Univerzitet u Sarajevu
Elektrotehnički fakultet
Ugradbeni sistemi 2023/24

Dokumentacija implementacije

PlanetTracker

Članovi grupe:
Benjamin Hadžihasanović, 19319
Faruk Zahiragić, 19382
Edin Živojević, 19314

Uvod

Unutar priloženog koda se nalaze neki drajveri koje smo koristili za ekran i senzor. Radi transparentnosti, kodovi koje smo mi razvili se nalaze u fajlovima:

- *server.py* - za udaljeni uređaj
- *main.py* - glavni program
- *mqtt_connection.py* - za spajanje sa internetom i mqtt-om
- *image_holder.py* - za učitavanje slika planeta
- *orientation.py* - orijentacija uređaja
- *rotary.py* - upravlja enkoderom i ledicama
- *data.py* - sadrži neke konstante glavnog programa

Također smo unutar drajvera za magnetometar dodali funkciju za klaibraciju i offsete kao privatne atribute.

Implementaciju projekta možemo podijeliti u sljedeće etape:

- Dobavljanje koordinata planeta
- Orijentacija uređaja
- Korisnički interfejs
- Glavni program

U nastavku ćemo objasniti pojedine etape projekta.

Dobavljanje koordinata planeta

Za dobavljanje koordinata nam je potreban dodatni uređaj koji će služiti kao *server*. Za razmjenu podataka koristimo MQTT, a kao broker korisitmo javni EMQX broker. Koriste se dvije teme *planet_tracker/request* i *planet_tracker/response*. Udaljeni uređaj je pretplaćen na *request* podtemu i čeka zahtjev od korisničkog uređaja, te šalje poruku na podtemu *response/ime_planete*. Korsinički uređaj je ptretplaćen na ovu temu i prima poruku u json formatu. Naprimjer, neka je korisnik odabrao Merkur, predstavljen kao 0 u listi planeta. Tada korisnik šalje poruku *planet_tracker/request* sa porukom "0". Server prima pruku, te korisiti python biblioteku PyEphem da odredi koordinate planete te šalje korisniku poruku na temu *planet_tracker/response/mercury* u formatu:

$$\{'az' : \{azimuth\_u\_stepenima\}, 'alt' : \{altituda\_u\_stepenima\}\}$$

Kod koji se izvršava na udaljenom računaru se nalazi u fajlu *server.py* i trenutno se izvršava na našem Raspberry Pi računaru kao servis. Korisnički uređaj koristi klasu MQTTCONN za slanje i primanje poruka, a tokom inicijalizacije vrši povezivanje na internet i broker.

Orijentacija uređaja

Za orijentaciju koristimo BMX055 senzor, koji u sebi sadrži tri senzora žiroskop, akcelerometar i magnetometar. Sa senzorom komuniciramo preko I2C, gdje svaki senzor ima vlastitu adresu. Adrese senzora se mogu postaviti na osnovu pinova SDO1, SDO2, CSB3. Mi koristimo standardne adrese, koje se dobivaju povezivanjem pomenutih pinova na uzemljenje. Podatke sa senzora čitamo pomoću drajvera *bma2x2.py* i *bmm150.py* napisanih u micropythonu. Koristimo akcelerometar i magnetometar. Instance objekata ovih drajvera koristimo u klasi Orientation. U ovoj klasi je sadržana sva logika određivanja orijentacije uređaja u 3D prostoru. Za ovo se obično koriste *fusion algoritmi* poput *Madgwick* ili *AHRS* algoritma. Zbog ograničenog vremena na izvedbi projekta smo koristili intuitivniji pristup. Naime, preko senzora dobijamo vektore gravitacionog ubrzanja i magnetne sile u koordinatnom sistemu uređaja. Nas zanima kako je uređaj usmjeren u odnosu na sjever, tačnije kako je usmjerena Z osa koja izlazi iz ekrana. Za to je potrebno odrediti karakteristične vektore *našeg* koordinatnog sistema, istok-sjever. Pošto magnetna sila ne odgovara tačno sjevernom vektoru, jer je usmjereno više dolje, prema zemlji, potrebno je obaviti par vektorskih proizvoda. Naime, vektorski proizvod vektora koji je usmjeren dolje, prema zemlji (ustvari -g) i magnetnog vektora sigurno daje vektor koji pokazuje prema istoku. Sada vektorskim proizvodom istoka i vektora *dolje*, dobijamo sjeverni vektor. Kako bi olakšali računanje azimuta, pretpostavljamo da je zabranjena rotacija uređaja oko Z ose (izlazi iz ekrana). Sada možemo odrediti ugao u kojem je uređaj usmjeren pomoću skalarnog proizvoda Y ose uređaja (izlazi iz vrha uređaja, i leži u sjever-istok ravni). Kako inverzna funkcija kosinusa vraća isključivo vrijednosti iz intervala $[0 - PI]$, a nas zanima vrijednost ugla od sjevera do Y ose uređaja u intervalu $[0 - 2PI]$, dodali smo provjeru da li se vektorski proizvod ova dva vektora poklapa sa vektorom gravitacije, ukoliko se poklapa onda je nova ispravljena vrijednost $2PI - azimuth$. Potrebno je još dodati 90 stepeni na dobijeni rezultat jer nas zanima u kojem pravcu korisnik gleda.

```
# Funkcija koja vraca istocni vektor
def _get_east(self, D):
    return self._cross_product(D, self.mag_vals)

# Funkcija koja vraca sjeverni vektor
def _get_north(self):
    D = [ -item for item in self.acc_vals ]
    return self._cross_product(self._get_east(D), D)

# Funkcija koja vraca azimut
def get_azimuth(self):
    N = self._get_north()

    azimuth = math.acos(self._dot_product(N, self.Y))
    cross = self._cross_product(N, self.Y)

    if self._dot_product(cross, self.acc_vals) > 0:
        azimuth = self.TWOPI - azimuth

    azimuth = (azimuth + self.PI2) % self.TWOPI

    return azimuth
```

Nagib uređaja lahko nalazimo jer je to zapravo $asin(g_z)$, gdje g_z Z komponenta gravitacionog ubrzanja.

U ovoj klasi također imamo tajmer koji se može incijalizovati ili deinicijalizovati da periodično čita vrijednosti sa senzora i ažurira trenutne vrijednost na način da ih prvo filtrira, tako što uzima u obzir prijašnje vrijednosti:

```
# Funkcija koja implementira low-pass filter
def _low_pass_filter(self, old_vector, new_vector, ratio):
    return [
        old_vector[i] * (1 - ratio) + new_vector[i] * ratio for i
    in range(3)
    ]

# Funkcija koja azurira vrijednosti senzora
def _update_vals(self, t):
    new_acc = self.acc.xyz()
    new_mag = self.mag.xyz()

    self.acc_vals = self._low_pass_filter(self.acc_vals, new_acc,
    self.acc_trust)
    self.mag_vals = self._low_pass_filter(self.mag_vals, new_mag,
    self.mag_trust)
```

Gdje su *mag_trust* i *acc_trust* privatne varijable koje je moguće postaviti sa odgovarajućim metodama i koje pretstavljaju povjerenje u nove vrijednosti senzora.

Na kraju preostaje da se odredi u kojem smjeru se treba uređaj kretati i udaljenost između ciljne tačke i trenutnog položaja uređaja, što rješavamo sa dosta trigonometrije:

```
# Funkcija koja odredjuje ugao i distancu izmedju trenutne i ciljne tacke
def get_directions(self, current_azimuth, current_altitude, target_azimuth, target_altitude):
    x_delta = target_azimuth - current_azimuth
    x_delta = (x_delta + self.PI) % self.TWOPI - self.PI
    y_delta = target_altitude - current_altitude

    heading_angle = math.atan2(y_delta, x_delta)

    distance_angle = math.acos(math.cos(target_altitude) * math.cos(current_altitude) * math.cos(target_azimuth - current_azimuth) + math.sin(target_altitude) * math.sin(current_altitude))

    return heading_angle, distance_angle
```

Korisnički interfejs

Korisnički interfejs se sastoji od ledica i ekrana. Ledice označavaju trenutno izabranu planetu i njima u potpunosti upravlja enkoder pomoću klase Rotary, koja prima broj pinova na koje je priključen enkoder i handler funkcije koja se poziva prilikom klika na enkoder. Klasa kao privatni atribut sadrži trenutno odabranu ledicu, tj., cijeli broj u opsegu $[0-7]$, što ustvari pretstavlja trenutno izabranu planetu. Rotacijom enkodera se bavi privatna funkcija klase, rotary_encoder_handler(), koja na osnovu okretanja enkodera mijenja trenutni broj/ledicu/planetu. Odabrana ledica, odnosno cijeli broj se dobavlja odgovarajućom metodom. Klasa također sadrži metode za aktivaciju i deaktivaciju handlera, jer se u nekim modovima programa ne koriste.

Ekran upravljamo pomoću biblioteke LVGL sa micropython bindingom. Izdvojili smo logiku učitavanja i upravljanja pozicijom slika planeta, koje pokazuju u kojem smjeru se uređaj treba zakrenuti, u zasebnu klasu Image_holder, koja prima objekat ekrana, kao atribut na kojem se slika treba učitati. Pozicija slike u centru ekrana se postavlja na osnovu vrijednosti dobivenih iz funkcije Orientation.get_directions(). Centar ekrana ima vrijednosti 0, 0, tako da je potrebno samo postaviti x i y koordinate slike na osnovu proslijeđenog ugla i udaljenosti od centra. Maksimalna udaljenost od centra u pixelima je 100, jer je poluprečnik centralnog kruga 100 piksela.

```
# Pomijeranje slike na osnovu ugla i distance u odnosu na centar
def move_image(self, rad, distance):
    if distance > 100:
        distance = 100

    x = int(distance * math.cos(rad))
    y = int(distance * math.sin(rad))

    self.image.set_pos(x, y)
```

Inicijalizaciju ekrana i odgovarajućih prikaza obavljamo u glavnom programu.

Glavni program

U glavnom programu vršimo inicijalizaciju odgovarajućih klasa. Kreiramo I2C za senzore i SPI za displej, koji se proslijeđuje ili9341 drajveru, kojim upravlja LVGL. Zatim kreiramo početni ekran koji sadrži labelu, koja na početku informiše korisnika da se uređaj povezuje na internet i broker. Nakon toga se pojavljuje prikaz da se odabere planeta pomoću enkodera. Za inicijalizaciju enkodera je potreban i handler koji se poziva na klik enkodera. Ova funkcija provjerava u kojem modu je program. Program ima dva moda. Tokom početnog moda se bira planeta, te je rotacija enkodera omogućena. Ukoliko se klikne enkoder u početnom modu, deaktivira se rotacija enkodera, postavlja se izabrana planeta, učitava slika planete i drugi ekran, dobavljaju koordinate, te pokreću tajmeri za čitanje senzora i ažuriranje ekrana. Ažuriranje ekrana obavlja funkcija update_screen(), a koju smo postavili da se obavlja svakih 300ms. Ova funkcija koristi instancu Orientation klase za određivanje orijentacije uređaja, te na osnovu njih poziva move_image() nad učitanom slikom i ažurira labele sa koordinatama. Ukoliko se ponovo klikne enkoder, program se vraća u početni mod. Pored toga, omogućeno je i vršenje kalibracije magnetometra pritiskom na taster 4, korištenjem prekida i prekidne rutine start_calibration(). Ukoliko je program u drugom modu, prvo se učitava početni ekran i deinicijaliziraju odgovarajući objekti te se zatim obavještava korisnik o kalibraciji. Kalibracija prikuplja 600 uzoraka sa dilejem 0.1s. Dakle traje 1min. Za lakšu kalibraciju smo dodali metodu unutar drajvera za magnetometar koja vrši *Hard Iron* kalibraciju, koja kompenzuje moguća magnetna polja koja kvare rezultate tako što postavlja offsete, koji se oduzimaju od pročitanih vrijednosti senzora. Ne vršimo *Soft Iron* kalibraciju, tako da može doći do izobličenja podataka uslijed nekih metalnih objekata koji se mogu naći blizu senzora. Senzor inače ima već postavljene offsete koji bi trebali biti ispravni za fizičku konfiguraciju senzora kao u priloženoj video demonstraciji.