

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

## **License Thesis**

# **Control and balancing – applied to Lego robots**

### **Abstract**

The thesis' goal is to accomplish a network communication between a mobile application and a MINDSTORMS EV3 two wheeled self balancing robot. To achieve this goal there are two main factors that need to be taken into consideration. Firstly the mobile application that needs to make the communication with the robot, and secondly an algorithm which makes the balancing of the robot possible.

The application and the algorithm which handles the balancing were written in Java. The EV3 control unit is responsible for navigating the robot with the help of the leJOS Linux based framework. The robot remains in a balanced position by using its PID (Proportional Integral Derivative) controller and its gyroscope sensor. For it to remain in a balanced state, its velocity, tilt angle and angular velocity needs to approximate to zero. Also its position needs to approximate to zero if it were to remain in a standing balanced position.

The PID's input argument is the error variable which is the difference between the expected value and the current value. In our case we need the error to approximate to zero in order to not lose its balance. Because of this the expected value is always zero. In conclusion the error is made up of four components: the robots tilt angle, angular velocity, velocity and position. By properly modifying these components we can manipulate the robot's movement.

The mobile application connects to the robot through the network and sends eligible commands to it. To accomplish and automated connection between the two components I've used the UPnP (Universal Plug and Play) library, which uses SSDP (Simple Service Discovery Protocol) protocol.

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 ȘI INFORMATICĂ  
SPECIALIZAREA INFORMATICĂ

**Lucrare de licență**

**Control și echilibrare – aplicat la  
roboți Lego**



CONDUCĂTOR ȘTIINȚIFIC:  
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

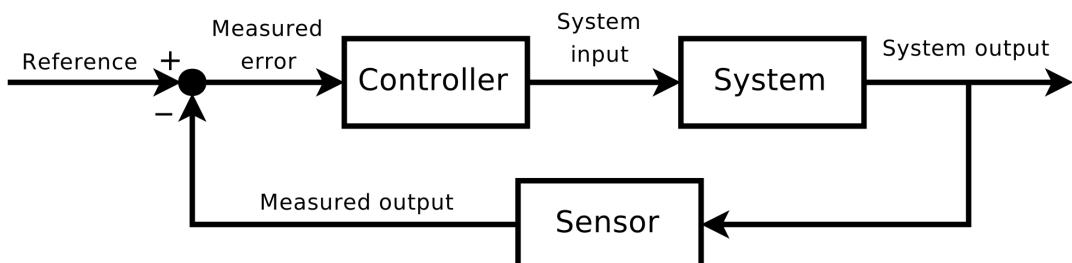
# Tartalomjegyzék

<b>1. Bevezető</b>	<b>3</b>
<b>2. LEGO</b>	<b>6</b>
2.1. LEGO megalakulása . . . . .	6
2.2. LEGO MINDSTORMS . . . . .	6
2.2.1. RXC . . . . .	6
2.2.2. NXT . . . . .	7
2.2.3. EV3 . . . . .	7
2.3. Ev3 motorok . . . . .	8
2.3.1. Nagy motor . . . . .	9
2.3.2. Közepes motor . . . . .	9
2.4. Ev3 szenzorok . . . . .	9
2.4.1. Színszenzor . . . . .	9
2.4.2. Giroszkóp szenzor . . . . .	10
2.4.3. Nyomásérzékelő . . . . .	10
2.4.4. Ultrahang szenzor . . . . .	11
2.4.5. Infravörös szenzor . . . . .	11
2.4.6. Az egyensúlyozó robot felépítése . . . . .	11
<b>3. Egyensúlyozás</b>	<b>14</b>
3.1. PID szabályzó algoritmus . . . . .	14
<b>4. Megvalósítás</b>	<b>16</b>
4.1. EV3 programozása . . . . .	16
4.2. Androidos alkalmazás és kommunikáció . . . . .	19
4.3. Egyensúlyozás problémái . . . . .	21

## 1. fejezet

# Bevezető

A szakdolgozat témája robotok egyensúlyozását megvalósító rendszerek tanulmányozása. Ezen rendszereket nevezhetjük zárt vagy nyílt ciklusosnak. Nyílt ciklusos rendszer esetén nincs visszacsatolás, ezért nem megfelelő olyan problémák megoldására amelyeknél ellenőriznünk kell a rendszer kimenetének eredményét, tehát nem szükséges a visszacsatolás. Egyszerű problémák megoldása esetén használatos. A zárt ciklusos rendszerek visszacsatolással működnek, irányítható a rendszer. Visszacsatolásos működés(1.1 ábra) annyit tesz, hogy a kimenet hatására egy visszajelzést kapunk, amely tulajdonképpen a rendszer bemenete lesz így a visszajelzés szabályozza a kimenetet, ezáltal komplex feladatok megvalósítására alkalmas.



1.1. ábra. Zárt ciklusos rendszer működési elve [https://en.wikipedia.org/wiki/Control\\_theory](https://en.wikipedia.org/wiki/Control_theory)

Zárt ciklusos rendszerek esetén szabályzó algoritmusról beszélünk, amelyek használata számos technológiában jelen van. Ilyenek közé sorolható a Swagway<sup>1</sup> illetve a hasonló szerkezetű és működésű Segway<sup>2</sup>.

A dolgozat során olyan projekt kerül bemutatásra, amely a LEGO MINDSTORMS EV3[4] készletből épített kétkerekű egyensúlyozó robot irányítását teszi lehetővé hálózaton keresztül, telefonos alkalmazás segítségével.

A kétkerekű robot a Gyro Boy<sup>3</sup> modell alapján készült el. Főbb komponensei közé sorolható az EV3 vezérlőegység<sup>4</sup>, giroszkóp szenzor<sup>5</sup> és a két nagy szervomotor<sup>6</sup>, amelyek egy tengelyen helyezkednek el. Az előbb említett elemekről bővebben szó lesz a 2.3 illetve 2.4 fejezetben. Mivel a kerekek egy tengelyen

1. <https://swagway.com>

2. <http://www.segwaymagyarorszag.com/>

3. [http://robotsquare.com/wp-content/uploads/2013/10/45544\\_gyroboy.pdf](http://robotsquare.com/wp-content/uploads/2013/10/45544_gyroboy.pdf)

4. [http://lego.wikia.com/wiki/45500\\_EV3\\_Intelligent\\_Brick](http://lego.wikia.com/wiki/45500_EV3_Intelligent_Brick)

5. <http://shop.lego.com/en-US/EV3-Gyro-Sensor-45505>

6. [http://lego.wikia.com/wiki/45502\\_EV3\\_Large\\_Servo\\_Motor](http://lego.wikia.com/wiki/45502_EV3_Large_Servo_Motor)

## 1. FEJEZET: BEVEZETŐ

helyezkednek el ezért instabil a szerkezete és könnyen, rövid idő alatt elveszti egyensúlyi állapotát. E probléma megoldására a robot dőlesi szöge, szög változásának a sebessége és a robot sebessége 0-hoz kell, hogy tartson valamint annak érdekében, hogy egy helyben próbálja megtartani egyensúlyát a robot pozíciója is 0-hoz kell közelítsen.

A probléma megoldására alkalmas a PID<sup>7</sup> szabályzó algoritmus használata, amely ipari körökben elterjedt. Viszonylag egyszerű a felépítése, kezelhetősége és az implementálhatósága. A PID szabályzó zárt ciklusos rendszer, melynek a bemeneti értéke a hiba, amely az elvárt érték és az aktuális érték különbsége. Esetünkben mivel 0-t kell közelítsünk annak érdekében, hogy ne veszítse el egyensúlyi állapotát a robot ezért az elvárt érték öröké 0 lesz, vagyis a hibát négy komponens alkotja: a robot dőlesi szöge, szögsebessége, a robot sebessége és pozíciója, amelyek külön-külön súlyozva vannak. A szög és szögsebesség érték meghatározásához a giroszkóp szenzort használjuk és a szervomotorok beépített szenzorai által lekérhető fordulat szám segítségével számoljuk ki a robot sebességét és pozíóját.

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 részeként elkészített 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 védelemét, a tovább bővíthetőséget illetve a szerializációt a Google Protocol Buffers<sup>8</sup> 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.

A robot szabályzó algoritmus Java-ban íródott, amelynek a futtatási környezetét a leJOS firmware biztosítja. Linux alapú és magába foglalja a JVM-t (Java virtual machine), amely lehetővé teszi, hogy a robot programozható legyen Java-ban. Számos firmware-t fejlesztettek ki annak érdekében, hogy a LEGO MINDSTORMS által fejlesztett vezérlőegységek programozhatóak legyenek magas szinten is. Pár ismert firmware: leJOS<sup>9</sup> José Solórzano hozta létre 1999 végén, nyílt forráskódú, támogatja az objektum orientált programozást, ROBOTC<sup>10</sup> támogatja a C programozást, ev3dev<sup>11</sup> amely a szkript nyelveket támogatja (Phyton, NodeJS, Ruby).

Az előbb említett firmware-k teszik lehetővé a komplex feladatok megoldását, amelyek nem lehetségesek az EV3 alapértelmezett rendszerével. E rendszerhez a LEGO MINDSTORMS biztosított egy grafikus felületet, amely a kisebb korosztály számára készült, hogy "programzhassák" a saját kezűleg épített robotokat.

A dolgozat négy fejezetből áll. Az első fejezet által betekintést nyerünk a dolgozat témajáról.

A második fejezet röviden bemutatja a LEGO megalakulását, a LEGO MINDSTORMS által kifejlesztett generációkat majd bemutatja az EV3 készlethez tartozó motorokat és szenzorokat. Tartalmazza

7. [https://en.wikipedia.org/wiki/PID\\_controller](https://en.wikipedia.org/wiki/PID_controller)

8. <https://developers.google.com/protocol-buffers/>

9. <http://www.lejos.org>

10 <http://www.robotc.net>

11 <http://www.ev3dev.org/>

## **1. FEJEZET: BEVEZETŐ**

azon eszközök részletes leírását amelyek a projekt során felhasználásra kerültek.

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

A negyedik fejezet célja, hogy bemutassa e szakdolgozat alatt létrehozott projekt működését és az elkészített Android mobil alkalmazást. Valamint a projekt elkészítése során felmerült problémákat, ezek megoldását illetve lehetséges megoldását.

## **2. fejezet**

# **LEGO**

**Összefoglaló:** A fejezetben, bemutatásra kerül a LEGO megalakulása, fejlődése. E mellett sor kerül a LEGO MINDSTORMS által kifejlesztett technológiák bemutatására, amelyek közül a giroszkóp szenzor és a nagy motorok felhasználásra kerülnek a továbbiakban.

### **2.1. LEGO megalakulása**

A LEGO [2][3] története 1932-ben kezdődött, Ole Kirk Christiansen<sup>1</sup> asztalos vállalkozást alapított Dánia, Billund nevezetű falujában, amely a fa játékok gyártásával foglakozott. 1934-ben vette fel a cégt 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 cégt. 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 téglák" 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 [5], 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 számított mivel, teljesen autonóm. Képes számítógép nélkül is működni de mára már nem gyakori a használata. Az évek során sokat fejlődött és ma már egyre több oktatási intézmény használja a robotika oktatásban.

#### **2.2.1. RXC**

Az RXC [8] amelyet a 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](http://lego.wikia.com/wiki/Ole_Kirk_Christiansen)

## 2. FEJEZET: LEGO

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. A kapacitásának ellenére számos projektben felhasználták, ilyenekben mint a Brick Sorter<sup>2</sup> mely szín szerint szortírozza az adott elemeket, RCX Remote Control<sup>3</sup> mely két RCX vezérlőegység közti kommunikációval irányítja a robot.



2.1. ábra. RXC brick

### 2.2.2. NXT

A második generáció a 2.2 ábrán látható, az NXT [6][7], 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 bemeneti portot, három kimeneti portot 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. Az NXT csomaggal már komplexebb projektek is készültek mint az RXC-vel. Megemlíteném a Segway with Robot Driver<sup>4</sup> című projektet, amely megvalósított egy olyan robot amely irányítja a szintén NXT vezérlőegység által működtetett Segway-t.

### 2.2.3. EV3

A harmadik generáció, az EV3 [1][7] 2013-ban jelent meg, a 2.3 ábra szemlélteti. Az "EV" evolúciót jelenti és a 3, hogy harmadik generáció. A legfejlettebb LEGO MINDSTROMS programozható építőelem amely kezeli a motorokat, szenzorokat és lehetőséget ad Bluetooth-on keresztül való kommunikációra.

2. <http://robotsquare.com/2012/02/15/brick-sorter-4/>

3. <http://robotsquare.com/2012/02/14/rcx-remote-control/>

4. <http://robotsquare.com/2012/09/04/segway-with-robot-driver/>

## 2. FEJEZET: LEGO



2.2. ábra. NXT brick

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 használunk 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 mellett 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 menüpontjainak kezelését.[7]

Mivel a LEGO MINDSTORMS generációk közül a legfejlettebb ezért a projekt által felhasznált vezérlőegység az EV3.



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

## 2. FEJEZET: LEGO

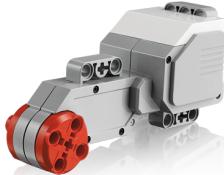
jelentek meg. Az EV3 nagy motorjának teljesítménye megegyezik az NXT-vel de a felépítését optimalizálták, illetve a EV3 közepes motorja teljesen új az NXT-hez képest.

### 2.3.1. Nagy motor

A nagy motor, amely a 2.4 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 nyomatéuk és a precizitásuk azonos.



2.4. ábra. Nagy motor



2.5. ábra. Közepes motor

### 2.3.2. Közepes motor

A közepes motor, amely a 2.5 ábrázol, tulajdonsága megegyezik a nagy motorjaival. Ugyancsak tartalmaz forgásérzékelőt és lehetséges a helyzetmeghatározás 1 fok eltéréssel, viszont alacsonyabb terhelésre terveztek, forgatónyomatéka 8 N/cm, ha blokkolt állapotba kerül akkor a nyomatéka 12 N/cm és a sebessége 240-250 rmp.

## 2.4. Ev3 szenzorok

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

### 2.4.1. Színszenzor

A digitális színszenzor, amely a 2.6 ábrázol, hét különböző színt ismer fel. A színpelismerő funkcionálitásán kívül mérhető vele a fényvisszaverődés erőssége vagy a környezeti fény intenzitását. E három

## 2. FEJEZET: LEGO

üzemmód lehető teszi a felhasználóknak, hogy akár kövessen egy fekete vonalat vagy megkülönböztetéssel 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 2.7 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 2.8 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 szögfordulási mutatója

### 2.4.3. Nyomásérzékelő

Az analóg nyomásérzékelője egy piros gombbal van felszerelve ami látható a 2.9 ábrán. Funkcionalitása nem túl bonyolult viszont annál hasznosabb. Érzékeli a gomb lenyomását illetve felszabadulását, amelyet egy integrált számlálóval segítségével nyomon lehet követni.

## 2. FEJEZET: LEGO



2.9. ábra. Nyomásérzékelő

### 2.4.4. Ultrahang szenzor

A digitális ultrahang szenzor látható a 2.10 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 végezünk 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 2.11 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 2.12 ábrán.



2.11. ábra. Infravörös szenzor



2.12. ábra. Infravörös jeladó

### 2.4.6. Az egyensúlyozó robot felépítése

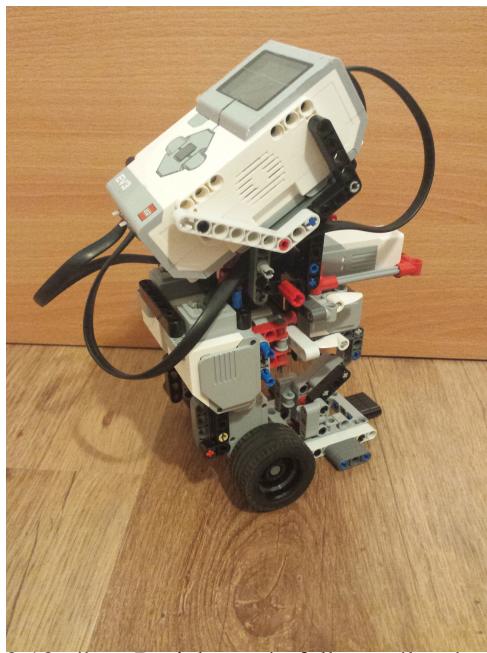
A kétkerekű egyensúlyozó robot, amely a 2.13 ábrán látható a Gyro Boy<sup>5</sup> modell alapján készült el, mely az EV3 csomagból építhető meg. Szimmetrikus súlyelosztása révén elősegíti az egyensúly megtar-

5. [http://robotsquare.com/wp-content/uploads/2013/10/45544\\_gyroboy.pdf](http://robotsquare.com/wp-content/uploads/2013/10/45544_gyroboy.pdf)

## 2. FEJEZET: LEGO

tását. Megemlítenék még más egyensúlyozó robotot is, az NXTway-GS<sup>6</sup> modellt, amely NXT csomaggal valósítható meg, illetve a Rover Kit<sup>7</sup>, amely mikrovezérlő által irányított egyensúlyozó robot.

A robot főbb komponensei közé sorolható az EV3 vezérlőegység, giroszkóp szenzor és a két nagy szervomotor. A motorok egy tengelyen helyezkednek, amelyekre fel vannak erősítve a kerekek. A kerekek átmérője 55mm, ezt az értéket felhasználjuk a pozíció illetve a sebesség kiszámolásakor. A giroszkóp szenzor egy tengelyen mentén mér, a robot központi részén helyezkedik el, közel az egyensúlyi pontjához.



2.13. ábra. Projekt során felhasznált robot

A robothoz tartozik egy állvány, amely a kiinduló pontja és egyensúlyba tartja amíg az egyensúlyozást megvalósító algoritmus elindul.

Az egyensúlyozás megvalósítására az algoritmus a PID szabályzót használja fel, amely egy zárt ciklusos rendszer. A PID szabályzó bemeneti értéke a hiba és a kimeneti a motorokra leadott erő.

A hibát az elvárt érték és az aktuális érték különbségéből kapjuk meg. Esetünkben, a kiindulási pont a robot egyensúlyi állapota, azaz a robot dőlési szöge, szög változásának a sebessége, a robot sebessége és a pozíciója nulla, amely az elvárt érték. A szög és szög változásának sebességét a giroszkóp szenzor által kérjük le. A leJOS firmware több módot is biztosít a giroszkóp használatára. Esetünkbe a `rate` módot használjuk, amely a szög változásának a sebességét méri. Mérések esetén, mivel a szenzor nem mér pontosan ezért többször mintavételezünk és ezeket átlagoljuk, hogy kiszűrjük a zajokat. Az így megkapott szög változásának sebességéből kiszámítható a dőlési szög a következő képlettel:

$$\varphi = \omega \cdot t,$$

6. [http://lejos-osek.sourceforge.net/NXTway-GS\\_Building\\_Instructions.pdf](http://lejos-osek.sourceforge.net/NXTway-GS_Building_Instructions.pdf)

7. <http://www.sainsmart.com/sainsmart-balancing-robot-kit.html>

## 2. FEJEZET: LEGO

ahol az  $\omega$  jelöli a szög változásának sebességét illetve a  $t$  az eltelt időt.

A szervomotorok esetén lekérhetjük a fordulat számát, amelyet a motorba beépített szenzor egy számlaló segítségével számol. A két motor fordulatszámát átlagoljuk, ezáltal kiszűrjük a zajokat. Az így megkapott értéket átalakítjuk szögbé majd kiszámítjuk a robot sebességét valamint pozícióját.

A sebességet és a pozíciót a következő képletekkel határozzuk meg:

$$v = \frac{\alpha - \alpha'}{t} \cdot d \quad (2.1)$$

$$x = \alpha \cdot d, \quad (2.2)$$

ahol az  $\alpha$  jelöli a motor elfordulási szögét, az  $\alpha'$  az előző időpillanatban a motor elfordulási szögét, a  $t$  az eltelt időt, a  $d$  a kerék átmérőjét, a  $v$  a robot sebességét illetve az  $x$  robot pozícióját.

Az előbbiekben kiszámított négy értéket kivonjuk a nekik megfelelő elvárt értékekből, amelyek nul-lák. Ezt követően konstansok által súlyozva majd összeadva az értékeket megkapjuk a PID bemenetét vagyis a hibát.

### 3. fejezet

## Egyensúlyozás

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

### 3.1. PID szabályzó algoritmus

A szabályzó rendszereket kétféleképpen nevezhetjük nyílt vagy zárt ciklusosnak. Nyílt ciklusos rendszerek esetén nincs visszacsatolás, ezért nem megfelelő komplex problémák megoldására. Előnyös a használata kisebb kapacitás esetén, mivel alacsony költségű. A zárt ciklusos rendszerek visszacsatolással működnek, vagyis a rendszer kimenetének hatására visszajelzést kapunk.

A PID (Proportional Integral Derivative) elterjedt zárt ciklusos rendszer, az ipari szabályzó körökben köszönhetően a viszonylag egyszerű féllepítésének és a könnyen kezelhetőségének. Számos felhasználási lehetősége van, ilyenek a nyomás, hőmérséklet, sebesség szabályozás. E szabályzó rendszer bementi értéke az aktuális hiba.

A hibát az elvárt érték és az aktuális érték különbsége határoz meg. Esetünkben az aktuális értékek a következők: robot dőlesi szöge, szög változásának sebessége, a robot pozíciója illetve sebessége. Az előbb felsorolt értékek esetén az elvárt értékek ahhoz, hogy a robot egyensúlyi állapotba maradjon 0-hoz kell közelítsenek.

A szabályzó kimeneti értéke a motorokra leadott erő. Esetünkbe tehát a visszacsatolás az előbb felsorolt négy értékben nyilvánul meg, mivel a kerekre leadott erő befolyásolja ezen értékeket.

A PID, mint a nevéből is látható három tagból tevődik össze: P (Proportional) az arányos tag, I (Integral) a hibák integrált tagja és a D (Derivative) a hibák időbeli változásának derivált tagja. Abban az esetben ha az előbb felsorolt tagok közül nem vesszünk figyelembe minden tagot, akkor a következő szabályzókról beszélhetünk: P, PI és PD.

PID szabályzó matematikai képlete:

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

ahol  $t$  jelöli az időt,  $e(t)$  a bemeneti hibát illetve az  $u(t)$  a kimenet. A  $K_P, K_I, K_D$  paraméterek rendre az arányos, integrált és a derivált tagok súlya, amelyek meghatározásakor a szabályzó érzékenységét valamint befolyásolják teljesítményét.

Kizárálag az arányos tag használata esetén a szabályzó rendszer olyan esetknél használható, ahol

### 3. FEJEZET: EGYENSÚLYOZÁS

megengedett egy bizonyos nagyságú hiba az kimenetben miután a rendszer stabilizálódott. Az arányos tag paramétere a  $K_P$ , amely egy konstans érték, ha túl nagy instabil rendszert eredményez.

Az integrált tag önmagában nem használható mint az arányos tag. Az integrált tag esetén a hibák összege mindenkorán változik míg az arányos tag nulla nem lesz. Tehát a feladata, hogy egy pontosabb és precízebb szabályzó rendszer biztosítása. Mivel kód szinten nem valósítható meg az integrálás ezért a következő képlet segítségével közelítsük meg:

$$u(t) = K_I \cdot \sum e(t),$$

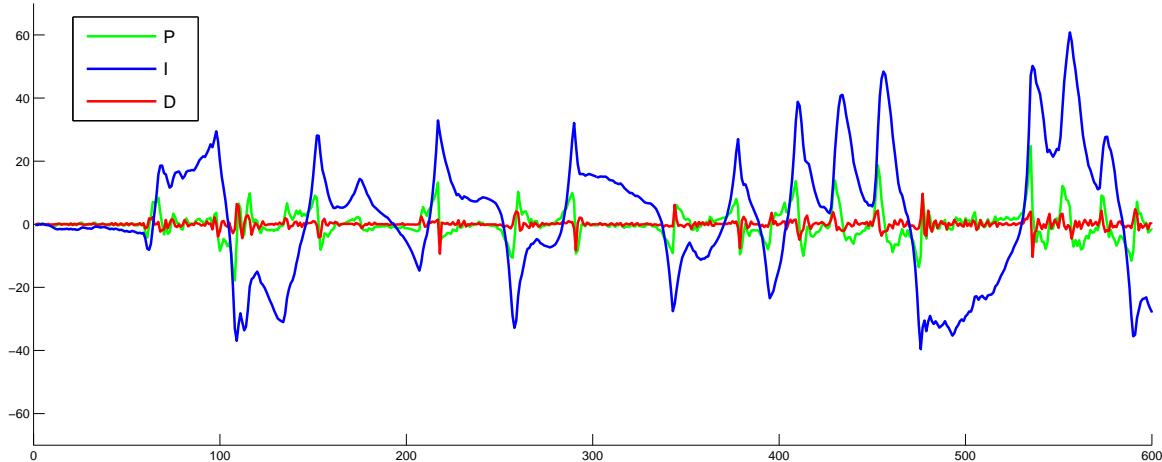
ahol az előbbihez hasonlóan  $K_I$  konstans.

A derivált tag értéke arányos a hiba változási sebességével. Feladata a hirtelen változások késleltetése, ezáltal egy stabilabb rendszert biztosítva. Ez esetben is felmerül az előbb említett probléma ugyanis kód szinten a deriválás sem valósítható meg ezért a következő képlettel közelítsük meg:

$$u(t) = K_D \cdot (e(t) - e(t - 1)),$$

ahol szintén a  $K_D$  konstans.

A 3.1 ábrán látható a PID szabályzó tagjainak változása egy teszt esetén. Jól látható az ábrán, hogy a derivált tag ellentétes előjelű az arányos taghoz viszonyítva, ennek köszönhetően a hirtelen változások késleltetve lesznek. Az integrált tag és az arányos tag esetén jól látható, hogy a hasonló oszcillációt végeznek.



3.1. ábra. Egy tesz során kapott PID tagjainak értékeit ábrázolja. Az x tengelyen jelöljük a iterációk számát, az y tengelyen a PID tagok értékének mértékét.

## 4. fejezet

# Megvalósítás

**Összefoglaló:** A dolgozat tárgyát képező projekt egy EV3 készletből épített kétkerekű egyensúlyozó robot irányítását valósítja meg hálózaton keresztül, telefonos alkalmazás segítségével. E fejezetben 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 nyelvek használatát támogatják, ilyen a ROBOTC<sup>1</sup> Carnegie Mellon egyetem fejlesztette ki, amely támogatja a C programozást illetve megemlíteném még a Debian<sup>2</sup> által kifejlesztett ev3dev-at<sup>3</sup> amely a szkript nyelveket támogatja (Phyton, NodeJS, Ruby) . 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). Futtatási környezetet biztosít a Java programozóknak, támogatja az objektum orientált programozást. Mindezek lehetővé teszik a socket alapú kommunikációt, szinkronizálhatóságot, szálak alkalmazását és Java típusok használatát.

Az EV3 vezérlőegységen található négy bemeneti(S1, S2, S3, S4) illetve négy kimeneti(A, B, C, D) port. A bemeneti portok esetén a szenzorok csatlakoztathatók valamint a kimeneti portok esetén a motorokat. A leJOS hozzáférést biztosít ezekhez a portokhoz. Amint látható a 4.1 illetve a 4.2 ábrán explicit megkel adjuk, hogy milyen porton található a motor illetve a szenzor. Motorok esetén importálnunk kell `lejos.hardware.port.MotorPort` csomagot és így elérjük a szükséges kimeneti port interface-t. Szenzorok esetén is hasonló az eljárás, azzal a különbséggel hogy ebben az esetben a `lejos.hardware.port.SensorPort` csomagot kell importáljuk a megfelelő interface eléréséhez.

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

1. <http://www.robotc.net>

2. <https://www.debian.org/>

3. <http://www.ev3dev.org/>

#### 4. FEJEZET: MEGVALÓSÍTÁS

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

##### 4.2. Listing. 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/másodperc -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.

##### 4.3. Listing. Giroszkóp szenzor `rate` mód használata

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

SampleProvider gyroMode = gyroSensor.getRateMode();

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

##### 4.4. Listing. Giroszkóp szenzor `angle` mód használata

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

SampleProvider gyroMode = gyroSensor.getAngleMode();

7 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 -100 és 100 közötti értéket adhatunk át. Az átadott érték határozza meg, hogy mekkora erőt fejtse ki a motor, 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 megvalósításához szükséges az előbb említett metódus által lekérni a két motor fordulat számát, amelyeket átlagoljuk, ezáltal kiszűrjük a szenzor mérésénél keletkezett zajokat. Az így megkapott értéket átalakítjuk szögbe majd kiszámítjuk a 2.4.6 alfejezetben megjelenő (2.1) és (2.2) képletek segítségével. A `stop` metódus hívása esetén a motor blokkolt állapotba kerül.

A motor szabályzott mód (4.6 ábra) esetén némely funkcionális 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.

#### 4. FEJEZET: MEGVALÓSÍTÁS

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 ki kellet számolni. Ez esetben lehetőség van ezen értékek lekérésére a `getPosition` és `getSpeed` metódusok által.

Esetünkben lényeges különbség a két mód között, hogy a szabályzott mód esetén jóval lassabb a motorok reagálási ideje mint a nem szabályzott mód esetén. E mellett a szabályzott mód használatakor ha kiadunk egy utasítást akkor kell figyeljünk, hogy utána ne adjunk olyan más utasítást amely miatt az előzőt leállítaná.

4.5. Listing. A nagy motor `regulated` mód használata

```
2 EncoderMotor motor = new UnregulatedMotor(MotorPort.A)
motor.resetTachoCount();
motor.setPower(40);
7 int tacho = motor.getTachoCount();
motor.stop();
```

4.6. Listing. 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);
10 float position = motor.getPosition();
```

Annak érdekében, hogy az EV3 vezérlőegységen futassuk és könnyedén kitelepítsük a programokat az Eclipse IDE fejlesztői környezetet használjuk és a leJOS plugin-t, amelyek mindezt megkönnyíti.

Mivel az EV3 vezérlőegységen az alapértelmezett firmware van telepítve ezért külön SD kártyára kell 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. Mivel az EV3 ARM9-es processzorral rendelkezik ezért fontos, hogy a JRE ARM verzióját töltök le. 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 állítsuk be az EV3\_HOME környezeti változónak a feltelepített leJOS plugin elérési útvonalát (`C:\ProgramFiles\leJOSEV3`) és a könyvtárán belüli bin könyvtárat (`C:\ProgramFiles\leJOSEV3\bin`) a Path-nek. Ezek a beállítások teszik lehetővé az operációs rendszer számára, hogy futtatás esetén megkapja a szükséges fájlok elérési útvonalát.

#### 4. FEJEZET: MEGVALÓSÍTÁS

## 4.2. Androidos alkalmazás és kommunikáció

A projekt része egy telefonos alkalmazás, mely célja hogy hálózaton keresztül kapcsolódjon a robothoz és küldje a megfelelő utasításokat, annak érdekében hogy a felhasználó tudja irányítani a robot mozgását. Az alkalmazás Android stúdióban 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" funkcióval, mely célja hogy a legutóbbi IP és PORT címet visszatölts az alkalmazás elindításakor. 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 kör rajzolunk ki. A nagy kör jelöli vizuálisan a határokat a felhasználónak és ezáltal meghatározunk egy konkrét intervallumot az irányításhoz szükséges értékeknek valamint így kizárnak annak a lehetőséget, hogy olyan értékeket olvassunk a kisebb kör mozgatásának hatására amelyek nem megfelelők. A kisebbik kör segít a felhasználóval tudatni, hogy a nagy kör peremén belüli rész az általunk értelmezett illetve figyelembe vett felület az irányításhoz. 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 kör átalakításokat végezünk koordináták között.

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

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

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

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

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 polárkoordinából átalakítani euklideszi koordinátába:

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

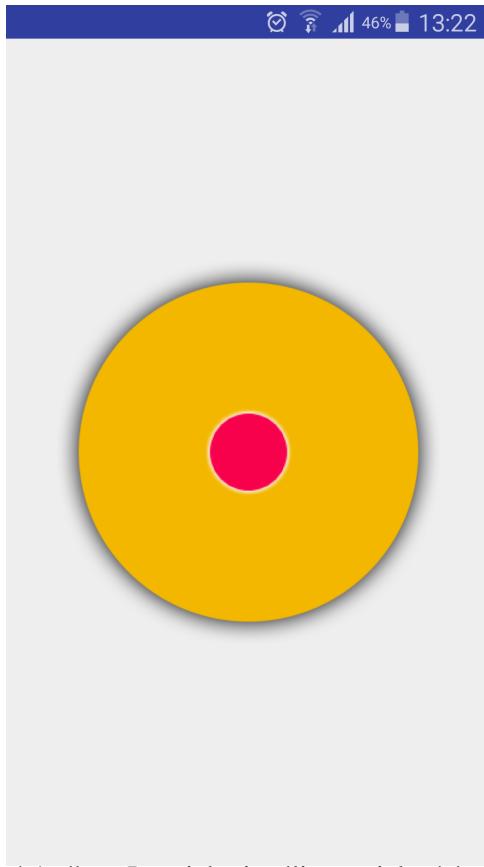
$$x = R * \cos(\varphi)$$

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

ahol R a két kör sugarának különbsége.

Tudva a mozgatás irányát, socketen keresztül küldjük a parancsokat. Az adatok szerializációját illetve

#### 4. FEJEZET: MEGVALÓSÍTÁS



4.1. ábra. Joystick vizuális megjelenítése

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 megadják 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 azonosító-ként szerepel a bináris kódolás során. Ezekben kívül megadható három típus mező minden attribútumnak, amelyek a következők: `required`, `optional`, `repeated`. A `required` mezővel beállíthatjuk, hogy az adott attribútum kötelezően értéket kapjon különböző 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.

4.7. Listing. Az adatok strukturáját definiáló `.proto` állomány

#### 4. FEJEZET: MEGVALÓSÍTÁS

```
1 package protobuf;
2 option java_package = "app.cs.ubb.edu.ev3.protobuf";
3 option java_outer_classname = "DataProtos";
4
5 message Data {
6
7     required int32 left = 1;
8     required int32 right = 2;
9     required float forward = 3;
10    required float speed = 4;
11    required float angle = 5;
12    required bool client = 6;
13}
```

### 4.3. Egyensúlyozás 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 szálra 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.

4.8. Listing. CyclicBarrier példányosítása

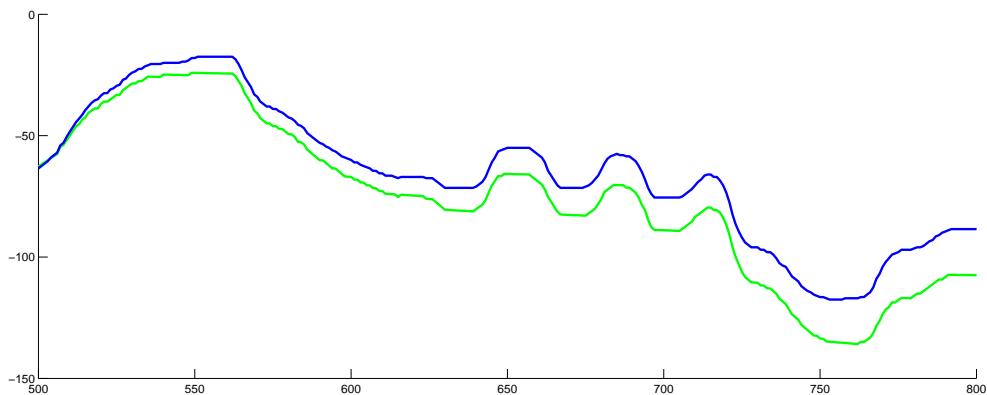
```
2 CyclicBarrier barrier = new CyclicBarrier(2);
```

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 szálra fut a giroszkóp kezelése és külön-külön a két motort szabályzó algoritmusára ezért minden iterációban a giroszkóp leolvasását egyszerre kellett végrehajtsák, különben más-más értékekkel dolgoztak, mely egyensúly vesztést okozott. Ehhez szükséges volt az előbb létrehozott CyclicBarrier konstruktorának átadott értékét növelése eggyel,

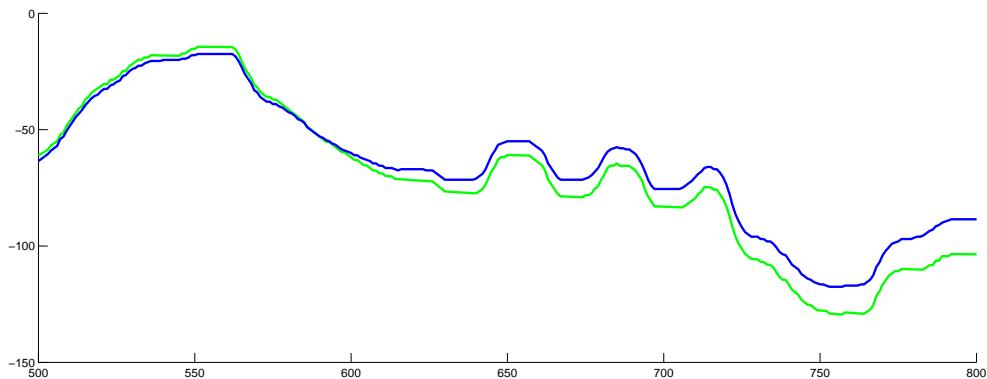
#### 4. FEJEZET: MEGVALÓSÍTÁS

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 leterhelí 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



4.3. ábra. Jobb 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 sebességeinek és pozíciójának kiszámításakor átlagolva volt a két motor fordulatszáma. Esetünkben ez módo-sult 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 különbözőt ezért a sebesség és a pozíció is. E különbség iterációnként

#### 4. FEJEZET: MEGVALÓSÍTÁS

növekedett. A sebesség és a pozíció a PID bementi hibájának a két komponense ezért a kimeneti érték is különözőt, amely meghatározta a motorokra adott erő nagyságát. Tehát a motorok által leadott erő különbözőt 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 pozíóját. 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] Ev3 vezérlőegység tartalma. URL <http://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313>. Utolsó megtekintés dátuma: 2016-05-16.
- [2] A lego történelmi idővonal, . URL [http://www.lego.com/en-us/aboutus/lego-group/the\\_lego\\_history/1930](http://www.lego.com/en-us/aboutus/lego-group/the_lego_history/1930). Utolsó megtekintés dátuma: 2016-05-09.
- [3] A lego történelme, . URL <https://hu.wikipedia.org/wiki/LEGO>. Utolsó megtekintés dátuma: 2016-05-09.
- [4] A lego mindstorms ev3 hivatalos oldala, . URL <http://www.lego.com/en-US/mindstorms/?domainredirect=mindstorms.lego.com>. Utolsó megtekintés dátuma: 2016-05-16.
- [5] A lego mindstorms történelme, . URL <http://www.lego.com/en-us/mindstorms/history>. Utolsó megtekintés dátuma: 2016-05-09.
- [6] Az nxt leírása, . URL <http://shop.lego.com/en-CA/NXT-Intelligent-Brick-9841>. Utolsó megtekintés dátuma: 2016-05-16.
- [7] Nxt összehasonlítása az ev3-al, . URL <http://botbench.com/blog/2013/01/08/comparing-the-nxt-and-ev3-bricks/>. Utolsó megtekintés dátuma: 2016-05-16.
- [8] Az rxc leírása. URL [http://lego.wikia.com/wiki/Mindstorms\\_RCX](http://lego.wikia.com/wiki/Mindstorms_RCX). Utolsó megtekintés dátuma: 2016-05-16.