TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Janno Peterson

Pythoni SDK LEGO WeDo 2.0-le

Bakalaureusetöö (9 EAP)

Juhendaja: Aivar Annamaa

Tartu 2017

Pythoni SDK LEGO WeDo 2.0-le

Lühikokkuvõte:

Käesoleva bakalaureusetöö eesmärk on luua Pythoni teek, mille abil oleks võimalik mugavalt LEGO Education WeDo 2.0-ga ühenduda ning selle suhelda. Töös antakse ülevaade WeDo 2.0-st ning Bluetooth Low Energy tehnoloogiast, mida WeDo 2.0 suhtlemiseks kasutab. Töö tulemusena valminud teegi abil on võimalik luua Pythonis kasutajaliideseid WeDo 2.0-i programmide loomiseks või Pythoni skripte, mille abil WeDo 2.0-i juhtida.

Võtmesõnad:

Kujundus, paigutus, mall

CERCS: P175 (Informaatika, süsteemiteooria)

Python SDK for LEGO WeDo 2.0

Abstract:

The aim of this Bachelor's thesis is to create a Python library which could be used to connect and control LEGO Education WeDo 2.0 devices. The thesis will give an overview of WeDo 2.0 and Bluetooth Low Energy, which WeDo 2.0 uses for communication. The result of this thesis is a library, which can be used for creating Python applications for WeDo 2.0 or just Python scripts for controlling WeDo 2.0 devices.

Keywords:

LEGO, WeDo 2.0, SDK, Python

CERCS: P175 (Informatics, systems theory)

Sisukord

Sissejuhatus 5
1. WeDo 2.0-i ülevaade 6
1.1 LEGO Education WeDo 2.0 6
1.2 Komplekt 6
1.2.1 Smarthub 6
1.2.2 Mootor 7
1.2.3 Kallutusandur 7
1.2.4 Liikumisandur 8
1.3 Tarkvara 9
1.3.1 Tarkvara kasutamine 9
1.3.2 Piltblokid ja programmide toimimine 10
1.4 WeDo 2.0-i Androidi SDK 11
1.4.1 Arhitektuuri ülevaade 12
2. Bluetooth Low Energy 15
2.1 Bluetooth ja Bluetooth Low Energy 15
2.2 Bluetooth LE arhitektuur 16
2.2.1 Kontroller 17
2.2.2 Host 17
2.2.3 Rakendus 18
2.3 GATT 19
2.3.1 GATT profiili hierarhia 19
2.3.2 GATT-i toimingud 20
2.4 Pygatt 21
3. Töö protsess ja tulemus 23
3.1 Töö protsess 23
3.2 Pythoni SDK arhitektuur 25
3.3 Teegi kasutamine ja selle toimimine 29
3.3.1 Teegi installeerimine 30
3.3.2 Smarthub’iga ühenduse loomine 30
3.3.3 Funktsioonide kasutamine 31
3.3.4 Toetatud funktsionaalsused 32

4. Kokkuvõte 34
5. Viidatud kirjandus 35
Litsents 37

Sissejuhatus

Tänapäevases infoühiskonnas on järjest suurenev nõudlus oskuslike programmeerijate ning infotehnoloogia spetsialistide järele. Seetõttu on viimastel aastatel aina rohkem hakatud koolides õpetama programmeerimist ning algust tehakse sellega ka juba algklassides. Üks levinumaid viise, kuidas lastele programmeerimist õpetada, on läbi lastele mõeldud robootika. Üks toode, mis sellist lähenemist toetab, on LEGO Education WeDo 2.0. WeDo 2.0 on algklasside õpilastele mõeldud LEGO robot, mille juhtimiseks kirjutatakse programme arvutis või nutiseadmes ning millega ühendutakse Bluetooth Low Energy tehnoloogia abil. Roboti programmeerimine toimib värviliste piltblokkide ahelasse paigutamisega, mis aitab lastele anda sissejuhatuse programmeerimise loogikasse. Hetkel ei leidu aga võimalust WeDo 2.0-i programmide koostamiseks mõnes lihtsakoelisemas tekstipõhises programmeerimiskeeles nagu näiteks Pythonis, sest Pythoni tuge LEGO ise WeDo 2.0-le ei paku.

2016. aasta suvel tegi LEGO kättesaadavaks ametlikud WeDo 2.0-i tarkvaraarenduse komplektid (SDK-d) platvormidele Windows, Mac OS, Android ja iOS. Nende komplektide lähtekoodi uurides on võimalik mõista, kuidas arvutid või nutiseadmed robotiga suhtlevad ning selle info põhjal on võimalik ka ise Bluetooth Low Energy abil robotile käsklusi saata.

Käesoleva töö eesmärgiks on luua teek programmeerimiskeeles Python, mis võimaldaks robotiga WeDo 2.0 mugavalt ühenduda ning teegis sisalduvate funktsioonide abil robotiga suhelda. Antud teek võimaldaks WeDo 2.0-i abil Pythonis tekstipõhist programmeerimist õpetada.

Töö käigus valmiva Pythoni teegi arendamiseks on töö autor otsustanud uurida Androidi platvormile loodud ametlikku LEGO WeDo 2.0-i tarkvaraarenduse komplekti. Bluetooth Low Energy tehnoloogial põhinevat suhtlust roboti ajuga viiakse läbi avatud lähtekoodiga Pythoni teegi pygatt abil.

Töö on jaotatud kolmeks suuremaks peatükiks. Esimeses peatükis antakse ülevaade WeDo 2.0-i ning WeDo 2.0-i Androidi SDK arhitektuuri kohta. Teises peatükis antakse tehniline ülevaade Bluetooth Low Energy tehnoloogia kohta ning tutvustatakse Pythoni teeki pygatt. Kolmandas peatükis antakse ülevaade WeDo 2.0-i Pythoni SDK loomise protsessist, kirjeldatakse loodud SDK arhitektuuri, põhjendatakse arenduse käigus tehtud valikuid ning tutvustatakse töö käigus loodud Pythoni teegi kasutamist.

1. WeDo 2.0-i ülevaade

Käesolevas peatükis antakse ülevaade LEGO Education WeDo 2.0-i olemusest, selle komplekti kuuluvast riistvarast ning tarkvarast, mida selle juhtimiseks kasutatakse. Täpsemalt tuuakse välja, millised funktsionaalsused on programmeerimiseks võimaldatud ning kuidas programmeerimine toimib. Peatüki lõpus kirjeldatakse ka Androidi platvormile loodud ametlikku LEGO WeDo 2.0-i SDK arhitektuuri.

1.1 LEGO Education WeDo 2.0

LEGO Education WeDo 2.0 on 2016. aasta alguses välja lastud programmeeritav robot, mis on suunatud lastele alates seitsmendast eluaastast. WeDo 2.0-i eesmärk on äratada lastes huvi teaduse vastu ning arendada nende kognitiivseid oskusi: õppimise käigus keskendutakse probleemide lahendamisele, modelleerimisele, andmete tõlgendamisele ja analüüsimisele ning arvutusliku mõtlemise arendamisele [1].

1.2 Komplekt

WeDo 2.0-i põhikomplekt sisaldab järgmisi osasid [1]:

1) 280 LEGO klotsi;
2) WeDo 2.0 Smarthub;
3) mootor;
4) kallutusandur;
5) liikumisandur.

1.2.1 Smarthub

WeDo 2.0 Smarthub ehk roboti „aju“, mis on näidatud joonisel 1, on elektrooniline LEGO ehitusklots, millel on kaks porti sisend-väljundseadmete jaoks (sinna alla kuuluvad mootor, liikumisandur ja kallutusandur), roheline nupp, mille vajutamine võimaldab robotit mõne muu seadmega paaritada ning RGB LED valgusti. Lisaks suudab Smarthub mängida helisid etteantud sagedustel. Aju kasutab vooluallikana kahte AA patareid või taaslaetavat akut. Ajusse on integreeritud Bluetooth Low Energy tehnoloogia, mis võimaldab tal arvutitega, tahvelarvutitega ja mobiilidega ühenduda ning suhelda. [1, 2]

Joonis 1. WeDo 2.0 Smarthub [2].

1.2.2 Mootor

WeDo 2.0-i mootorit, mis on kujutatud joonisel 2, on võimalik panna pöörlema päripäeva ja vastupäeva. Tegu on lihtsakoelise mootoriga, mis ei anna roboti ajule tagasisidet selle kohta, kui palju ta parasjagu pöörelnud on, mistõttu ei ole võimalik seda mingi kindla pöördenurga võrra pöörlema panna, vaid mootor üritab pöörelda parasjagu sellise võimsusega, mille roboti aju talle ette seadnud on. [1]

Joonis 2. WeDo 2.0-i mootor [3].

1.2.3 Kallutusandur

WeDo 2.0-i kallutusandur, mis on kujutatud joonisel 3, edastab roboti ajule andmeid selle kohta, millise kalde all ta parasjagu on [1]. Andmeid edastatakse ajule kallutusanduri kahe telje kalde põhjal 90 kraadi ulatuses (-45 kuni 45 kraadi). Näiteks, kui kallutada kallutusandur 20 kraadi tahapoole nii, et ta küljele kallutatud ei ole, siis annaks ta ühe telje puhul kalde väärtuseks -20 ning teise telje puhul 0 kraadi.

Joonis 3. WeDo 2.0-i kallutusandur [4].

Ametlikus LEGO WeDo 2.0-i tarkvaras on kallutusanduri andmete töötlemine lihtsustatud nii, et kalde väärtuseks antakse üks viiest võimalikust väärtusest: „kalle ette“, „kalle taha“, „kalle vasakule“, „kalle paremale“ või „kalle puudub“ [5].

1.2.4 Liikumisandur

WeDo 2.0-i liikumisandur, mida on võimalik näha joonisel 4, annab roboti ajule infot selle kohta, kas tema ees on hetkel mingi objekt ning kui on, siis kui kaugel see on. Liikumisandur suudab objekte tuvastada kuni 15 cm kauguselt. Roboti aju suudab lisaks sellele ka loendada ning arvet pidada, mitut liikumist ta enda ees näinud on [1].

Joonis 4. WeDo 2.0-i liikumisandur [6].

WeDo 2.0-i ametlikus tarkvaras on liikumisanduri kasutamist lihtsustatud nii, et programmile antakse tagasisidet vaid selle kohta, kas mingi objekt on liikumisanduri nägemisulatuses, kas andur tuvastab enda ees mingit liikumist, kas miski liigub hetkel liikumisandurile lähemale või kas miski liigub andurist kaugemale [5].

1.3 Tarkvara

LEGO WeDo 2.0-i ametlik tarkvara on toetatud järgnevatel operatsioonisüsteemidel [7]:

1) Windows (alates Windows 7-st) – Windows 7 puhul on tarkvara toimimiseks vajalik BLED112 Bluetooth Low Energy adapter,
2) Mac OS (alates Mac OS X 10.10-st),
3) Chrome OS (alates versioonist 50),
4) iOS (alates iOS 8.1-st) – toimib alates 3. generatsiooni iPad ja iPad mini puhul,
5) Android 4.4.2 Kitkat – toimib osade Androidi tahvelarvutite puhul.

Tarkvara on tasuta allalaaditav LEGO Education veebilehel [8].

1.3.1 Tarkvara kasutamine

Peale tarkvara käivitamist on esmalt vajalik ühendada enda seade WeDo 2.0 ajuga: selleks tuleb vajutada aju pealpool olevale rohelisele nupule, vajutada programmi aknas roboti lisamise nupule ning vajutada avanenud listis sellele roboti nimetusele, millega soovitakse ühendust luua. Kui ühendus on loodud, siis on võimalik ka koheselt hakata roboti juhtimiseks programme käivitama. Programmeerimine toimub piltblokkide ahelasse paigutamise abil, mis on nähtav ka joonisel 5.

Joonis 5. WeDo 2.0-i näidisprogramm platvormil Windows 10.

Programmi käivitamisel hakatakse piltblokkide funktsioone vasakult paremale ükshaaval käivitama nii, et uut blokki ei käivitata enne, kui eelmine on täidetud.

1.3.2 Piltblokid ja programmide toimimine

Piltblokid võib üldiselt jagada kolme kategooriasse: töövooga seotud blokid, käsklustega seotud blokid ning sisendblokid [5].

Töövooga seotud blokid aitavad läbi viia roboti programmi loogikat. Sinna alla kuuluvad näiteks sellised blokid nagu „Start Block“, mis alustab roboti programmi tööd, kui sinna peale vajutada, ja „Wait For Block“, mille külge saab haakida mõne sisendbloki, mis esindab kas mingit numbrilist väärtust X, mille korral oodatakse järgmise käsu juurde minemiseni X sekundit, või mingit tõeväärtusega seotud sisendit – näiteks „Tilt Down“, mille korral liigutakse käskude ahelas järgmise käsuni alles siis, kui kallutusandur on ettepoole kallutatud. Kogu loodud blokkide või käskude jada on võimalik ka korduma panna, kasutades blokki „Repeat Block“, mille alla saab koondada mistahes blokkide jada. Bloki „Repeat Block“ külge saab haakida ka sisendbloki: kui sisendblokk esindab arvu X, siis korratakse antud tsüklit X korda – teisisõnu käitub see blokk kui *for*-tsükkel. Kui sisendblokk esindab mingit tõeväärtust, siis korratakse tsüklit iga kord, kui tsükli lõpus on antud tõeväärtus tõene – teisisõnu käitub antud olukorras see blokk nagu *do-while*-tsükkel. Kui sisendblokk puudub, siis korratakse antud tsüklit lõpmatuseni ning tsükkel toimib nagu *while*-tsükkel, millel on garanteeritult igal korral tõeväärtus tõene.

Käsklustega seotud blokkide alla kuuluvad mootoriga seotud blokid, arvude kuvamise blokid ning LED RGB valgusti blokk. Mootori blokkide abil saab roboti küljes oleva mootori tööle panna ning otsustada seejuures, millises suunas mootor pöörleb, kui kiiresti ta pöörleb (sisendiks saab anda arvulise väärtuse 0-10) ning kui kaua ta pöörleb. Arvude kuvamise blokkidega on võimalik programmis olevale ekraanile kuvada näiteks sensorite poolt tagastatavaid väärtusi või juba ekraanil olevate väärtustega arvutusi teha (liita, lahutada, korrutada, jagada). Väärtust, mis on viimasena ekraanile kuvatud, on võimalik ka sisendparameetrina mõnele teisele blokile anda. LEG RGB valgusti blokiga saab muuta roboti aju peal oleva valgusti poolt näidatavat värvi – selleks on vaja parameetrina talle ette anda arv (vahemikus 0-10), mis tõlgitakse programmisiseselt ümber üheks kindlaks värviks.

Sisendblokke kasutatakse töövooga seotud blokkidele või käsklustega seotud blokkidele sisendparameetrite andmiseks. Selleks, et siduda ühte sisendblokki mingi blokiga, mis vajab sisendparameetrit, liigutatakse sisendblokk selle bloki alumisse ossa, mille tagajärjel nad

ühenduvad. Sisendblokid jagunevad kaheks: sensorite info sisendblokid ning programmisisese info sisendblokid.

Sensorite info sisendblokid annavad infot liikumisanduri või kallutusanduri hetkenäitude kohta. Hetkenäidud võivad olla kahel kujul: numbrilise väärtusena või tõeväärtusena. Liikumisanduri puhul tagastatakse numbrilise väärtusena liikumisanduri poolt tuvastatud objekti kaugus vahemikus 0-10 (cm). Tõeväärtuste puhul on valida kolme juhtumi vahel: „Distance Change Closer“, „Distance Change Further“ või „Any Distance Change“.

Kallutusanduri puhul tagastatakse numbrilise väärtusena arv selle kohta, mis suunas kallutusandur hetkel kallutatud on (0 – neutraalne, 3 – tahapoole, 5 – paremale, 7 – vasakule, 9 - ettepoole). Tõeväärtuste puhul on valida viie erineva juhtumi vahel: „Tilt Up“, „Tilt Down“, „Tilt This Way“, „Tilt That Way“ ning „Any Tilt“.

Programmisisese info sisendblokkide puhul antakse sisendväärtused programmi sees kasutaja enda poolt. Sisendiks on võimalik anda enda kirjutatud teksti, enda kirjutatud arvu, suvalise numbri (programm annab sel juhul sisendiks suvalise numbri vahemikus 1-9) või selle, mis on hetkel kirjas programmi sees oleval ekraanil (võib olla arvu või teksti kujul).

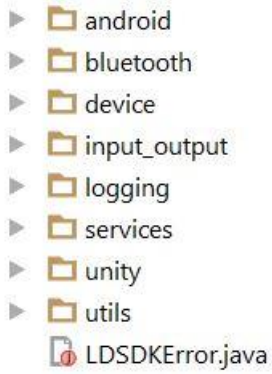
1.4 WeDo 2.0-i Androidi SDK

2016. aasta suvel tegi LEGO avalikuks ametlikud WeDo 2.0-i SDK-d kõigi enda tarkvara platvormide jaoks: Windows, Mac OS, iOS ning Android. Antud SDK-sid on LEGO kasutanud enda WeDo 2.0-i tarkvara loomiseks. SDK-sse on programmeeritud kõik, mis on vajalik, et robotiga ühenduda, talle infot saata ning tema sensoritelt infot lugeda. Seda ära kasutades on vajalik sisuliselt ainult kasutajaliidese loomine, et valmis saada töötava tarkvaraga, millega oleks võimalik WeDo 2.0-le programme kirjutada ning neid käivitada.

Kuna antud bakalaureusetöö eesmärk on luua WeDo 2.0-i SDK Pythonis, siis otsustas töö autor lähtuda selle loomisel avalikest LEGO SDK-dest. Täpsemalt valis töö autor uurimisaluseks Androidi SDK. Valiku põhjendus seisneb selles, et Androidi SDK on programmeeritud programmeerimiskeeles Java ning erinevalt teiste SDK-de poolt kasutatavatest keeltest, on töö autor antud keelega varem kokku puutunud ning tunneb selle süntaksit, mis teeb selle SDK lähtekoodi uurimise oluliselt lihtsamaks.

1.4.1 Arhitektuuri ülevaade

Joonisel 6 on näidatud Androidi SDK üldine struktuur kaustade kaupa. Struktuur on üldiselt jagatud kaheksaks alamkaustaks: *android*, *bluetooth*, *device*, *input_output*, *logging*, *services*, *unity* ning *utils*.



Joonis 6. WeDo 2.0-i Androidi SDK struktuur.

Kaustas *android* sisalduvates failides kasutatakse ära Androidi Bluetooth'iga seotud klasse, et skannida Bluetooth Low Energy (BLE) tehnoloogial põhinevate seadmete poolt edastatud üldist infot: mis on seadme nimi, millised teenused seadme poolt pakutavad on, kas seadmega on võimalik ühenduda, milline on seadme signaalitugevus, andmeid seadme tootja kohta ning veel mõningaid andmeid.

Kaustas *bluetooth* olevad failid tegelevad üldise BLE suhtlusega, kasutades Androidi seadme sisse ehitatud Bluetooth'i adapterit. Need failid aitavad kõigepealt leida ümbritsevas keskkonnas kõik avalikud BLE seadmed ning, kasutades eelpool mainitud kaustas *android* olevaid faile, saada nende seadmete kohta algset infot. Seejärel võimaldavad need failid leitud seadmetest ühe seadmega ühendus luua ning seda ühendust haldama hakata: anda ühendatud robotile edasi käske, lugeda erinevate BLE teenuste väärtusi ning jälgida erinevate BLE teenustes sisalduvate tunnuste väärtuste muutumisi.

Kausta *device* failide ülesandeks on hallata neid BLE seadmeid, mida *bluetooth* kaustas olevate failide abil ümbrusest skannimisel leitakse ning pidada arvet hetkel paaritatud seadme näitajate üle. Leitud seadmete põhjal luuakse `LegoDevice`-objektid, milles hoitakse infot selle kohta, mis tüüpi seadmetega tegu on – kui tegu on WeDo 2.0 robotiga, siis uuritakse ka välja, missugused I/O seadmed roboti portidesse ühendatud on ning milline on roboti patareide või aku hetkeseis.

Kaustas *input_output* leiduvad failid võib jagada kaheks: käskudega seotud failid ning andmete formaatidega seotud failid. Käskudega seotud failid on *OutputCommand.java* ning

InputCommand.java. Käskudega seotud failide ülesandeks on luua etteantud parameetrite põhjal baidijada, mida robot mõistaks ning mida saaks seejärel roboti ajule saata, et ta soovitud käsku täidaks. Failide *OutputCommand.java* ja *InputCommand.java* vahe seisneb selles, et *OutputCommand.java* tegeleb selliste käskudega, mille põhjal robot midagi koheselt muudab, näiteks mootori pööramisega seotud käsud või RGB LED valgusti värvi muutmise käsud; *InputCommand.java* seevastu annab robotile käske, mis on seotud roboti anduritelt loetavate väärtuste lugemisega või nende väärtuste formaatide sätestamisega.

Andmete formaatidega seotud failid on *DataFormat.java* ja *InputFormat.java*. *DataFormat.java* ülesanne on luua `DataFormat` objekt, mis hoiaks endas infot mingisuguse valiidse andmeformaadi kohta, mille põhjal oleks võimalik aju sensoritelt infot lugeda. Üks `DataFormat` hoiab endas sensori sätte numbrilist väärtust, andmete ühiku numbrilist väärtust, ühe loetava arvu suurust baitides ning seda, mitu sellist arvu baidijadas on. *InputFormat.java* ülesanne on luua `InputFormat`-objekt, mis aitab sätestada ühte andmete lugemise formaati ühe kindla sensori kohta. Selleks hoiab ta endas pordi numbrit, kus antud sensor paikneb; selle sensori tüübinumbrit; selle sensori hetkesätet ning tõeväärtust selle kohta, kas antud sensori väärtuste muutumist jälgitakse. `InputFormat` ja `DataFormat` aitavad hallata seda, millised sensorid millisel kujul andmed edastavad ning kuidas neid andmeid tõlgendama peaks.

Kaustas *logging* olevad failid on abiks SDK töö haldamisel ning potentsiaalsete vigade leidmisel. Nad väljastavad infot SDK-s toimiva kohta kas niisama infona, hoiatustena või veateadetena.

Kaustas *services* on klassid kõikide WeDo 2.0-i sensorite ning teenuste kohta: mootor, kallutussensor, toonimängija, liikumissensor ja paljud teised. Kõikide teenuste kohta, mis robotiga ühendumisel saadavad on, luuakse neile vastavad teenuste objektid, mis hoiavad endas infot selle kohta, millises pordis nad on, millises formaadis on nende poolt tagastatavad andmed ning millised on nende sensorite poolt viimati tagastatud andmed. Lisaks sellele kasutatakse neid objekte, et saata robotile käske – selleks kasutatakse *bluetooth* kaustas olevat faili *BluetoothIO.java*.

Kaustas *unity* on failid, mille eesmärk on ühilduda Unity mängumootoriga, mille abil on võimalik Androidi rakendusi luua. Antud failid kasutavad ära kõikides eespool nimetatud kaustades sisalduvaid faile, et võtta kokku kogu SDK tööprotsess: skannida kõiki seadeid,

luua ühendus ühega neist, vaadata, milliseid teenuseid ta kasutab ning hallata edaspidi suhtlust loodud rakenduse kasutajaliidese ning roboti aju vahel.

Kaustas *utils* on kaks faili: *ByteUtils.java*, milles leiduvaid funktsioone kasutatakse kümnendsüsteemi arvudest baidijadade moodustamiseks ja baidijadadest kümnendsüsteemi arvude väljalugemiseks, ning *HandlerHelper.java*, milles kasutatakse Androidi klassi `Handler`, et robotile saadetavad käsud ning robotilt saadavad andmed üksteise järel kohale jõuaksid ning et neid oleks võimalik õiges järjekorras töödelda.

2. Bluetooth Low Energy

Käesolevas peatükis antakse ülevaade Bluetooth Low Energy tehnoloogiast. Esmalt selgitatakse, mis on Bluetooth LE ning milleks seda kasutatakse. Seejärel antakse tehniline ülevaade Bluetooth LE arhitektuurist ning selgitatakse seejuures lahti kõige tähtsamad terminid ja põhimõtted. Peatüki lõpus käsitletakse Pythoni teeki pygatt, mida käesoleva töö autor on kasutanud Bluetooth LE suhtluseks töö käigus loodud WeDo 2.0-i Pythoni SDK-s.

2.1 Bluetooth ja Bluetooth Low Energy

Bluetooth on standard traadita side tehnoloogia jaoks, mille abil on võimaldatud seadmete vaheline andmeedastus ning suhtlus. Bluetooth on loodud andmete edastuseks väikeses raadiuses (kuni 100 m), tehes seda madala energiakuluga. Bluetoothi on kasutatud failide edastuseks arvutite, mobiiltelefonide ja muude nutiseadmete vahel, juhtmevabade kõrvaklappidega muusika kuulamiseks, printeriga suhtlemiseks, vabakäeseadmete kasutamiseks ja veel paljuks muuks [9].

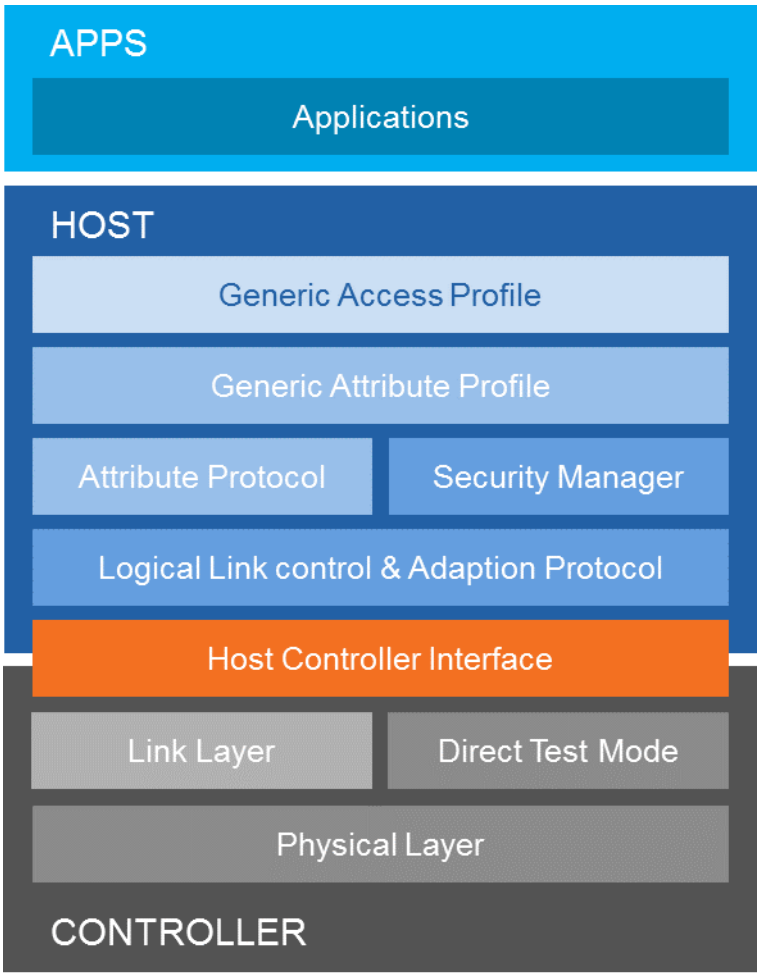
Bluetooth Low Energy (BLE) lisati Bluetoothi standardisse 2010. aastal Bluetooth Core spetsifikatsiooni versiooni 4.0 raames [10]. BLE loodi eesmärgiga võimaldada toodete loomist, mis toetaksid andmevahetust sarnaselt klassikalisele Bluetoothile, kuid väiksema energiakulu, keerukuse ja hinnaga [11]. Väiksem energiakulu saavutatakse tänu sellele, et BLE puhul on saadetavad andmepaketid väiksemamahulised ning neid saadetakse harvem kui klassikalises Bluetoothis – seetõttu on BLE-d kõige enam sobiv kasutada seadmetes, mis ei nõua suuremahulist ega pidevat andmeside kasutamist [9].

Bluetooth LE leiab peamiselt rakendust järgnevates valdkondades [9]:

1) värkvõrk (*Internet of Things*);
2) meditsiiniseadmed, näiteks termomeetrid, vererõhuaparaadid ning glükomeetrid;
3) sporditarvikud, näiteks pulsimõõtjad, GPS positsioneerimisseadmed, sammuloendurid;
4) koduautomaatika, näiteks ruumide kütte ja valgustuse juhtimine;
5) keskkonnaga seotud sensorid, näiteks temperatuurisensorid, baromeetrid ning õhupuhtuse sensorid.

2.2 Bluetooth LE arhitektuur

Bluetooth LE arhitektuur, mida võib näha joonisel 7, jaguneb kolmeks suuremaks osaks: kontroller (*controller*), host (*host*) ja rakendus (*application*) [12, 13]. Kontroller on tüüpiliselt füüsiline seade, mis suudab edastada ja vastu võtta raadiosignaale ning teab, kuidas neid signaale andmepakettidena tõlgendada [13]. Kontroller sisaldab endas Bluetooth LE protokollivirna (*protocol stack*) alumisi kihte [12]. Host sisaldab BLE protokollivirna ülemisi kihte, mille alusel ta haldab seadmete omavahelist suhtlust (sealhulgas suhtluse turvamist) ning määrab selle, kuidas seadme poolt edastatavatele andmetele ligi pääseda [13]. Rakendus on arhitektuuri kõige ülemine kiht ning selle alla kuuluvad kasutajaliides, loogika ja andmete haldamine, mis on loodud vastavalt antud rakenduse tööülesandele [12].



Joonis 7. Bluetooth Low Energy arhitektuur [14].

Lisaks on hosti ning kontrolleri vaheliseks kommunikatsiooniks olemas standardiseeritud protokoll HCI (*Host Controller Interface*). HCI on loodud selleks, et erinevate kompaniide poolt loodud kontrollerid ja hostid oleksid võimelised omavahel probleemideta suhtlema. [12]

2.2.1 Kontroller

Kontroller sisaldab nelja järgnevat komponenti [13]:

1) füüsiline kiht (*physical layer*);
2) lülikiht (*link layer*);
3) *Direct Test Mode*;
4) HCI kontrolleripoolne osa.

Füüsiline kiht sisaldab endas riistvara, mis on võimeline raadiolainete abil infot edastama ning seda vastu võtma. Füüsilises kihis olev raadioliides kasutab kommunikatsiooniks 2,4 GHz raadiosageduse ISM-laineala, mis jagatakse 40-ks kanaliks sagedusvahemikus 2,4 GHz – 2,4835 GHz. Neist 40-st kanalist 37 on kasutusel andmete edastuseks kahe seadme vahel ning ülejäänud kolme kasutatakse seadmete avastamiseks ning nende vahel ühenduse loomiseks.

Lülikiht defineerib protokolli, mille alusel kaks seadet omavahel ühenduse saavad luua: üks seadmetest peab end „reklaamima“ ehk saatma laiali vastavaid andmepakette, milles sisaldub info, et antud seadmega on võimalik ühenduda. Samal ajal peab teine seade skannimise ajal selle teate kätte saama. Teate kätte saanud seade võib selle peale „reklaami“ saatnud seadmele ühendumise taotluse saata, mille peale saavad seadmed hakata üksteisele otse andmepakette saatma. Seejuures on defineeritud kaks rolli: ühenduse taotluse esitanud seade saab ülemaks (*master*) ning ennast „reklaaminud“ seade saab alluvaks (*slave*). [15]

Direct Test Mode on loodud füüsilise kihi testimiseks. Testijal on võimalik anda füüsilisele kihile käsklus saata laiali hulk testpakette või neid vastu võtta. Seejärel saab testija kontrollida ning analüüsida, kas saadetud testpaketid jõudsid kohale ning kas füüsiline kiht toimib nii nagu peaks. [13]

HCI on kontrolleri ja hosti vahelüliks – see võimaldab läbi käskude ja sündmuste saatmise kontrolleril ja hostil omavahel suhelda [13].

2.2.2 Host

Host sisaldab viit järgnevat komponenti [13]:

1) *Logical Link Control and Adaptation Protocol* (L2CAP);
2) *Attribute Protocol* (ATT);
3) *Security Manager Protocol* (SMP)*;*
4) *Generic Attribute Profile* (GATT);

5) *Generic Access Profile* (GAP).

L2CAP täidab kahte eesmärki. Esiteks seob ta endas kokku ATT ja SMP ning loob nendest protokollidest tulevate andmete põhjal Bluetooth LE standardse andmepaketi formaadi (ja vastupidi). Teiseks, tegeleb L2CAP andmete killustamisega ning uuesti kokkupanemisega: ta võtab ülemistest kihtidest suuri pakette ning teeb neist väiksemad tükid, et need mahuksid BLE maksimaalse suurusega pakettidesse, mis mahutavad kuni 27 baiti. Kontrolleri poole pealt võtab ta killustunud paketid ning kombineerib need jällegi kokku suuremateks pakettideks, mida ülemised kihid seejärel kasutada saavad. [12]

ATT määrab kahe ühendatud BLE seadme vahelise suhtluse korralduse. Iga seade on kas klient (*client*), server või mõlemad (hoolimata sellest, kumb neist on ülem ja kumb on alluv). Klient on see, kes teeb serverile päringu ning server vastab sellele. Serveris on andmed organiseeritud atribuutidena. Igal atribuudil on 16-bitine pide (*handle*), unikaalne identifikaator UUID (*universally unique identifier*), hulk pääsuõiguseid (*permissions*) ning mingi väärtus (*value*). Atribuudi pide on identifikaator, mille abil antud atribuudi väärtusele ligi pääsetakse. Atribuudi UUID abil täpsustatakse atribuudi väärtuse tüüp. Olenevalt atribuudi pääsuõigustest võib klient teha serverile päringuid näiteks erinevate atribuutide õiguste kohta või mingi kindla atribuudi kirjutamise või lugemise päringuid. [12]

SMP on protokoll, mis korraldab seadmete turvalist ühenduse loomist, mille käigus jagatakse seadmete vahel võtmeid, autenditakse üksteise võtmed ning luuakse seejärel krüpteeritud suhtlusega ühendus [13].

GATT on Bluetooth LE-le omane andmete organiseerimisega seotud raamistik, mis põhineb ATT-il. See kasutab ATT-i arhitektuuri, et grupeerida kokku seadme iga tööülesandega seotud atribuudid, et seadme erinevaid tööülesandeid mugavamalt hallata ning nende tööülesannetega seotud väärtusi seadmete vahel organiseeritult jagada. [11]

GAP on BLE protokollivirna kõige ülemine kiht ning kontrollib koostöös madalamate kihtidega seda, kuidas seadmed üksteist avastavad, üksteisega ühenduvad ja kuidas need üksteisele informatsiooni edasi annavad [11].

2.2.3 Rakendus

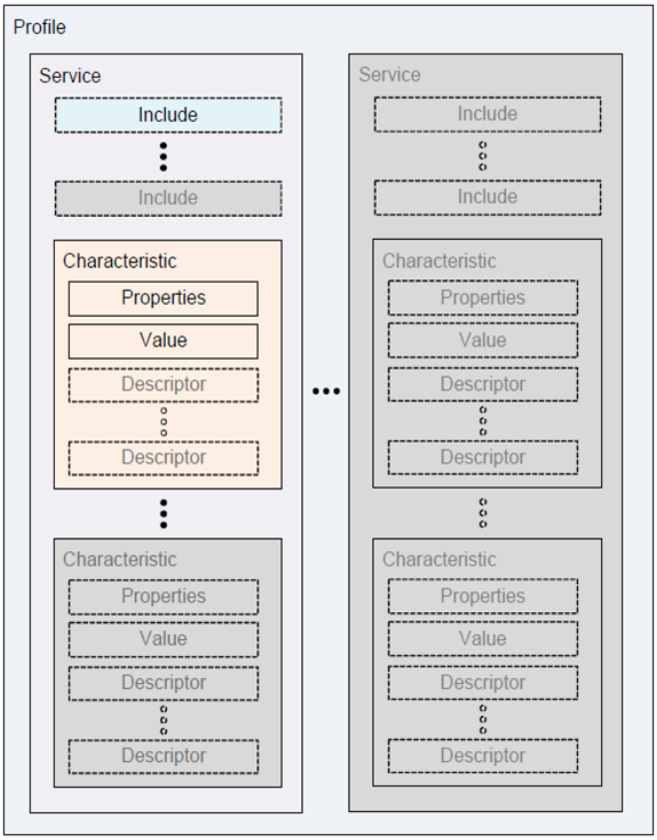
Rakendus on BLE arhitektuuri kõige ülemine osa ning kasutab ära BLE protokollivirna (hosti ja kontrollerit), et talle määratud tööülesannet täita. Rakenduse ülesehitus sõltub täielikult sellest, mis on tema ülesandeks ning ei ole otseselt Bluetoothi spetsifikatsioonide poolt ette määratud. [12]

2.3 GATT

GATT (*Generic Attribute Profile*) on raamistik, mis määrab BLE-põhilise andmemudeli ning sellega seotud protseduurid, mille abil seadmed saavad üksteise andmeid avastada, lugeda ja muuta. Bluetoothi enda poolt on eeldefineeritud suur hulk GATT-põhiseid profiile, mis aitavad ellu viia mingeid kindlaid tööülesandeid (*use case*): näiteks *Find Me Profile*, mis laseb seadmetel üksteist füüsiliselt üles leida (näiteks kasutada nutitelefoni, et leida enda võtmeid) või *Health Thermometer Profile*, mis aitab läbi BLE edastada temperatuuriga seotud andmeid. Lisaks Bluetoothi poolt eeldefineeritud profiilidele on BLE seadmeid loovatel firmadel võimalik ka ise mingile kindlale tööülesandele suunatud GATT-põhine profiil luua. [12]

2.3.1 GATT profiili hierarhia

GATT on üles ehitatud ATT protokolli peale. ATT töö põhineb adresseeritavatel andmetel, mis on atribuutide kujul. Atribuut on andmeolem (*data entity*), mis hoiab infot iseenda kohta ning mingit väärtust, mis on olenevalt antud atribuudi pääsuõigustest ligipääsetav unikaalse pide kaudu. Atribuutide põhjal loob GATT hierarhia, mis organiseerib atribuudid ning aitab luua kindlat raamistikku, mille põhjal oleks võimalik GATT-il põhinevaid profiile luua. [12] Hierarhia on kujutatud joonisel 8.



Joonis 8. GATT profiili hierarhia [16].

Atribuudid on GATT-is grupeeritud teenusteks (*services*). Iga teenus sisaldab endas karakteristikuid (*characteristics*), mis omakorda võivad sisaldada deskriptoreid (*descriptors*). [12]

Teenustesse on grupeeritud karakteristikud, mis on seotud mingi kindla sisulise eesmärgiga. Lisaks võib teenuses sisalduda viiteid teistele teenustele, kasutades selleks atribuuti (*include declaration*), mille väärtuseks on viide mingile teisele teenusele. [12]

Karakteristikutes on vähemalt kaks atribuuti: atribuut, mis hoiab infot antud karakteristikus olevate andmete omaduste (*properties*) kohta (seda atribuuti nimetatakse ka karakteristiku deklaratsiooniks), ning atribuut, milles hoitakse antud karakteristiku väärtust. Lisaks võib karakteristik sisaldada ka deskriptoreid, mis olemasolu korral annavad lisainfot antud karakteristiku ning tema andmete kohta. [12]

2.3.2 GATT-i toimingud

GATT-i toimingud on rangelt defineeritud protseduurid, mille alusel GATT-il põhinevate andmete edastus aset saab leida.

Saadaval on järgnevad toimingud [12]:

1) teenuste ja karakteristikute leidmine;
2) karakteristikute ja deskriptorite väärtuste lugemine;
3) karakteristikute ja deskriptorite väärtuste kirjutamine;
4) serveri poolt saadetavad uuendused.

Teenuste leidmiseks on kaks võimalust: esiteks on võimalik küsida serverilt kõikide teenuste listi ning teiseks on võimalik otsida mingit teenust selle teenuse UUID järgi. Juba leitud teenuse puhul on võimalik leida selles teenuses viidatavad teenused (*included services*). Karakteristikute leidmiseks on samuti kaks võimalust: kas küsida serverilt kõik leitud teenuse karakteristikud listina või otsida karakteristikut UUID järgi. Leitud karakteristiku puhul on võimalik küsida serverilt ka kõik antud karakteristiku deskriptorid. [12]

Karakteristiku ja deskriptori väärtuseid on võimalik lugeda nende pide (*handle*) järgi ning karakteristiku väärtust on võimalik lugeda ka tema UUID järgi [12].

Kui antud karakteristiku või deskriptori puhul on pääsuõigused sellised, et väärtuste kirjutamine on lubatud, siis on võimalik karakteristiku või deskriptori väärtust kirjutada, saates serverile paketi, mis sisaldab endas muudetava väärtuse atribuudi pidet ning uut väärtust.

Kui server on käsu kätte saanud, siis saadab ta kliendile ka sellekohase vastuse. Karakteristiku puhul on võimalik väärtust kirjutada ka nii, et serverilt vastust ei oodata. Sellise lähenemise puhul on ka võimalus, et server võib käsu täitmata jätta ning selle kohta mitte midagi kliendile öelda. [12]

Serveri poolt saadetavad uuendused toimivad sellisel põhimõttel, et server saadab kliendile teate siis, kui mingi karakteristiku väärtus muutunud on. See hoiab kokku energiat ning suhtlemisel kasutatavat andmemahtu, sest klient saab teada, kui mingi väärtus muutunud on, ilma et ta peaks iga natukese aja tagant seda ise serverilt kontrollima. Serverilt tulevaid uuendusi on võimalik üles seada kahel viisil: nii, et server ootab kliendile saadetud teatele vastust või nii, et server saadab teateid ilma vastust ootamata. [12]

2.4 Pygatt

Pygatt on avatud lähtekoodiga Pythoni teek, mille abil on võimalik Bluetooth LE seadmetega GATT-il põhinevat suhtlust läbi viia. Pygatt koosneb kahest eraldiseisvast tööriistast: BGAPI-st ja gatttool'ist. Pygatt'i kasutamisel on võimalik enda arvutist ja operatsioonisüsteemist olenevalt valida, kumba neist kahest kasutada. [17]

BGAPI kasutamiseks on vajalik Bluegiga BLED112 adapteri kasutamine. BLED112, mida on võimalik näha joonisel 9, on Bluetooth LE adapter, mida on võimalik arvutiga USB-liidese kaudu ühendada. BGAPI-t on võimalik kasutada Windowsi, Mac OS-i ning Linuxi operatsioonisüsteemiga arvutite puhul – vajalik on vaid BLED112 adapteri olemasolu. [17]

Joonis 9. Bluegiga BLED112 [18].

Gatttool kasutab töötamiseks arvutisisest Bluetoothi tuge ning põhineb BlueZ'il. BlueZ on Linuxi operatsioonisüsteemile loodud tarkvara, mis implementeerib Bluetoothi protokollivirna. Sellest tulenevalt saab pygatt'is gatttool'i kasutada vaid Linuxi operatsioonisüsteemiga arvutite puhul. Windowsi ja Mac OS-i operatsioonisüsteemiga arvutite puhul on pygatt'i kasutamiseks ainult üks võimalus: kasutada pygatt'is BGAPI tööriista koos BLED112 adapteriga. [17]

Pygatt'is on gatttool'i ning BGAPI tööriistal toetatud ühesugused operatsioonid. BLE seadmete leidmiseks on olemas skannimisega tegelev funktsioon, mis tagastatab listi leitud seadmetest ning leitud BLE seadmega on võimalik luua ühendus selle seadme aadressi põhjal.

Kui mingi seadmega on ühendus loodud, siis on võimalik ka spetsiifilisemaid BLE suhtlusega seotud funktsioone kasutada: võimalik on leida kõik seadme karakteristikud (tagastatakse UUID-de listina), lugeda ja kirjutada karakteristikute väärtusi nii UUID kui ka karakteristiku pide põhjal ning saada seadmelt teateid karakteristikute uuenduste kohta. Serveri teadete aktiveerimiseks kasutatakse funktsiooni, millele antakse ette karakteristiku UUID ning funktsioon, mida hakatakse välja kutsuma, kui antud karakteristiku väärtus on muutunud – välja kutsutava funktsiooni argumentideks annab pygatt muutunud karakteristiku pide ning karakteristiku uue väärtuse.

Võrreldes üldiste GATT-põhiste protseduuridega ei ole toetatud näiteks teenuste avastamine ning seetõttu on pygatt'iga keerulisem aru saada, mis karakteristikuid mis teenustes kasutatakse, kuid olemas on kõik mis vajalik, et seadmega ühendus luua ning sellega BLE suhtlust läbi viia.

3. Töö protsess ja tulemus

Käesolevas peatükis annab töö autor ülevaate WeDo 2.0-i Pythoni SDK loomise protsessist ja töö tulemusest. Täpsemalt selgitatakse lahti loodud SDK arhitektuur ning see, kuidas SDK-d kasutada. Lisaks on välja toodud kõik funktsioonid, mida SDK kasutajal on võimalik WeDo 2.0-ga suhtlemiseks kasutada.

3.1 Töö protsess

Töö algas planeerimisest: vaja oli leida Pythoni teek, mille abil saaks Bluetooth LE suhtlust korraldada. Taolisi mooduleid oli käputäis, kuid neist enamik töötasid vaid Linuxi platvormil ning ainult üks neist töötas Windowsil – selleks oli teek nimega pygatt ning selles olev BGAPI tööriist. Sellegipoolest nõuab pygatt Windowsil ja Mac OS-il toimimiseks BLE adapteri BLED112 olemasolu. Sel põhjusel nõuab ka loodud WeDo 2.0-i Pythoni SDK kasutamine BLED112-te, kuid see-eest töötab see kõigil kolmel platvormil: Windowsil, Mac OS-il ning Linuxil. BLED112 adapteri sai töö autor enda juhendajalt ning asus selle toimimist testima.

2016. aasta detsembris sai autor juhendaja kaudu ka WeDo 2.0-i komplekti kõigi osadega. Järgnesid katsed pygatt'i abil Smarthub'iga ühendust luua. Katsed olid edukad, kuid Smarthub'ile käskude andmiseks oli vaja täpsemalt teada, mismoodi käske baidijadade kujul moodustatakse. Selles oli suureks abiks üks Portugali päritolu blogi nimega O Falcão[1], kus blogi autor oli teinud erinevaid katseid WeDo 2.0-le käskude andmisega ning sensoritelt väärtuste lugemise kohta. Lisaks oli blogis selgitatud mõningate käskluste kohta, millise loogika põhjal neid moodustatakse. Töö autor asus pygatt'i abil neid katseid replikeerima, milles saatis teda edu – selle põhjal oli teada, et pygatt'i on tõepoolest võimalik WeDo 2.0-ga suhtlemiseks kasutada.

Järgmine suurem samm oli WeDo 2.0-i Androidi SDK uurimine ning otsustamine, kuidas loodav Pythoni SDK üles ehitada. Autor otsustas pärast mõningast uurimist Androidi SDK struktuuri suuremal määral järgida ning hakata Pythoni SDK-d sarnaselt arendama. Vastu võeti otsus alustada arendust Androidi SDK-s olevate kaustade *input_output*, *services* ja *byte_utils* põhjal, sest nende kaustade sisu ei ole otseselt seotud Androidiga – nendes olevates failides defineeritakse see, kuidas Smarthub'ile saadetavaid käske moodustatakse ning see, millised seadmed selle külge käivad. SDK arenduseks ja koodi hoidmiseks lõi autor

[1] http://ofalcao.pt/blog/series/wedo-2-0-reverse-engineering

vastava GitHub'i koodirepositooriumi[2]. Kaustades *input*_output, *services* ning *byte_utils* olevad failid said suuremas osas Pythonisse ümbertõlgitud, kuid failid, mis olid seotud *callback*-funktsioonidega, jättis autor algselt Pythonisse tegemata, sest selle keerukus paistis liialt suur ning arvas, et võimalik on ellu viia ka lihtsam lahendus. *Callback*-funktsioonide kasutamise mõte Androidi SDK-s seisneb selles, et Smarthub'ilt saadakse pidevalt infot, kui sensoritelt loetavad väärtused muutuvad ning neid väärtusi muudetakse SDK-s sellele vastavalt. Eelmainitud blogis oleva info abil tehtud testidest selgus aga, et võimalik on ka lihtsalt lugeda sensorite väärtusi siis, kui neid vaja on – see on märksa lihtsam lahendus, kui ühendada kõikide sensorite töö vastavate *callback*-funktsioonidega.

Järgmine suurem probleem oli otsustada, kuidas lahendada Androidi SDK-s olevate kaustade *bluetooth* ja *device* sisu ülekandmine Pythoni SDK-sse. Lähemal uurimisel selgus, et *bluetooth* ja *device* kaustades on väga palju faile, mille ülesandeks on hallata infot mitmete ümbritsevate BLE seadmete kohta (see tuleb kasuks graafilise kasutajaliidesega rakenduste puhul). Töö autor leidis, et Pythonis sellisel funktsionaalsusel erilist mõtet ei ole, sest Pythoni SDK kasutajal peaks olema võimalus ühenduda WeDo 2.0-i seadmega ning seejärel hakata sellega suhtlema, kuid see, missugused seadmed veel arvuti ümbruses saadaval on, ei ole kasutaja jaoks oluline. Sel põhjusel otsustas töö autor võtta neist kahest kaustast Pythoni SDK-sse vaid selline loogika, mis on vajalik Smarthub'iga ühendumiseks ning sellega suhtlemiseks. Selle funktsionaalsuse võtavad Pythoni SDK-s kokku failid *bluetooth_io.py*, *bluetooth_helper.py*, *connect_info.py* ning *service_manager.py*.

Teoreetiliselt töötava Pythoni SDK-ni oli jäänud vaid üks samm: luua fail, milles oleks võimalik luua Smarthub'iga ühendus ning, kasutades teistes kaustades olevaid faile ära, toimida liidesena Smarthub'iga suhtlemisel. Et Smarthub'iga ühendumisel on vaja infot selle Smarthub'i kohta meelde jätta (näiteks millised seadmed millistes portides selle küljes on), siis otsustas autor, et luua võiks klassi, mille isendi loomisel luuakse konstruktoris ühendus Smarthub'iga ning info, mis antud ühenduse ja Smarthub'i kohta meelde peaks jätma, salvestatakse antud isendi muutujatesse. Nii lõigi töö autor faili *smarthub.py*, milles on klass `Smarthub`, mille konstruktoris luuakse Smarthub'iga ühendus. Lisaks sellele paiknevad klassis isendimeetodid, mis aitavad kasutajal mugavalt Smarthub'iga suhelda.

Faili *smarthub.py* loomine võttis kõige rohkem aega, sest selles olevate funktsioonide loomisel ja testimisel tuli välja suurel hulgal pisivigu, mis olid Pythoni SDK loomise algusest

[2] https://github.com/jannopet/LEGO-WeDo-2.0-Python-SDK

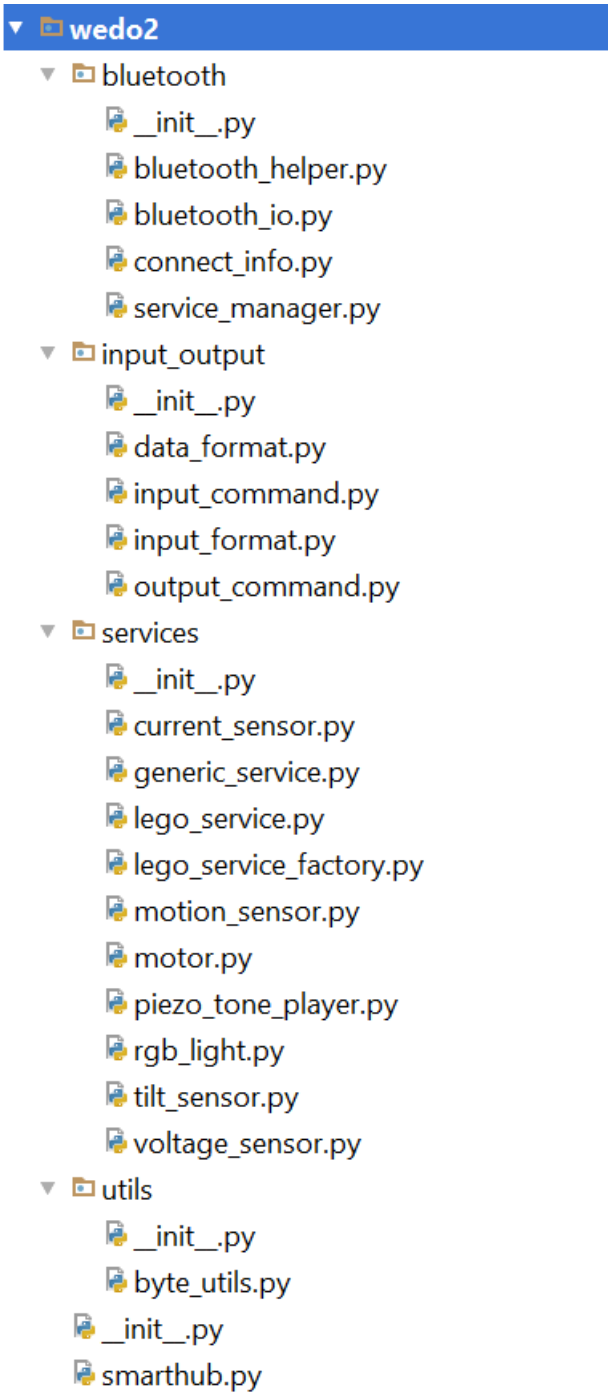
erinevatesse failidesse sisse jäänud. Antud vigu oli äärmiselt ajakulukas üles leida ning seega ka parandada. Vaadates arendamisele tagasi leiab autor, et aega oleks võimalik olnud suuresti kokku hoida, kui Androidi SDK eeskujul oleks loodud logimisega tegelev fail ning arendamise käigus teistes failides seda logimiseks kasutatud.

Valminud SDK PyPI-sse üles panemisel tuli välja, et ei leidu ühtegi PyPI-s olemasolevat pygatt'i versiooni, milles toimiks Windowsis BLED112 adapteri leidmine ilma vigadeta. Seega sai lõpus *smarthub.py* faili lisatud ka klass `_Patched_BGAPIBackend`, milles on üle kirjutatud pygatt'i klassi `BGAPIBackend` meetod `_open_serial_port()`. Tänu sellele parandusele on võimalik PyPI-st loodud teeki installeerida nii, et pygatt ilma vigadeta toimiks.

Kui tulevikus peaks ilmuma näiteks parem Pythoni teek Bluetooth LE suhtluseks (mis ei nõuaks Windowsis ja Mac OS-is lisaadapteri olemasolu), siis oleks selle implementeerimine olemasolevasse Pythoni SDK-sse suhteliselt lihtne: vaja oleks ainult failis *smarthub.py* muuta WeDo 2.0-ga ühendumisega seotud koodiridu ning failis *bluetooth_io.py* väljakutsutavate pygatt'i funktsioonide nimesid.

3.2 Pythoni SDK arhitektuur

Loodud Pythoni SDK arhitektuur on jaotatud neljaks alamkaustaks *bluetooth*, *input_output*, *services*, *utils* ning failiks *smarthub.py*. Lisaks põhifailidele on igas kaustas fail *__init__*.py, mis ei sisalda koodi, aga selle olemasoluga võimaldab Python erinevates kaustades olevatel failidel üksteise sisu importida. Pythoni SDK struktuur, mis on nähtav joonisel 10, on inspireeritud WeDo 2.0-i Androidi SDK struktuurist.



Joonis 10. WeDo 2.0-i Pythoni SDK struktuur.

Kaustas *bluetooth* on neli faili: *bluetooth_io.py*, *bluetooth_helper.py*, *service_manager.py* ning *connect_info.py*. Faili *bluetooth_io.py* ülesandeks on suhelda pygatt'i BGAPI tööriista abil WeDo 2.0-ga. Selles sisalduvad nii spetsiifilised WeDo 2.0-i käske ellu viivad meetodid (näiteks mootori pöörlemisega ning RGB LED valgusti värvi muutmisega seotud funktsioonid) kui ka sensoritelt andmete lugemisega seotud meetodid (näiteks etteantud pordist hetkeväärtuse lugemine või selles pordis oleva sensori väärtuste formaadi muutmine).

Failis *bluetooth_helper.py* on funktsioonid, mille abil luuakse stringi kujul olevad UUID-d, mis esindavad WeDo 2.0-i karakteristikute aadresse. Seda kasutatakse ära *bluetooth_io.py* failis, kus moodustatud UUID-dele käske saadetakse.

Faili *service_manager.py* ülesandeks on kontrollida, mis seadmed ning sensorid WeDo 2.0-l saadaval on ning täpsemalt teada saada, mis portides need asuvad. WeDo 2.0-s kasutatakse nimetust „port“ mitte ainult Smarthub'i küljes olevate portide kohta, vaid ka Smarthub'i sees olevate seadmete kohta – näiteks Smarthub'i sees oleva toonimängija port on 5 ning Smarthub'i peal oleva RGB LED valgusti pordiks on 6. Infot seadmete portide kohta saab hiljem ära kasutada, et teada, mis pordi kaudu iga seadmega suhtlema peaks.

Fail *connect_info.py* aitab hoida andmeid selle kohta, mis tüüpi seade millises pordis asub. Seda kasutatakse ära failis *service_manager.py*, kus iga seadme kohta saadud andmete põhjal luuakse üks `ConnectInfo` klassi isend.

Kaustas *input_output* on neli faili: *data_format.py*, *input_command.py*, *input_format.py*, *output_command.py*. Failis *data_format.py* olevast klassist `DataFormat` loodud isend määrab ühe formaadi, milles WeDo 2.0 andmeid tagastada saab. Näiteks liikumissensor saab andmeid tagastada selle kohta, kui kaugel tema ees olev objekt on või mitut liikumist ta enda ees näinud on, ning lisaks saab ta andmeid edastada erinevates ühikutes. See, millises seadistuses liikumissensor parajasti on, määrab ka selle, millisel kujul tema poolt tagastatav baidijada on. Klassi `DataFormat` isend ütleb ühe sellise seadistuse kohta, mitmeks osaks tagastatav baidijada jagatud on ning mitu baiti igas osas on. Teades formaati, mida sensor hetkel kasutab, on võimalik tema poolt tagastatavaid väärtusi õigesti kümnendsüsteemi arvudeks teisendada.

Failides *input_command.py* ja *output_command.py* luuakse roboti ajule edastatavaid käske vastavalt sensoritelt loetavate andmete kohta (näiteks käsud, mis ütlevad ajule, mille kohta või milliste ühikutega mingi sensor andmeid tagastama peaks) ning otseste funktsioonide kohta (näiteks käsud, mis mõjutavad mootori pöörlemist). Käsud luuakse baidijadade kujul, mida WeDo 2.0-i aju mõistab. Antud faile kasutatakse ära failis *bluetooth_io.py*, kus edastatakse ajule klasside `InputCommand` ja `OutputCommand` põhjal loodud baidijadad.

Failis *input_format.py* oleva klassi `InputFormat` isend määrab baidijadana ühe sensori seadistuse. Ta hoiab baidijadana infot selle kohta, millises pordis antud sensor asub ning mille kohta ja millistes ühikutes ta andmeid tagastama peaks. Antud baidijada kombineeri-

takse klassi `InputCommand` põhjal loodud baidijadaga ning sellest saadakse käsk, mida roboti ajule edastada. Kui aju selle kätte saab, siis asub ta edaspidi vastavalt etteantud formaadile andmeid tagastama.

Kaustas *services* asuvad failid WeDo 2.0-i sensorite ning seadmete defineerimiseks. Seadmete ülemklass `LegoService` on defineeritud failis *lego_service.py* ning selle klassi alamklassid on kõik spetsiifiliste seadmetega seotud klassid (näiteks klass `Motor` failis *motor.py* või klass `TiltSensor` failis *tilt_sensor.py*). Nende klasside isendid luuakse failis *lego_service_factory.py* peale seda, kui failis *service_manager.py* on leitud kõik Smarthub'iga ühendatud ja saadaval olevad seadmed. Loodud isendites hoitakse muutujatena infot selle kohta, mis portides nad asuvad, ning kui tegu on sensoritega, siis ka infot selle kohta, mille kohta ning millises formaadis nad hetkel väärtusi tagastavad. Lisaks sellele on seadmete failides enda spetsiifiliste funktsionaalsustega seotud meetodid (näiteks klassis `TiltSensor` on meetod `get_angle()`, ning klassis `MotionSensor` on meetod `get_distance()`), mis pöörduvad *bluetooth_io.py* faili poole, kus vajalikud Smarthub'ile saadetavad käsud moodustatakse.

Kaustas *utils* on fail nimega *byte_utils.py*. Selles failis on meetodid, mida kasutatakse baidijadade arvudeks teisendamiseks. Selle faili meetodeid kasutatakse kaustas *services* olevates seadmete failides, et sensoritelt loetud baidijadu kasutajale mõistetavateks kümnendsüsteemi arvudeks teisendada.

Fail *smarthub.py* on ainuke fail, mida SDK kasutaja reaalselt kasutama peaks: selle abil saab kasutaja WeDo 2.0-i ajuga ühenduse luua ning selles leiduvate funktsioonidega robotile käske anda ja sensoritelt andmeid lugeda. Kogu madalama taseme loogika, mis on robotiga suhtlemiseks vajalik, viiakse läbi muudes SDK failides.

Kõige suuremad erinevused WeDo 2.0-i Androidi SDK-ga seisnevad selles, et Pythoni SDK-s on ära jäetud kaustad *unity*, *logging*, *device* ning *android*. Kuna *unity* kausta failide mõte Androidi SDK-s on anda võimalus Unity's graafilise kasutajaliidese loomiseks ning kogu ülejäänud SDK ära kasutamine, siis esindab Pythoni SDK-s sisuliselt sarnast eesmärki fail *smarthub.py*.

Suure osa Androidi SDK kaustade *device* ja *bluetooth* sees olevate failide eesmärk oli hallata paljusid ümbruses olevaid BLE seadmeid. Antud töö autor otsustas Pythoni SDK-s sellise lahenduse kasuks, kus tavakasutajal oleks võimalikult mugav läheduses oleva seadmega ühenduda ning et seda tehtaks automaatselt, ilma et selleks eraldi kasutajaliidesega suhtlust

vaja oleks. Sel põhjusel sai Androidi SDK *device* ja *bluetooth* kaustades olevate failide tegevust Pythoni SDK-s märgatavalt vähendatud. Kaustad *logging* ning *android* jäeti ära seetõttu, et logimise funktsionaalsust antud SDK-s implementeeritud ei ole ning *android* kausta sisu ei ole Pythonis toimuvaga kuidagi seotud.

Kõige märkimisväärsem erinevus loodud Pythoni SDK ja uuritud Androidi SDK vahel on see, et Androidi SDK-s põhineb sensoritelt andmete lugemine Smarthub'ilt uuenduste saamisel: kui näiteks kallutussensori väärtus muutub, siis uuendatakse *callback*-funktsioonide abil kallutussensori hetkeväärtust, mida hoitakse muutujana klassi `TiltSensor` isendis. Pythoni SDK-s on väärtuste lugemine tehtud lihtsama variandiga, mille käigus loetakse sensori hetkeväärtus ainult siis, kui SDK kasutaja seda nõuab. Töö autor otsustas sellise lahenduse kasuks, sest kuigi see muudab loodud SDK töö Androidi SDK-s implementeeritud variandiga võrreldes veidi aeglasemaks, lihtsustas selline lähenemine märgatavalt SDK struktuuri ning selle programmeerimist. Lisaks sai seda lahendust testitud ning autor leidis, et sensorite väärtused saab sellisel moel piisavalt kiiresti kätte, et õigustada märkimisväärselt lihtsamat lahendust.

3.3 Teegi kasutamine ja selle toimimine

Teegi kasutamiseks on vajalik arvuti, mis jooksutab Pythonit (alates versioonist 3.4). Kuna teegis kasutatakse Bluetooth LE suhtluseks pygatt moodulit, siis on vajalik ka BLED112 adapteri olemasolu.

Teek on üles ehitatud eesmärgiga olla võimalikult lihtsasti kasutatav. Sel põhjusel otsustas autor luua ühe faili, mida teegi kasutaja kasutama peaks ning mille abil saaks kasutaja WeDo 2.0-ga ühenduse luua ning seda kontrollima hakata. Selleks failiks on *smarthub.py*, mis hoiab endas klassi `Smarthub`. Klassi `Smarthub` isendi loomisel luuakse Smarthub'iga ühendus ning selle isendi kaudu saab välja kutsuda klassis `Smarthub` paiknevaid WeDo 2.0-i funktsionaalsusega seotud meetodeid. Sellise lahenduse põhjal on võimalik ka mitme Smarthub'iga ühendus luua (luues lihtsalt erinevaid `Smarthub` isendeid) – selle poolest on loodud Pythoni SDK võimsam kui LEGO ametlikud SDK-d, kus võimalik on luua ühendus vaid ühe Smarthub'iga korraga.

3.3.1 Teegi installeerimine

Loodud Pythoni SDK on üles laaditud PyPI keskkonda[3]. Selle installeerimiseks on vajalik Pythoni moodul pip. Installeerimiseks on vaja käsureal sisestada järgnev käsk:

```
pip install wedo2
```

Koos Pythoni SDK-ga installeeritakse automaatselt ka pygatt'i teek.

3.3.2 Smarthub'iga ühenduse loomine

Klassis `Smarthub` on konstruktor, mille käivitamisel hakatakse pygatt'iga skannima ümbruses olevaid BLE seadmeid. Seadmed järjestatakse seejärel nende signaalitugevuse alusel ning ühendus luuakse selle seadmega, mille signaal kõige tugevam on. Autor otsustas sellise lahenduse kasuks, sest ta leidis, et kasutajal oleks mugavam, kui ta ei peaks hakkama ühenduse loomiseks midagi rohkemat tegema, vaid kõik tehtaks tema eest ära. Sellise lahenduse puhul on eeldatud, et ühenduse loomise ajal hoitakse suurima signaalitugevuse saavutamiseks WeDo 2.0 aju enda arvuti läheduses, kuid see ei tohiks olla takistuseks, sest ühenduse loomiseks on niikuinii vaja Smarthub'i peal olevat nuppu vajutada, seega Smarthub peaks igatahes kasutaja arvuti läheduses olema. Samuti eeldatakse, et enne konstruktori väljakutset on vajutatud Smarthub'i peal olevat nuppu – seda seetõttu, et seadmete skannimine toimub umbes ühe sekundi jooksul ning kui selle aja jooksul skannimine ühtegi seadet ei leia, visatakse erind.

Alternatiiviks oleks olnud näiteks konstruktori väljakutsumisel anda üheks konstruktori parameetriks enda WeDo 2.0-i seadme nimi, mille alusel oleks kõigi skannimisel leitud seadmete seast valitud just see seade, mille nimi ette oli antud. Sellisel lahendusel on kaks potentsiaalselt negatiivset külge: esiteks, ei pruugi kasutajale olla teada, mis tema WeDo 2.0-i seadme nimi on, ning teiseks võivad mitmel seadmel samasugused nimed olla, seega ei saa näiteks klassiruumi kontekstis (kus on palju seadmeid korraga) kindel olla, et ühendus siiski õige seadmega luuakse.

Kui Smarthub'iga on ühendus loodud, siis kasutatakse faili *service_manager.py*, et antud Smarthub'i kõik saadaval olevad sensorid ning seadmed tuvastada. Tuvastatud seadmete põhjal luuakse klassi `LegoService` vastavate alamklasside isendid ning neid hoitakse klassi `ServiceManager` isendis hulga andmetüübis (ingl k. *set*). Kui seni kõik õigesti toiminud on, siis on Smarthub valmis järgnevaid käske saama.

[3] https://pypi.python.org/pypi/wedo2/1.1.1

Eeldusel, et kasutajal on olemas vajalik riistvara ning antud teek on installeeritud, tuleb kasutajal `Smarthub` klassi importimiseks ning `Smarthub` isendi loomiseks kasutada enda Pythoni programmis kahte järgnevat rida:

```
from wedo2.smarthub import Smarthub
isend = Smarthub()
```

Neile kahele reale võivad juba järgneda otseselt loodud isendiga seotud meetodite väljakutsed. Isendiga ühenduse lõpetamiseks tuleb välja kutsuda meetod `disconnect()`. Tähele tasub panna ka seda, et enne konstruktori väljakutset peaks kasutaja vajutama Smarthub'i peal olevale nupule.

3.3.3 Funktsioonide kasutamine

Funktsioonid klassis `Smarthub` on jagatud vastavalt seadmetüüpidele (mootor, toonimängija, RGB LED valgusti, kallutussensor, liikumissensor, pingesensor ning voolutugevuse sensor) sektsioonidesse.

Funktsiooni väljakutsumisel pöördutakse `ServiceManager` isendi poole, mille kaudu kontrollitakse, kas antud funktsiooniga seotud seade on saadaval või ei. Kui ei ole, siis väljastatakse selle kohta teade, et antud seade ei ole saadaval. Kui seade aga on saadaval, siis tagastab `ServiceManager` selle seadme isendi (näiteks klassi `MotionSensor` isendi). Seejärel pöördutakse juba otseselt selle seadme isendi poole ning kutsutakse omakorda temas vajalik funktsioon välja.

Kuna paljude sensorite puhul on mitu erinevat toimimisega seotud sätet, siis on olemas ka funktsioonid, mis nende sensorite sätteid muudavad. Näiteks kallutussensori puhul on kaks võimalikku sätet: esimese sätte (*Direction Mode*) puhul tagastab Smarthub eeldefineeritud täisarvulise väärtuse kallutussensori asendi kohta (0 – neutraalne, 3 – tahapoole, 5 – paremale, 7 – vasakule, 9 - ettepoole), kuid teise sätte (*Angle Mode*) puhul tagastatakse ennik, kus esimene väärtus annab kaldenurga vasakule (-45 kuni 0 kraadi) või paremale (0 kuni 45 kraadi) ning teine väärtus annab kaldenurga tahapoole (-45 kuni 0 kraadi) või ettepoole (0 kuni 45 kraadi). Teise sätte kehtestamiseks on vajalik meetodi `set_tilt_mode_to_angle()` väljakutsumine.

Kui loodud on `Smarthub`-objekt nimega „isend", siis näiteks selleks, et Smarthub'i küljes olev mootor päripäeva 50%-lise võimsusega pöörlema panna, seejärel kallutussensori kallet küsida ning seejärel Smarthub'iga ühendus lõpetada, tuleks kasutada järgnevaid koodiridu:

```
isend.turn_motor(50)  # Pannakse mootor pöörlema 50% võimsusega
kalle = isend.get_tilt()  # Küsitakse kallutussensori kallet
isend.disconnect()    # Lõpetatakse isendiga ühendus
```

3.3.4 Toetatud funktsionaalsused

Järgnevas loetelus on välja toodud kõik `Smarthub` klassis toetatud funktsionaalsused:

1) mootori pöörlema panemine etteantud võimsusel meetodiga `turn_motor(power)` (negatiivsete väärtuste puhul pööratakse mootorit vastupäeva ning positiivsete korral päripäeva);

2) mootori peatamine äkitselt meetodiga `motor_brake()`;

3) mootori peatamine sujuvalt meetodiga `motor_drift()`;

4) noodi mängimine meetodiga `play_note(note, octave, duration)`, kus muutuja `note` on number vahemikus 1–12 (1 – C, 2 – C#, 3 – D, …, 12 – B), `octave` on number vahemikus 1–6 ning `duration` on aeg millisekundites vahemikus 0–65536;

5) etteantud sagedusel tooni mängimine meetodiga `play_frequency(frequency, duration)`, kus `frequency` on vahemikus 0–1500 hertsi ning `duration` on aeg millisekundites vahemikus 0–65536;

6) esitatava tooni mängimise lõpetamine meetodiga `stop_playing()`;

7) RGB LED valgusti sätestamine indekseeritud väärtuste jaoks meetodiga `set_rgb_light_mode_to_discrete()` – see on ka vaikimisi säte RGB LED valgusti jaoks;

8) RGB LED valgusti sätestamine RGB kujul esitatud väärtuste jaoks meetodiga `set_rgb_light_mode_to_absolute()`;

9) RGB LED valgusti värvi muutmine indeksi abil meetodiga `change_color_index(index)`, kus `index` on väärtus vahemikus 0–10 (0 – tule kustutamine, 1 – roosa, 2 – lilla, 3 – tumesinine, 4 – helesinine, 5 – heleroheline, 6 – tumeroheline, 7 – kollane, 8 – oranž, 9 – punane, 10 – valge);

10) RGB LED valgusti värvi muutmine RGB väärtuste abil meetodiga `change_color(red, green, blue)`, kus `red`, `green` ja `blue` on väärtused vahemikus 0–255;

11) kallutussensori sätestamine tagastamaks eeldefineeritud väärtusi meetodiga `set_tilt_mode_to_direction()` – see on ka vaikimisi säte kallutussensori jaoks;

12) kallutussensori sätestamine tagastamaks kaldenurka meetodiga `set_tilt_mode_to_angle()`;

13) kallutussensori kalde küsimine meetodiga `get_tilt()`, kus väärtus tagastatakse vastavalt sellele, milline säte kallutussensoril hetkel on (*Direction mode* või *Angle mode*);

14) liikumissensori sätestamine tagastamaks ees oleva objekti kaugust meetodiga `set_motion_sensor_to_detect()` – see on ka vaikimisi säte liikumissensori jaoks;

15) liikumissensori sätestamine tagastamaks arvu selle kohta, mitu objekti tema ees liikunud on meetodiga `set_motion_sensor_to_count()`;

16) liikumissensori ees oleva objekti kauguse küsimine meetodiga `get_object_distance()`, mis tagastab väärtuse vahemikus 0–9 ning kui objekti nii lähedal ei näe, siis tagastab väärtuseks 10;

17) liikumissensori eest läbi käinud objektide arvu küsimine meetodiga `get_motion_count()`;

18) Smarthub'i patareil oleva pinge küsimine meetodiga `get_voltage()`, mis tagastab väärtuse millivoltides;

19) Smarthub'i patareil oleva voolutugevuse küsimine meetodiga `get_current()`, mis tagastab väärtuse milliamprites;

20) Smarthub'iga ühenduse katkestamine meetodiga `disconnect()`.

Eelnimetatud funktsioonide kirjeldused on kommentaaride kujul lahti selgitatud ka failis *smarthub.py*.

4. Kokkuvõte

Antud bakalaureusetöö eesmärgiks oli luua LEGO Education WeDo 2.0-i jaoks teek programmeerimiskeeles Python, mis võimaldaks WeDo 2.0-ga ühenduda ning teegis sisalduvate funktsioonide abil WeDo 2.0-ga suhtlust läbi viia.

Töö esimeses peatükis anti ülevaade WeDo 2.0-st ning selle toimimisest. Lisaks kirjeldati, kuidas on üles ehitatud LEGO WeDo 2.0-i Androidi SDK arhitektuur. Töö teises peatükis anti ülevaade Bluetooth Low Energy tehnoloogiast ning Pythoni SDK loomiseks kasutatud Pythoni teegist pygatt. Töö kolmandas peatükis kirjeldati töö protsessi ning loodud tarkvara arhitektuuri, selle toimimist ja selle kasutamist.

Töö tulemusena valmis Pythoni SDK LEGO WeDo 2.0-le, mille abil on võimalik WeDo 2.0-i Smarthub'iga ühenduda ning sellega suhelda. Valminud Pythoni SDK toetab Windowsi, Mac OS-i ning Linuxi platvorme, kuid selle kasutamiseks on vajalik Bluetooth Low Energy adapteri BLED112 olemasolu.

Valminud SDK abil on võimalik luua Pythonis kasutajaliideseid WeDo 2.0-ga suhtlemiseks ning ka lihtsalt Pythoni skripte, millega WeDo 2.0-i juhtida. Pythoni SDK-ga on võimalik ka mitme WeDo 2.0-ga korraga ühenduda ning suhelda, mis ei ole LEGO ametlikes SDK-des toetatud.

Antud SDK-d on võimalik edasi arendada, muutes faili *smarthub.py* kasutajate jaoks paindlikumaks ja universaalsemaks. Lisaks saab SDK-d muuta vastavalt sellele, kui WeDo 2.0-le peaks lisanduma uut riistvara või kui esile peaks kerkima mõni uus Pythoni teek Bluetooth LE suhtluseks, mis oleks parem kui hetkel kasutatav pygatt.

5. Viidatud kirjandus

[1] LEGO® Education WeDo 2.0 Core Set. https://education.lego.com/en-us/products/lego-education-wedo-2-0-core-set/45300 (11.03.2017)

[2] WeDo 2.0 Smart Hub. https://education.lego.com/en-us/products/wedo-2-0-smart-hub/45301 (09.04.2017)

[3] WeDo 2.0 Medium Motor. https://education.lego.com/en-us/products/wedo-2-0-medium-motor/45303 (09.04.2017)

[4] WeDo 2.0 Tilt Sensor. https://education.lego.com/en-us/products/wedo-2-0-tilt-sensor/45305 (09.04.2017)

[5] WeDo 2.0 Programming Block Descriptions. https://education.lego.com/en-us/support/wedo-2/programming-block-descriptions (11.03.2017)

[6] WeDo 2.0 Motion Sensor. https://education.lego.com/en-us/products/wedo-2-0-motion-sensor/45304 (09.04.2017)

[7] WeDo 2.0 System Requirements. https://education.lego.com/en-us/support/wedo-2/system-requirements (10.04.2017)

[8] WeDo 2.0 Downloads. https://education.lego.com/en-us/downloads/wedo-2 (10.04.2017)

[9] Gupta N. Inside Bluetooth low energy, Second Edition. USA: Artech house. 2016.

[10] Bluetooth SIG. Our History. https://www.bluetooth.com/about-us/our-history (19.04.2017)

[11] Mikhaylov K, Plevritakis N, Tervonen J. Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI. Sensors, 2012. http://www.mdpi.com/2224-2708/2/3/589/htm (19.04.2017)

[12] Townsend K, Cufí C, Davidson R. Getting started with Bluetooth low energy: tools and techniques for low-power networking. USA: "O'Reilly Media, Inc.". 2014.

[13] Heydon R. Bluetooth Low Energy: The Developer's Handbook. USA: Prentice Hall. 2012

[14] Bluetooth Core Specification. https://www.bluetooth.com/specifications/bluetooth-core-specification (19.04.2017)

[15] Gomez C, Oller J, Paradells J. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. Sensors, 2012. http://www.mdpi.com/1424-8220/12/9/11734/htm (20.04.2017)

[16] Generic Attributes (GATT) and the Generic Attribute Profile. https://www.bluetooth.com/specifications/generic-attributes-overview (20.04.2017)

[17] Peplin C. PyGATT - Bluetooth Low Energy (BLE) and Python. http://christopher-peplin.com/2015/10/22/pygatt-python-bluetooth-low-energy-ble/ (11.03.2017)

[18] Bluegiga BLED112 *Bluetooth*® Smart Dongle. http://www.silabs.com/products/wireless/bluetooth/bluetooth-low-energy-modules/ble121lr-bluetooth-smart-long-range-module1 (22.04.2017)

Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Janno Peterson**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Pythoni SDK LEGO WeDo 2.0-le,
(*lõputöö pealkiri*)

mille juhendaja on Aivar Annamaa,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **10.05.2017**