

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

CPS demonstrátor kialakítása

Témalabor, 2022

Rádai Ronald
Konzulens: Huszerl Gábor
2022

1 Bevezető

A félév során a feladatom egy CPS demonstrátor kialakítása, amely alkalmas egy CPS rendszer alapvető struktúráját és működését bemutatni a hallgatónak. Egy ilyen CPS rendszer eszközökből áll, illetve valamilyen szolgáltatásból, amely adatokat, információkat oszt meg az eszközök között. Az adatmegosztó szolgáltatás szerepét nálunk a DDS, az eszközök szerepét az 1. ábrán látható távirányítós kamion tölti be, amelyet a látványosság érdekében többféle be- és kimenettel szereltünk fel.



1. ábra A demonstrátor kamion illusztrációja

A témalabor során szignifikáns szerepet töltött be a kamion felkészítése a DDS-sel való vezérlésre. A vezérlést egy Raspberry Pi-hez hasonló, bár működésben részben eltérő miniszámítógép/SBC (Single Board Computer) segítségével valósítottam meg, melynek neve BeagleBone.

Ezen kívül a feladat magában foglalta a kamion funkcionális bővítését, módosítását is. Felszerelésre került egy feszültség szabályzó áramkör, amely a BeagleBone-t hivatott táplálni, egy elméletileg 100 decibeles duma, amely egy tranzisztoron keresztül kerül meghajtásra, és egy fényhíd. A működtetés során a kamionban található 5 vezetékes szervó motort, amely a kormányzásért felelős, érdemes volt kicserélni egy 3 vezetékesre, mert ennek a vezérlése jóval egyszerűbb.

A kamion projekttel előttem is foglalkoztak már a tanszéken. 2022. tavaszi félévben Fekete-Szabó Illés "DDS vezérelt okos kisautó" c. önálló laboratórium keretei között ért el eredményeket a projekt kapcsán Huszerl Gábor segítségével.

A munkámat ez nagyban segítette, hiszen a kamion eredeti motorvezérlő egységéről a távirányításért felelős IC által kiadott jelek már lemérésre és dokumentálásra kerültek, az előbb említett IC le lett forrasztva, és a helyére csatlakozókkal ellátott vezetékek lettek forrasztva. A panel így teljesen fel volt készítve a BeagleBone fogadására. A lentebb említett alkatrészek is beszerzésre kerültek, így azok megérkezésére nem kellett várnom.

Az RTI cég DDS implementációja több nyelvet is támogat, viszont én a munkám során C++98-at használtam.

A továbbiakban először is ismertetem a projektben felhasznált komponenseket, majd az első találkozásom a DDS-sel, aztán a BeagleBonera való fordítás menetét, majd a működő kód tesztelését a kamion nélkül, aztán az első kört a kamionnal és a kód optimalizálását a kamionra, a komponensek kapcsolatát, majd végül a jövőbeli terveket a kommunikáció struktúrájára.

2 Tájékoztató a projekt komponenseiről

Az alábbiakban ismertetni fogom a használt middleware-t, a felhasznált számítógépet, majd a különböző hardware elemeket.

2.1 DDS

A DDS (Data Distribution Service) [1] egy olyan szolgáltatás, amely middleware szinten képes adatot megosztani egy rendszer különböző komponensei között. Működése során kiemelkedő megbízhatóságot és alacsony késleltetést biztosít, ezért gyakran használják kritikus környezetekben és az ipari IoT világában.

A kommunikáció topicokra osztható, ezek olyan struktúrák, amelyeknek közös az adattípusa a kommunikáció során. Ezekben a topicokban adatokat osztanak meg a komponensek egymás között. Az ilyen topicokba való adat írást végzik a publisherek, akik a kommunikáció során az adatmegosztók szerepét töltik be. Ezeket az adatokat a subscriberek olvassák, akik valamilyen logika szerint feldolgozzák, tárolják ezeket az adatokat. Az adatok publisherektől subscriberekig jutásának viselkedését (késleltetés, biztonság, hibatolerancia, idő, erőforráshasználat) bizonyos QoS paraméterek segítségével tudjuk beállítani.

A feladat során az RTI cég implementációját [2] fogom használni, egész pontosan az RTI Connext DDS 6.1.1-es verzióját.

2.2 BeagleBone

A BeagleBone [3] egy Raspberry Pi-hez hasonló miniszámítógép, amely 65db digitális I/O porttal rendelkezik, összességében 92db felhasználható pin található rajta, amelyek között megtalálható I2C, analóg, PWM, UART és LCD kijelző csatlakozáshoz használható pinek is. A kártyán lévő processzor egy ARM-v7 alapokon nyugvó ARM-Cortex-A8 processzor. A kártya 512 MB DDR3 RAM-mal rendelkezik, található rajta egy 4GB kapacitású eMMC tárhely is, amelyre az operációs rendszer telepíthető.

Kezdetben a projektet egy a 2. ábrán látató vezetékes BeagleBone Black-en kezdtük, de amint beépítésre került a kamionba, az UTP kábel húzgalásának elkerülése céljából váltottunk a Wireless verzióra.



2. ábra BeagleBone Black

2.3 Szervó motor

A HS-81 [4] egy 3 vezetékes szervó motor, amely a 3. ábrán is látható. Ez adott volt a projekt elején, mert már az előző félévben beszerzésre került.

Az előnye az eredeti, 5 vezetékes szervó motorral szemben, hogy az elmozdulást érzékelő és befolyásoló elektronika a motor vázán belül van elhelyezve, így az irányítás során nekem az elmozdulás szögét nem kell figyelnem. Működése egyszerű, egy pár vezeték a tápellátásért felelős, az utolsó vezeték pedig egy PWM-mel hajtott jel bemenet, amely az elmozdulás szögét hivatott meghatározni. A tápellátást a beaglebone biztosítja, a "sys 5v" kimeneten. A PWM jel periódusa 3000µsec, egyenes állásban a jel működési ideje 2150 µsec, teljes balra fordulás esetén 2600 µsec, teljes jobbra fordulás esetén 1700 µsec. Ha nem akarjuk teljesen kifordítani a kormányt, akkor az adott intervallumon

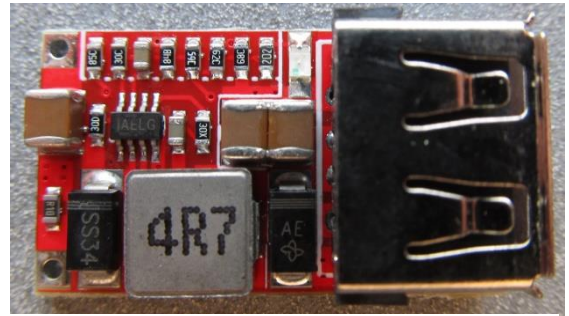


3. ábra HS-81 szervó motor

tetszőleges érték hatására a kormányzás lineárisan fog működni. (A szervó nem pontosan ezen az intervallumon működik, de a kamion helyes működéséhez ezek az értékek lettek beállítva. A valós működési intervallum: [1500 μ sec, 2600 μ sec])

2.4 Feszültség szabályzó áramkör

A feszültség szabályzó áramkör [5] szerepét egy külföldi oldalról rendelt Buck converter fogja ellátni, amely a 4. ábrán látható. Erre azért van szükség, mivel a kamionban található akkumulátor (ami a motort is hajtja) 7,6V üzemi feszültségű, viszont a BeagleBone működtetéséhez 5V-os feszültségre van szükségünk. Az áramkör elméletileg 3A-ig képes ezt a feszültséget szolgáltatni, amely a jelenlegi felhasználáshoz bőven elegendő.



4. ábra Feszültség szabályzó áramkör

2.5 Duda

Egy érdekes feature lehet, ha a kamionunkkal dudálni is tudnánk, így ennek a felszerelése sem volt kérdés. Szerencsére a többi alkatrészhez hasonlóan ez is adott volt, így erre sem kellett várni. A duda egy elméletileg 100 decibelre képes 5V-os kürt lenne, eredetileg távirányítós drónok megtalálásához használt kürt, amely az 5. ábrán is látható. [6]



5. ábra Dudaként használt kürt

2.6 Fényhíd

Az eredeti beépített lámpán kívül tervben van egy fényhíd [7] felszerelése is, amely egy viszonylag erős világítást szolgáltat a kamionnak. A felszerelés helyére még nem született végleges döntés, de valahol a fülke tetején lenne előnyös az elhelyezése.



6. ábra Felhasznált fényhíd

3 Ismerkedés a DDS-sel

Az első feladatom egy Hello World alkalmazás elkészítése volt, amelyben a publisher másodpercenként egy "Hello World" tartalmú stringet küld a subscribernek.

Ehhez a DDS szolgáltatást telepítenem kellett a gépemre, de különösebb nehézséget ez nem okozott. Az RTI [2] oldaláról letöltöttem a Windows-ra szánt telepítő exe-t, majd a szokásos módon feltelepítettem, majd beállítottam az NDDHOME környezeti változót. Ezt követően egy Getting Started guide [8] segítségével elkezdtem az első feladatomat. A kódgenerátor segítségével generáltam a példát, ezt a következő sorral tettem:

```
rtiddsgen -language c++ -platform <x64Win64VS2017> -create makefiles -create
typefiles -d c++98 -ppDisable hello_world.idl
```

A kódban elvégeztem az útmutató által javasolt módosítást, majd a Visual Studio segítségével fordítottam és futtattam a két kódot.

A feladat viszonylag gyorsan sikerült, így nekiálltam írni egy alap irányítást megvalósító publisher-subscriber párt. Kezdetben csak az alap 4 irányt szerettem volna implementálni (előre,

hátra, balra, jobbra), de olyan módon, hogy egyidejűleg akár több aktív is lehessen (balra előre). Erre a következő adatstruktúrát definiáltam:

```

1      enum going
2      {
3          going_null = 0,
4          forward = 1,
5          backward = 2
6      };
7      enum direction
8      {
9          direction_null = 0,
10         left = 1,
11         right = 2
12     };
13
14     struct DriveAssembly
15     {
16         going drive;
17         direction steer;
18     };

```

Az adatok struktúrája könnyen érthető, lényegében enumokkal tároljuk a vezetés adatait. A going 3 értéket vehet fel, vagy előre megyünk, vagy hátra, vagy sehova. A direction szintén, vagy balra fordulunk, vagy jobbra, vagy egyenesen megyünk, majd ebből a két enumból van egy-egy a struktúrában, amely a kamion vezetéséért lesz felelős.

A kód felépítésének lényegi részét a következőkben ismertetem. A publisher és a subscriber kódja nagyban megegyezik, így itt csak a publisher kódját fogom értelmezni. A kódrészletekből kihagytam a hibakezeléssel és felszabadítással kapcsolatos részeket, hiszen az nem tesz hozzá a kód érthetőségéhez, és a kódrészletek hosszát nagyban növeli. Az argumentumok feldolgozása után indul a következő függvény, amelyben a kód fő ciklusa is található:

```

1      int run_code(unsigned int domain_id, unsigned int sample_count)
2      {

```

Ezt követi a DDS specifikus inicializálás, ahol létrehozunk egy participant-ot, majd egy publishert, aztán egy topicot, és egy data writert. Ezek után a sample lesz az adat, amit a data writer a topicba ír majd.

```

3          DDSDomainParticipant* participant =
4              DDSTheParticipantFactory->create_participant(
5                  domain_id,
6                  DDS_PARTICIPANT_QOS_DEFAULT,
7                  NULL /* listener */,
8                  DDS_STATUS_MASK_NONE);
9
10         DDSPublisher* publisher = participant->create_publisher(
11             DDS_PUBLISHER_QOS_DEFAULT,
12             NULL /* listener */,
13             DDS_STATUS_MASK_NONE);
14
15         DDS_ReturnCode_t retcode =
16             DriveAssemblyTypeSupport::register_type(participant, type_name);
17         DDSTopic* topic = participant->create_topic(
18             "DriveTruckTopic",
19             type_name,
20             DDS_TOPIC_QOS_DEFAULT,

```

```

21         NULL /* listener */,
22         DDS_STATUS_MASK_NONE);
23
24     DDSDataWriter* writer = publisher->create_datawriter(
25         topic,
26         DDS_DATAWRITER_QOS_DEFAULT,
27         NULL /* listener */,
28         DDS_STATUS_MASK_NONE);
29     DriveAssembly* sample = DriveAssemblyTypeSupport::create_data();
30

```

Itt következik a kód fő ciklusa, ahol minden iterációban megnézzük a gombok állapotát, és az alapján írunk a topicba.

```

31     unsigned int message_counter = 0;
32     for (unsigned int count = 0; !shutdown_requested && count <
33         sample_count; ++count)
34     {
35         // Set driving variables to zero
36         sample->drive = going_null;
37         sample->steer = direction_null;
38
39         // Check pressed keys
40         if (GetAsyncKeyState(VK_UP) < 0)
41             sample->drive = forward;
42         else if (GetAsyncKeyState(VK_DOWN) < 0)
43             sample->drive = backward;
44         if (GetAsyncKeyState(VK_LEFT) < 0)
45             sample->steer = left;
46         else if (GetAsyncKeyState(VK_RIGHT) < 0)
47             sample->steer = right;
48
49         //If we press any of the direction keys
50         if (sample->drive != going_null || sample->steer !=
51             direction_null)
52         {
53             retcode = driveAssembly_writer->write(*sample,
54                 DDS_HANDLE_NIL);
55             if (retcode != DDS_RETCODE_OK)
56             {
57                 std::cerr << "write error " << retcode <<
58                     std::endl;
59             }
60             else
61             {
62                 std::cout << "Loop counter: " << count << ", Sent command counter: " <<
63                     message_counter << ", Sent command: " << (sample->drive == forward ?
64                     "Forward, " : (sample->drive == backward ? "Backward, " : "Stopped, ")) <<
65                     (sample->steer == left ? "Left" : (sample->steer == right ? "Right" :
66                     "Straight")) << std::endl;
67                 ++message_counter;
68             }
69         }
70     }
71
72     DDS_Duration_t send_period = { 0, WAITTIME * 1000000 };
73     NDDSUtility::sleep(send_period);
74 }

```


Az eredményeket az 58. sorban látható módon a konzol ablakra írtam ki. Fontosnak tartottam, hogy az irányítás kényelmes és folyamatos legyen, így a publisher működése során pollingot használva 50ms időnként (ez a 64. sorban látható `send_period`) a `windows.h` `getAsyncKeyState` függvényei segítségével derítettem ki, hogy a 4 nyíl billentyű közül melyek vannak lenyomva (a gombok kezelése a 34-46. sor között található). Ezen kívül figyeltem arra is, hogy a hálózaton felesleges kommunikáció ne történjen, ennek érdekében csak akkor küldtem üzenetet, amikor valamelyik billentyű le volt nyomva (49. sor feltétel miatt).

A subscriber oldalon a fő ciklus várakozik, amíg le nem telik 200ms, vagy nem kap egy üzenetet. Megérkezett üzenet esetén azt feldolgozza, amennyiben pedig 200ms időintervallum alatt egy üzenetet sem kapott, leállítja a kamion mozgását.

A 7. ábrán látható az eddigi eredmény. A "Loop counter" a publisher main loop futások számát jelzi, a "Sent command counter" az elküldött üzenetek számát, a "Sent command" pedig az elküldött üzenetet.

C:\Users\gtadj\Documents\Backup\Work\5.2023\Ten	C:\Users\gtadj\Documents\Backup\Work\5.2023\Temalab\extras\old_versions\communicators_v0.1\Truckdriver\objs\64Win
Drive: Forward, Direction: Left	Loop counter: 1283, Sent command counter: 572, Sent command: Forward, Left
Drive: Forward, Direction: Left	Loop counter: 1284, Sent command counter: 573, Sent command: Forward, Left
Drive: Forward, Direction: Left	Loop counter: 1285, Sent command counter: 574, Sent command: Stopped, Left
Drive: Stopped, Direction: Left	Loop counter: 1286, Sent command counter: 575, Sent command: Stopped, Right
Drive: Stopped, Direction: Left	Loop counter: 1287, Sent command counter: 576, Sent command: Stopped, Right
Drive: Stopped, Direction: Left	Loop counter: 1288, Sent command counter: 577, Sent command: Forward, Left
Drive: Backward, Direction: Left	Loop counter: 1289, Sent command counter: 578, Sent command: Forward, Left
Drive: Backward, Direction: Left	Loop counter: 1290, Sent command counter: 579, Sent command: Forward, Left
Drive: Backward, Direction: Left	Loop counter: 1291, Sent command counter: 580, Sent command: Forward, Left
Drive: Stopped, Direction: Left	Loop counter: 1292, Sent command counter: 581, Sent command: Forward, Left
Drive: Stopped, Direction: Left	Loop counter: 1293, Sent command counter: 582, Sent command: Forward, Left
Drive: Forward, Direction: Left	Loop counter: 1294, Sent command counter: 583, Sent command: Forward, Right
Drive: Forward, Direction: Left	Loop counter: 1295, Sent command counter: 584, Sent command: Stopped, Right
Drive: Forward, Direction: Right	Loop counter: 1296, Sent command counter: 585, Sent command: Stopped, Right
Drive: Forward, Direction: Straight	Loop counter: 1297, Sent command counter: 586, Sent command: Forward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1298, Sent command counter: 587, Sent command: Forward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1299, Sent command counter: 588, Sent command: Forward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1300, Sent command counter: 589, Sent command: Stopped, Left
Drive: Stopped, Direction: Straight	Loop counter: 1301, Sent command counter: 590, Sent command: Stopped, Left
Drive: Stopped, Direction: Straight	Loop counter: 1302, Sent command counter: 591, Sent command: Stopped, Left
Drive: Stopped, Direction: Straight	Loop counter: 1303, Sent command counter: 592, Sent command: Backward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1304, Sent command counter: 593, Sent command: Backward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1305, Sent command counter: 594, Sent command: Backward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1306, Sent command counter: 595, Sent command: Stopped, Left
Drive: Stopped, Direction: Straight	Loop counter: 1307, Sent command counter: 596, Sent command: Stopped, Left
Drive: Stopped, Direction: Straight	Loop counter: 1308, Sent command counter: 597, Sent command: Forward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1309, Sent command counter: 598, Sent command: Forward, Left
Drive: Stopped, Direction: Straight	Loop counter: 1310, Sent command counter: 599, Sent command: Forward, Right
Drive: Stopped, Direction: Straight	Loop counter: 1311, Sent command counter: 600, Sent command: Forward, Straight

7. ábra Alapirányok DDS-sel

4 Fordítás a BeagleBone-ra

A fordítás menete sajnos közel sem volt triviális köszönhetően az RTI által kényelmesnek vélt (és azonos rendszerre való fordítás esetén valóban kényelmes) kódgenerátornak [9]. Szerencsére a problémákon sikerült felülkerekednem, de a folyamat hosszadalmas volt, és nagyon sok energiát igényelt. A következő fejezetet ezen probléma kifejtésének szenteltem.

Mivel a BeagleBone Linuxot futtat, és ARM processzort tartalmaz, ezért nem lehet telepíteni rá a RTI DDS fordításhoz szükséges csomagot, így a Bone-on fordítani nem lehet a kódot. Erre a megoldást az jelenti, hogy biztosítanak target bundle-eket (ezek olyan csomagok, amelyek az adott számítógéphez specifikus függőségeket tartalmaznak, de bármilyen másik operációs rendszerre telepíthetők, hogy onnan tudjuk fordítani a programunkat arra az operációs rendszerre, ahol a fordítás nem lehetséges), amelyek segítségével fordíthatunk bármelyik másik operációs rendszerről, ahol támogatott a DDS telepítése.

A fordítás előtti felkészülést az jelentette, hogy letöltöttem a target bundle-t, és az RTI által fejlesztett RTI Launcher segítségével az ARMv7-hez szükséges függőségeket hozzáadtam a telepített csomagokhoz.

4.1 Fordítás Windows alól a generált Makefile segítségével

Kezdetben az RTI által ajánlott fordítási módszerrel próbálkoztam Windows 11 rendszerről, sajnos kevés sikerrel. A folyamat úgy nézett ki, hogy egy readme-ben megadott paranccsal simán lefordítjuk a kódot, és ezt felmásoljuk a targetünkre. Az említett parancs a következő volt:

```
make -f makefile_driveAssembly_armv7Linux4gcc7.5.0
```

Kezdetben a probléma az volt, hogy a "make" parancs nem volt található a gépen, hiszen Windows rendszerről lévén szó alpból nem jár hozzá make. Erre gyorsan jött is a megoldás, telepítettem a Chocolatey-t [10], amely egy Windows rendszerekhez készült csomagkezelő. Így sikerült telepítenem a make-et, amely elméletileg le tudta volna fordítani a kódomat a targetre.

Sajnos a következő fordítási próbálkozásnál nem ez történt, a makefile UNIX specifikus elemeket tartalmazott. Megpróbáltam átírni, hogy Windows rendszereken is lefusson, de ezt részben a hossza miatt, részben pedig az egyszerűbb megoldás reményében feladtam. Hosszas ötletelés után jött a következő ötlet.

4.2 Fordítás cross compile-lal Visual Studio segítségével

Egy kézenfekvő megoldás lett volna cross compile segítségével fordítani, mivel ez támogatott volt Visual Studio-ban, és ugyanezt a fejlesztőkörnyezetet használtam a PC-s Publisher fejlesztése közben is. Sajnos a kódgenerátor által generált projektet nem sikerült ARM Linux-ra targetelni, ugyanis platform váltást nem találtam az IDE-ben.

Lehetőség lett volna még, hogy létrehozok egy Linuxos ARM projektet, és az eddig elkészült fájlokat ebbe a projektbe másolom be. Viszont az RTI DDS függőségeket ebben az esetben nem sikerült a projektbe importálni. Az RTI csak a kódgenerátor által támogatja a projektek létrehozását.

4.3 Fordítás WSL alól Debian 11 segítségével

Gyorsan kiderült, hogy gyorsabb és egyszerűbb megoldás a programot Linux alól fordítani. Mivel a target rendszer is Linux alapú, így bíztam benne, hogy kevesebb akadállyal fogok találkozni.

Mivel a kódolást a saját gépemem szerettem volna végezni, ezért az egyszerűbb automatizálás érdekében nem lett volna jó megoldás virtuális gépről fordítani (ebben az esetben a host operációs rendszertől teljesen elszeparálva lettek volna a file-ok, így az automatikus fordítást nagyon nehéz, szinte lehetetlen lett volna megoldani, mivel a megírt kód automatikus virtuális gépre jutása egy nagyon nehéz feladat), ezért a választásom a Windows Subsystem for Linux-ra [11] esett. (A próbálkozásaim során végig az 1-es verziót használtam.)

Mivel a BeagleBone-ra telepített operációs rendszer egy Debian [12] disztribúció, így elsőnek én is egy Debian disztribúcióval próbálkoztam a kompatibilitási problémák elkerülése érdekében. A disztribúció telepítése után telepítettem az RTI Connex DDS implementációt, majd a target bundle telepítése után neki is álltam a fordításnak. Szerencsére sikeresen lefordult a kód, viszont sajnos az eszközre való másolás után nem indult el, a libc egy újabb verzióját hiányolta. A hibára rákérdezve kiderítettem, hogy ez arra utal, hogy a kernel verziója túl régi, hogy futtatni tudja a programot, mivel a libc library verziója és a kernel verziója összefüggésben van. (Debian 9-es verzió futott a Bone-on, amely eredetileg 3-as kernel verzióval jött ki, majd 4.9-ig lehetett frissíteni.)

Neki is álltam a BeagleBone-on lévő Linux verzió frissítésének. Mivel, amivel dolgoztunk, egy Black változata volt a Bone-nak, így a legfrissebb telepíthető Debian verzió a 10 volt. Sajnos miután újra próbáltam futtatni a programot, kiderült, hogy így sem elég friss a kernel verziója, hiszen a futtatáshoz a libc 4.29-es verziójára lett volna minimum szükség, de csak a 4.28-as verzió volt elérhető.

4.4 Fordítás Ubuntu 18.04 segítségével

Megpróbáltam a WSL alatt található operációs rendszert alacsonyabb verzióra hozni, viszont ez sajnos nem sikerült, mivel a WSL-lel használható legrégebbi kép Debian 11-et használt. Az egyetlen Linux verzió, ami elég idősnek tűnt, és támogatott volt WSL-ben, az Ubuntu 18.04-es [13] verziója volt. Szerencsére ennek a rendszernek a kernel verziója megfelelő volt, így a RTI Connex DDS implementáció, majd a target bundle telepítése után a fordítás sikeres volt. Ekkor az eszközre másolva a programot a kód végre futni kezdett.

4.5 Automatizált deploy script megírása

Szerettem volna a fordítást, a kód eszközre való továbbítását és a futtatást automatizáltan megoldani, hiszen ez egy viszonylag fáradalmas feladat kézzel. A legegyszerűbb megoldásnak egy batch script megírása tűnt, amely a következőket tartalmazza: Átmásolja a mappából, ahol a fejlesztés történik, a forráskódot a WSL filerendszerébe, a megfelelő mappába, majd a WSL segítségével lefuttat egy parancsot, amely lefordítja a kódot az Ubuntu-n, aztán SCP [14] segítségével az eszközre másolja a futtatható kódot, majd SSH-n [15] keresztül csatlakozik a Bone-hoz, és futtatja a felmásolt programot.

Az elkészült kód a következő:

```
@echo off

set ip=192.168.1.2
set
source=C:\Users\gtadj\Documents\Backup\Work\5.2023\Temalab\communicators\Truckdriver\driveAssembly_subscriber.cxx
set
destination=C:\Users\gtadj\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu18.04LTS_79rhkp1fndgsc\LocalState\rootfs\home\ronald\truckDriver\src
set
compiled=C:\Users\gtadj\AppData\Local\Packages\CanonicalGroupLimited.Ubuntu18.04LTS_79rhkp1fndgsc\LocalState\rootfs\home\ronald\truckDriver\src\objs\armv7Linux4gcc7.5.0\driveAssembly_subscriber

REM Copying file to compiling folder
copy /y %source% %destination%

REM Compiling driveAssembly_subscriber.cxx file (with other files)
ubuntu1804 run "cd ~/truckDriver/src/ && sudo rm -d objs -r && sudo make -f makefile_driveAssembly_armv7Linux4gcc7.5.0"

REM Sending driveAssembly_subscriber file to BeagleBone
scp %compiled% debian@%ip%:.

REM Starting project on remote device
ssh debian@%ip% "config-pin P9.14 pwm ; config-pin P9.16 pwm ; config-pin P9.22 pwm ; ./driveAssembly_subscriber"
```

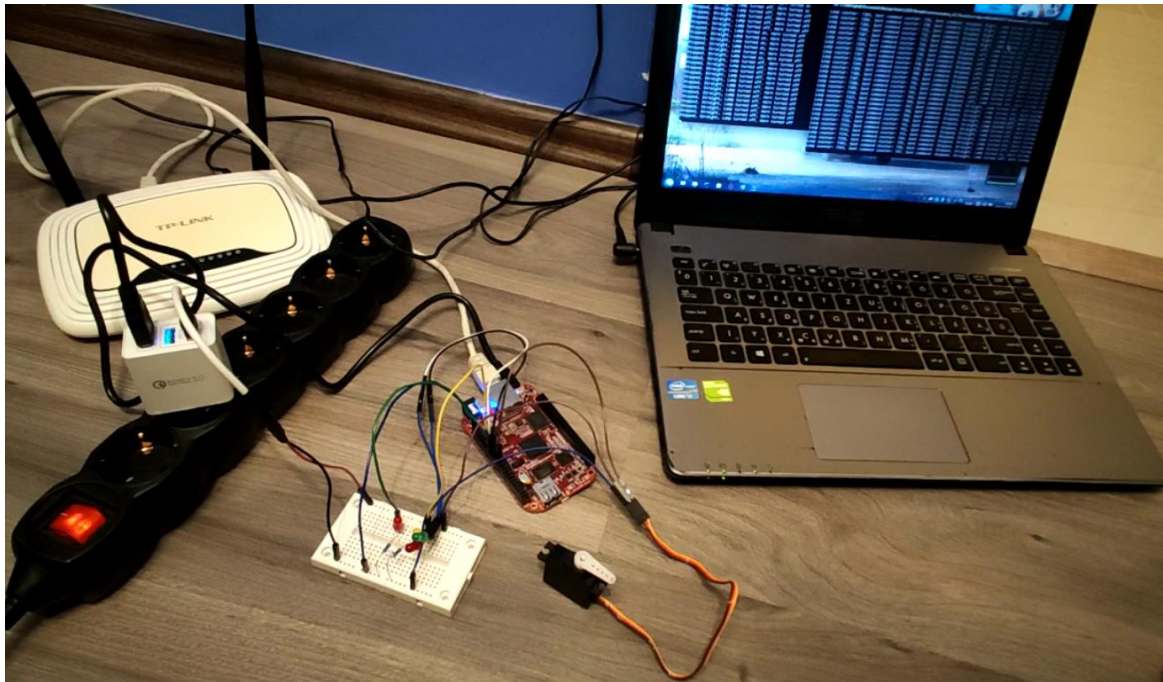
5 Jelek kiadása a BeagleBone-nal

A kamion motorjának működtetéséhez PWM jelre volt szükségünk. Sajnos a PWM jel kiadására nem találtam működőképes könyvtárat BeagleBone-ra, így azt fileműveletekkel szabályozom. Mielőtt egy lábat PWM módban szeretnénk használni előtte azt configolni kell PWM-hez. Ezt a terminálban a "config-pin {P8/P9}.{XX} pwm" utasítással tehetjük meg, melyben a {P8/P9} a Bone-on lévő pin header azonosítója, az {XX} pedig a pin header-ben elfoglalt száma. A pin

konfigurálást minden újraindítás után meg kell tennünk, így az megtalálható a deploy scriptben is. A PWM lábak vezérléséhez a kamionhoz a tanszéken készített Wiki oldal [16] nyújtott segítséget, itt találtam olyan információt, ami segítségével sikerült a megfelelő outputot kiadnom a lábakon. A PWM jel kiadásához a `/dev/pwm/ehrpwmXX/period` file-ba kellett a PWM jel periódus idejét nanoszekundumban beírni, a `/dev/pwm/ehrpwmXX/duty_cycle` file-ba a PWM jel munka idejét kell beírni nanoszekundumban, és a `/dev/pwm/ehrpwmXX/enable` file-ba a 1 értéket, ha engedélyezve van a kimenet, 0 értéket, ha nincs engedélyezve. Az XX mindenhol egy olyan kétjegyű számot, vagy betű és szám kombinációt jelöl, amely jellemzi az adott lábat. A GPIO lábak vezérlése sokkal egyszerűbb volt, itt nem kellett a terminálban semmit konfigurálni, a `/sys/class/gpio/gpioXX/export` file-ba kellett beírni a pin számát, a `/sys/class/gpio/gpioXX/direction` file-ba az irányt (be/kimenet - in/out), a `/sys/class/gpio/gpioXX/value` file-ba az értéket (0/1), a `/sys/class/gpio/gpioXX/enable` file-ba pedig, hogy engedélyezve van-e a kimenet. A kód módosítása után multiméterrel ellenőriztem, hogy a lábakon az elvárt kimenetet tapasztalom-e.

6 Kód tesztelése a kamion nélkül

Mielőtt a kamionba szereltem a BeagleBone-t, szerettem volna kipróbálni nélküle, hogy minden funkció működik-e. A teszteléshez elsőnek egy router-ben beállítottam egy statikus IP címet a Bone-nak, majd csatlakoztam hozzá, és kipróbáltam a deploy script működését a hálózaton keresztül. Szerencsére a várt módon felmásolódott az eszközre a kód. Ezután összekábeleztem a BeagleBone-t, a szervó motort és néhány LED-et 1-1 10k ellenállással párosítva egy breadboardra, hogy lássam az egyes funkciókat működés közben. Kezdetben az előre- és hátramenetet 1 LED szimbolizálta, illetve a dudára és a lámpára is volt 1-1 LED kötve. A 8. ábrán látható módon történt a tesztelés. Minden működött, így jöhetett a kamion összeszerelése, majd tesztelése.



8. ábra Működés tesztelése a kamion nélkül

7 Kód tesztelése a kamionnal

A sikeres kezdeti tesztek után összekábeleztuk a kamiont és a BeagleBone-t, majd kipróbáltuk a funkciókat a teljes rendszeren.

Az elkészített funkciók megfelelően működtek, viszont kiderült, hogy a kamion sebessége túlságosan gyors, változtathatóvá kell tenni a sebességet, hogy a kamion precízen manőverezhetővé váljon. Az adatstruktúrát módosítani kell, mert az első projektben erre nem gondoltam. A módosított adatstruktúra a következő lett:

```
struct DriveAssembly
{
    double drive;
    double steer;
    int32 lamp_mode;
    int32 horn_voice;
};
```

Itt a drive és a steer változók az irányításért felelősek, amíg az újonnan bekerült lamp_mode változó a lámpa kapcsolásához, a horn_voice a duka kapcsolásához szükséges. Ezek azért int32 típusúak, mert így támogathatjuk több módban való működést is. (pl. duka, szaggatott sípolás, vagy akár dallam lejátszása) Az irányításért felelős változók double típusát az indokolja, hogy így különböző sebességet tudunk támogatni. A 0 az alap állapot, amikor a kamion nem mozog, minden az eredeti állapotban van. A pozitív érték az előre menetelt vagy jobbra fordulást jelenti, ahol az 1 érték a maximális, a negatív érték a hátra menetelt vagy a balra fordulást jelenti, ahol a -1 maximális érték. A maximális és minimális kormányzóhoz tartozó PWM jel közötti jel értelmes outputot generál, így bármilyen tört értelmezhető megfelelő inputnak.

A fentiek alapján a Subscriber kód változtatásának a lényegi része a következő:

```
// Iterate over all available data
for (int i = 0; i < data_seq.length(); ++i)
{
    // Check if a sample is an instance lifecycle event
    if (!info_seq[i].valid_data)
    {
        std::cout << "Received instance state notification" << std::endl;
        continue;
    }
    // Print data
    //DriveAssemblyTypeSupport::print_data(&data_seq[i]);
    drive = data_seq[i].drive;
    steer = data_seq[i].steer;
    horn_voice = data_seq[i].horn_voice;
    lamp_mode = data_seq[i].lamp_mode;
    samples_read++;
}
```

Itt látható, hogy végigiterálunk az üzenetsoron, és az egyes változókba bekerül az utolsó üzenet értékei.

Ez után hívódik a kimenetek feldolgozásának függvénye:

A feldolgozás során először megnézzük, hogy a kapott parancs változott-e az előzőhöz képest. Amennyiben igen, azt a megfelelő lábon elküldjük. Először az előre/hátramenet információját dolgozzuk fel. Ha a kapott érték nagyobb, mint nulla, akkor előre megy a kamion, ehhez egy PWM jelet adunk ki a minimum és a maximum periódusidő között skálázva a munkaciklust az előremenetre dedikált lábra, a másik láb értéke 0 lesz. Ha a kapott érték kisebb, mint nulla, akkor hátramenet lábra tesszük ugyanazt a jelet, mint az előző esetben az előremenetre dedikált lábra

tettünk volna, és az előremenet láb értéke 0 lesz. Ha a kapott érték nulla, akkor mindkét láb értéke 0 lesz.

```

1 void process_Output()
2 {
3     if (drive != drive_last)
4     {
5         std::fstream pwm_forward;
6         std::fstream pwm_backward;
7         pwm_forward.open("/dev/pwm/ehrpwm1a/duty_cycle");
8         pwm_backward.open("/dev/pwm/ehrpwm1b/duty_cycle");
9         if (drive > 0 && drive <= 1)
10        {
11            pwm_backward << 0;
12            pwm_forward << DUTY_FORWARD_MIN + (PERIOD_DRIVE -
13                DUTY_FORWARD_MIN) * drive;
14        }
15        else if (drive < 0 && drive >= -1)
16        {
17            pwm_forward << 0;
18            pwm_backward << DUTY_FORWARD_MIN + (PERIOD_DRIVE -
19                DUTY_FORWARD_MIN) * (-drive);
20        }
21        else
22        {
23            pwm_forward << 0;
24            pwm_backward << 0;
25        }
26        pwm_forward.close();
27        pwm_backward.close();
28        drive_last = drive;
29    }
30 }

```

A kormányzás megvalósítása egy láb vezérlésével történik. Ha negatív értéket kapunk, akkor a kormány teljesen balra fordítjuk, ha pozitív értéket, akkor teljesen jobbra, ellenkező esetben középre irányítjuk. (Ezt a továbbiakban, ha tudunk, érdemes lehet módosítani, hiszen egy olyan Publisherrel, aki a kormányt félig szeretné elfordítani (0,5 értéket küld), jelenlegi állapot szerint a kormány teljesen balra fordul.)

```

28     if (steer != steer_last)
29     {
30         std::fstream fs;
31         fs.open("/dev/pwm/ehrpwm0a/duty_cycle");
32         if (steer < 0 && steer >= -1)
33             fs << DUTY_LEFT_STEER;
34         else if (steer > 0 && steer <= 1)
35             fs << DUTY_RIGHT_STEER;
36         else
37             fs << DUTY_STRAIGHT_STEER;
38         fs.close();
39         steer_last = steer;
40     }

```

A lámpa kimenetének feldolgozása következik. Itt gondoltam a többféle mód kezelésére, ezért a kódot úgy írtam meg, hogy az a jövőben könnyen bővíthető legyen. A jelenlegi kód a lekapcsolt és felkapcsolt állást tudja kezelni, nulla (vagy jelenleg az 1-es érték kivételével bármely szám esetén) a lámpa le lesz kapcsolva, 1 esetén a lámpát felkapcsolja.

```
41  if (lamp_mode != last_lamp_mode)
42  {
43      std::fstream fs;
44      fs.open("/sys/class/gpio/gpio60/value");
45      switch (lamp_mode)
46      {
47          case(1):
48              fs << 1;
49              break;
50          default:
51              fs << 0;
52              break;
53      }
54      fs.close();
55      last_lamp_mode = lamp_mode;
56  }
```

A dudu funkció kezelése a lámpával teljesen analóg módon működik.

```
57  if (horn_voice != last_horn_voice)
58  {
59      std::fstream fs;
60      fs.open("/sys/class/gpio/gpio48/value");
61      switch (horn_voice)
62      {
63          case(1):
64              fs << 1;
65              break;
66          default:
67              fs << 0;
68              break;
69      }
70      fs.close();
71      last_horn_voice = horn_voice;
72  }
73 }
```

A laptopon futó Publisher kódja is kis részben változott. A változtatás lényegi része a billentyűk kezelésében és az új funkciók billentyűinek figyelésében van, illetve készült egy *getSpeed()* függvény is, amely a megfelelő sebesség értékét hivatott meghatározni.

A *getSpeed()* függvény kódja:

```
1 double getSpeed()
2 {
3     if (GetAsyncKeyState(0x59) < 0)
4         return GOING_SLOW * 0.01;
5     if (GetAsyncKeyState(0x41) < 0)
6         return GOING_FAST * 0.01;
7     return GOING_NORMAL * 0.01;
8 }
```

A billentyűk kezelésének a kódja a következő lett:


```

1 // Check pressed keys
2 if (GetAsyncKeyState(VK_UP) < 0)
3     sample->drive = getSpeed();
4 else if (GetAsyncKeyState(VK_DOWN) < 0)
5     sample->drive = -getSpeed();
6 if (GetAsyncKeyState(VK_LEFT) < 0)
7     sample->steer = -1;
8 else if (GetAsyncKeyState(VK_RIGHT) < 0)
9     sample->steer = 1;
10 if (GetAsyncKeyState(0x4C)) // L button
11 {
12     lamp_toggle = true;
13     lamp_state = lamp_state == 0 ? 1 : 0;
14 }
15 if (GetAsyncKeyState(0x48)) // H button
16     sample->horn_voice = 1;

```

8 Kamion, a komponensek és a BeagleBone kapcsolata

A teljes tápellátást a kamion eredeti akkumulátora biztosítja, amely a kamion eredeti vezérlő áramköréhez van csatlakoztatva. Ez az áramkör felelős a motor meghajtásáért a kapott PWM jel alapján. Ezen kívül az áramkörön egy GND és egy tápellátás lábat is találtam, ahonnan az akkumulátorra tudunk további fogyasztókat kötni. Ezekre a lábakra kötöttem a 2.4 bekezdésben említett feszültség szabályzó áramkört, amely az akkumulátor 7.4V-os feszültségét 5V-ra alakítja át. Ez táplálja a BeagleBone-t. Az összes többi komponens a BeagleBone-ról van táplálva az 5V kimeneteiről, vagy a GPIO lábakon keresztül. A további elemek a 9. ábrán látható módon vannak bekötve. A használt lábak a P9 pin headeren vannak. 8: szervo táp 12: lámpa, 14: motor előre PWM jel, 15: motor hátra PWM jel, 22: szervo kormányzás PWM jel

P9					P8			
Function	Physical Pins		Function		Function	Physical Pins		Function
DGND	1	2	DGND	LEGEND Power, Ground, Reset Digital Pins PWM Output 1.8 Volt Analog Inputs Shared I2C Bus Reconfigurable Digital	DGND	1	2	DGND
VDD 3.3 V	3	4	VDD 3.3 V		MMC1_DAT6	3	4	MMC1_DAT7
VDD 5V	5	6	VDD 5V		MMC1_DAT2	5	6	MMC1_DAT3
SYS 5V	7	8	Szervo táp		GPIO_66	7	8	GPIO_67
PWR_BUTTON	9	10	SYS_RESET		GPIO_69	9	10	GPIO_68
UART4_RXD	11	12	Lámpa		GPIO_45	11	12	GPIO_44
UART4_TXD	13	14	Motor előre PWM		EHRPWM2B	13	14	GPIO_26
Duda	15	16	Motor hátra PWM		GPIO_47	15	16	GPIO_46
SPI0_CS0	17	18	SPI0_D1		GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C_SDA		EHRPWM2A	19	20	MMC1_CMD
SPI0_DO	21	22	Szervo kormányzás		MMC1_CLK	21	22	MMC1_DAT5
GPIO_49	23	24	UART1_TXD		MMC1_DAT4	23	24	MMC1_DAT1
GPIO_117	25	26	UART1_RXD		MMC1_DATA0	25	26	GPIO_61
GPIO_115	27	28	SPI1_CS0		LCD_VSYNC	27	28	LCD_PCLK
SPI1_DO	29	30	GPIO_112		LCD_HSYNC	29	30	LCD_AC_BIAS
SPI1_SCLK	31	32	VDD_ADC		LCD_DATA14	31	32	LCD_DATA15
AIN4	33	34	GND_ADC		LCD_DATA13	33	34	LCD_DATA11
AIN6	35	36	AIN5		LCD_DATA12	35	36	LCD_DATA10
AIN2	37	38	AIN3		LCD_DATA8	37	38	LCD_DATA9
AIN0	39	40	AIN1		LCD_DATA6	39	40	LCD_DATA7
GPIO_20	41	42	ECAPWMO		LCD_DATA4	41	42	LCD_DATA5
DGND	43	44	DGND		LCD_DATA2	43	44	LCD_DATA3
DGND	45	46	DGND		LCD_DATA0	45	46	LCD_DATA1

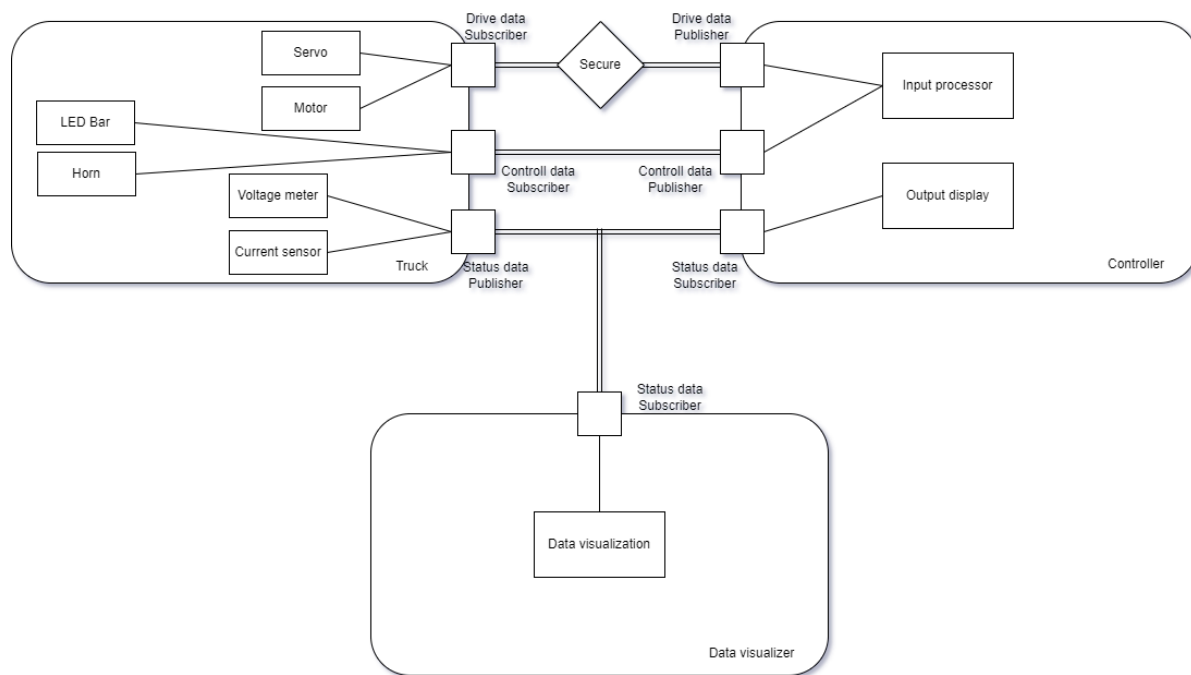
9. ábra A BeagleBone kapcsolata a komponensekkel.

9 Projekt kilátásai, jövőbeli tervek

Sok ötlet, terv született, amit még meg lehetne valósítani:

- Demó mód: A kamion önállóan bemutatja a funkcióit.
- A 2.6 fejezetben említett fényhíd felszerelése: Sajnos a félév során nem jutott idő a fényhídra, viszont a kommunikáció jelenlegi működése szerint szoftver szinten a megoldása nagyon kis munkával megoldható lenne.
- Xbox kontrollerral való irányítás: Az irányítás egy egyszerűbb formája lenne egy hasonló célokra (részben vezetésre) tervezett kontrollerrel való vezetés.
- Feszültség és árammérő áramkör beszerelése: tervben volt még egy áramkör beszerelése, amivel monitorozni lehetne az aktuális feszültséget és a pillanatnyi áramfogyasztást, ennek a beszerelésére sajnos nem jutott idő.
- Adat vizualizációs Subscriber: egy olyan Subscriber létrehozása, amely az aktuális feszültség és áramerősség adatokat vizuálisan megjeleníti
- Új kommunikációs modell implementálása: A félév vége felé a kommunikáció átgondolására is fordítottam időt, ennek eredménye a 10. ábrán látható kommunikációs rendszermodell, amely ugyan nem SysML 1.6 modell, de az alapvető szabályokat követi. Ez a kommunikáció egy új, jobban átgondolt változatát mutatja be, amely a jövőben implementálásra vár.

Az ábra jobb oldalán a vezérlő komponens található, amely a vezetési és vezérlési adatok szempontjából publisher, a státusz adatok szempontjából subscriber. A bal oldalon a kamion található a BeagleBone-nal, amely a vezetési és vezérlési adatok szempontjából subscriber, a státusz adatok szempontjából pedig publisher. Az ábra alján egy olyan komponens szerepel, amely képes az adatokat vizuálisan megjeleníteni, vagy esetleg egy adatbázisban eltárolni. Fontos még megemlíteni, hogy a Drive data-n található Secure szó a kapcsolat biztonságára utal, ez az RTI DDS Security plugin segítségével lenne megoldható.



10. ábra Kommunikációs rendszermodell

A projektet az Önálló laboratórium alatt is szeretném folytatni. A fentebb említett terveken kívül számtalan ötlet van még, amelyet meg lehetne valósítani.

10 Hivatkozások

- [1] Object Management Group (OMG), "What is DDS?," . [Online]. Available: <https://www.dds-foundation.org/what-is-dds-3/>.
- [2] RTI, „DDS: An Open Standard for Real-Time Applications,” . [Online]. Available: <https://www.rti.com/products/dds-standard>.
- [3] BeagleBoard.org Foundation, „BeagleBone Black,” 14 06 2022. [Online]. Available: <https://beagleboard.org/black>.
- [4] Hitec RCD USA, „HS-81 Standard Micro Servo,” Hitec, [Online]. Available: <https://hitecrcd.com/products/servos/micro-and-mini-servos/analog-micro-and-mini-servos/hs-81/product>.
- [5] „DC-DC Buck Module,” [Online]. Available: https://www.banggood.com/DC-DC-Buck-Module-6-24V-12V-or-24V-to-5V-3A-USB-Step-Down-Power-Supply-Charger-Efficiency-97_5-pencent--p-1103973.html.
- [6] „5V 100DB Loud Alarm Buzzer,” [Online]. Available: <https://www.banggood.com/5V-100DB-Loud-Alarm-Buzzer-With-WS2812-Colorful-LED-for-RC-Drone-NAZE32-F3-F4-F7-Flight-Controller-p-1417383.html>.
- [7] „AUSTAR LED Light,” [Online]. Available: <https://www.banggood.com/AUSTAR-LED-Light-Aluminum-Alloy-Frame-For-CC01-or-D90-or-SCX10-or-4WD-RC-Car-Parts-p-976648.html>.
- [8] C. RTI, „Getting Started guide,” Real-Time Innovations, Inc., [Online]. Available: https://community.rti.com/static/documentation/connex-dds/current/doc/manuals/connex-dds_professional/getting_started_guide/cpp98/before.html.
- [9] Real-Time Innovations, „Code Generator,” Real-Time Innovations, [Online]. Available: <https://www.rti.com/products/tools/code-generator>.
- [10] Chocolatey Software, Inc., „The package manager for Windows,” Chocolatey Software, Inc., 2022. [Online]. Available: <https://chocolatey.org>.
- [11] Microsoft, „Windows Subsystem for Linux Documentation,” Microsoft Corporation, 2022. [Online]. Available: <https://learn.microsoft.com/en-us/windows/wsl/>.
- [12] D. project, „Debian,” Software in the Public Interest, Inc., [Online]. Available: <https://www.debian.org>.
- [13] „Ubuntu,” Canonical Ltd., 2022. [Online]. Available: <https://ubuntu.com>.
- [14] „Using SCP to copy files between computers, with examples,” SSH COMMUNICATIONS SECURITY, INC., [Online]. Available: <https://www.ssh.com/academy/ssh/scp>.
- [15] „SSH Secure Shell home page, maintained by SSH protocol inventor Tatu Ylonen,” SSH COMMUNICATIONS SECURITY, INC., [Online]. Available: <https://www.ssh.com/academy/ssh>.
- [16] ftsrg, „BeagleBone Wiki,” [Online]. Available: <https://github.com/ftsrg-edu/cpsdbz/wiki/BeagleBone.wiki>.