Listing 1. Firmware programu do wskazywania pozycji ISS

// ISS Tracker - David Hunt

// Wersja 1.4

// Licencja - Attribution Non-Commercial Share Alike (by-nc-sa)

// pobierz wszystkie niezbędne biblioteki za pomocą menedżera bibliotek

#include < AccelStepper . h >

// do sterowania silnikami krokowymi

#include < ezTime . h >

#include < **WiFi** . h >

#include < HTTPClient . h >

// aby pobrać dane z API

#include < ArduinoJson . h >

// aby wyodrębnić informacje o lokalizacji ISS

#include < U8g2lib . h >

// wyszukaj U8g2 w menedżerze bibliotek

#include < Wire . h >

time\_t next\_pass\_timestamp = 1572652800 ; // 2 listopada 2019 jako punkt odniesienia

const char ssid [ ] = "Twoja sieć tutaj" ;  // SSID sieci (nazwa)

const char pass [ ] = "Hasło routera" ;       // twoje hasło sieciowe

const int touch\_threshold = 40 ;  // wejście touchpin (T0) jest GPIO4

const pływak steps\_per\_degree = XXL Standard 4096 / 360 ; // 4096 kroków na obrót w trybie pół kroku

double my\_latitude = 51,89970 ; // zmień tutaj swoją lokalizację

double my\_longitude = - 2.12084 ; // znajdź swoje współrzędne za pomocą google maString next\_pass\_JSON; // dane o lokalizacji są pobierane w formacie JSON

int next\_risetime ;

int next\_duration ;

U8G2\_SSD1306\_128X32\_UNIVISION\_F\_HW\_I2C u8g2 ( U8G2\_R0 ) ;

// Definicje pinów silnika w numerach GPIO

#define motor\_pin\_az1  13     // IN1 w sterowniku ULN2003

#define motor\_pin\_az2  12     // IN2 w sterowniku ULN2003

#define motor\_pin\_az3  14     // IN3 w sterowniku ULN2003

#define motor\_pin\_az4  27     // IN4 na Sterownik ULN2003

# zdefiniować motor\_pin\_el1  26     // IN1 na sterowniku ULN2003

# zdefiniować motor\_pin\_el2  25     // IN2 na sterowniku ULN2003

# zdefiniować motor\_pin\_el3  33     // IN3 na sterowniku ULN2003

# zdefiniować motor\_pin\_el4  32     // IN4 na sterowniku ULN2003

#define MotorInterfaceType 8 // 8 = tryb pół kroku, 4 = tryb pełnego kroku

// Inicjalizacja za pomocą sekwencji pinów IN1-IN3-IN2-IN4 do korzystania z biblioteki AccelStepper z silnikiem krokowym 28BYJ-48:

AccelStepper azimuth\_motor = AccelStepper ( MotorInterfaceType , motor\_pin\_az1 , motor\_pin\_az3 , motor\_pin\_az2 , motor\_pin\_az4 ) ;

AccelStepper elevation\_motor = AccelStepper ( MotorInterfaceType , motor\_pin\_el1 , motor\_pin\_el3 , motor\_pin\_el2 , motor\_pin\_el4 ) ;

Strefa czasowa my\_timezone ;

// użyj internetu, aby sprawdzić czas następnego przelotu

void get\_next\_pass ( double lat , double lon , int & p\_risetime , int & p\_duration ) {

 HTTPClient http ;

 const char \* api\_host = "http://api.open-notify.org/iss-pass.json?lat=" ;

 char my\_lat\_position [ 10 ] ;

 char my\_lon\_position [ 10 ] ;

 char api\_address [ 255] ;

 //

 zamień współrzędne zapisane jako podwójne na łańcuchy z dokładnością do 4 dp dtostrf ( lat , 7 , 4 , my\_lat\_position ) ;

 dtostrf ( lon , 7 , 4 , my\_lon\_position ) ;

 // utwórz adres URL za pomocą polecenia get

 strcpy ( api\_address , api\_host ) ;

 strcat ( api\_address , my\_lat\_position ) ;

 strcat ( api\_address , "& lon =") ;

 strcat ( api\_address , my\_lon\_position ) ;

 // pobierz tylko czas i datę następnego przebiegu

 strcat ( api\_address , "& n = 1" ) ;

 http . begin ( api\_address ) ;

 int httpCode = http . GET ( ) ;

 if ( httpCode > 0 ) {

   String payload = http . getString ( ) ;

   // Serial.println (httpCode);

   // Serial.println (ładunek);

   const **size\_t** capacity = JSON\_OBJECT\_SIZE ( 2 ) + JSON\_OBJECT\_SIZE ( 3 ) + 200;

   DynamicJsonDocument doc ( pojemność ) ;

   DeserializationError error = deserializeJson ( doc , payload ) ;

   p\_duration = doc [ "odpowiedź" ] [ 0 ] [ "czas trwania" ] ; // zapisz tę wartość do next\_duration

   p\_risetime = doc [ "response" ] [ 0 ] [ "risetime" ] ; // zapisz tę wartość do next\_risetime

 }

 else {

**Serial** . println ( "Błąd żądania HTTP" ) ;

 }

 http . koniec ( ) ; // Zwolnij zasoby

}

// przekonwertuj to do postaci czytelnej dla człowieka w dniach, godzinach, minutach i sekundach

String time\_to\_next\_flyby ( int next\_observation ) {

 int time\_remaining ;

 int next\_rise\_days ;

 int next\_rise\_hours ;

 int next\_rise\_minutes ;

 int next\_rise\_seconds ;

 Strunowy countdown\_time ;

 time\_remaining = next\_observation - teraz ( ) ;

 next\_rise\_days = time\_remaining / ( 3600 \* 24 ) ;

 time\_remaining % = ( 3600 \* 24 ) ;

 next\_rise\_hours = time\_remaining / 3600 ;

 time\_remaining % = 3600 ;

 next\_rise\_minutes = time\_remaining / 60 ;

 next\_rise\_seconds = time\_remaining 60 % ;

 countdown\_time + = next\_rise\_days ;

 countdown\_time + = "dni" ;

 countdown\_time + = next\_rise\_hours ;

 if ( next\_rise\_minutes < 10 ) {

   countdown\_time + = ": 0" ; } // dodaj wiodące zero dla pojedynczej cyfry

 else {

   countdown\_time + = ":" ;

 }

 countdown\_time + = next\_rise\_minutes ;

 if ( next\_rise\_seconds < 10 ) {

   countdown\_time + = ": 0" ; } // dodaj wiodące zero dla pojedynczej cyfry

 else {

   countdown\_time + = ":" ;

 }

 countdown\_time + = next\_rise\_seconds ;

 return countdown\_time ;

}

// znajdź aktualne położenie ISS nad Ziemią

// i oblicz kąty, aby wskazać na nią z naszej lokalizacji

void point\_to\_ISS ( ) {

 HTTPClient http ;

 podwójna szerokość geograficzna ;

 podwójna długość geograficzna ;

 if ( http . begin ( "http://api.open-notify.org/iss-now.json" ) ) {

   int httpCode = http . GET ( ) ;

   // httpCode będzie ujemny, jeśli wystąpi błąd,

   jeśli ( httpCode > 0 ) {

     String ISS\_Location\_Data = http . getString ( ) ;

     //Serial.println(httpCode); // to będzie 200 dla pomyślnego pobrania

     //Serial.println(ISS\_Location\_Data); // dane Json

     const **size\_t** capacity = JSON\_OBJECT\_SIZE ( 2 ) + JSON\_OBJECT\_SIZE ( 3 ) + 200 ;

     DynamicJsonDocument doc ( pojemność ) ;

     DeserializationError error = deserializeJson ( doc, ISS\_Location\_Data ) ;

     // const char \* message = doc ["wiadomość"];

     long timestamp = doc [ "timestamp" ] ;

     latitude = doc [ "iss\_position" ] [ "latitude" ] ;

     longitude = doc [ "iss\_position" ] [ "longitude" ] ;

     // Wydrukuj wartości przydatne do debugowania, aby sprawdzić, czy działa

     //Serial.println(timestamp);

     //Serial.println(latitude, 4);

     //Serial.println(longitude, 4);

   }

   else {

**Serial** . print ( "Błąd HTTP:" ) ;

**Szeregowy** . println ( http . errorToString ( httpCode ) . c\_str ( ) ) ;

   }

   http . koniec ( ) ; // Zwolnij zasoby

 }

 else {

**Serial** . wydrukować( „Błąd HTTP: nie można się połączyć” ) ;

 }

 // ESP32 oblicza kąty w radianach (nie w stopniach)

 double latitude\_in\_radians = latitude \* PI / 180 ;

 double longitude\_in\_radians = longitude \* PI / 180 ;

 double my\_latitude\_in\_radians = my\_latitude \* PI / 180 ;

 double my\_longitude\_in\_radians = my\_longitude \* PI / 180 ;

 // oblicz kierunek azymutu na ISS

 double x = cos (latitude\_in\_radians ) \* sin ( longitude\_in\_radians - my\_longitude\_in\_radians ) ;

 double y = cos ( my\_latitude\_in\_radians ) \* sin ( latitude\_in\_radians ) - sin ( my\_latitude\_in\_radians ) \* cos ( latitude\_in\_radians ) \* cos ( longitude\_in\_radians - my\_longitude\_in\_radians ) ;

 podwójne b = atan2 ( x, y ) ;

 //Serial.print("Bearing to ISS: ");

 b = b \* 180 / PI ; // konwersja na stopnie

 //Serial.println(b);

// obliczyć namiar na wysokość do ISS

 double earth\_radius = 6371 ; // średnia wartość w km

 double orbit = 412 ; // średnia wartość w km - może sprawdź też tę wartość?

 double x1 = earth\_radius \* cos ( my\_latitude\_in\_radians ) \* cos ( my\_longitude\_in\_radians ) ;

 double y1 = earth\_radius \* cos ( my\_latitude\_in\_radians ) \* sin ( my\_longitude\_in\_radians );

 double z1 = earth\_radius \* sin ( my\_latitude\_in\_radians ) ;

 double x2 = earth\_radius \* cos ( latitude\_in\_radians ) \* cos ( longitude\_in\_radians ) ;

 double y2 = earth\_radius \* cos ( latitude\_in\_radians ) \* sin ( longitude\_in\_radians ) ;

 double z2 = earth\_radius \* sin (latitude\_in\_radians ) ;

 double dist\_to\_nadir = sqrt ( sq ( x2 - x1 ) + sq ( y2 - y1 ) + sq ( z2 - z1 ) ) ;

 //Serial.print("Distance to nadir: ");

 //Serial.println(dist\_to\_nadir);

 podwójne geocentric\_angle\_in\_radians = 2 \* asin ( dist\_to\_nadir / ( 2 \* earth\_radius ) ) ;

 //Serial.print("Geocentric angle w stopniach: ");

 //Serial.println(geocentric\_angle\_in\_radians \* 180 / PI);

 podwójna odległość\_do\_ISS = sqrt ( sq ( orbita + promień\_ziemia ) + sq ( promień\_ziemi ) - 2 \* ( orbita + promień\_ziemi ) \* promień\_ziemi \* cos ( geocentric\_ang\_in\_radians ) ) ;

 //Serial.print("Distance to ISS: ");

 //Serial.println(distance\_to\_ISS);

 double ISS\_angle = ( 180 / PI ) \* asin ( ( orbit + earth\_radius ) \* ( sin ( geocentric\_angle\_in\_radians ) / distance\_to\_ISS ) ) ;

 // azymutalny silnik krokowy ma 4096 pół kroków na obrót i przełożenie 113/22

 azymut\_motor . moveTo ( XXL Standard 4096 / 360 \* 113 / 22 \* b ) ;

 // silnik elewacji ma 4096 pół kroków na obrót

 Elevation\_motor . moveTo ( XXL Standard 4096 / 360 \* ISS\_angle ) ;

 // uruchom silnik, czekając na aktualizację współrzędnych co 5 sekund

 double holding\_time = now ( ) ;

 while ( now ( ) - holding\_time < 5 ) {

   azymuth\_motor . run ( ) ;

   Elevation\_motor . run ( ) ;

   wydajność ( ) ;

 }

}

// wydrukuj wiadomości na OLED używając dwóch wierszy tekstu

void display\_message ( String line\_one , String line\_two ) {

 u8g2 . setFont ( u8g2\_font\_8x13\_tr ) ;

 //u8g2.setFont(u8g2\_font\_fur14\_tr);

 u8g2 . firstPage ( ) ;

 u8g2 . setCursor ( 0 , 15 ) ;

 u8g2 . print ( line\_one ) ;

 u8g2 . setCursor ( 0 ; 31 ) ;

 u8g2 . print ( line\_two ) ;

 u8g2 . nextPage ( ) ;

}

void setup ( ) {

**Serial** . rozpocząć ( 115200 ) ;

 opóźnienie ( 3000 ) ; // spróbuj tego, aby umożliwić OLED nawiązanie komunikacji?

 u8g2 . begin ( ) ; // dla OLED

 u8g2 . enableUTF8Print ( ) ; //

 // Ustaw maksymalne kroki silnika na sekundę:

 azymuth\_motor . setMaxSpeed ( 800 ) ; // duże przełożenie, więc przyspiesz to

 azymuth\_motor . setAcceleration( 100,0 ) ;

 Elevation\_motor . setMaxSpeed ( 400 ) ;

 Elevation\_motor . setAcceleration ( 100,0 ) ;

// ######################### sekcja kalibracji ##################### ##########

 display\_message ( "Strzałka zatrzymaj, gdy" , "skierowana w dół" ) ;

 while ( touchRead ( T0 ) > touch\_threshold ) {

   elevation\_motor . ruch ( 400 ) ;

   Elevation\_motor . run ( ) ;

 }

 // przycisk został naciśnięty, gdy wskaźnik elewacji jest skierowany w dół

 display\_message ( "Silnik podniesienia" , "ustaw na zero" ) ;

 Elevation\_motor . setCurrentPosition ( 0 ) ;

 opóźnienie ( 1000 ) ;

 display\_message ( "Stop strzałka na" , "lewa strona" ) ;

 while ( touchRead ( T0 ) > touch\_threshold ) {

   azymut\_motor . ruch ( 400 ) ;

   azimuth\_motor . run ( ) ;

 }

 // przycisk jest naciśnięty, gdy wskaźnik azymutu jest skierowany do przodu

 display\_message („Silnik azymutu” , „ustawiony na zero” ) ;

 opóźnienie ( 2000 ) ; // pauza, aby poinformować, że został naciśnięty przycisk

 azimuth\_motor . setCurrentPosition ( 0 ) ;

 // przesuń elewację, aby wskazywała na Polaris

 // ten kąt jest taki sam jak szerokość geograficzna obserwatora, punkt odniesienia w dół, więc +90

 elevation\_motor . moveTo ( steps\_per\_degree \* ( my\_latitude + 90 ) ) ;

 Elevation\_motor . runToPosition ( ) ;

 display\_message ( "Wskaż Polaris" , "Następnie naciśnij przycisk" ) ;

 while ( touchRead ( T0 ) > touch\_threshold ) { } // czekaj na naciśnięcie przycisku

 display\_message ( "Teraz skalibrowano" , "" ) ;

 opóźnienie ( 2000 ) ; // pauza, aby powiadomić cię o naciśnięciu przycisku

 // ######################## koniec sekcji kalibracji ######## ####################

 // ######################### Połącz się z WiFi #################### ###################

**Serial** . println ( "Śledzenie ISS" ) ;

 display\_message ( "Wyszukiwanie" , ssid ) ;

**Szeregowy** . println ( ssid ) ;

**WiFi** . begin ( ssid , pass ) ;

 while ( **WiFi** . status ( ) ! = WL\_CONNECTED ) {

   opóźnienie ( 500 ) ;

**Szeregowy** . print ( „.” ) ;

 }

**Seryjny** . println ( "Połączono z Wi-Fi" ) ;

 display\_message ( "Połączono z" , ssid ) ;

 // ######################### Synchronizuj czas z UTC ################### ############

 // Odkomentuj poniższy wiersz, aby zobaczyć, co robi za kulisami

 setDebug ( INFO ) ;

 waitForSync ( ) ; // nie rób nic, dopóki nie nadejdzie właściwy czas

**Szeregowy** . println ( "Bieżący czas UTC:" + UTC . dateTime ( ) ) ;

 my\_timezone . setLocation ( F ( "GB" ) ) ; // <a href=" https://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones ">   https://en.wikipedia.org/wiki/List\_of\_tz\_database ...>

 opóźnienie ( 3000 ) ;

**Szeregowy** . print ( F ( "Czas lokalny:" ) ) ;

**Szeregowy** .println ( my\_timezone . dateTime ( ) ) ;

}

void loop ( ) {

 events ( ) ; // dzięki temu silniki będą się poruszać

 //Serial.println("COOKIE: "+ UTC.dateTime (COOKIE));

 // Serial.println (dateTime (next\_pass\_timestamp, "D dM H: i: s"));

 if ( now ( ) + next\_duration > next\_risetime ) { // następny przelot tylko wtedy, gdy obecny

   minie get\_next\_pass ( my\_latitude , my\_longitude, next\_risetime , next\_duration ) ;

 }

**Seryjny** . print ( "Następny przelot:" ) ;

 //Serial.print(dateTime(next\_risetime, "D dM H: i: s"));

**Szeregowy** . println ( time\_to\_next\_flyby ( next\_risetime ) ) ;

 point\_to\_ISS ( ) ;

 opóźnienie ( 1000 ) ;

 //display\_message(UTC.dateTime("H:i:s T "), my\_timezone.dateTime (" H: i: s "));

 display\_message ( UTC . dateTime ( "H: i: s T") , time\_to\_next\_flyby ( next\_risetime ) ) ;

}