
**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Komáromi Sándor

# WIRELESS KONTROLLER FEJLESZTÉSE ROBOTVEZÉRLÉSHEZ

KONZULENS

Nagy Ákos

BUDAPEST, 2023

# Tartalomjegyzék

# 1 Bevezetés

Ebben a dokumentumban a szeretném bemutatni egy három szegmenses robotkar összekapcsolását a ROS2 (Robot Operating System) rendszerrel. A robotkart a Simonyi Károly Szakkollégium Lego Köre fejleszti az INDACT projekt keretei között. A projekt célkitűzése az Iparban használt technológiák megismerése, valamint reprodukálása, hogy a benne részt vevő tagok jobban megismerhessék azokat. Így esett a választásom erre a témára is, hiszen a ROS2 tökéletes választás robotikai eszközök vezérlésére. Ugyan az iparban nem használják, viszont a különböző ipari fejlesztésekben annál inkább.

Ebben a dokumentumban első sorban azokat a technológiákat fogom bemutatni melyeket a munkám során használtam, valamint azoknak alkalmazását, tényleges felhasználását valódi rendszerekben. Továbbá bemutatom azon tényleges eszközöket, melyeket felhasználtam munkámhoz. A dokumentum elsősorban használati útmutató jellegű lesz, célja, hogy munkám reprodukálható legyen.

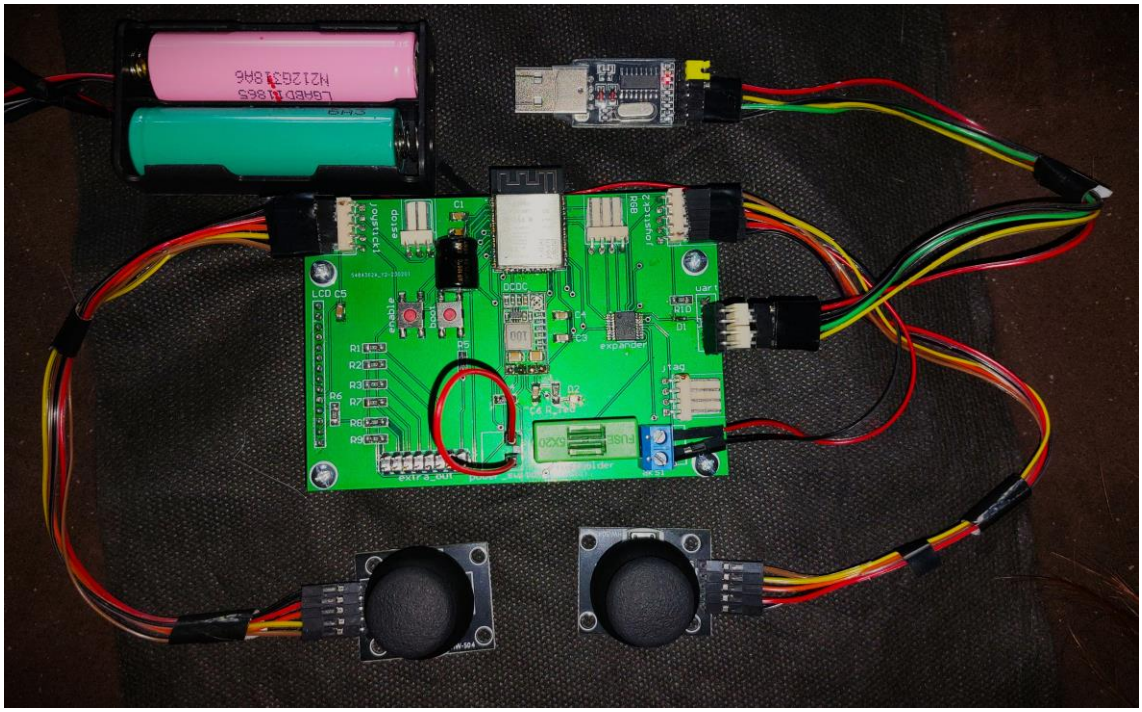
## 1.1 A megoldandó probléma

Feladatom a címben is olvasható wireless kontroller, valamint a fent említett robotkar összekötése, a kettő közötti kommunikáció megvalósítása úgy, hogy mind a kontroller által vezérelt robot kiváltható legyen másikkal, mind a robot vezérlése kiváltható legyen akár másik interfacel, vezérléssel. Ennek elérése érdekében választottam a ROS2-t hiszen, ez nemcsak a kommunikáció felállításában ad lehetőségeket, hanem a robot vezérlésében is. A dokumentum kizárólag a korábban már említett robotkarról szól, azonban használható más robottal való kapcsolat felépítésére is.

# 2 Használt eszközök

## 2.1 Wireless kontroller

A két eszköz közül, melyek közötti kapcsolat megteremtése a célom, az egyik elkészítése egy diplomaterv feladata, melyet Puskás Timea készít. Az ő feladata a kontroller hardveres és szoftveres elkészítése. A kontroller tartalmaz majd különféle joystikeket, gombokat, kapcsolókat, valamint egy E-stop gombot vészleállítás érdekében. Továbbá egy érintő kijelző további robot specifikus tulajdonságok elvégzéséhez. A kontroller minden tulajdonságát egy ESP32-es mikrokontroller vezérli, a kapcsolat felépítéséhez a vezérlő wifi modulját használjuk fel. A távirányító jelenlegi állapotát az 1. ábra mutatja.

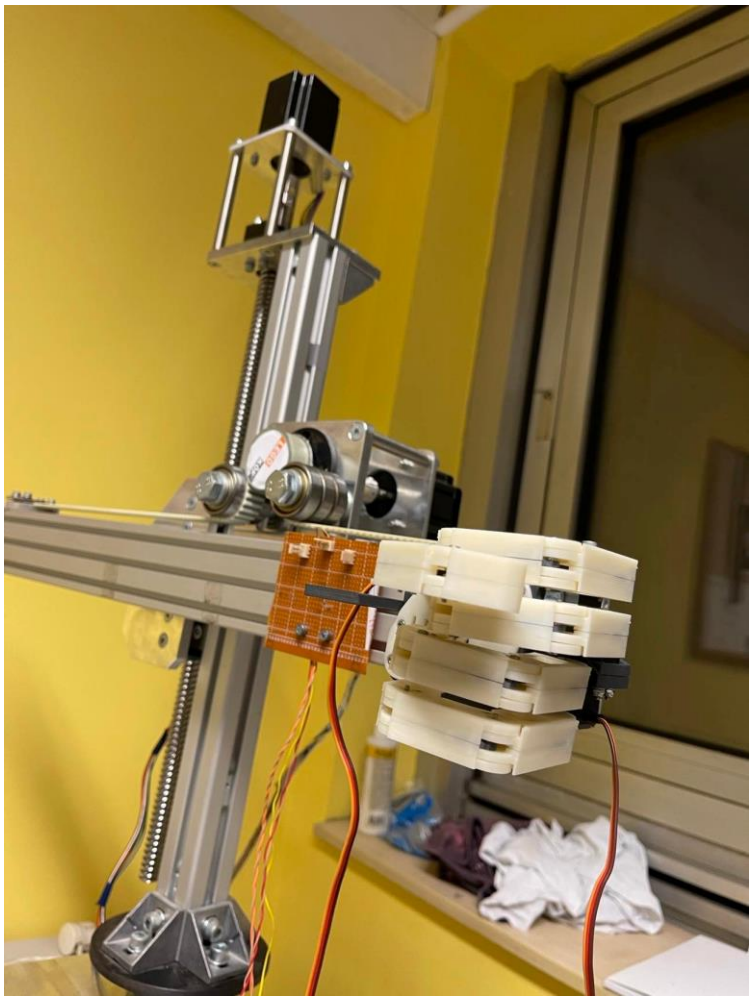


**1. ábra: A wireless kontroller jelenlegi állapota**

## 2.2 Robotkar

A vezérlendő robot egy 3 szegmenses RTT robotkar, azaz egy rotációs és két transzlációs csuklót tartalmaz. Henger koordináta rendszerben mozog, valamint léptetőmotorok hajtják. A motorok vezérlői PWM jelet várnak a vezérlésül, így a kiszámolt abszolút pozíciót még át kell számolni relatív elmozdulásba, majd azt PWM jelbe. Azonban a munkám kizárólag a kommunikáció megvalósítását érintette, így csak

az abszolút pozíció megadására törekedtem. A robotkar számítási kapacitását egy STM32f746-os mikrokontroller biztosítja, melynek megfelelő lábmennyisége, egyéb feladatok ellátására, valamint bővítéséi lehetőségekre alkalmas. A robotkarhoz egy megfogó is készült, későbbiekben ennek vezérlése is cél. A robotkarról készített fotót a 2. ábra mutatja.



**2. ábra: A vezérlendő robotkar**

# 3 A felhasznált eszközök

Az alap probléma megoldására több lehetőség is akad, ahogyan azt már említettem én a ROS2 mellett döntöttem. Azonban a ROS2 önmagában nem alkalmas mikrokontrollereken való futtatásra, a magas számítási igényei miatt. Ezen probléma megoldására alkalmaztam egy a ROS2 architektúrájába illő szoftvercsomagot, a MicroROS-t.

## 3.1 ROS2

A ROS2 egy szoftverplatform robotikai alkalmazások fejlesztésére, más névegy egy robotikai software developement kit (SDK). Széles körben támogatja a robotikai alkalmazások tárházát, az oktatástól, a kutatáson át, a terméktervezésig. Több különböző, de egymással összefüggő szoftver komponenst tartalmaz, melyek mind beleillenek egy közös tervezési mintába, valamint kompatibilisek egymással.

## 3.2 MicroROS

A MicroROS a ROS2 közösség által fejlesztett csomagjai közé tartozik, lefedve egy kritikus részét a palettának, a microcontrollereket. Fő feladata, hogy kapcsolatot teremtsen az adott linux rendszeren futó ROS2, valamint egyéb mikrokontrollerek között. Elrejti a ROS2 számára a kommunikációs hálózatot, így a mikrokontrolleren futó node úgy fog látszani mintha a linux rendszerben működne.

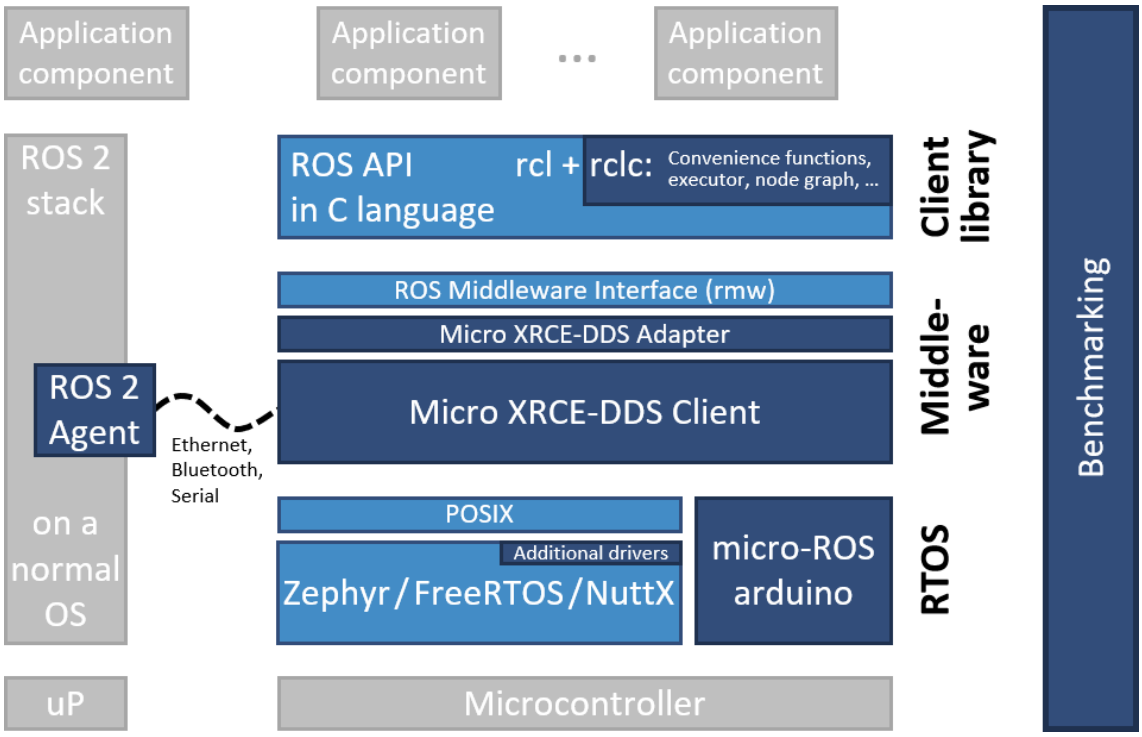
A kommunikáció felépítésére több lehetőséget is támogat a rendszer. Alapvetően három kommunikációs formához készült átfogó támogatás, azonban a kommunikációhoz szükséges függvényeket a felhasználó bármilyen saját rendszerhez hozzáillesztheti. A szoftvercsomag UART, USB, valamint TCP/IP kommunikációs formákat támogat.

A csomag alap célkitűzése az, hogy a platformfüggetlenül bármelyik mikrokontrollerre elérhetővé tegyen minden ROS2-be beépített funkcionalitást. Így alapjaiban az RCL (ROS Client Library) c nyelvű implementációját használták a csomag elkészítéséhez.

A MicroROS egy multiplatformú rendszert kínál, mely különböző fejlesztői környezethez, különböző telepítést biztosít, így elérve egy széleskörűen biztosított palettát.

### 3.2.1 MicroROS felépítése

A MicroROS egy kommunikációért felelős Middleware-t nyújt, mely elfedi a kommunikációs protokollt, erre épül fel a ROS c nyelven implementált Client Library-ja, amelyet használ a felhasználó. A számítógéppel, melyen a ROS2 fut ,az azon elindított Agent alkalmazás veszi fel a kapcsolatot, ugyancsak elrejtve minden kommunikációt a ROS többi nodeja számára.

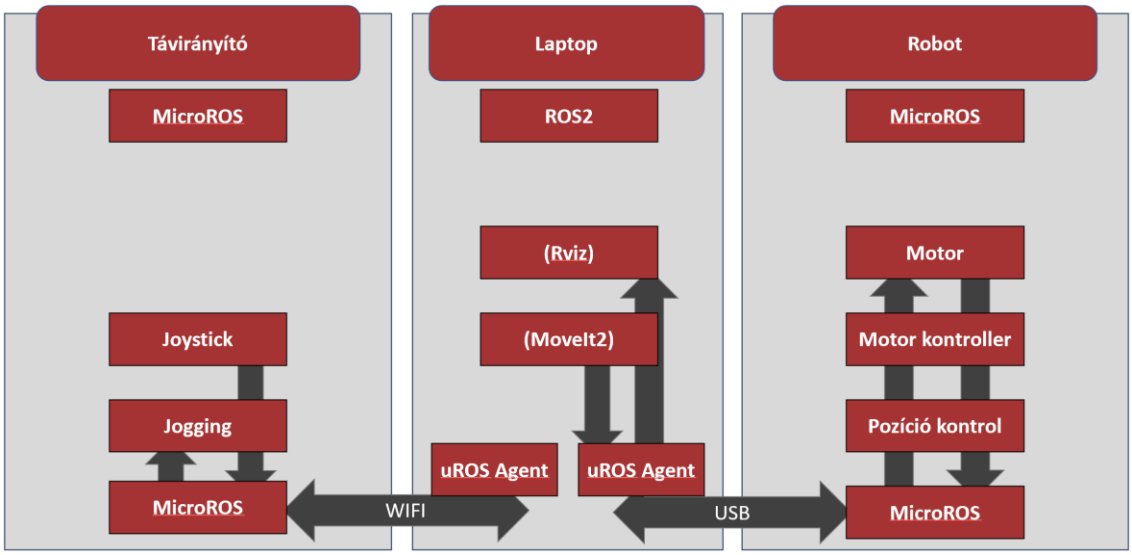



**3. ábra: MicrROS szoftver architektúra**

# 4 A vezérlés leírása

Ebben a fejezetben egy robotkar távirányitóval való vezérlésének elvét foglalom össze, részletezve a különböző kommunikációhoz szükséges üzenettípusokat is.

Az irányítás alapját az képezi, hogy a távirányító ismeri a robotkar jelenlegi állapotát, és az alapján adja ki az új pozíció jelet. Így a távirányító mindig teljes egészében részét képezi a szabályozási körnek. A 4. ábra alapján jól látható az, hogy a felhasználó a joystick mozgatásával az adott pozícióhoz hozzáadja az új elmozdulásokat, majd ezt küldi ki a robotra. Az üzenetet egy „command” topic-ra küldi, melyet majd már a robot fogad. A kapcsolatot a laptopon futó két MicrROS Agent hozza létre, így a Robot automatikusan fogadni tudja a kiadott parancsot, melyet átadva a feldolgozó szálnak a robot végrehajt. A Robot a mindenkori pozícióját visszaküldi egy „state” topicra, így a távirányító is tudja azt.




**4. ábra: Irányítási algoritmus**

Az irányítandó robotok között kizárólag a kiküldendő üzenet ad különbséget, egyébként minden esetben ugyanígy kell irányítani. A tesztelt esetben egy robotkart irányítottam. A kiadott parancs egy JointTrajectory típusú üzenet, amely tartalmazza az egyes jointok megnevezését valamint azok irányítását is pozícióban, sebességben, gyorsulásban, vagy nyomatékban. Ezek közül én egy egyszerűbb esetet választottam, mikor csak a robot új pozícióit adtam ki. A robot JointSate üzenetet küldött vissza jelenlegi állapotáról, mely hasonlóan a fentiekhez, csak pozíció információt tartalmazott.

# 5 MicroROS telepítése a fejlesztőkörnyezethez

Ezt a fejezetet elsősorban a fejlesztői környezet bemutatására szeretném használni, így elérve, hogy könnyedén reprodukálható legyen a feladatom. Miután egy STM32-vel dolgoztam magát a MicroROS-t az STM32 fejlesztői környezetéhez, az STM32CubeIDE-hez, csatlakoztattam hozzá. Így annak beállításait több helyen változtatni kellett. A MicroROS egy úgynevezett static library-át adtam hozzá a CubeIDE-hez, ez mindig letölti a teljes szoftvercsomagot, és a szükséges fileokat. Ennek telepítéséhez szükség van MicrROS által nyújtott fájlokra, így a „micro_ros_stm32cubemx_utils" github repositorit le kell cloneolni a projektfileok közé, bármilyen egyéb tevékenység elött.

## 5.1 Pre-build steps

Annak érdekében, hogy a szoftvercsomag mindig naprakész legyen, minden build esetén ellenőrizni kellett annak naprakészségét. Ezért először a pre-build beállításokat kellett módosítanom. Ez a `Project -> Properties -> C/C++ Build -> Settings -> Build Steps Tab` alatt található. Itt a pre-build steps mezőben egy bat file elindítását írtam. A filet közvetlenül a project mappába kell elhelyezni, az elindításért felelős parancs a következő:

- `${workspace_loc:/${ProjName}}\pre-buildCommands.bat`.

A file elindít egy docker containert, mely letölt minden szükséges filet. A két összetettebb parancsot tartalmaz, melyeket a 5. ábra mutat be.

```
@echo off

title pre-Build Commands
echo This is the pre-build script!
docker pull microros/micro_ros_static_library_builder:humble
docker run --rm -v C:\Users\sanyi\Documents\dev\Repositories\
INDACT_RobotArm\ROS_Test_Firmware:/project --env
MICROROS_LIBRARY_FOLDER=micro_ros_stm32cubemx_utils/
microros_static_library_ide microros/micro_ros_static_library_builder:humble
```

**5. ábra: Pre-build steps**

## 5.2 Include könyvtár hozzáadása

Annak érdekébe, hogy a build közben az IDE a microrsból könyvtárjait is mellé telepítse, szükséges hozzáadni ezek elérhetőségét mind a compilerhez mind a linkerhez. Ezt a `Project -> Properties -> C/C++ Build -> Settings -> Toold Settings Tab -> MCU GCC Compiler -> Include Path` almenüben az alábbi hozzáadása szükséges:

- `../micro_ros_stm32cubemx_utils/microros_static_library_ide/libmicroros/include`

Az `MCU GCC Linker -> Libraries` menüben a fentihez hasonlóan kell hozzáadni:

- Libraries: `microros`
- Library serach path: `"${workspace_loc:/${ProjName}/micro_ros_stm32cubemx_utils/microros_static_library_ide/libmicroros}"`

## 5.3 Kommunikációhoz szükséges fileok

A cloneolt könyvtárból az alábbiakat át kell másolni a projektfileok közé a kommunikáció működése érdekében:

- `extra_sources/microros_time.c`,
- `extra_sources/microros_allocators.c`,
- `extra_sources/custom_memory_manager.c`,
- Egy választott transport file az `extra_sources/microros_transports` mappából.

## 5.4 További feladatok

A továbbiakban csak egy-két feladat maradt hátra. Az „.ioc” fájlban be kell állítanunk ahhoz a freeRTOS taskhoz, mely microROS kommunikációt intéz majd, 10 kB-nál nagyobb tárhelyet, ezt úgy tudjuk megtenni ha a Stack Size-hoz 3000-ret írunk hiszen a word típusban megadott szám így 12 kB-memóriát foglal majd a tasknak.

Továbbá szükséges beállítani a választott kommunikációs protokol beállításait is. Én USB-t választottam így az „.ioc” fájlban a `Connectivity` fül alatt bekapcsoltam az USB kommunikációt, valamint a `Middleware -> USB_DEVICE` fül alatt kiválasztottam a `Communication Device Class (Virtual Port Com)` módot.

Ezekután már nem volt más dolgom csak összefésülni a saját main.c fájlomat a példának ajánlottal, mert én másik kommunikációs formát választottam.