Röviden a robot használatáról

Ebben a doksiban röviden megpróbálom összefoglalni a szükséges tudnivalókat a robotról és a programozásáról, amikre szükségetek lesz a tanfolyamunk során.

A robot részei, felépítése

A robot váza 3D nyomtatott, a hajtásáért 2db motor + kerék felel. A motorokon vannak enkóderek, amikkel így lehetőség van lemérni a kerekek pontos elfordulását.

Érzékelők terén a robot rendelkezik egy színérzékelővel (az alján), egy infrás távolságérzékelővel és egy IMU-val (giroszkóp + gyorsulásmérő). A távolságérzékelő egy szervó segétségével elfordítható jobbra és balra, így a robot tud "*nézelődni*".

A vezérlésért egy STM32-es mikrovezérlő felel - ez a robot "*szíve*". A telemetria adatokat egy ESP8266 küldi és fogadja.

A robot programozása

A munkátok megkönnyítése érdekében egy olyan szoftveres környezetet alakítottunk ki nektek, ami elrejti szinte teljesen az alacsony szintű hardveres dolgokat, számításokat és lekérdezéseket - így gyakorlatilag klasszikus C-ben, a robot API-nk függvényei segítségével tudtok zökkenőmentesen fejleszteni.

Az `app_main.c` fájlba kell dolgoznotok, ezt szabadon módosíthatjátok.

Mozgással kapcsolatos függvények

- `int setMotorSpeed(uint8_t mot_lr, float speed)` - beállítja az adott motor sebességét
  - *mot_lr* - lehet MOT_L (bal motor) vagy MOT_R (jobb motor)
  - *speed* - a kívánt sebesség a [-100; 100] intervallumon - **ne** vigyétek tartósan 40 fölé
  - *return* - siker esetén 0-t ad vissza
- `int getEncoderPosition(uint8_t mot_lr)` - lekéri az adott motorhoz tartozó enkóder abszolút pozícióját (elfordulását)
  - *mot_lr* - lehet MOT_L (bal motor) vagy MOT_R (jobb motor)
  - *return* - az enkóder abszolút pozíciója (int32_t értelmezési tartományon)
- `void setServoPosition(int8_t position)` - beállítja a távolságérzékelős szervó pozícióját
  - *position* - a szervó kívánt helyzete [-90°; +90°]

Érzékelőkkel kapcsolatos függvények

- `void getColorHsv(Color* color)` - ezzel lehet kiolvasni az éppen érzékelt színt a HSV formátumban (lásd erről szóló előadás dia)
  - *color* - pointer egy `Color` típusú struct-ra, amibe a kiolvasott érték fog kerülni

- `uint16_t getIrDistance()` - visszaadja az infravörös érzékelő által mért távolságot, kb. 30Hz a frissítési frekvenciája
    - *return* - a legutóbb olvasott távolság mm-ben
- `void getAccData(Vec3* acc)` - visszaadja a legutóbb olvasott gyorsulási értékeket
    - *acc* - pointer egy `Vec3` típusú struct-ra, amibe a kiolvasott értékek kerülni fognak. A gyorsulási értékek *g*-ben vannak megadva
- `void getGyroData(Vec3* gyro)` - visszaadja a legutóbb olvasott giroszkópos értékeket
    - *gyro* - pointer egy `Vec3` típusú struct-ra, amibe a kiolvasott értékek kerülni fognak. A giroszkópos értékek *°/s*-ben vannak megadva
- `void getMagData(Vec3* mag)` - visszaadja a legutóbb olvasott iránytű értékeket
    - *mag* - pointer egy `Vec3` típusú struct-ra, amibe a kiolvasott értékek kerülni fognak. Az iránytű értékek *uT*-ben vannak megadva
- `void getOrientation(Orientation* orientation)` - a robot a háttérben az IMU adatai alapján nyilván tartja az aktuális abszolút orientációját (dőlésszögét az x és y tengelyek mentén)
    - *orientation* - pointer egy `Orientation` típusú struct-ra, amibe az aktuális orientáció kerül (pitch és roll °-ban)
- `float getTemp()` - visszaadja a legutóbb olvasott hőmérsékletet
    - *return* - a legutóbb olvasott hőmérséklet °C-ban

Kommunikációval kapcsolatos függvények

- `int lcdPrintf(uint8_t row, uint8_t col, const char *fmt, ...)` - kiír egy formázott szöveget a kijelzőre
    - *row* - a sor, ahova kiírja a szöveget (0 vagy 1)
    - *col* - az oszlop, ahova kiírja a szöveget (0-tól 15-ig)
    - *fmt* - a formázott szöveg, mint a `printf` esetében
    - ... - a formázott szöveg paraméterei
    - *return* - a kiírt karakterek száma
- `int uartPrintf(const char *fmt, ...)` - kiír egy formázott szöveget a soros portra
    - *fmt* - a formázott szöveg, mint a `printf` esetében
    - ... - a formázott szöveg paraméterei
    - *return* - siker esetén 0-t ad vissza
- `int espPrintf(const char *fmt, ...)` - kiír egy formázott szöveget a webes telemetria felületre
    - *fmt* - a formázott szöveg, mint a `printf` esetében
    - ... - a formázott szöveg paraméterei
    - *return* - siker esetén 0-t ad vissza

- `int espRead(char* data)` - beolvassa a webes telemetria felületről a legutóbbi üzenetet
    - *data* - pointer egy karaktertömbre, amibe az üzenet kerül, amennyiben több üzenet is felhalmozódott, akkor ezek egy-egy `\n` karakterrel lesznek elválasztva
    - *return* - ha van új üzenet, akkor 1-et ad vissza, egyébként 0-t

Időzítéssel kapcsolatos függvények

- `void delayMs(uint32_t delay)` - blokkol a megadott időtartamig
    - *delay* - a blokkolás időtartam ms-ben
- `void delayUs(uint32_t delay)` - blokkol a megadott időtartamig. **Figyelem!** A függvény a jelenlegi implementációból adódóan legfeljebb kb. 18000us (18ms) blokkolásra képes, ennél hosszabb időtartam esetén a függvény nem garantálja a pontos blokkolási időt.
    - *delay* - a blokkolás időtartam us-ben
- `uint32_t getTimeMs()` - visszaadja a legutóbbi reset óta eltelt időt ms-ben
    - *return* - az eltelt idő ms-ben

A függvények lefutási ideje

Arra érdemes figyelni, hogy a függvények közül **egyik sem blokkol jelentősebb ideig**, az ezt jelenti, hogy az adott robot API függvény általában néhány microsec alatt lefut. Ez alól két kivétel van: a `delayMs` és a `delayUs` függvények.