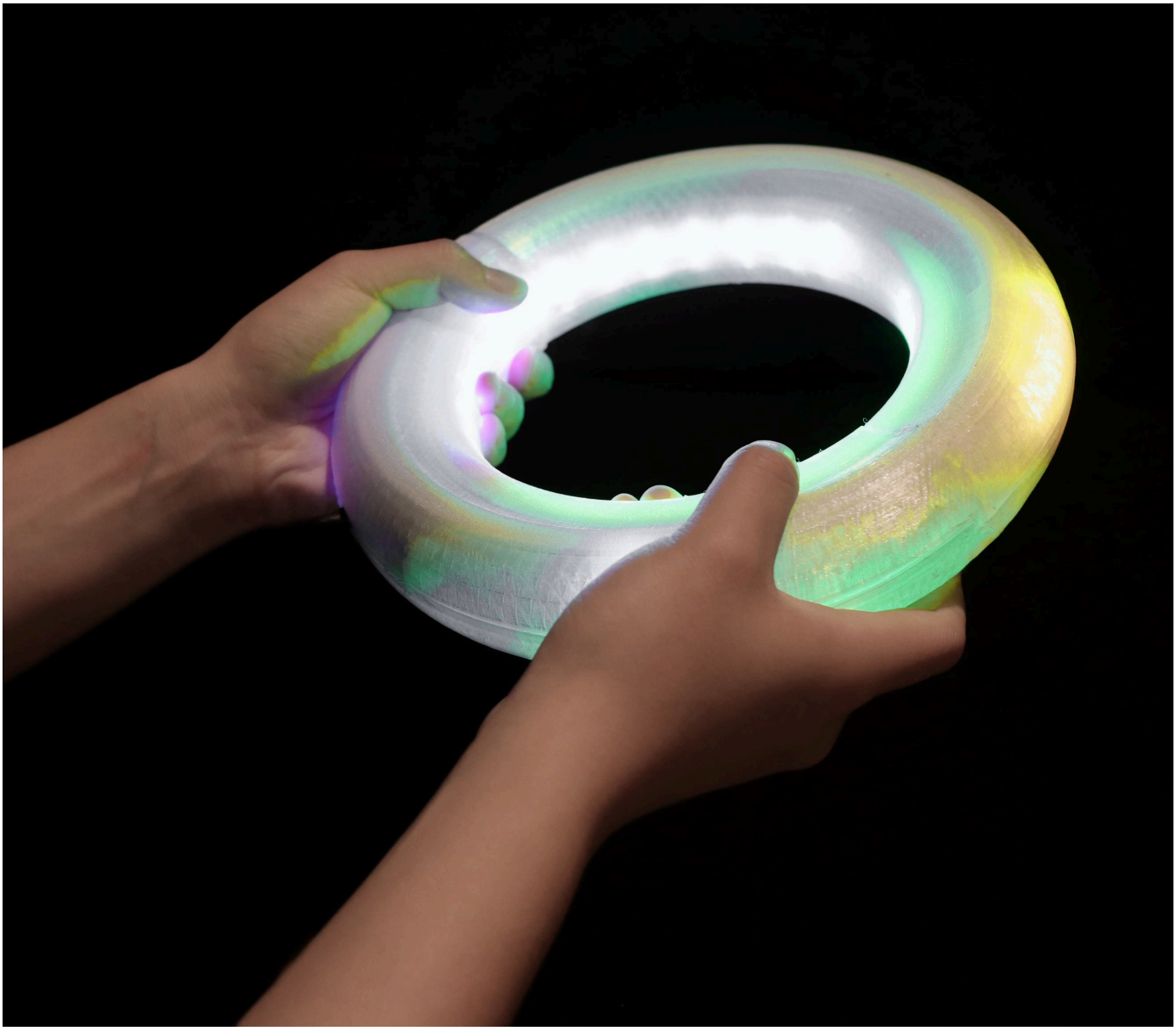
The current state of the EHOURA prototype records user input and produces output as intended. This includes:

- *producing consistent lighting angles on the outer ring based on solar positioning regardless of shifts in the direction the GPS faces when the user moves while holding EHOURA,*
- *setting a new mood tracking light with 30 seconds of non-activity after turning the potentiometer, and*
- *powering the device on/off by pressing the button.*

The main technological challenges in implementing the EHOURA prototype included that the GPS module's reception of satellite signals can vary based on location (higher reception outdoors than indoors). The 3D printing process additionally involved several reprints as the error margin was inconsistent across printed halves of the model, in such a way that they could not be fit together.

One potential improvement in the future could be to explore replacing the current button for powering the device with either a pressure sensor and softer material for the shell, or a button with a portion of the shell casing the button. Currently, the button module is quite small but protrudes through the shell's side in a way that may visually interrupt the outer light sequence. Both suggestions would retain the same purpose as current, while improving the visual appeal of the prototype, discovery of the button, and stress-reductive qualities (squeezing the device vs. pinpointing the button to power on). A similar improvement could also be applied to the potentiometer.

04
DISCUSSION



VIDEO: https://vimeo.com/946845922?share=copy

05

VIDEO

```
#include <Adafruit_NeoPixel.h>
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include "bmm150.h"
#include "bmm150_defs.h"
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include <HTTPClient.h>

#include <ArduinoJson.h>
#include <WiFi.h>

#define GPS_RX_PIN 16             // Define the RX point of GPS module
#define GPS_TX_PIN 17             // Define the TX point of GPS module
#define FIRST_STRIP_PIN 26        // Define the LED PIN
#define FIRST_STRIP_NUMPIXELS 39  // LED numbers
#define SMOOTHING_FACTOR 0.2      // Smooth animation factor
#define CHANGE_THRESHOLD 1.0
#define UPDATE_INTERVAL 300000
#define GPS_UPDATE_INTERVAL 300000


HardwareSerial gpsSerial(2);
//SoftwareSerial ss(GPS_RX_PIN, GPS_TX_PIN);
TinyGPSPlus gps;
Adafruit_NeoPixel strip(FIRST_STRIP_NUMPIXELS, FIRST_STRIP_PIN, NEO_GRBW + NEO_KHZ800);
Adafruit_MPU6050 mpu;
BMM150 bmm = BMM150();

int lastKnownHour = -1;
const char* ssid = "HANG"; // WIFI username
const char* password = "123456787"; // WIFI password
const char* api_key = "0163423784034162a7b23a1ff791a8a3";  // api key

unsigned long lastAPICallTime = 0;
unsigned long lastHourUpdateTime = 0;  // last update interval
float smoothedHeading = 0;  // smoothed heading
float storedSunAzimuth = 0, storedMoonAzimuth = 0;
float latitude, longitude;  // global variable to store GPS data
float sunAzimuth = 0.0;
float moonAzimuth = 0.0;

void breathingEffect(bool fadeOut = false) {
  static float t = 0.0;
  static bool increasing = true;
  static unsigned long lastUpdate = millis();

  const int updateInterval = 30;
  uint32_t lightPurple = strip.Color(150, 0, 150);
  uint32_t lightBlue = strip.Color(0, 150, 150);
```

06
CODE

01

INTERACTION

Arnold, 45, is an astronaut who experiences challenges with time perception and physical orientation in space. At approximately 3 PM of his given schedule, Arnold is feeling particularly disoriented and distressed. Arnold discovers EHOURA attached to the wall along the ISS corridor and reaches to its abstract form out of curiosity.

EHOURA displays a yellow glow at the top right of the outer ring, which shifts to one particular direction as Arnold is floating around, like a compass.

Arnold thinks, it would be helpful to him to visualize his emotions in the device. Arnold uses knob on the device to change the color of the light on the inner ring to reflect his emotions.

Arnold passes by a window. Seeing the sun outside, he realizes the yellow glow represents that it is daytime in his 24-hour routine, and the angle of the light on the ring represents the position of the sun from him.

Discovering the visual lighting, he feels more grounded and at home with his emotions. Altogether, EHOURA provides a means for Arnold to gain a sense of clarity during emotional imbalances and challenges with space-time orientation.

Over time, Arnold continues to seek out Ehoura when he experiences temporal and orientation-related distress, using Ehoura to better understand his mood patterns and effectively regulate his emotional well being during his stay in the ISS.

02

OPERATION & COMPONENTS

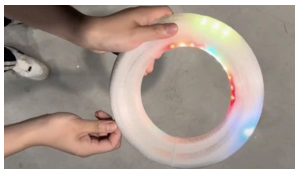
EHOURA consists of a curiously minimalistic torus form which displays a variety of meaningful light effects.

Press the button to power the device.

The GPS module takes 30s to power on. An animated display of lights will show along the LED strip during this time.

Rotate Ehoura to find the sun/moon.

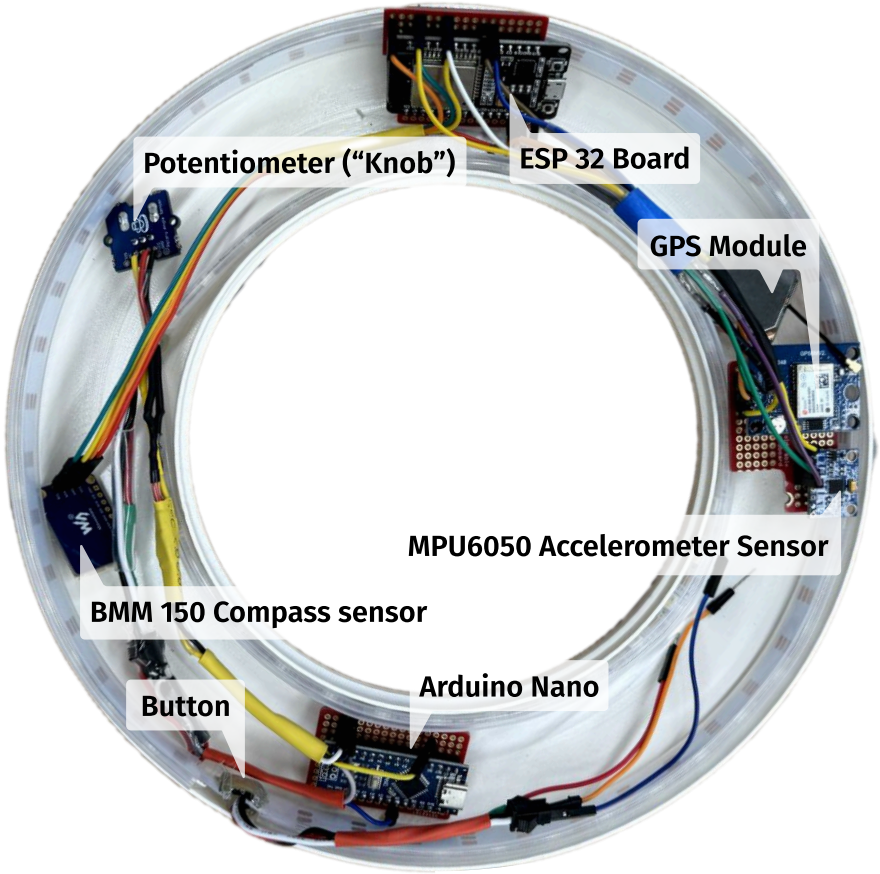
Yellow/blue lights on the outer LED strip show the sun/moon's azimuth from the user, based on current GPS data, when the compass senses product movement.



Turn the knob to record an emotion.

A new light on the inner ring's LED strip will change color as the potentiometer ("knob") is turned, and its color set after 30s of non-activity.

EHOURA's shell is made of a 3D printed torus casing of plastic translucent filament, sliced in half and extruded in the sliced section such that the halves fit as a case. All electronic parts are soldered and fit hidden in this casing, except a button and potentiometer which are exposed through two holes drilled on the side of the shell.



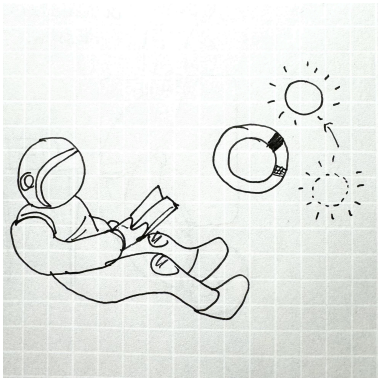
03

DESIGN PROCESS ARTIFACTS

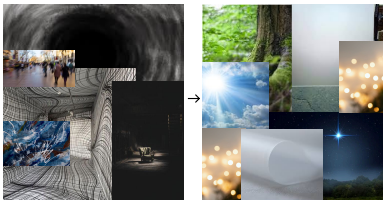
A. Problem & Prompt

"Design an interactive device at a small physical scale responsive to the challenges of those inhabiting the International Space Station."

B. Project Proposal



C. Ideation I



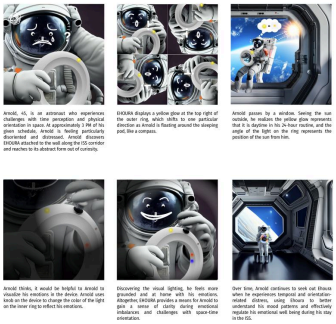
Before After

D. Ideation II

E. Design I

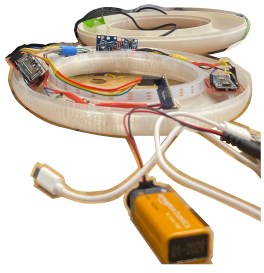
F. Feedback I

G. Design II



H. Design III

I. Assembling the Model



EHOURA

Back in orbit.

```
 if (fadeOut) {
   while (t > 0) {
     t -= 0.02;
     t = fmax(0.0f, t);
     uint32_t color = interpolateColor(lightPurple, lightBlue, t);
     for (int i = 0; i < FIRST_STRIP_NUMPIXELS; i++) {
       strip.setPixelColor(i, color);
     }
     strip.show();
     delay(30);
   }
   t = 0;
   increasing = true;
   return;
 }

  if (millis() - lastUpdate >= updateInterval) {
   lastUpdate = millis();
   if (increasing) {
     t += 0.05;
     if (t >= 1.0) {
       t = 1.0;
       increasing = false;
     }
   } else {
     t -= 0.05;
     if (t <= 0.0) {
       t = 0.0;
       increasing = true;
     }
   }

   uint32_t color = interpolateColor(lightPurple, lightBlue, t);
   for (int i = 0; i < FIRST_STRIP_NUMPIXELS; i++) {
     strip.setPixelColor(i, color);
   }
   strip.show();

   Serial.print("t: ");
   Serial.println(t);
  }
}

uint32_t interpolateColor(uint32_t color1, uint32_t color2, float t) {
  int r1 = (color1 >> 16) & 0xFF;
  int g1 = (color1 >> 8) & 0xFF;
  int b1 = color1 & 0xFF;

  int r2 = (color2 >> 16) & 0xFF;
  int g2 = (color2 >> 8) & 0xFF;
  int b2 = color2 & 0xFF;
```

06

CODE (continued)

```
  int r = r1 + (r2 - r1) * t;
  int g = g1 + (g2 - g1) * t;
  int b = b1 + (b2 - b1) * t;

  return ((r & 0xFF) << 16) | ((g & 0xFF) << 8) | (b & 0xFF);
}

void setup() {
  Serial.begin(115200);
  setupWiFi();
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);
  strip.begin();
  strip.show();
  Wire.begin();

  if (bmm.initialize() != BMM150_OK) {
    Serial.println("BMM150 sensor not found");
    while (1)
      ;
  }
  breathingEffect(false);
}

void loop() {
  static bool isFirstGPSReceived = false;
  static float latitude = 0, longitude = 0;
  static unsigned long lastAPICallTime = 0;
  unsigned long currentMillis = millis();
  static unsigned long lastGPSUpdateTime = 0;

  if (!isFirstGPSReceived) {
    if (currentMillis - lastGPSUpdateTime > 60000 || lastGPSUpdateTime == 0) {
      Serial.println("Reading GPS...");
      lastGPSUpdateTime = currentMillis;
      bool newData = false;

      while (gpsSerial.available() > 0) {
        char c = gpsSerial.read();
        if (gps.encode(c)) {
          newData = true;
        }
      }

      if (newData && gps.location.isValid()) {
        latitude = gps.location.lat();
        longitude = gps.location.lng();
        Serial.println("Got GPS data.");
        Serial.print("Latitude: ");
        Serial.println(latitude);
        Serial.print("Longitude: ");
        Serial.println(longitude);
        isFirstGPSReceived = true; // Mark that GPS data has been received
```

06

CODE (continued)

```
    }
   }
  } else {
   // Update sun and moon azimuths every five minutes or if never updated
   if (currentMillis - lastAPICallTime > UPDATE_INTERVAL || lastAPICallTime == 0) {
    lastAPICallTime = currentMillis;
    updateAzimuthsFromAPI(latitude, longitude, "sun");
    updateAzimuthsFromAPI(latitude, longitude, "moon");
   }
   updateAzimuths(); // Regular LED update based on current azimuths
  }

  // Continue or stop the breathing effect based on GPS reception
  breathingEffect(!isFirstGPSReceived);
  delay(50); // Control loop frequency
}


void setupWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
  }
  Serial.println(" connected");
}

void updateAzimuths() {
  float rawHeading = getHeading();
  if (fabs(smoothedHeading - rawHeading) > CHANGE_THRESHOLD || fabs(smoothedHeading -
rawHeading) > (360 - CHANGE_THRESHOLD)) {
   smoothedHeading = smoothedHeading * (1 - SMOOTHING_FACTOR) + rawHeading *
SMOOTHING_FACTOR;
  }

  float sunAzimuth = updateAzimuthsFromAPI(latitude, longitude, "sun");
  float moonAzimuth = updateAzimuthsFromAPI(latitude, longitude, "moon");

  int sunIndex = calculateLEDIndex(sunAzimuth - smoothedHeading);
  int moonIndex = calculateLEDIndex(moonAzimuth - smoothedHeading);

  animateLEDTransition(sunIndex, moonIndex);
}

float updateAzimuthsFromAPI(double latitude, double longitude, String body) {
  unsigned long currentMillis = millis();
  if (currentMillis - lastAPICallTime > UPDATE_INTERVAL || lastAPICallTime == 0) {
   WiFiClientSecure client;
   client.setInsecure();
   HTTPClient http;
```

06

CODE (continued)

```
    String serverPath = "https://api.ipgeolocation.io/astronomy?apiKey=" + String(api_key) + "&lat=" +
String(latitude, 6) + "&long=" + String(longitude, 6);

    http.begin(client, serverPath);
    int httpResponseCode = http.GET();

    if (httpResponseCode > 0) {
      String response = http.getString();
      DynamicJsonDocument doc(1024);
      deserializeJson(doc, response);
      if (doc.containsKey(body + "_azimuth")) {
        float azimuth = doc[body + "_azimuth"].as<float>();
        http.end();
        if (body == "sun") {
          storedSunAzimuth = azimuth;
        } else if (body == "moon") {
          storedMoonAzimuth = azimuth;
        }
        lastAPICallTime = currentMillis;
        return azimuth;
      }
    }
    http.end();
  }
  return body == "sun" ? storedSunAzimuth : storedMoonAzimuth;
}


float getHeading() {

  bmm150_mag_data value;
  bmm.read_mag_data();
  value.x = bmm.raw_mag_data.raw_datax;
  value.y = bmm.raw_mag_data.raw_datay;
  value.z = bmm.raw_mag_data.raw_dataz;

  float heading = atan2(value.y, value.x) * 180 / PI;
  heading = heading < 0 ? heading + 360 : heading;
  const char* direction;
  if (heading < 22.5 || heading >= 337.5) direction = "North";
  else if (heading < 67.5) direction = "North-East";
  else if (heading < 112.5) direction = "East";
  else if (heading < 157.5) direction = "South-East";
  else if (heading < 202.5) direction = "South";
  else if (heading < 247.5) direction = "South-West";
  else if (heading < 292.5) direction = "West";
  else direction = "North-West";

  return heading;
}
```

06

CODE (continued)

```
void animateLEDTransition(int sunIndex, int moonIndex) {
  static int lastSunIndex = 0;
  static int lastMoonIndex = 0;

  int stepCount = 10;

  int sunSteps = shortestPath(lastSunIndex, sunIndex, FIRST_STRIP_NUMPIXELS);
  int moonSteps = shortestPath(lastMoonIndex, moonIndex, FIRST_STRIP_NUMPIXELS);

  for (int i = 0; i <= stepCount; i++) {
    int currentSunIndex = (lastSunIndex + sunSteps * i / stepCount + FIRST_STRIP_NUMPIXELS) %
FIRST_STRIP_NUMPIXELS;
    int currentMoonIndex = (lastMoonIndex + moonSteps * i / stepCount + FIRST_STRIP_NUMPIXELS) %
FIRST_STRIP_NUMPIXELS;
    displaySunAndMoon(currentSunIndex, currentMoonIndex);
    delay(10);
  }
  lastSunIndex = sunIndex;
  lastMoonIndex = moonIndex;
}

// Calculate the shortest path between two indices on a circular array
int shortestPath(int fromIndex, int toIndex, int totalElements) {
  int forwardPath = (toIndex - fromIndex + totalElements) % totalElements;
  int backwardPath = -((fromIndex - toIndex + totalElements) % totalElements);
  return (abs(forwardPath) < abs(backwardPath)) ? forwardPath : backwardPath;
}

void displaySunAndMoon(int sunIndex, int moonIndex) {
  int hour = getCurrentHour();
  uint32_t sunColor = getSunColor(hour);    // get color based on hour
  uint32_t moonColor = getMoonColor(hour);

  strip.clear();

  setSunGradient(sunIndex, sunColor, FIRST_STRIP_NUMPIXELS);
  setMoonGradient(moonIndex, moonColor, FIRST_STRIP_NUMPIXELS);

  strip.show();
}


void setSunGradient(int index, uint32_t color, int numPixels) {
  strip.setPixelColor(index, color);
  uint32_t dimColor = strip.Color(strip.gamma8((color >> 16) & 0xFF) / 2,
                                  strip.gamma8((color >> 8) & 0xFF) / 2,
                                  strip.gamma8(color & 0xFF) / 2);

  strip.setPixelColor((index - 1 + numPixels) % numPixels, dimColor);
  strip.setPixelColor((index + 1) % numPixels, dimColor);
}
```

06

CODE (continued)

```
void setMoonGradient(int index, uint32_t color, int numPixels) {
  strip.setPixelColor(index, color);
  uint32_t dimColor = strip.Color(strip.gamma8((color >> 16) & 0xFF) / 3,
                                  strip.gamma8((color >> 8) & 0xFF) / 3,
                                  strip.gamma8(color & 0xFF) / 3);

  strip.setPixelColor((index + 1) % numPixels, dimColor);
}

int calculateLEDIndex(float azimuth) {
  azimuth = fmod(azimuth, 360);
  if (azimuth < 0) azimuth += 360;
  return int((azimuth / 360.0) * FIRST_STRIP_NUMPIXELS) % FIRST_STRIP_NUMPIXELS;
}

int getCurrentHour() {
  unsigned long currentMillis = millis();
  if (currentMillis - lastHourUpdateTime >= 3600000 || lastHourUpdateTime == 0) {
    while (gpsSerial.available() > 0) {
      if (gps.encode(gpsSerial.read())) {
        if (gps.time.isValid()) {
          int hour = gps.time.hour();
          int minute = gps.time.minute();
          int second = gps.time.second();

          bool isDaylightSaving = true;
          int timezoneOffset = isDaylightSaving ? -4 : -5;
          hour = (hour + timezoneOffset + 24) % 24;

          Serial.print("Local Time (EDT/EST): ");
          Serial.print(hour);
          Serial.print(":");
          Serial.print(minute);
          Serial.print(":");
          Serial.println(second);
          if (hour != lastKnownHour) {
            lastKnownHour = hour;
            lastHourUpdateTime = currentMillis;
            Serial.print("Updated Hour: ");
            Serial.println(lastKnownHour);
          }
        }
      }
    }
  }
  return lastKnownHour;
}


uint32_t getSunColor(int hour) {
  if (hour == -1) {
```

06

CODE (continued)

```
  if (hour == -1) {
    return strip.Color(255, 165, 0);
  }

  if (hour >= 6 && hour < 12) {          // morning
    return strip.Color(255, 165, 0);     // orange
  } else if (hour >= 12 && hour < 18) {  // noon
    return strip.Color(255, 255, 0);     // light yellow
  } else if (hour >= 18 && hour < 20) {  // afternoon
    return strip.Color(255, 102, 0);     // dark orange
  } else {                               // night
    return strip.Color(159, 70, 10);     // grey
  }
}

uint32_t getMoonColor(int hour) {
  if (hour == -1) {
    return strip.Color(100, 149, 237);
  }

  if (hour >= 18 || hour < 6) {          // night to morning
    if (hour >= 18 && hour < 22) {       // afternoon
      return strip.Color(100, 149, 237); // dark blue
    } else if (hour >= 22 || hour < 4) { // late night
      return strip.Color(25, 25, 112);   // dark blue
    } else {                             // dawn
      return strip.Color(65, 105, 225);  // royal blue
    }
  } else {
    return strip.Color(128, 128, 128);
  }
}
```

06

CODE (continued)