BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

Control and balancing – applied to Lego robots

Abstract

A szakdolgozat célja egy egyensúlyozó, kétkerekű MINDSTORMS EV3 robot irányításának megoldása hálózaton keresztül, telefonos alkalmazás segítségével.Ennek a célnak az eléréséhez szükséges két nagyobb feladat elvégzése. A kommunikációt megvalósító telefonos alkalmazás illetve egyensúly megtartása irányítás alatt.

Az alkalmazás és az egyensúlyozási algoritmus Java-ban íródott. A robot irányításáért felelős az EV3 vezérlőegység, amelyen Linux alapú leJOS keretrendszer fut. A robot PID (`Proportional Integral Derivative`) kontroller alkalmazásával és giroszkóp szenzor használatával marad egyensúlyban. Ahhoz hogy egyensúlyba maradjon a robot sebessége, dőlési szöge, szög változásának a sebessége 0-hoz kell közelítsen illetve annak az érdekében, hogy egy helyebe maradjon pozíciója is 0-hoz kell tartson. A PID bemeneti értékét az aktuális hiba adja meg, amelyet négy különböző komponens határozz meg: szög, szögsebesség, robot sebesség és pozíció. A komponensek megfelelő módosítása lehetővé teszi a robot mozgásának manipulálását.

Mobil részre egy telefonos alkalmazást fejlesztettünk, amely hálózaton keresztül kapcsolódik a robothoz és küldi a megfelelő parancsokat. Az alkalmazás és a robot közti kapcsolat létrehozásának megkönnyítésére használjuk az SSDP(`Simple Service Discovery Protocol`) automatikus kapcsolódást UPnP(`Universal Plug and Play`) segítségével.

This work is the result of my own activity. I have neither given nor received unauthorized assistance on this work.

JULY 2016 MÁRTON ZETE-ÖRS

ADVISOR:
ASSIST PROF. DR. HUNOR JAKAB

BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

Control and balancing – applied to Lego robots

SCIENTIFIC SUPERVISOR:
ASSIST PROF. DR. HUNOR JAKAB

STUDENT:
MÁRTON ZETE-ÖRS

JULY 2016

UNIVERSITATEA BABEŞ-BOLYAI, CLUJ–NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

Control şi echilibrare – aplicat la roboţi Lego

CONDUCĂTOR ŞTIINŢIFIC:
LECTOR DR. HUNOR JAKAB

ABSOLVENT:
MÁRTON ZETE-ÖRS

IULIE 2016

BABEŞ-BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Államvizsga-dolgozat

Kontroll és egyensúlyozás – alkalmazás Lego robotnál

TÉMAVEZETŐ:

DR. JAKAB HUNOR,
EGYETEMI ADJUNKTUS

SZERZŐ:

MÁRTON ZETE-ÖRS

2016 JÚLIUS

Tartalomjegyzék

1. fejezet

Bevezető

A szakdolgozat által bemutatásra kerül egy projekt, amely a LEGO MINDSTORMS EV3[2] készletből épített kétkerekű egyensúlyozó robot irányítását teszi lehetővé hálózaton keresztül, telefonos alkalmazás segítségével. Emellet sor kerül a robot főbb elemeinek a bemutatása, felhasznált technológiák és a vezérlést kezelő Android alkalmazás.

A kétkerekű robot a Gyro Boy[1] modell alapján készült el. Főbb komponensei közé sorolható az EV3 vezérlőegység[2], giroszkóp szenzor[3] és a két nagy szervomotor[4], amelyek párhuzamosan vannak elhelyezve. Az előbb említett elemekről bővebben szó lesz a 2.3 illetve 2.4 fejezetben. Mivel a kerekek párhuzamosan helyezkednek el ezért nem marad egyensúlyi állapotban. E probléma megoldására alkalmas a PID[5] szabályzó algoritmus használata, amely ipari körökben elterjedt a viszonylag egyszerű felépítéséért, kezelhetőségéért és implementálhatóságáért. A szabályzó bemeneti értékét az aktuális hiba határozza meg, amelyet négy értékből határozunk meg: szög, szögsebesség, robot sebesség és a robot pozíciója, amelyek külön-külön súlyozva vannak. A szög és szögsebesség érték meghatározásához szükséges a giroszkóp szenzor és a szervomotorok beépített szenzorjai teszik lehetővé a sebesség és pozíció megállapítását.

A robot irányításának érdekében szükséges a PID szabályzó algoritmus módosítása és esetleges újabb szabályzók bevezetése annak érdekében, hogy irányítás alatt ne veszítse el az egyensúlyi állapotát. A felhasználónak lehetőséget ad a projekt Android alkalmazása, hogy hálózaton, socketeken keresztül csatlakozzon a robothoz és az irányításnak megfelelő adatokat továbbítsa. Ezen adatok beviteli módját egy "touch joystick" teszi lehetővé, amellyel négy irányba lehetséges a robot vezérlése. Az adatok védelmét, a tovább bővíthetőséget illetve a szerializációt a Google Protocol Buffers[6] biztosítja. A Protocol Buffers platform és nyelvfüggetlen, könnyen kezelhető és gyors. Lehetőséget nyújt az adatok tetszőleges felépítésére, amelynek a forráskódját egy speciális generátor segítségével könnyen kigenerálható. E strukturált adatok írását illetve olvasását biztosítja a generált kód.

Az EV3 készlethez biztosított a LEGO MINDSTORMS egy grafikus felületet, amellyel a kisebb korosztály "programozhatja" a saját kezűleg épített robotokat. E projekt esetében leJOS [7] keretrendszer

1. http://robotsquare.com/wp-content/uploads/2013/10/45544_gyroboy.pdf
2. http://lego.wikia.com/wiki/45500_EV3_Intelligent_Brick
3. http://shop.lego.com/en-US/EV3-Gyro-Sensor-45505
4. http://lego.wikia.com/wiki/45502_EV3_Large_Servo_Motor
5. https://en.wikipedia.org/wiki/PID_controller
6. https://developers.google.com/protocol-buffers/
7. https://en.wikipedia.org/wiki/LeJOS

fut az EV3 vezérlőegységen. A leJOS Linux alapú és magába foglalja a JVM-t (Java virtual machine), amely lehetővé teszi, hogy a robot programozható legyen Java-ban.

A dolgozat 3 fejezetből áll. Az első fejezet bevezetőként szolgál a projektbe.

A második fejezet röviden bemutatja a LEGO megalakulását, a LEGO MINDSTORMS által kifejlesztett generációkat majd bemutatja az EV3 készlethez tartozó motorokat és szenzorokat. Tartalmazza azon eszközök részletes leírását amelyek a projekt által használatosak.

A harmadik fejezet által bemutatásra kerül a PID szabályzó működése és használata e projekt esetén.

2. fejezet

LEGO

Összefoglaló: *A fejezetben, bemutatásra kerül a LEGO [1] megalakulása, fejlődése. E mellet sor kerül a LEGO MINDSTORMS által kifejlesztett technológiák bemutatása, amelyek közül egy pár felhasználásra kerül a továbbiakban.*

2.1. LEGO megalakulása

A LEGO története 1932-ben kezdődött, Ole Kirk Christiansen[1] asztalos vállalkozást alapított Dánia, Billund nevezetű falujában, amely a fa játékok gyártásával foglakozott. 1934-ben vette fel a cég a LEGO nevet, amely a LEG GODT Dán szóból ered. 1947 után kezdtek el műanyagból előállítani játékokat amelyek fő célja az volt, hogy könnyedén egymásba illeszthetőek illetve szétszedhetőek legyenek. Az alapító fia Godtfred Christian lett az igazgató 1958-ban, ekkor alakult meg a ma ismert LEGO vállalat és vezetésével fellendült a cég. Az első számítógép által vezérelt robot 1986-ban jelent meg, amelyet követően 1988-ban a LEGO és az MIT (`Massachusetts Institute of Technology`) együttműködésével elkezdődött az "inteligens tégla" fejlesztése, mely lehetőséget ad a programozhatóságra.

2.2. LEGO MINDSTORMS

A LEGO első játéka amit forgalmazott Ole Kirk Christians vezetésével a fából készült "LEGO Duck" évekkel később megjelentek a műanyag játékok. Ma már világszerte ismert a LEGO építőjáték, egymással összeilleszthető, kombinálható elemeket tartalmaz így szinte bármi megépíthető belőle és rendkívül hasznos oktatási eszköz.

A LEGO MINDSTORMS, egy programozható robotikai építőkészlet. Lehetőséget ad, hogy megépítsd, programozd és irányítsd a robotot. 1998-ban jelent meg az első generáció az RXC (`Robotic Command eXplorers`), akkoriban nagy előrelépés volt mivel, teljesen autonóm, képes számítógép nélkül is működni de mára már elavult. Az évek során sokat fejlődött és ma már egyre több oktatási intézmény használja a robotika oktatásban.

2.2.1. RXC

Az RXC amelyet a 2.2.1 ábra szemlélteti, az első generációs LEGO MINDSTORMS. Ma már nem igen használatos, elavult programozható vezérlőegység, mivel csupán 8 bites mikrokontroller és 32 Kbyte

1. http://lego.wikia.com/wiki/Ole_Kirk_Christiansen

RAM-al rendelkezik. Tartalmaz három szenzor és három motor csatlakozó portot. Ezek mellet egy LCD kijelzőt amin látható az akkumulátor töltöttségi szintje, a bemeneti és kimeneti portok állapota illetve egyéb információk megjelenítésere is alkalmas.

2.1. ábra. RXC brick

2.2.2. NXT

A második generáció a 2.2.2 ábrán látható, az NXT, 2006-ban jelent meg illetve az NXT 2.0 2009-ben, amely több építőelemet és szenzort tartalmaz. Több mindenben különbözik az NXT az RXC-hez képest. Szembetűnő esztétikai változások történtek, javítottak a kapacitáson, növelték a portok számát és megjelentek a szervomotorok.

Az NXT tartalmaz négy szenzort, három motor csatlakozót és egy 2.0 USB portot. Beépített bluetooth kommunikációs adatperrel rendelkezik, megjelent a grafikus kijelző és tartalmazott egy 8Khz hangszórót. Az NXT 32 bites AMTEL ARM7 mikrokontrollerrel, 256 Kbyte flash memóriával és 64 Kbyte RAM-al rendelkezik.

2.2. ábra. NXT brick

2.2.3. EV3

A harmadik generáció, az EV3 2013-ban jelent meg, a 2.2.3 ábra szemlélteti. Az "EV" evolúciót jelenti és a 3, hogy harmadik generáció. A legfejeltebb LEGO MINDSTROMS programozható építőelem amely kezeli a motorokat, szenzorokat és lehetőséget ad Wi-Fi-n vagy Bluetooth-on keresztül való kommunikációra, ezért is választottuk ezt a generációt.

Az EV3 egy ARM9 nevű processzorral van felszerelve, amely 300 Mhz, 16 Mbyte flash memóriával és 64 Mbyte RAM-al rendelkezik. A Linux alapú operációs rendszere nagyban segíti a programozhatóságát. Egy 2.0 mini USB PC port van a számítógéppel való kommunikációhoz, aminek átviteli sebessége 480 Mbit/s, az SD kártya olvasója 32 Gbyte-ot ismer fel. Tartalmaz egy USB portot amely lehetővé teszi, hogy használjunk Wi-Fi dongle-t, ezáltal megkönnyíti a programok kitelepítését, fájlok kezelését és lehetséges a kommunikáció okos készülékekkel is.

Az NXT-hez képest sokat fejlődött, nagyobb a kapacitása, lehetőség van Wi-Fi és SD kártya használatára. Ezek mellet négy szenzor port van három helyett és négy motor csatlakozó port. Nagyobb LCD kijelzőt raktak és a felületén hat gomb van négy helyett, könnyítve az operációs rendszer menüjének kezelését.

2.3. ábra. EV3 brick

2.3. Ev3 motorok

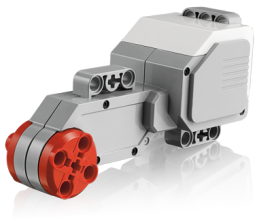
Az NXT generációval megjelentek a szervomotorok, amelyek legfőbb tulajdonsága a pontosság. Két típusa jelent meg a közepes és a nagy szervomotor. Mind a kettőnél megmaradt a pontosság de a méretük, az erejük és a reagálási idejük eltérnek. Tovább fejlesztették a motorokat, amelyek az EV3 generációval jelentek meg. Az EV3 nagy motorjának teljesítménye megegyezik az NXT-vel de a felépítését optimalizálták, illetve a EV3 közepes motorja teljesen uj az NXT-hez képest.

2.3.1. Nagy motor

A nagy motor, amely a 2.3.1 ábrázol, erőteljes, ideális a általunk megépített robot irányítására. A motorba beépített forgásérzékelő által információkat lehet lekérni az pillanatnyi állapotáról, amely fokban vagy teljes fordulatban mér.

A nagy motorok sebessége 160-170 rmp, forgatónyomaték 20 N/cm és ha blokkolt állapotba kerül akkor a nyomatéka 40N/cm. A motor beépített forgásérzékelője lehetővé teszi a pontos vezérlést, +/- 1 fok pontossággal.

A robot mozgatását a nagy motorok segítségével fogjuk elérni, ha bár lassabbak mint a közepes motorok de nagyobb a nyomatékuk és a precizitásuk azonos.



2.4. ábra. Nagy motor

2.3.2. Közepes motor

A közepes motor, amely a 2.3.2 ábrázol, tulajdonsága megegyezik a nagy motorjaival. Ugyancsak tartalmaz forgásérzékelőt és lehetséges a helyzetmeghatározás 1 fok eltéréssel, viszont alacsonyabb terhelésre tervezték, forgatónyomatéka 8 N/cm, ha blokkolt állapotba kerül akkor a nyomatéka 12 N/cm és a sebessége 240-250 rmp.

2.5. ábra. Közepes motor

2.4. Ev3 szenzorok

Az NXT csomagban számos szenzor megtalálható, ilyenek mint az érintés, hang, szín és ultrahang szenzorok, amelyeket felhasználva több különböző funkcionalitásokkal bővíthetjük a megépített robotunkat. Az EV3 megjelenése magával hozott még két új szenzort, a giroszkóp és az infravörös szenzorokat. Ezek mellet fejlesztették a már meglévő szenzorokat is.

2.4.1. Színszenzor

A digitális színszenzor, amely a 2.4.1 ábrázol, hét különböző színt ismer fel. A színfelismerő funkcionalitásán kívül mérhető vele a fényvisszaverődés erőssége vagy a környezeti fény intenzitását. E három

üzemmód lehető teszi a felhasználóknak, hogy akár kövessen egy fekete vonalat vagy megkülönböztetésen szín szerint tárgyakat és még sok más felhasználási lehetősége van.

2.6. ábra. Szín szenzor

2.4.2. Giroszkóp szenzor

A digitális giroszkóp szenzor, látható a 2.4.2 ábrán, amelyet mi is felhasználunk, az EV3-al egy időben jelent meg. A giroszkóp igen elterjedt szenzor, megtalálható okostelefonokban, különféle navigációs rendszerek vagy akár irányításért felelős vezérlőegységek használják. A EV3 giroszkóp szenzora egy tengely mentén tud mérni. Látható a 2.4.2 ábrán, hogy a giroszkópon fel van tüntetve két nyíl és ennek segítségével betudjuk állítani a pozícióját, ha jobb oldali nyíl irányába mozdítjuk akkor mínusz értéket kapunk és ha ellentétesen akkor egyértelműen pozitívat.

A giroszkóp három módban használható. Mérhető az elfordulási szög fokban -90 és 90 intervallumba, a szög gyorsulása illetve lehetséges a két mód használata egyszerre.A pontossága szögmérés esetén +/-3 fok, szög gyorsulása eseté pontosabb. Maximális információ megosztási sebessége 440 fok/másodperc és a mintavételezési sebessége 1 kHz. A sebességét kihasználva simítjuk a +/-3 fok szórását oly módon, hogy többször mintavételezünk.

2.7. ábra. Giroszkóp szenzor

2.8. ábra. A giroszkóp szögelfordulási mutatója

2.4.3. Nyomásérzékelő

Az analóg nyomásérzékelője egy piros gombbal van felszerelve ami látható a 2.4.3 ábrán. Funkcionalitása nem túl bonyolult viszont annál hasznosabb. Érzékeli a gomb lenyomását illetve felszabadulását, amelyet egy integrált számlálóval segítségével nyomon lehet követni.

2.9. ábra. Nyomásérzékelő

2.4.4. Ultrahang szenzor

A digitális ultrahang szenzor látható a 2.4.4 ábrán, amely +/- 1 cm hibával meghatározza, hogy milyen távolságra van az előtte lévő tárgy. A hatótávolsága 250 cm és működése a magas frekvencia kibocsátására alapszik. Számolja az előtte lévő tárgyról visszaverődő frekvenciák érkezésének idejét, ezáltal határozza meg a távolságot. Az akadály pontos irányának meghatározása úgy lehetséges ha több mérést végzünk oldal mozgással. Így lehetőség van, hogy kiszámoljuk az akadály irányát.

2.10. ábra. Ultrahang szenzor

2.4.5. Infravörös szenzor

A digitális infravörös szenzor látható a 2.4.5 ábrán, amelynek a maximális hatótávolsága 70 cm jelentősen kisebb az ultrahang szenzorhoz képest. Funkcionalitása inkább vezérlésre alkalmas, 2 m hatótávolságról is érzékeli a LEGO MINDSTORMS saját infravörös jeladóját, amely latható a 2.4.5 ábrán.

2.11. ábra. Infravörös szenzor

2.12. ábra. Infravörös jeladó

3. fejezet

Egyensúlyozás

Összefoglaló: *E fejezet célja, hogy bemutassa a PID szabályzó algoritmus működését. E mellet sorkelűr az egyensúlyozás működésének leírására, valamint hogy miként lehet felhasználni ez esetbe a PID szabályzó algoritmust.*

3.1. PID szabályzó algoritmus

A PID `(Proportional Integral Derivate)` elterjedt algoritmus az ipari szabályzó körökben köszönhetően a viszonylag egyszerű félépítéséért és a könnyen kezelhetőségéért. Számos felhasználási lehetősége van, ilyenek a nyomás, hőmérséklet, sebesség szabályozás.

A PID, mint a nevéből is látható három tagból tevődik össze: P `(Proportional)` proporcinális tag, I `(Integral)` a hibák integrált tagja és a D `(Derivate)` a hibák időbeli változásának derivált tagja. Több változata is van, azok az esetek mikor nem valósulnak meg az előbb felsoroltak mindegyike, ilyenkor beszélünk P, PI, PD szabályokról.

PID szabályzó matematikai képlete:

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{d}{dt} e(t),$$

ahol bemeneti hiba jelöle az e(t), u(t) a kimenet. A K_P, K_I, K_D rendre a proporcinális, integrált és a derivált súlyát határozzák meg.

Esetünkben a P az aktuális hiba, I a hibák összességének integráltja, D az aktuális és az előző hiba időbeli változásának deriváltja. A tagok súlyát meghatározó konstansok jelentős befolyással vannak a kimenetre.

A K_P befolyásolja a szabályzó érzékenységét, instabilitást okozhat. A magas erősítés, amely nagy oszcillációt idéz elő, ellentétben áll az alacsony erősítéssel, hiszen az utóbbi kicsi oszcillációt idéz elő. Ezen estekben a robot elveszti az egyensúlyi állapotát.

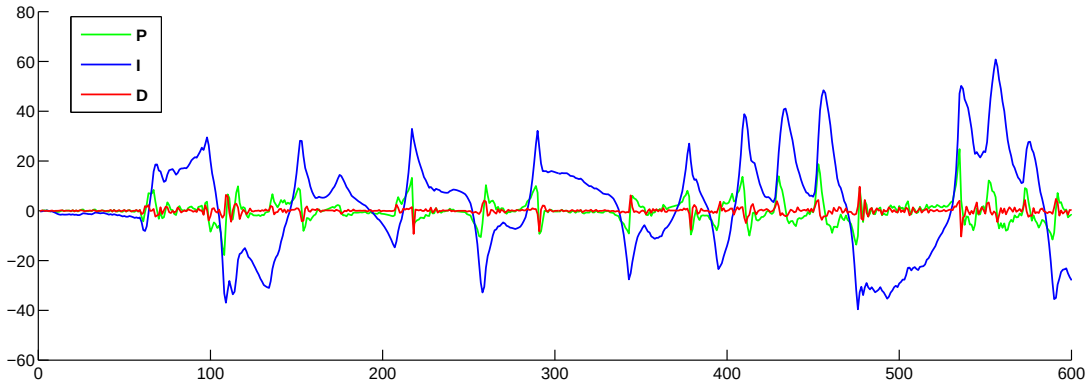
A K_I erősítő esetén figyelembe kell venni, hogy az integrált tag folytonosan nő és a túl nagy érték túllendülést eredményez.

A K_D a hiba változásának mértékét határozza meg. Amennyiben az aktuális és előző hiba közti különbség nagy, hirtelen változást idéz elő a szabályzó kimenetén. A derivált szerepe, hogy csökkentse a kimenet hirtelen változását, megakadályozva a hirtelen sebesség növekedést. Magas erősítés okozhat instabilitási problémákat.

A 3.1 ábrán látható a PID szabályzó tagjainak változása. Az I tag domináns a P illetve D tagokkal

szemben, látható hogy az integrált tag folyamatosan növekszik. Annak érdekében, hogy túllendülést okozzon az integrált, a derivált tagnak kell ellensúlyozza.

A P tag az előbb említett két tag nagysága között változik. E tag esetén a túl nagy oszcilláció azt idézné elő, hogy a robot hirtelen reagálna az eldőlésre ezért átlendítené ellentétes irányba annyira, hogy eldőlne. Ezzel ellentétbe, a kisebb oszcilláció hatására megfigyelhető lenne a roboton, hogy a vártnál később reagál az egyensúly korrigálására és ebben az esetben is eldőlne.



3.1. ábra. PID tagjai egyensúlyi állapotban

4. fejezet

Megvalósítás

Összefoglaló: *A projekt egy EV3 készletből épített kétkerekű egyensúlyozó robot irányítását valósítja meg hálózaton keresztül, telefonos alkalmazáson segítségével. E fejezet alatt bemutatásra kerülnek a megvalósítás során felmerült problémák, ezek megoldása és a felhasznált technológiák.*

4.1. EV3 programozása

A LEGO MINDSTORMS kifejlesztett egy programozási környezetet, mely célja, hogy a megépített robotot különböző funkcionalitásokkal lehessen felruházni. E környezet lehetővé teszi a kisebb korosztály számára is a robotok programozását. Különböző grafikus elemekből úgynevezett blokkokból épül fel a program, amely USB-n keresztül kitelepíthető az EV3 vezérlőegységen futó LEGO MINDSTORMS által fejlesztett firmware. Az előbb említett programozási környezet nem alkalmas komplexebb problémák megoldására. Ezért több firmware-t is kifejlesztettek melyek magas szintű programozási nyelveket támogatnak. Esetünkben a leJOS firmware-t használjuk.

A leJOS firmware-t José Solórzano hozta létre 1999 végén és azóta is folyamatosan fejlesztik. Linux alapú, nyílt forráskódú, magába foglalja a JVM-t (Java virtual machine), a neve is rámutat a Java programozhatóságra JOS(Java Operating System). Lehetővé teszi, a robot programozását Java-ban, támogatja a objektum orientált programozást. Mindezek lehetővé teszik a socket alapú komunikációt, szinkronizálhatóságot, szálak alkalmazását és Java típusok használatát. A leJOS hozzáférést biztosít az EV3 vezérlőegységen levő portokhoz, ezáltal kezelhetjük a giroszkóp szenzort (4.2 ábra) illetve a nagy motorokat (4.1 ábra). Elrejti a szenzorok implementációját, lehetővé téve a magas szintű absztrakció használatát.

Listing 4.1. Az `A` porton keresztül hozzáférés a motorhoz

```
1 EncoderMotor motor = new UnregulatedMotor(MotorPort.A);
```

Listing 4.2. Az `S2` porton keresztük hozzáférés a giroszkop szenzorhoz

```
2 EV3GyroSensor gyroSensor =  new EV3GyroSensor(SensorPort.S2);
```

A 4.3 és a 4.4 ábrákon látható a leJOS firmware által biztosított két mód, a giroszkóp szenzor használatához. A `rate` mód a szögsebességet méri, amelyet szög/másodperce -ben kapunk meg. A 4.3 ábrán látható a mód kiválasztása és ezt követően a mintavételezés, melynek eredménye a `sample` tömb első eleme lesz. Használható `angle` módban is, amely 4.4 ábrán látható. E mód a szenzor kezdő orientációjához képest mér. Mintavételezés hasonlóan történik, mint az előbb említett módnál, annyi különbséggel

hogy ez esetben szöget mér a kezdő pozíciójához képest. Mindkét mód esetében a `reset` metódus hívással újra kalibrálhatjuk a szenzort.

Listing 4.3. Giroszkóp szenzor `rate` mód használata

```
float[] sample = new float[1];
EV3GyroSensor gyroSensor =  new EV3GyroSensor(SensorPort.S2);

SampleProvider gyroMode = gyroSensor.getRateMode();

gyro.fetchSample(sample, 0);
```

Listing 4.4. Giroszkóp szenzor `angle` mód használata

```
float[] sample = new float[1];
EV3GyroSensor gyroSensor =  new EV3GyroSensor(SensorPort.S2);

SampleProvider gyroMode = gyroSensor.getAngleMode();

gyro.fetchSample(sample, 0);
```

A nagy motorok esetében is két módot biztosít a leJOS firmware, amelyek a 4.5 és 4.6 ábrán láthatóak. Rendre értelemszerűen a nem szabályzott és a szabályzott módok.

A motor nem szabályzott mód (4.5 ábra) használata esetén az irányítást a `setPower` metódus hívással valósítjuk meg, amelynek ha az átadott érték pozitív akkor előre forog illetve ha negatív akkor hátra. A `resetTachoCount` valamint `getTachoCount` metódusokkal kezeljük a motorban levő forgás mérő szenzort, melynek számlálójának értékét lekérhetjük vagy lenullázhatjuk. Esetünkbe szükséges a pozíció illetve a sebesség meghatározása, e megvalósítását az előbb említett metódus teszi lehetővé. Az eltelt idő, a kerék sugara és a fordulat szám segítségével, amelyet `getTachoCount` térit vissza, kiszámítható a robot sebessége valamint a pozíciója. A `stop` metódus hívása esetén a motor blokkolt állapotba kerül.

A motor szabályzott mód (4.6 ábra) esetén némely funkcionalitás eltér az előbb említett módhoz képest. Ez esetben a motor sebessége fok/másodperc-ben megadható a `setSpeed` metódus által. A forgás sebessége függ az akkumulátor töltöttségi szintjétől, a maximális sebessége 740 fok/másodperc. A `rotate` metódussal adjuk meg, hogy hány fokot forduljon a motor. E metódusnak két változata van. Megadhatjuk csak a fokot vagy egy logikai értékkel megadható, hogy miután elérte az adott forgási fokot magától megálljon a motor. Ez esetben ha forgás közben meghívódik egy másik metódus, mely parancsot ad a motornak akkor az előbb kiadott utasítás végrehajtása leáll. A `waitComplete` metódussal lehetőség van, hogy bevárja a forgás befejezését, tehát addig vár míg végrehajtja a motor az azelőtt kapott utasítást. Az előbb említett mód esetén a pozíciót és a sebességet kikellet számolni. Ez esetben lehetőség van ezen értékek lekérésére a `getPosition` és `getSpeed` metódusok által.

Listing 4.5. A nagy motor `regulated` mód használata

```
EncoderMotor motor = new UnregulatedMotor(MotorPort.A)

motor.resetTachoCount();
motor.setPower(40);

int tacho = motor.getTachoCount();

motor.stop();
```

Listing 4.6. A nagy motor `unregulated` mód használata

```
RegulatedMotor motor = new EV3LargeRegulatedMotor(MotorPort.A);

motor.resetTachoCount();
motor.setSpeed(600);
motor.rotate(720);
motor.waitComplete();
motor.rotate(-460, true);

float position = motor.getPosition();
```

Annak érdekében, hogy az EV3 vezérlőegységen futtassuk és könnyedén kitelepítsük a programokat az Eclipse IDE fejlesztői környezetre van szükség és a leJOS plugin-ra.

Mivel az EV3 vezérlőegységen az alapértelmezett firmware van telepítve ezért külön SD kártyára felkel telepíteni a leJOS-t. Legalább 2GB-os SD kártya de ne legyen 32GB-nál nagyobb és ne SDXC típusú legyen, mert nem ismeri fel az EV3 hardware. Az SD kártyát szükséges formázni FAT32 típusú partícióra. A leJOS számítógépre való telepítése során szükség lesz az 1.7 JDK-ra(Java Development Kit). Az előkészített program segítségével feltelepíthető a leJOS firmware az SD kártyára, ehhez még kell a JRE(Java Runtime Environment) is. Sikeres telepítés után az SD kártyát behelyezve az EV3 vezérlőegységbe elindítható a firmware, ha az alapértelmezett rendszer indul el akkor megkel ismételni az SD kártyára való telepítést. Ezt követően telepítsük az Eclipse plugin-t majd berakjuk az EV3_HOME-t környezeti változónak és a bin könyvtárat a path-be.

4.2. Androidos alkalmazás és kommunikáció

A projekt része egy telefonos alkalmazás, mely célja hogy hálózaton keresztül kapcsolódjon a robothoz és kommunikáljon vele. Az alkalmazás Android stúdióba készítettük és a kommunikációt java socketen keresztül valósítottuk meg, amit a leJOS, Linux alapú firmware tesz lehetővé.

Az alkalmazást elindítva megadhatjuk a robot IP és PORT címét amin keresztül csatlakozik. A csatlakozás során ellenőrizzük a bekért adatok helyes formátumát és hogy lehetséges vagy sem a kapcsolat. Az alkalmazás kezelését elősegíti egy általunk létrehozott "Remember me" funkcionalitás, mely célja hogy a legutóbbi IP és PORT címet visszatöltse az alkalmazás élindításakor. E megvalósítása a SharedPreferences API-n keresztül történik, érték és kulcs párok alapján tárolódnak fájlba az adatok. Ezen adatok hozzáférési pontja a SharedPreferences objektum, amely könnyen kezelhető metódusokat biztosít ezek olvasására illetve írására.

A sikeres kapcsolódást követően egy 2D-s joystick segítségével lehet irányítani a robotot négy irányba. A joystick vizuális megjelenítésére két kört rajzolunk ki a telefon képernyőjére canvas segítségével (4.1 ábrán látható), amely felületéhez hozzárendeljük a megfelelő eseményfigyelőt(OnTouchListener). Annak érdekében, hogy a felhasználó ne tudja kimozdítani a nagyobb körön belüli kisebb kört átalakításokat végzünk koordináták között.

A képernyőt megérintve az eseményfigyelő által megkapjuk az x és y koordinátákat, ezeket a pontokat átalakítjuk polárkoordinátákba:

$$(x, y) \Longrightarrow (r, \varphi)$$

$$r = \sqrt{x^2 + y^2}$$

$$\varphi = arctg(y, x)$$

Tudva a két kör sugarának különbségét és a polárkoordinátákat, leellenőrizhető hogy a kis kör nagy kör sugarán kívül esik vagy sem. Abban az esetben ha a nagy körön kívül esik a kirajzolási pont, akkor a sugár mentén rajzoljuk ki a kisebb kört a szög függvényében. Ehhez szükséges polárkoordinátából átalakítani euklideszi koordinátába:

$$(r, \varphi) \Longrightarrow (x, y)$$

$$x = R * cos(\varphi)$$

$$y = R * sin(\varphi),$$

ahol R a két kör sugarának különbsége.

4.1. ábra. Joystick vizuális megjelenítése

Tudva a mozgatás irányát, socketen keresztül küldjük a parancsokat. Az adatok szerializációját illetve titkosítását a Google Protocol Buffers által biztosítjuk, amely lehetővé teszi, hogy megszerkesszük az adatok struktúráját majd egy speciális generátorral létrehozzuk ezen strukturált adatok kezelésére a hozzá tartozó Java osztályt. E létrehozott osztályon keresztül könnyedén kezelhetjük a strukturált adatokat.

A Google Protocol Buffers használatához létre kell hozzunk egy `.proto` kiterjesztésű állományt

(4.7 ábra), amelyben definiáljuk az adataink struktúráját. Az állomány első sorában deklaráljuk a csomag(package) nevét, második sorban konkrétan megadjuk a Java csomag hierarchiáját és a harmadik sorban megadjak az osztály nevet, amellyel rendelkezni fog a legenerált osztály. Abban az esetben ha nem adunk meg konkrét nevet a `java_outer_classname` mezőben akkor a `.proto` fájl nevét fogja megkapni. A további sorokban megadjuk az adattagokat, amelyek rendelkeznek típus névvel ez lehet bool, int32, float, double és string. Minden attribútumnak megadunk egy számot, amely egyedi azonosítóként szerepel a bináris kódolás során. Ezeken kívül megadható három típus mező minden attribútumnak, amelyek a következők: `required, optional, repeated`. A `required` mezővel beállíthatjuk, hogy az adott attribútum kötelezően értéket kapjon különbem RuntimeException vagy IOException hibát eredményez. A `optional` mezőt annak az adattagnak állítjuk be, amely nem biztos, hogy értéket kap futási időben, ebben az esetben megadhatunk `.proto` állományban egy alapértelmezett értéket ennek az adattagnak. A `repeated` típussal lehetséges annak a jelzése, hogy az adott adattag ismétlődni fog.

Listing 4.7. Az adatok strukturáját definiáló .proto állomány

```
package protobuf;
option java_package = "app.cs.ubb.edu.ev3.protobuf";
option java_outer_classname = "DataProtos";

message Data {

        required int32 left = 1;
        required int32 right = 2;
        required float forward = 3;
        required float speed = 4;
        required float angle = 5;
        required bool client = 6;
}
```

4.3. Egyensúlyozási problémái

Gilyen Hunor által elkészített licensz-dolgozat során, megépült a kétkerekű LEGO MINDSTORMS EV3 Gyro Boy, amely képes egy helyben megtartani az egyensúlyi állapotát. E cél megvalósítására a PID szabályzót implementálta le Java-ban.

A dolgozat során az általa elkészített projektet vettük alapnak és ebből indultunk ki, hogy megvalósítsuk négy irányba való elmozdulást úgy, hogy megtartsa közbe az egyensúlyi állapotát.

Az általa definiált projekt struktúra nem volt előnyös számunkra. Egy központi osztálya volt, amelyben definiált egy belső osztályt. Ezen osztályon belül megvalósította magát az egyensúlyozást a PID szabályzó által és ez az osztály implementálta a `Runnable` interfészt. Ezen kívül létrehozott két modell osztályt a szenzorok által kapott értékek eltárolására. A program indulásakor létrehozott egy `Thread` objektumot, amelynek átadta az egyensúlyozást megvalósító osztály példányát.

A projekt struktúrájának javítása érdekében külön választottuk a PID szabályzó algoritmust és a giroszkóp szenzor olvasásához szükséges metódusokat. E két osztály külön szálon indítjuk el. A szinkronizálást a `CyclicBarrier` segítségével valósítottuk meg, amely a Java 7 része. A példányosításakor (4.8 ábrán látható) megadható, hogy hány szálal szeretnénk dolgozni. Szinkronizálás során a `await` metódus hívással a szál várakozik mindaddig míg az összes szál nem hívja meg ez a metódus. Mikor egy szál meghívja ez a metódus akkor a `CyclicBarrier` növeli a számlálóját és mindaddig várakozik a

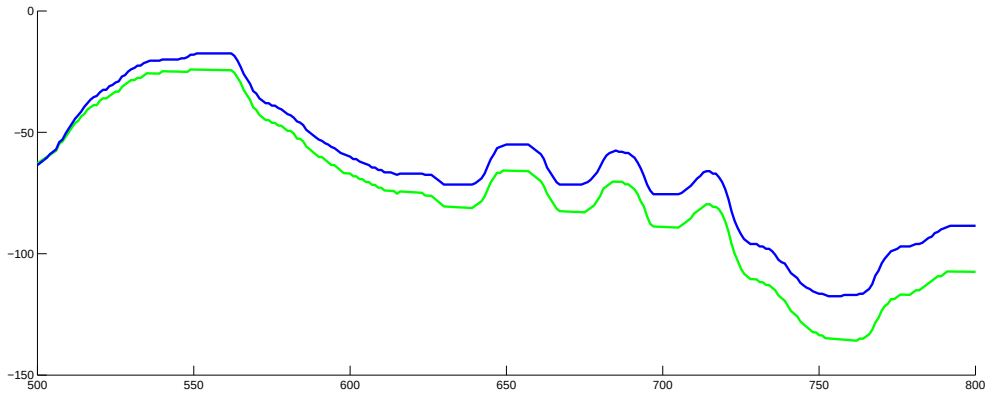
szál amíg ez a számláló nem éri el a példányosításkor megadott értéket. Valamint létrehoztunk egy központi osztályt, amely feladata a szálak kezelése és a szenzorokhoz illetve motorokhoz való hozzáférést biztosító objektumok példányosítása.

Listing 4.8. CyclicBarrier példányosítása

```
2 CyclicBarrier barrier = new CyclicBarrier(2);
```

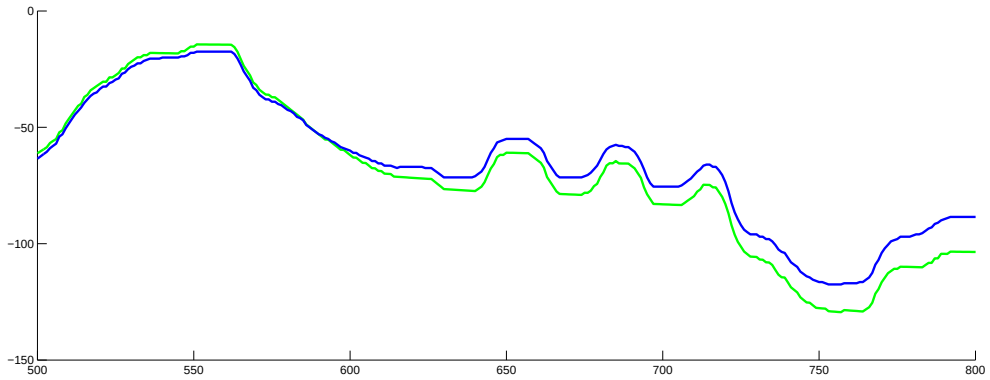
Kezdetben az volt az elképzelésünk, hogy külön válasszuk a jobb és bal motorokat irányító algoritmust, ezáltal egymástól függetlené téve őket. Ehhez szükséges volt a szabályzó algoritmus általánosítása illetve a két motor szinkronizálásának megoldása. Mivel külön szálon fut a giroszkóp kezelése és külön-külön a két motort szabályzó algoritmusa ezért minden iterációban a giroszkóp leolvasását egyszerre kellet végrehajtsák, különben más-más értékekkel dolgoztak, mely egyensúly vesztést okozott. Ehhez szükséges volt az előbb létrehozott `CyclicBarrier` konstruktorának átadott értékét növelése eggyel, mivel most már külön-külön kérte le a giroszkóp szenzor értékét a jobb és bal motorokat vezérlő szál. Emelet még egy újabb `CyclicBarrier` bevezetése volt szükséges, hogy a motorokat is szinkronizáljuk egymáshoz. E módosítást követően a robot váratlan időn belül elvesztette az egyensúlyát.

Mivel a robot LCD kijelzője nem megfelelő méretű, hogy tesztelésre alkalmas adatokat jelenítsünk meg futás közbe ezért szükség van a fájlba való kiirtásra és ezt követően az értékek értelmezése Matlab segítségével. E módszer igen csak időigényes és nehézkes, mivel a túl sok fájlba való írás leterheli a robot processzorát és növelődik egy iterációnak a végrehajtási ideje. Az iterációnként eltelt idő szerint számítsuk a robot sebességét és PID integrált illetve derivált tagját is. Tehát ha jelentősen tolódik egy iteráció lefutási ideje akkor az kritikusan befolyásolja az algoritmus működését.



4.2. ábra. Bal motor fordulatszám összehasonlítása a bal és jobb motor fordulatszám átlagával

A szinkronizálás helyességének tesztelése után, a PID szabályzó bemeneti hibáját meghatározó négy komponensnek a kiszámítását teszteltük. Gilyen Hunor által készített algoritmus esetében a robot sebességének és pozíciójának kiszámításakor átlagolva volt a két motor fordulatszáma. Esetünkben ez módosult és elhagytuk az átlagolást. Ennek eredményeként amikor egy adott iterációban minimális különbség lépet fel a két motor fordulatszáma között akkor ez nagymértékben befolyásolta sebesség és pozíció kiszámítását. Mivel a fordulatszám különbözött ezért a sebesség és a pozíció is. E különbség iterációnként



4.3. ábra. Jobb motor fordulatszám összehasonlítása a bal és jobb motor fordulatszám átlagával

növekedett. A sebesség és a pozíció a PID bementi hibájának a két komponense ezért a kimeneti érték is különbözött, amely meghatározta a motorokra adott erő nagyságát. Tehát a motorok által leadott erő különbözött és ezért a robot elvesztette egyensúlyát.

E probléma megoldására próbáltuk súlyozni az aktuális és az azelőtti mért fordulatszámot, hogy megközelítsük az átlagolt fordulatszámot. Az így megkapott értékel számoltuk ki a robot sebességét és pozícióját. A legjobb megközelítését, amelyet az átlagolt értékkel hasonlítottunk össze az látható a 4.2 és a 4.3 ábrákon. A kék szín jelöli a adott motor fordulatszámát illetve a zöld jelöli a motorok az átlagolt fordulatszámot. De még ez a közelítés sem volt elegendő, hogy a robot ne veszítse el az egyensúlyát.

Irodalomjegyzék

[1] A lego történelme. URL http://www.lego.com/en-us/aboutus/lego-group/the_lego_history/1930. Utolsó megtekintés dátuma: 2016-05-09.

[2] A lego mindstorms ev3 hivatalos oldala. URL http://www.lego.com/en-US/mindstorms/?domainredir=mindstorms.lego.com. Utolsó megtekintés dátuma: 2016-05-16.