LAPIN AMK
Lapland University of Applied Sciences

AJOSIMULAATION TOTEUTUS UNITY 3D -PELIMOOTTORILLA

Tomi Tapio

Opinnäytetyö
Tietotekniikan koulutusohjelma
Insinööri (AMK)

2014

LAPIN AMK
Lapland University of Applied Sciences

LAPIN AMMATTIKORKEAKOULU

TEOLLISUUDEN JA LUONNONVAROJEN OSAAMISALA

Tietotekniikan koulutusohjelma

Opinnäytetyö

AJOSIMULAATION TOTEUTUS UNITY 3D -PELIMOOTTORILLA

Tomi Tapio

2014

Toimeksiantaja Ohjelmistotekniikan laboratorio pLAB

Ohjaaja Maisa Mielikäinen

Hyväksytty 23.4.2014

Työ on luettavissa Theseus-verkkokirjastossa.

ALKUSANAT

Työn toimeksiantaja oli Lapin ammattikorkeakoulun ohjelmistotekniikan laboratorio pLAB. Haluaisin kiittää pLABin laboratorioesimiestä Pertti Rauhalaa mainion aiheen ideoinnista sekä pLABin ammattitaitoista henkilökuntaa. Osoitan kiitoksen ohjaavalle opettajalleni Maisa Mielikäiselle sekä opettaja Petri Hannulalle työn ohjauksesta ja sisällön ideoinnista. Kiitän myös ahkeraa siskoani opinnäytetyön kirjallisen osuuden vinkeistä. Lopuksi suurin kiitos avopuolisolleni Roselle sekä opiskeluihin liittyvissä että niihin liittymättömissä asioissa.

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön
tiivistelmä

Teollisuus ja luonnonvarat
Tietotekniikan koulutusohjelma

Tekijä Tomi Tapio Vuosi 2014

Toimeksiantaja Ohjelmistotekniikan laboratorio pLAB
Työn nimi Ajosimulaation toteutus Unity 3D -pelimoottorilla
Sivu- ja liitemäärä 62 + 3

Opinnäytetyön aiheena oli ajosimulaation toteutus Unity 3D -pelimoottorilla. Työn idea syntyi, kun ENVI-oppimisympäristöön tarvittiin liikealustalla toimiva ambulanssisimulaattori, jonka avulla opiskelijat voisivat oppia erilaisia hoitotilanteita. ENVI on terveydenalan opiskelijoiden virtuaalinen oppimisympäristö. Työn toimeksiantaja oli ohjelmistotekniikan laboratorio pLAB, joka sijaitsee Lapin ammattikorkeakoulun tiloissa Rantavitikalla.

Opinnäytetyön tavoitteena oli toteuttaa Unity 3D -ohjelma Moog-liikealustan ohjaamiseen UDP-yhteyden kautta. Liikealustan liikkeen tuli olla mahdollisimman sulavaa. Lisäksi tavoitteeksi sisällytettiin virtuaalilasien yhteiskäyttö liikealustan kanssa. Lopputuloksen tuli myös olla helposti integroitavissa uusiin Unity 3D -projekteihin. Työn keskeisenä ongelmana oli saada Unity 3D -pelimoottori kommunikoimaan liikealustan kanssa UDP-yhteyden avulla sekä saada liikealustan liikkuminen tasaiseksi.

Sovellus toteutettiin C#-ohjelmointikielellä Unity 3D -pelimoottorilla. Pelimoottorin ja liikealustan välinen kommunikointi tapahtuu UDP-yhteyden välityksellä. Työn tuloksena syntyi Unity 3D -ohjelmisto, joka ohjaa liikealustaa. Liikealusta ohjautuu pelimaailman auton mukaisesti, lukuun ottamatta äkkinäisiä liikkeitä, jolloin liikealustan liike on sulavaa. Pelimaailmassa olevan auton ajaminen tapahtuu ratti-poljin-yhdistelmän avulla ja sovelluksen käyttäjä pystyy kokemaan virtuaalimaailman virtuaalilasien kautta. Ambulanssisimulaattorin toteutus aloitetaan sovelluksen pohjalta. Lopputulos toteutettiin myös niin, että se on helposti liitettävissä uusiin Unity 3D -projekteihin ja peliobjekteihin.

Avainsanat Unity 3D, Liikealusta, Ajosimulaatio, Oculus Rift

LAPIN AMK
Lapland University of Applied Sciences

School of Industry and Natural Resources
Information Technology Programme

Abstract of Thesis

Author	Tomi Tapio	Year	2014
Commissioned by	Software Engineering Laboratory pLAB		
Subject of thesis	Driving Simulation Implementation with Unity 3D Game Engine		
Number of pages	62 + 3		

The subject of this Bachelor's thesis was driving simulation implementation with the Unity 3D game engine. The thesis was part of further development of a project called ENVI, which is a virtual learning environment for the health care students. There was a need for an ambulance simulator on the motion platform so that the students can practice different scenarios of health care. This project was commissioned by pLAB. pLAB is software engineering laboratory located in the Lapland University of Applied Sciences.

The objective of this thesis was to create a Unity 3D software to control the Moog motion platform. In addition, a virtual reality headset, called Oculus Rift, was attached to this thesis to increase the immersion in virtual reality. The end result also had to be easy to integrate with the new Unity 3D projects. One of the common problems was how to get the Unity 3D game engine communicate with the motion platform via a UDP connection and how to get the motion platform moving smoothly.

The programming language used was C-sharp and the software was accomplished with the Unity 3D game engine. The software sends data packets to the motion platform via a UDP-connection.

The result of this thesis was software which moves the motion platform. The movement is related to the position of the Unity 3D game object in the virtual environment, without the exceptions of sudden movements, so that the movement of the motion platform is smooth. A person can observe the virtual reality with the Oculus Rift headset. The user is also able to drive a car with a steering wheel and pedals. The implementation of the ambulance simulator is started based on the software. The end result was also implemented the way that it is easy to attach to the new Unity 3D project and game objects.

Key words Unity 3D, Motion Platform, Driving Simulation, Oculus Rift

SISÄLTÖ

KUVIOLUETTELO 1

ESIMERKKIKOODI -LUETTELO 3

KÄSITELUETTELO 4

1 JOHDANTO 5

2 TAUSTA 6

3 KÄYTETYT TYÖVÄLINEET JA TEKNOLOGIAT 8

3.1 UNITY 3D -PELIMOOTTORI 8

3.1.1 Yleistä 8

3.1.2 Lisenssit 11

3.1.3 Editori 13

3.1.4 Ohjelmointiympäristö 14

3.2 MOOG-LIIKEALUSTA 15

3.2.1 Liikealustan esittely 15

3.2.2 Vapausasteet ja raja-arvot 17

3.2.3 Liikealustan tilat 19

3.2.4 Datan lähettäminen liikealustalle 21

3.2.5 Datan vastaanotto liikealustalta 23

3.3 UDP-PROTOKOLLA 25

3.4 AJO-OHJAIN JA ISTUIN 27

3.5 OCULUS RIFT -VIRTUAALILASIT 28

4 SOVELLUKSEN SUUNNITTELU 30

5 SOVELLUKSEN TOTEUTUS 35

5.1 SOVELLUKSEN TOIMINTAPERIAATE 35

5.2 UDP-YHTEYS 39

5.3 TASAISEN LIIKKEEN TUOTTAMINEN 41

5.4 LIIKEALUSTAN LIIKE PELIOBJEKTISTA 43

5.5 RATIN JA POLKIMIEN LISÄYS 46

5.6 OCULUS RIFT:N LIITTÄMINEN 48

5.7 KÄYTTÖKOKEMUKSET 48

5.8 LIIKEALUSTAN DATAN VASTAANOTTO 49

6 LIIKEALUSTASOVELLUKSEN KÄYTTÄMINEN 53

6.1 SOVELLUKSEN HALLINTA 53

6.2 TOTEUTUKSEN INTEGROINTI 53

7 YHTEENVETO 57

LÄHTEET 59

LIITTEET 62

KUVIOLUETTELO

KUVIO 1. KESKIPAKOVOIMAN VAIKUTUS AUTON KÄÄNTYESSÄ (MUKAILLEN HYPERPHYSICS 2014) ... 6
KUVIO 2. KUUTION MUOTOINEN PELIOBJEKTI UNITY 3D –PELIMAAILMASSA ... 9
KUVIO 3. UNITY 3D -PELIMOOTTORIN KÄYTTÖLIITTYMÄ ... 13
KUVIO 4. MONODEVELOP-OHJELMOINTIYMPÄRISTÖ ... 14
KUVIO 5. MOOG 6DOF2000E-LIIKEALUSTA PLAB:N TILOISSA ... 15
KUVIO 6. MOOG 6DOF2000E MOTION SYSTEM INTERFACES (MUKAILLEN MOOG SYSTEMS GROUP SERIES 6DOF2000E -ESITE) ... 16
KUVIO 7. MOOG 6DOF2000E MOTION SYSTEM TIETOKONEEN STATUSKUVA ... 17
KUVIO 8. MOOG 6DOF2000E MOTION SYSTEM (MUKAILLEN MOOG MOTION SYSTEM DIVISION 2000B, 1.2) ... 18
KUVIO 9. MOOG-LIIKEALUSTAN TILAKAAVIO (MUKAILLEN MOOG MOTION SYSTEM DIVISION 2000A, 4.3) ... 20
KUVIO 10. MDA-TILAN MCW:N KOMENNOT JA NIIDEN 16-BITTISET ARVOT (MOOG MOTION SYSTEM DIVISION 2000A, 3.8) ... 21
KUVIO 11. BIG ENDIAN- JA LITTLE ENDIAN -TAVUJÄRJESTYKSIEN MUUNNOS (MUKAILLEN WIKIPEDIA 2014) ... 23
KUVIO 12. OTE LENGTH-VASTAUSPAKETIN SISÄLLÖSTÄ (MOOG MOTION SYSTEM DIVISION 2000A, 3.14) ... 25
KUVIO 13. UDP-PAKETIN PERUSMUOTO (MUKAILLEN H3C 2014) ... 26
KUVIO 14. KUVANKAAPPAUS LÄHETETYSTÄ MDA-DATAPAKETISTA WIRESHARK OHJELMAN KAUTTA ... 26
KUVIO 15. PLAYSEAT EVOLUTION-ISTUIN JA LOGITECH G27 LIIKEALUSTALLA ... 27
KUVIO 16. LOGITECH G27 LIIKEALUSTALLA ... 28
KUVIO 17. TYÖSSÄ KÄYTETYT OCULUS RIFT -VIRTUAALILASIT ... 28
KUVIO 18. OCULUS RIFT -LASIEN TUOTTAMA KUVA ... 29
KUVIO 19. TYÖVAIHEET JA NIIDEN VÄLINEN SIIRTYMÄ ... 31
KUVIO 20. HAHMOTELMA SOVELLUKSEN YKSINKERTAISTETUSTA TOIMINNASTA ... 32
KUVIO 21. KAAVIO ASENTO- JA LIIKETIETOJEN PÄIVITTÄMISESTÄ LIIKEALUSTALLE ... 33
KUVIO 22. SMOOTHDAMP() -FUNKTION TOIMINTA (MUKAILLEN SCRATCHAPIXEL 2014) ... 33
KUVIO 23. VUOKAAVIO SOVELLUKSEN TILOJEN VÄLISESTÄ SIIRTYMISESTÄ ... 36
KUVIO 24. KAAVIO LIIKEDATAN PÄIVITYSREITISTÄ LIIKEALUSTALLE NEWMDA-KOMENNOLLA ... 37
KUVIO 25. VUOKAAVIO LIIKEDATAN PÄIVITTÄMISESTÄ LIIKEALUSTALLE YHDEN KEHYKSEN AIKANA ... 38
KUVIO 26. SEKVENSSIKAAVIO KOMPONENTTIEN VÄLISESTÄ KOMMUNIKOINNISTA KOMENTOSANAN OLLESSA NEWMDA ... 39
KUVIO 27. KUVANKAAPPAUS INPUT MANAGER -ASETUKSISTA ... 47
KUVIO 28. KUVANKAAPPAUS OCULUS RIFT -KAMERAN NÄKYMÄSTÄ UNITY 3D -PELIMOOTTORILLA ... 48
KUVIO 29. ESIMERKKI LIIKEALUSTAN TILATIETOJEN BITTIMASKAUKSESTA AND-OPERAATTORILLA ... 52
KUVIO 30. KUVANKAAPPAUS TOTEUTUKSEN LIITTÄMISESTÄ UUTEEN PROJEKTIIN 54

KUVIO 31. KUVANKAAPPAUS UNITYPACKAGE-TIEDOSTON LUOMISESTA 54
KUVIO 32. KUVANKAAPPAUS INTEGROINTIVAIHEESTA 55
KUVIO 33. KUVANKAAPPAUS INPUT MANAGER -ASETUKSISTA INTEGROINTIVAIHEESSA 56
KUVIO 34. KUVANKAAPPAUS LIIKKEEN NOPEUDEN JULKISISTA MUUTTUJISTA..... 56

ESIMERKKIKOODI -LUETTELO

ESIMERKKIKOODI 1. TOTEUTETTU SENDDATA() -FUNKTIO..... 40
ESIMERKKIKOODI 2. OTE MOTIONPLATFORM -SKRIPTISTÄ..... 41
ESIMERKKIKOODI 3. OTE SMOOTHDAMP() -FUNKTION KÄYTÖSTÄ..... 42
ESIMERKKIKOODI 4. OTE CARVALUEHANDLER-SKRIPTIN ROLL-ARVON LUKEMISESTA LIIKEALUSTALLE..... 43
ESIMERKKIKOODI 5. OTE CARVALUEHANDLER-SKRIPTIN SIVUTTAISEN LIIKKEEN TUOTTAMISESTA LIIKEALUSTALLE 44
ESIMERKKIKOODI 6. OTE PELIOBJEKTIN KIIHTYVYYDEN JA NOPEUDEN VAIKUTUKSESTA LIIKEALUSTAAN 45
ESIMERKKIKOODI 7. OTE PINGPONG() -FUNKTION KÄYTÖSTÄ 46
ESIMERKKIKOODI 8. OTE PELIOHJAIMEN HANDBRAKE-NÄPPÄINKOMENNOSTA 47
ESIMERKKIKOODI 9. OTE UDP-VASTAANOTON TOTEUTTAMISESTA JA TIEDON LUKEMISESTA BITTITAULUKKOON 50
ESIMERKKIKOODI 10. OTE CHECKSTATE() -FUNKTIOSTA RESPONSE-SKRIPTISSÄ. 51

KÄSITELUETTELO

Asset	Resurssitiedostot Unity 3D -projektissa.
Asynkroninen kommunikointi	Kommunikointi, joka ei tapahdu reaaliajassa, vaan viiveitä esiintyy.
Bittimaskaus	Verrataan bittijonojen yksittäisiä bittejä keskenään. Käytetään muun muassa arvojen manipuloimisen, laskutoimituksien ja vertailujen yhteydessä.
Big Endian	Bittijärjestys, jossa eniten merkitsevä bitti ilmoitetaan ensimmäisenä.
DOF	Degree of freedom eli vapausaste.
ENVI	Lapin AMK:n virtuaalinen oppimisympäristö terveydenalan opiskelijoille.
FPS	Frames per second eli kuvataajuus.
Immersio	Läsnäolon tunne virtuaalitodellisuudessa (McMahan, A 2003).
Little Endian	Bittijärjestys, jossa vähiten merkitsevä bitti ilmoitetaan ensimmäisenä.
MDA	Motion Drive Algorithm. Liikealustan ohjaustila, joka sisältää sisäänrakennetut algoritmit liikealusta ohjaamiseen.
MBC	Motion Base Computer. Liikealustassa oleva tietokone, joka sisältää rajapinnan ohjaamiseen.
MCW	Motion Command Word. Liikealustan komentosana.
Simulaatio	Todellisuuden jäljentäminen (University of Central Florida 2013).
Simulaattori	Laitekokonaisuus, jonka avulla simulaatio tuotetaan. (University of Central Florida 2013).
TCP	Yhteydellinen protokolla tiedonsiirtoon Ethernet-verkossa.
UDP	Yhteydetön protokolla tiedonsiirtoon Ethernet-verkossa.

1 JOHDANTO

Virtuaaliset oppimisympäristöt ja erilaiset simulaattorit ovat nykypäivänä kehittyneet yhä hyödyllisemmiksi opetusvälineiksi. Tämän opinnäytetyön aiheena on ajosimulaation toteutus Unity 3D -pelimoottorilla. Aihe syntyi, kun ENVI-oppimisympäristöön oli tarve saada Moog-liikealustalla toimiva ambulanssisimulaattori oppimistilanteita varten. ENVI on terveydenalan opiskelijoille toteutettu virtuaalinen oppimisympäristö, jonka avulla opiskelijat voivat harjoitella hoitotilanteita vuorovaikutteisesti.

Opinnäytetyön tavoitteena oli saada liikealustan ohjaus toimimaan Unity 3D -pelimoottorilla UDP-yhteyden kautta. Lisäksi tavoitteena oli liikealustan liikkeen saaminen mahdollisimman sulavaksi. Liikealustaa ohjataan Unity 3D -peliobjektin asentojen ja liikkeiden mukaisesti. Todentuntuisuuden lisäämiseksi työhön sisällytettiin myös Oculus Rift -virtuaalilasien yhteiskäyttö liikealustan kanssa. Lopputuloksen tuli olla helposti integroitavissa uusiin Unity 3D käyttötarkoituksiin. Ajamista varten työssä valittiin myös sopiva ajopelituoli ja realistisen ajokokemuksen mahdollistava ajo-ohjain.

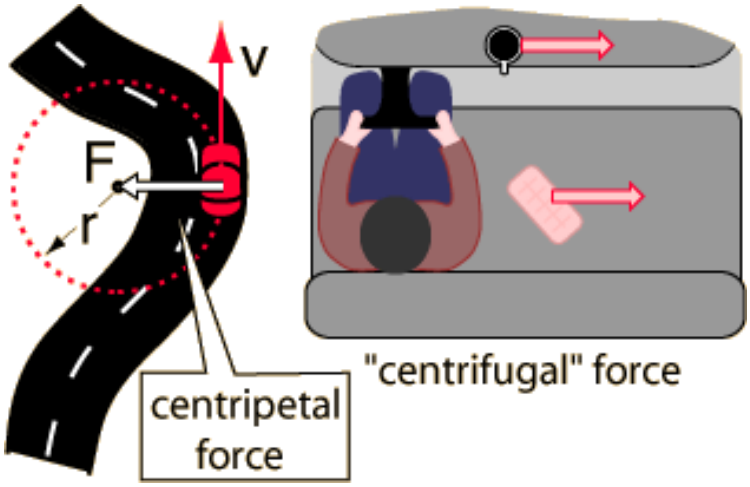
Opinnäytetyön luvussa 2 esitellään aiheen taustaa ja tarkoitusta. Luvussa 3 kerrotaan käytetyt työvälineet ja teknologiat. Luku 4 käsittelee työn aloitusta ja sovelluksen eri työvaiheita sekä luku 5 kertoo, millä tavalla sovellus toteutettiin. Kuudennessa luvussa kuvataan, miten toteutettua ajosimulaatiosovellusta käytetään ja miten se voidaan helposti integroida toiseen Unity 3D -projektiin. Lopuksi yhteenvedossa analysoidaan lopputulosta ja kerrotaan jatkokehitysideoita toteutukseen liittyen.

Opinnäytetyön toimeksiantaja on Lapin Ammattikorkeakoulun ohjelmistotekniikan laboratorio pLAB, joka on erikoistunut erityisesti ohjelmisto- ja mittausteknologioihin sekä reaaliaikaisiin 3D-visualisointiympäristöjen toteutuksiin, mutta toteuttaa myös sovelluksia aina www-sivuista mobiili- ja työpöytäsovelluksiin (pLAB 2013). Opinnäytetyön aiheen valitsin, koska olen kiinnostunut Unity 3D -pelimoottorista ja sen eri soveltamismahdollisuuksista. Opinnäytetyön valitseminen pLAB:sta tuntui myös luonnollisimmalta, koska olen opiskelujen ohessa työskennellyt siellä.

2 TAUSTA

Virtuaalitodellisuuden laitteet ja tekniikat ovat nykypäivänä kehittyneet yhä todentuntuisimmiksi, mikä mahdollistaa erilaisten oppimistilanteiden toteuttamisen yhä realistisempien simulaatioiden avulla. Toteutettavan ajosimulaation tehtävänä on tuottaa käyttäjälle todentuntuinen ajokokemus liikealustan liikkeiden, 3D-virtuaalilasien, ratti-poljin-yhdistelmän ja pelimaailman äänten avulla. Todentuntuisesta virtuaalitodellisuuden tuntumasta käytetään nimitystä immersio. Immersio voidaan ymmärtää niin, että sillä tarkoitetaan voimakasta läsnäolon tunnetta virtuaalimaailmassa (McMahan, A 2003).

Liikealustan liikkeiden todentuntuisuus perustuu suurelta osin auton liikkeiden vastavoimiin. Esimerkiksi kiihdyttäessä kuljettajaan vaikuttava voima työntää kuljettajaa taaksepäin, kun taas jarruttaessa eteenpäin. Sama vastavoimaisuus pätee myös kääntymisen liikkeiden kuvaamiseen. Tämä nähdään Kuviossa 1. Auton kääntyessä mutkassa vasemmalle sisällä olevat tavarat työntyvät oikealle.



Kuvio 1. Keskipakovoiman vaikutus auton kääntyessä (Mukaillen HyperPhysics 2014)

Terveydenalan opiskelijoille suunnattuun ENVI-oppimisympäristöön haluttiin toteuttaa ambulanssisimulaattori. Oppimisympäristön on kehittänyt Lapin ammattikorkeakoulun ohjelmistotekniikan laboratorio pLAB. Ensimmäinen versio ENVI:stä on valmistunut vuonna 2007. Oppimisympäristössä on toteutettuna erilaisia hätä- ja kriisitilanteita, joissa opiskelijat pystyvät harjoittele-

maan hoitotilanteita vuorovaikutteisesti (RAMK 2013). Toteutetuissa hätä- ja kriisitilanteissa opiskelijat pystyvät harjoittelemaan tilanteissa toimimista, kuten tajuttomana makaavan henkilön auttamista. Virtuaalisesta oppimisympäristöstä on kehkeytynyt hyödyllinen ja käytetty opetusmenetelmä.

Ambulanssisimulaattoriin haluttiin yhdistää Oculus Rift -virtuaalilasit, sillä ne tuovat lisää läsnäolon tunnetta virtuaaliympäristöön. Ambulanssin ajamisessa tärkeintä on liikkeen tasaisuus, jotta hoitotyö on mahdollista. Siksi liikealustan liikkeen tasaisuuteen tuli kiinnittää paljon huomiota sovellusta tehdessä. Tasaisuudella tarkoitettiin sitä, ettei liikealustan liike ole nytkähtelevää. Sen sijaan realistisen ajokokemuksen tuottamisen kannalta liikealustan oli reagoitava tien pintaan ja kuljettajan ajamiseen reaalimaailman tavoin. Esimerkiksi liikealustan tuli reagoida todentuntuisesti tien pinnan kuoppiin, mutta myös kuljettajan äkkinäisiin kiihdytyksiin ja jarrutuksiin. Simulaattorin avulla opiskelijat pystyvät monipuolisesti harjoittelemaan toimimista ambulanssissa eri harjoitteluympäristöjen avulla. Harjoittelutilanteissa voi esimerkiksi tien pinta vaihdella hiekasta asfalttiin tai keli tehdä tien pinnasta liukkaan. Ohjaaja voi myös kesken ajon asettaa tielle erilaista liikennettä tai estettä, kuten ajoneuvoja ja eläimiä.

ENVI tullaan viemään kokonaisuudessaan uudelle pelimoottorille, Unity 3D:lle. Ambulanssisimulaattori on osa ENVI:n uutta versiota. Uusi versio sisältää paljon uusia ominaisuuksia, kuten ensihoitotilanteita, joissa opiskelijat pystyvät samanaikaisesti liikkumaan virtuaalimaailmassa tablettitietokoneiden avulla.

3 KÄYTETYT TYÖVÄLINEET JA TEKNOLOGIAT

3.1 Unity 3D -pelimoottori

3.1.1 Yleistä

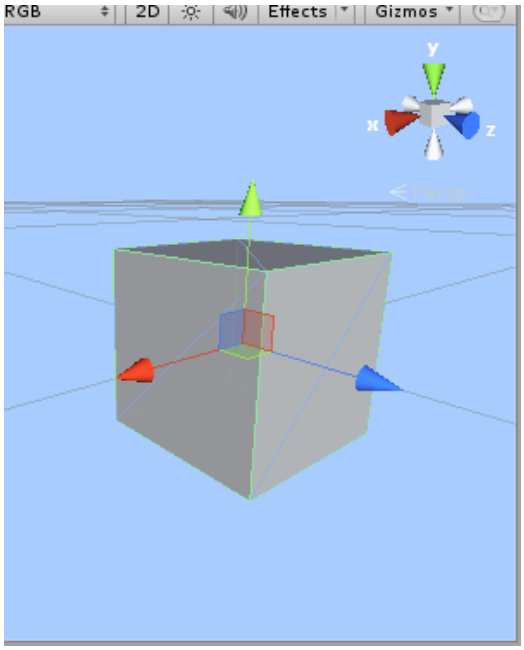
Unity 3D on alustariippumaton pelimoottori, jonka on kehittänyt Unity Technologies. Pelien lisäksi Unity 3D:llä voidaan tehdä muun muassa erilaisia visualisointeja sekä tuottaa interaktiivista kaksi- tai kolmiulotteista sisältöä esimerkiksi nettisivuille. Unity 3D on kasvanut yhdeksi suosituimmista pelimoottoreista. Unity 3D:llä on kirjoitushetkellä 2/2014 400 tuhatta aktiivista käyttäjää kuukausittain ja melkein 2 miljoonaa rekisteröitynyttä kehittäjää. Unityn tunnetuimpia asiakkaita ovat muun muassa Coca-Cola, NASA, LEGO, Disney, Microsoft sekä Warner Bros. (Unity 3D 2013a.)

Pelimoottorin uusin versio 4.3 julkaistiin marraskuussa 2013. Version uutuutena ovat muun muassa erilliset 2D-pelien kehitystyökalut. Windows-käyttöjärjestelmillä on mahdollista käyttää grafiikkojen luomiseen DirectX11-ohjelmointirajapintaa. Muilla käyttöjärjestelmillä Unity 3D:n renderöinti tapahtuu OpenGL-rajapinnan kautta. (Unity 3D 2013i.)

Unity 3D -pelimoottori sisältää NVIDIA PhysX -3D-fysiikkamoottorin. Fysiikkamoottori sisältää fysiikkamallinnuksen, joka mahdollistaa 3D- ja 2D-pelien tekemisen. Fysiikkamallinnusta tuottavista komponenteista mainittakoon muun muassa Rigidbody-, Joint-, Ragdoll-, Soft Bodies - ja Cloth-mallinnukset. Rigidbodyn avulla peliobjektiin saadaan liitettyä esimerkiksi painovoiman vaikutus, massa tai liikevoima. Soft Bodies -mallinnuksen avulla voidaan realistisesti mallintaa esimerkiksi pehmeiden objektien liikettä ympäristössä. Joint-komponenttien avulla pystytään liittämään objekteja toisiinsa ja määrittämään niille esimerkiksi vaadittava törmäysvoima, joka irrottaa ne. Joint-komponteista esimerkkeinä voisivat olla objektit, kuten jouset ja nivelet. Cloth-komponentit ovat tarkoitettu kangasmaisten objektien fysiikan kuvaamiseen. Cloth-komponettia voi esimerkiksi käyttää pelihahmon vaatteisiin. Ragdoll puolestaan tarkoittaa räsynukkemallinnusta. (Unity 3D 2013h.)

Unity 3D:ssä jokaisella peliobjektilla on Transform-komponentti, joka sisältää vektoritiedot skaalauksesta, peliobjektin asennosta ja sijainnista pelimaail-

massa. Kuviossa 2 nähdään kuution muotoinen peliobjekti ja sen Transform-komponentin 3D-vektori. Punainen nuoli osoittaa peliobjektin x-akselin suunnan, vihreä nuoli y-akselin suunnan ja sininen nuoli z-akselin suunnan.



Kuvio 2. Kuution muotoinen peliobjekti Unity 3D –pelimaailmassa

Unity 3D:ssä on peliobjekteihin mahdollista lisätä eri collider-komponentteja, jotka ovat objektikohtaisia törmäystunnistimia. Collider-tyypit on kuvattu tarkemmin taulukossa 1. Collider-komponenteista Mesh Collider omaa kaikista tarkimman törmäystunnistuksen, sillä sen muoto on täsmälleen sama kuin objektinkin, mutta tämän vuoksi se myöskin tekee eniten laskutoimituksia. Wheel Collider on pääasiassa suunniteltu eri ajoneuvojen renkaiden fysiikan mallinnusta varten.

Taulukko 1. Unity 3D Collider-tyypit

Collider	**Muoto**
Box Collider	Kuutio
Capsule Collider	Kapseli
Sphere Collider	Ellipsi
Mesh Collider	Objekti
Wheel Collider	Rengas

Peliobjekteille on mahdollista lisätä Physic Material -komponentti, jonka avulla voidaan säätää objektien kitkaa ja kimmoisuutta törmäyksissä (Unity 3D 2014e). Fysiikkakomponenttina mainittakoon vielä Constant Force -komponentti, jonka avulla objektiin saadaan lisättyä liikevoimaa. Esimerkkinä tästä voisi olla vaikkapa raketti, jonka nopeus halutaan kiihdyttää tiettyyn pisteeseen ampumisen jälkeen (Unity 3D 2013b).

Mikäli peliobjekti halutaan saada näkyväksi, sille pitää lisätä Mesh Filter -komponentti. Mesh Filter on komponentti, joka tallentaa peliobjektin muodon tiedot. Lisäksi peliobjektille tulee luoda Mesh Renderer, joka piirtää valitun materiaalin peliobjektille Mesh Filterin muotojen mukaisesti. (Unity 3D 2014c.)

Unity 3D:ssä peliobjekteista on mahdollista tehdä prefab-tiedostoja, jotka ovat esikoostettuja peliobjekteja. Prefab-tiedostojen etuna on se, että peliobjektista voi luoda useita instansseja ajon aikana. Prefab-tiedosto sisältää tiedot kaikista peliobjektiin liitetyistä ominaisuuksista ja komponenteista, joten niitä ei tarvitse erikseen lisätä.

Scene on kenttäkohtainen pelimaailma. Kaikkien pelimaailmassa olevien peliobjektien tiedot tallentuvat Sceneen. Scene on helppo tapa jaotella peliä osiin, kuten esimerkiksi kenttiin ja tasoihin, joista siirtyminen uuteen kenttään tapahtuu lataamalla kyseessä oleva Scene. (Unity 2014a.)

Unity 3D:llä on mahdollista kehittää pelejä ja sovelluksia mobiililaitteisiin, tietokoneille työpöytäsovelluksina, nettiselaimille ja pelikonsoleille. Tietokoneiden käyttöjärjestelmistä Unity 3D:llä on mahdollista tehdä sovelluksia Windows-, Linux- ja Mac-käyttöjärjestelmille. Mobiilikäyttöjärjestelmistä on tuettu iOS, Android, Windows Phone 8 ja Blackberry. Pelikonsoleista tuettuna ovat Playstation 3, Xbox 360 ja Wii U. Pelien kehittäminen pelikonsoleille on mahdollista kuitenkin vain yrityksille, jotka ovat pelialustojen haltijoiden virallisesti hyväksymiä kehittäjiä. (Unity 3D 2013e.) Unity 3D:llä on siis mahdollista kehittää sovelluksia käytännössä kaikille suosituimmille alustoille käyttäen pohjana vain yhtä projektia, joka tekee projektien kääntämisestä eri alustoille nopeaa ja helpohkoa.

3.1.2 Lisenssit

Unity 3D:stä on saatavilla maksuton versio Unity Free ja maksullinen Unity Pro -versio, jossa on lisäominaisuuksia. Ilmaisversiolla on mahdollista kehittää pelejä jokaiselle tuetulle alustalle lukuun ottamatta pelikonsoleita. Lisäksi Unity Pro -versioon voidaan hankkia BlackBerry-, Andoid- ja iOS-alustojen Pro-lisenssejä. Muiden tuettujen alustojen Pro-lisenssit lukeutuvat Unity Pro -versioon lukuun ottamatta konsolialustoja (Unity 3D 2013f). Pro-lisenssien lisäominaisuudet verrattuna maksuttomaan versioon näkyvät taulukossa 2.

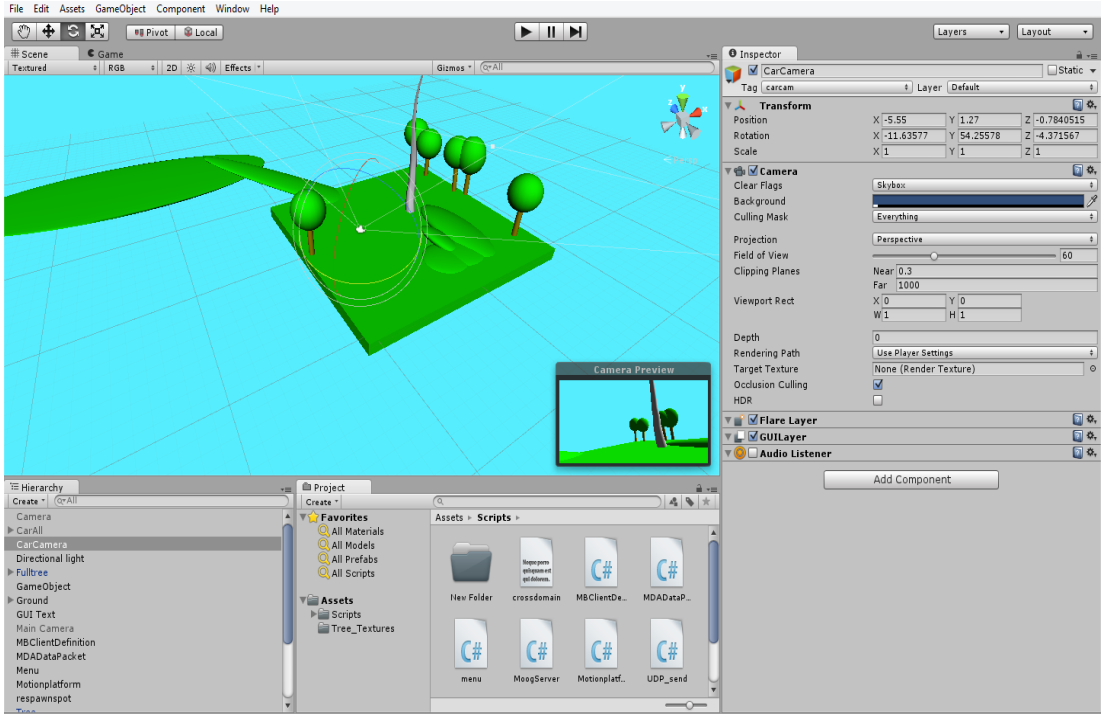
Taulukko 2. Unity Free- ja Unity Pro -lisenssien ominaisuuksien vertailutaulukko (Unity 3D 2013d)

General	**Unity Free**	**Unity Pro**
Physics	x	x
NavMeshes, path-finding, and crowd Simulation	x	x
Multiplayer Networking with RakNet	x	x
LOD support		x
Audio (3D Positional and Classic Stereo)	x	x
Audio Filter		x
Video Playback and Streaming		x
Fully Fledged Streaming with Asset Bundles		x
May be licensed and used by companies or incorporated entities that had a turnover in excess of US$100,000 in their last fiscal year.		x
Animation	**Unity Free**	**Unity Pro**
Mecanim	x	x
Mecanim: IK Rigs		x
Mecanim: Sync Layers & Additional Curves		x
Deployment	**Unity Free**	**Unity Pro**
One-Click Deployment	x	x
Web Browser Integration	x	x
Custom Splash Screen		x
Graphics	**Unity Free**	**Unity Pro**
Low-Level Rendering Access	x	x

Dynamic Fonts with markup	x	x
Shuriken Particle System	x	x
3D Texture Support		x
Realtime Directional Shadows	x	x
Realtime Spot/Point and soft shadows		x
HDR, tone mapping		x
Light Probes		x
Optimized Graphics	x	x
Shaders (Built-in and Custom)	x	x
Lightmapping	x	x
Lightmapping with Global Illumination and area lights		x
Dynamic Batching	x	x
Static Batching		x
Terrains (Vast, Densely Foliaged Landscapes)	x	x
Render-to-Texture Effects		x
Full-Screen Post-Processing Effects		x
Occlusion Culling		x
Deferred Rendering		x
Stencil Buffer Access		x
GPU Skinning		x
Code	**Unity Free**	**Unity Pro**
Navmesh: Dynamic Obstacles and Priority		x
Webplayer debugging	x	x
.NET Based Scripting With C#, JavaScript, and Boo	x	x
Access to Web Data through WWW Functions	x	x
Open an URL in the User's Browser	x	x
.NET Socket Support	x	x
Native Code Plugins Support		x
Inspector GUI for custom classes	x	x
Editor	**Unity Free**	**Unity Pro**
Integrated Editor	x	x
Instantaneous, Automatic Asset Importing	x	x
Integrated Animation Editor	x	x
Profiler and GPU profiling		x
External Version Control Support	x	x
Script Access to Asset Pipeline		x
Dark Skin		x

3.1.3 Editori

Unity 3D:n editorin näkymän voi jakaa viiteen eri osaan: Hierachy, Project Browser, Inspector Scene ja Game. Hierarchy sisältää luettelon kaikista Scenen sisäisistä peliobjekteista. Project Browser sisältää projektin assets-tiedostot eli projektissa käytettävät tiedostot. Inspector näyttää aktiiviseksi valitun peliobjektin kaikki ominaisuudet. (Unity 3D 2013c.) Inspectorin kautta pystytään muun muassa suoraan asettamaan komponenttien julkisille muuttujille arvoja ilman skripteihin koskemista. Scene-osiossa näkyy virtuaaliympäristö. Game-osiossa on esillä aktiivisen kameran mukainen maailma eli pelinäkymä.

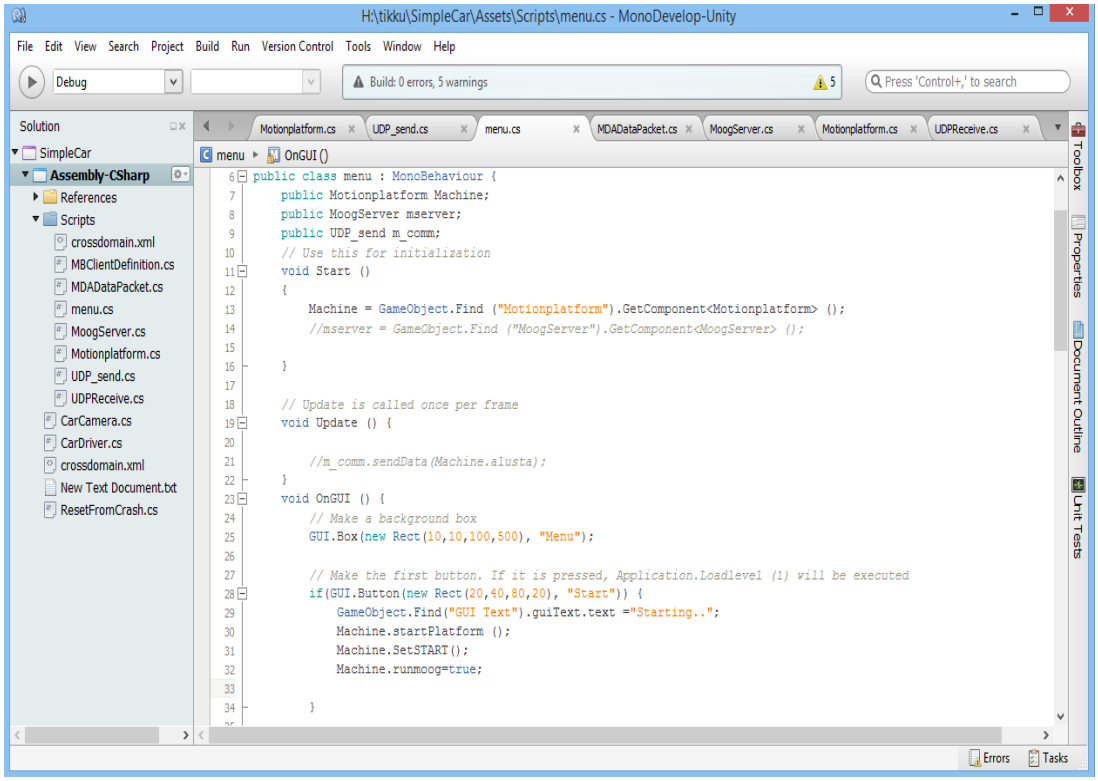


Kuvio 3. Unity 3D -pelimoottorin käyttöliittymä

Kuviossa 3 esitetään yksi tapa osioiden jakamiseen näytöllä. Pelimoottori antaa mahdollisuuden muuttaa osioiden näkyvyyttä editorissa. Editoriin on suunniteltu Drag and Drop -toiminto, jonka avulla esimerkiksi skriptin lisääminen peliobjektille tai peliobjektin siirtäminen Scene-osion virtuaalimaailmaan tapahtuu helppokäyttöisesti.

3.1.4 Ohjelmointiympäristö

Unity 3D tukee C#-, Javascript- ja Boo-ohjelmointikieliä. Pelimoottori toimii avoimen lähdekoodin .NET-kehitysalustalla nimeltä Mono. (Unity 3D 2013g.) Unity 3D:n asennus sisältää MonoDevelop-editorin skriptien kirjoittamiseen, mutta mahdollistaa myös muiden editorien käyttämiseen (Kuvio 4).



Kuvio 4. MonoDevelop-ohjelmointiympäristö

MonoBehaviour-luokka on pääluokka, josta jokainen skripti periytyy. Skriptin täytyy periytyä MonoBehaviour-luokasta, jotta skriptin voidaan liittää osaksi jonkin peliobjektin toimintaa. (Unity 3D 2014d.) Opinnäytetyössä on käytetty MonoBehaviour-luokan funktioista muun muassa *Update()*-, *Start()*-, *OnGUI()*- ja *FixedUpdate()* -funktioita. *Start()*-funktio suoritetaan heti pelin käynnistyessä, siksi siihen on hyvä laittaa esimerkiksi tarvittavat alustukset. *Update()*-funktio suoritetaan kerran jokaisen kehyksen aikana. *FixedUpdate()*-funktion suoritustiheyttä voi hallita Unity 3D:n editorin kautta Time Manager -asetuksista Fixed Timestep -kohdasta. *OnGUI()* -funktio puolestaan on tarkoitettu käsittelemään käyttöliittymän tapahtumia.

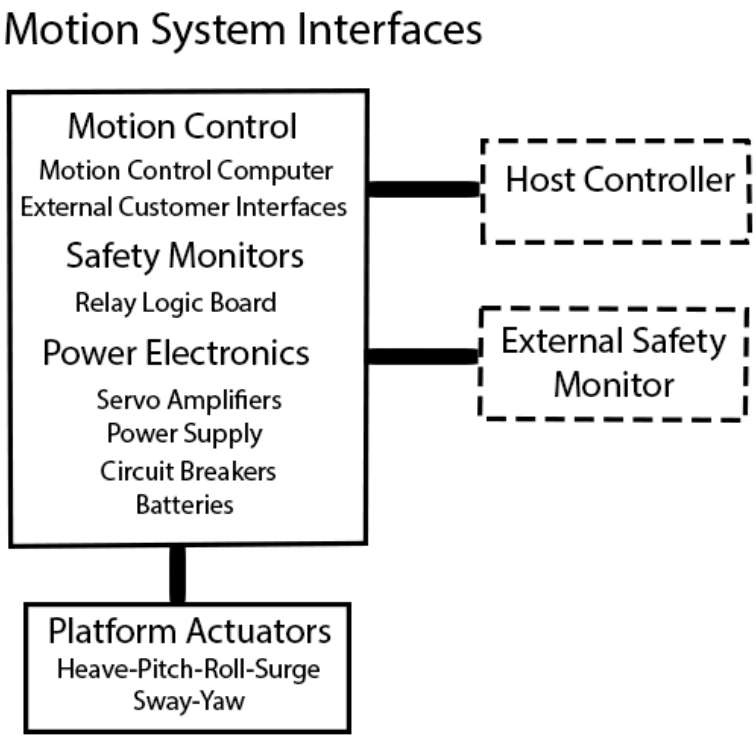
3.2 Moog-liikealusta

3.2.1 Liikealustan esittely

Moog 6DOF200E -liikealusta on Moog Inc. -yrityksen toteuttama liikealusta. Moog Inc. on erikoistunut erilaisten liikeohjaustuotteiden ja -järjestelmien toteutuksiin. (Moog 2014.) Liikealusta on sähköisesti toimiva kuuden vapausasteen liikkeen mahdollistava liikealusta (Kuvio 5).

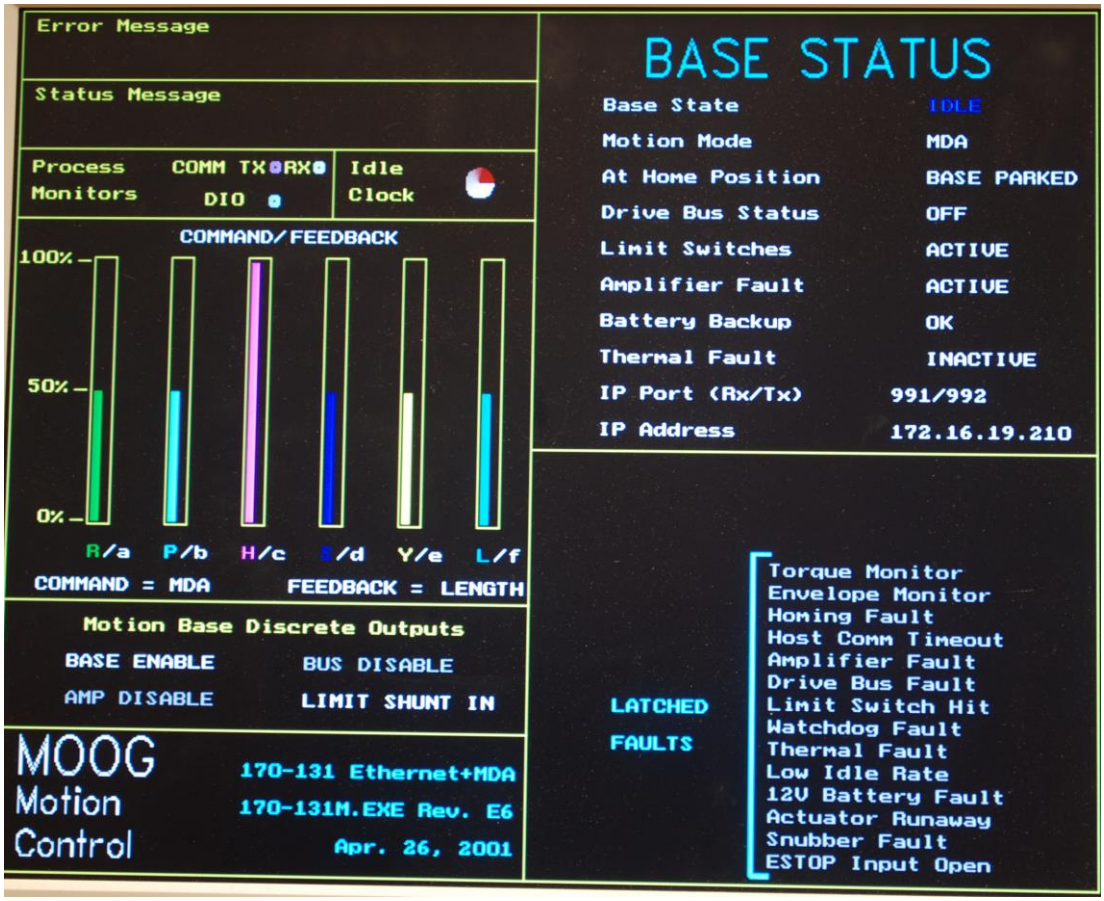
Kuvio 5. Moog 6DOF2000E-Liikealusta pLAB:n tiloissa

Liikealustalle voidaan asentaa enintään 1000 kg:n painoinen massa, johon tuotetaan liikettä kuuden servomoottoroidun sylinterin avulla (Piiroinen 2008, 13). Liikealusta sisältää muun muassa oman sähköjärjestelmän, servojen hallinnan, turvamonitorit, ylikuormituksen suojauksen ja kommunikointirajapinnan asiakassovellusta varten (Moog Motion System Division 2000b, 1.1). Kuvio 6 sisältää tarkemman kuvauksen edellä mainitusta.



Kuvio 6. Moog 6DOF2000E Motion System Interfaces (Mukaillen Moog Systems Group Series 6DOF2000E -Esite)

Liikealustassa on oma tietokone, joka sisältää ohjelmointirajapinnan kommunikointiin ethernet-verkon avulla. Liikealusta mahdollistaa sen tilojen välisen ohjaamisen UDP-yhteydellä. Liikealustan ja asiakassovelluksen kommunikointi tapahtuu master-slave -muodossa. Liikealustan tietokone tunnistaa yhteydenoton ainoastaan, kun asiakassovellus lähettää sille validin datapaketin. Pakettien lähettäminen ja vastaanotto tapahtuu asynkronisesti, jolloin kommunikointi ei ole reaaliaikaista. Tämän vuoksi lähettämisessä ja vastaanottamisessa voi esiintyä viiveitä. (Moog Motion System Division 2000a, 3.1.) Kuviossa 7 nähdään liikealustalta näytölle tuleva statuskuva. Sen avulla nähdään muun muassa liikealustan tilatiedot, mahdolliset virheet ja osoitetiedot.



Kuvio 7. Moog 6DOF2000E Motion System tietokoneen statuskuva

3.2.2 Vapausasteet ja raja-arvot

Liikealustan vapausasteet ovat kuvattu nimillä: Pitch eli kaltevuus, Roll eli kierto, Yaw eli kääntyminen, Heave eli nosto, Surge eli syöksy ja Lateral eli sivuttainen liike (Moog Motion System Division 2000b, 1.2). Vapausasteet on kuvattu kuviossa 8. Kuuden vapausasteen liikealusta mahdollistaa kuorman liikuttamisen x-, y- ja z-akselien mukaisesti.

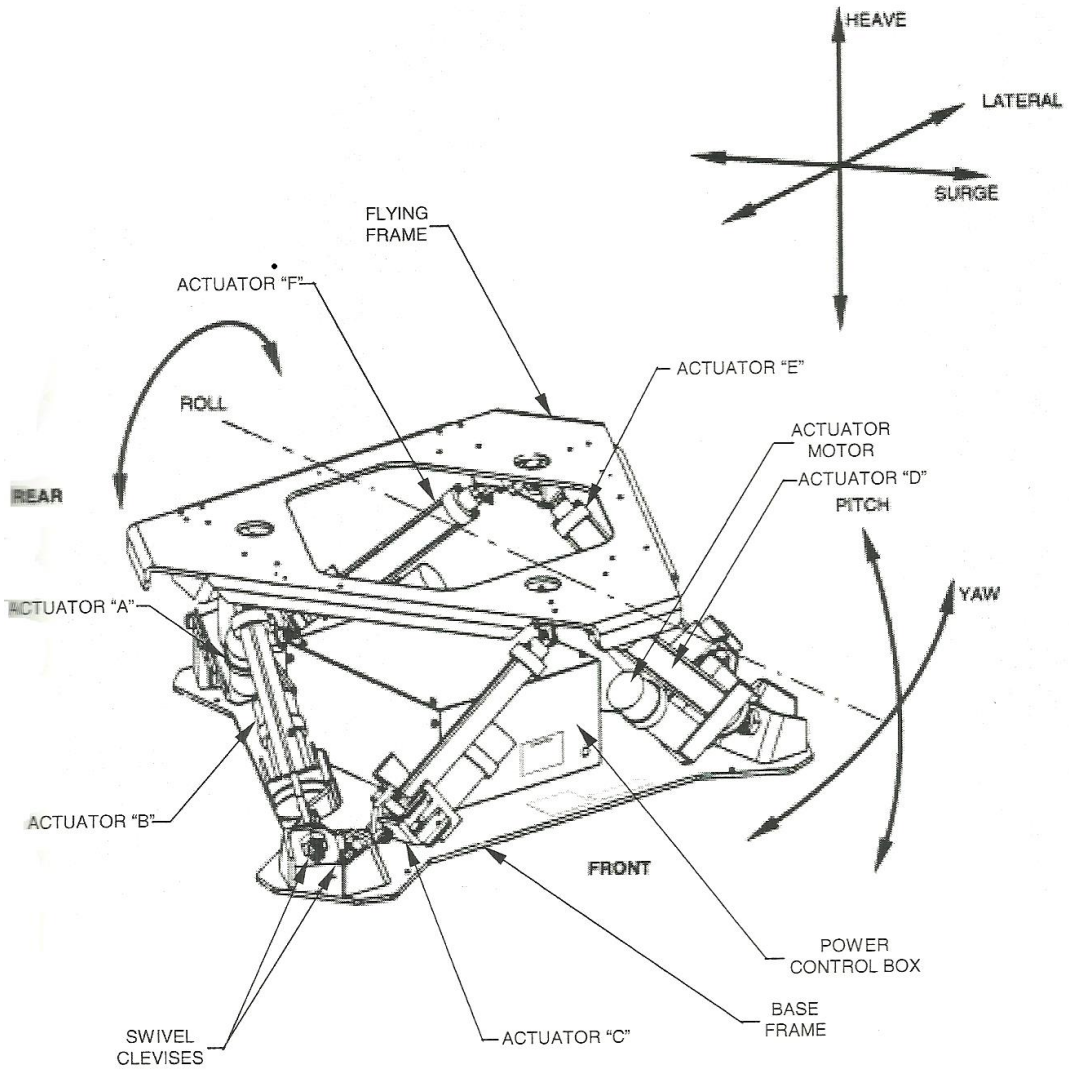


Figure 1-1
Moog 6DOF2000E Motion System

Kuvio 8. Moog 6DOF2000E Motion System (mukaillen Moog Motion System Division 2000b, 1.2)

Pitch-, Roll- ja Yaw-arvot kertovat kulman suuruuden ja niiden yksikkö on radiaani. Pitch-arvoja muuttamalla liikealusta kallistuu joko eteen tai taakse. Roll-arvot kuvaavat liikealustan kiertoa. Yaw-arvoja muuttamalla liikealusta kääntää kuormaa sivuttain. Heave-, Surge- ja Lateral-arvot kuvaavat liikealustan sijaintia x-, y- ja z-koordinaatistossa, mitkä luetaan metreinä. Heaven arvoja muuttaessa liikealusta liikkuu joko ylös tai alas. Surge-arvoja muuttaessa liikealusta liikkuu eteen tai taakse. Lateral-arvojen muuttaminen

puolestaan liikuttaa liikealustaa oikealle tai vasemmalle. Alustalla on raja-arvot jokaiselle vapausasteelle (Taulukko 3). Näiden arvojen ylittäminen aiheuttaa virhetilan, jolloin liikealustan arvot vapausasteille tulee alustaa uudelleen. (Moog Motion System Division 2000b, 6.1.)

Taulukko 3. Moog-liikealusta vapausasteiden raja-arvot (mukaillen Moog Motion System Division 2000a, 6.1)

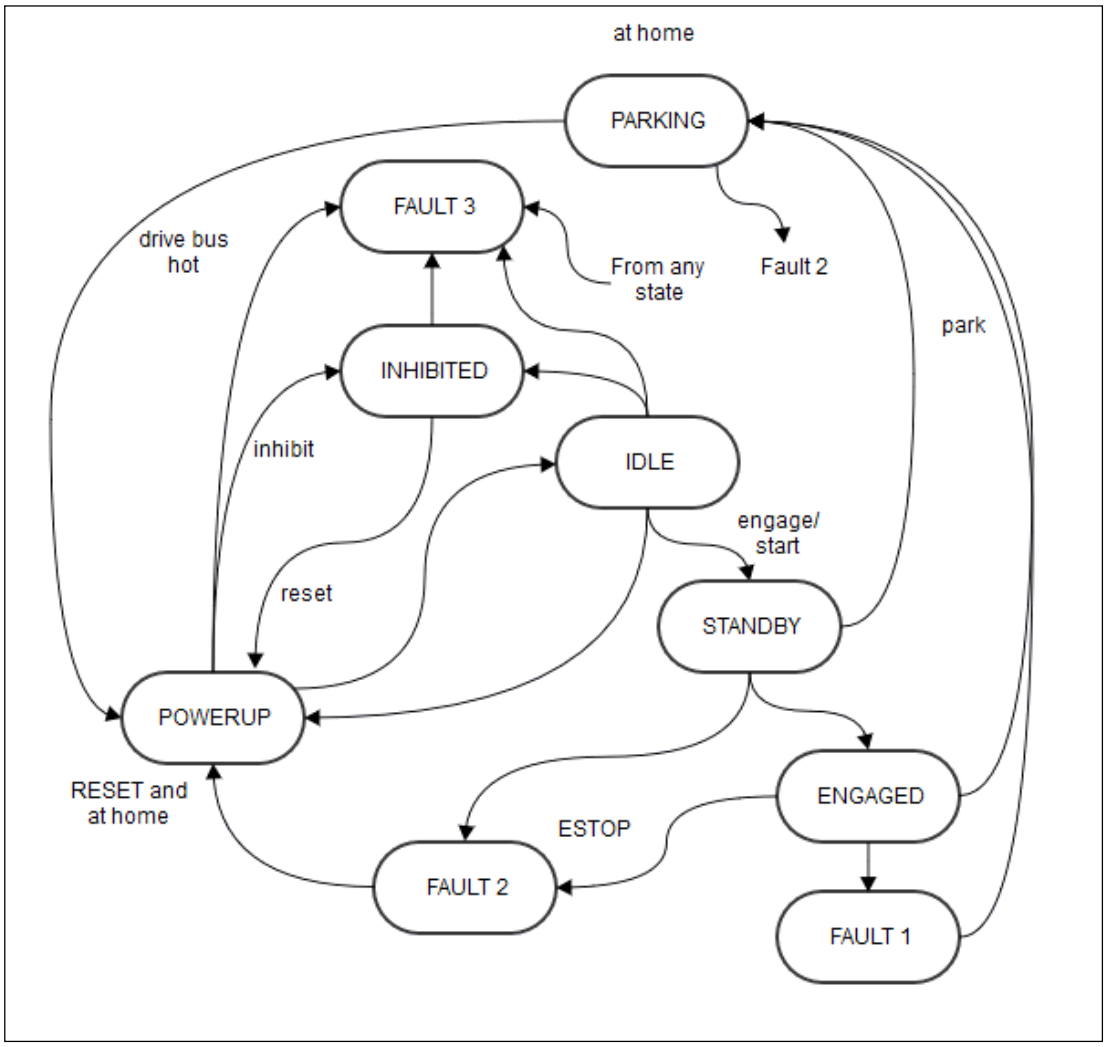
Vapausaste	**Pienin sallittu arvo**	**Suurin sallittu arvo**
Pitch (rad)	-0.57585	+0.57585
Roll (rad)	-0.50605	+0.50605
Yaw (rad)	-0.50605	+0.50605
Heave (m)	-0.4572	0.0
Surge (m)	-0.38100	+0.38100
Lateral (m)	-0.38100	+0.38100

3.2.3 Liikealustan tilat

Liikealustalla on yhdeksän toimintatilaa, joista kolme on virhetiloja (Kuvio 9.) Liikealusta käynnistyy POWERUP-tilaan. POWERUP-tila kertoo sen, että liikealusta on käynnistynyt oikein, mutta yhteyttä ei ole vielä luotu. Liikealusta siirtyy IDLE-tilaan, kun asiakaskoneesta on tullut validi datapaketti sekä akun ja kytkimien kunto on tarkistettu ja liikealusta on aloitusasennossa (Moog Motion System Division 2000a, 4.4). IDLE-tila kertoo sen, että yhteys asiakaskoneeseen on luotu.

Liikealusta siirtyy IDLE-tilasta STANDBY-tilan kautta ENGAGED-tilaan, kun asiakaskoneelta tulee validi ENGAGED-komento liikealustalle. STANDBY-tila kestää vain muutaman sekunnin, jonka aikana se hoitaa mahdollisten virhetilojen tarkistuksen ja nostaa liikealustan aloitusasentoon. ENGAGED-tilassa liikealusta on valmis ottamaan vastaan datapaketteja, jotka ohjaavat liikealustaa eri asentoihin. PARKING-tilassa liikealusta siirtyy muutamien sekuntien viiveellä takaisin perusasentoon, jonka jälkeen siirrytään automaattisesti IDLE-tilaan. INHIBITED-tila on liikealustan deaktivoimisen tila ja siitä voidaan päästä pois ainoastaan RESET-komennolla aloitusasennossa (Moog

Motion System Division 2000a, 3.12). Kuviossa 9 näkyy tarkempi kuvaus tilojen välisistä siirtymisistä.



Kuvio 9. Moog-liikealustan tilakaavio (Mukaillen Moog Motion System Division 2000a, 4.3)

ENGAGE-tilassa ollessaan UDP-pakettien lähetys on tapahduttava 60-100 kertaa sekunnissa, jotta yhteys pysyy aktiivisena ja mahdollistaa sulavan liikkeen tuottamisen liikealustalle. Mikäli datan vastaanottamisessa esiintyy yli 166 millisekunnin viive, se tuottaa yleensä virhetilan liikealustalle. Lähetystaajuuden pitää olla noin 60 Hz taajuudella, jotta liikealusta tunnistaa lähetetyt paketit. (Moog Motion System Division 2000a, 4.4.)

3.2.4 Datan lähettäminen liikealustalle

Liikealustalla on kolme ohjaustapaa; MDA-, DOF- ja Length-tilat, joita voi vaihtaa ainoastaan IDLE- ja POWERUP-tiloissa. Length-tilassa liikealustaa ohjataan sylinterien pituuksia muuttamalla. DOF-tilassa (Degree of Freedom) liikealustalle lähetetään dataa vapausasteiden mukaisesti. MDA-tila sisältää omat algoritmit liikealustan ohjaamiseen. MDA-tilassa lähetetään liikealustalle neljää eri syötedataa, joilla muutetaan liikealusta kulmanopeutta, kulmakiihtyvyyttä, lineaarista kiihtyvyyttä sekä liikealusta kallistumaa. (Moog Motion System Division 2000a, 2.1-3.5.) Liikealustan tiloja vaihdetaan datapaketissa olevan MCW:n eli Motion Command Word:n (Suom. komentosana) arvoa muuttamalla. Kuviossa 10 näkyy MDA-tilan komentosanat ja niiden bittiarvot.

WORD	DESCRIPTION	FORMAT (d31 … d0)			
0	MOTION COMMAND WORD (LSB holds command byte)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	x x x x x x x x	X X X X X X X X
	NEW MDA FILE NUMBER (see para 3.3)				
	DISABLE 220 (DC HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 0 1 1 1 0 0
	PARK 210 (D2 HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 0 1 0 0 1 0
	LOW LIMIT ENABLE 200 (C8 HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 1 0 0 1 0 0 0
	LOW LIMIT DISABLE 190 (BE HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 1 1 1 1 1 0
	ENGAGE 180 (B4 HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 1 1 0 1 0 0
	LENGTH MODE 172 (AC HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 1 0 1 1 0 0
	DOF MODE 170 (AA HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 1 0 1 0 1 0
	RESET 160 (A0 HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 1 0 0 0 0 0
	INHIBIT 150 (96 HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 1 0 1 1 0
	MDA MODE 140 (8C HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 0 1 1 0 0
	NEW MDA 128 (80 HEX)	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0
	NEW MDA FILE 155 (9B HEX)				

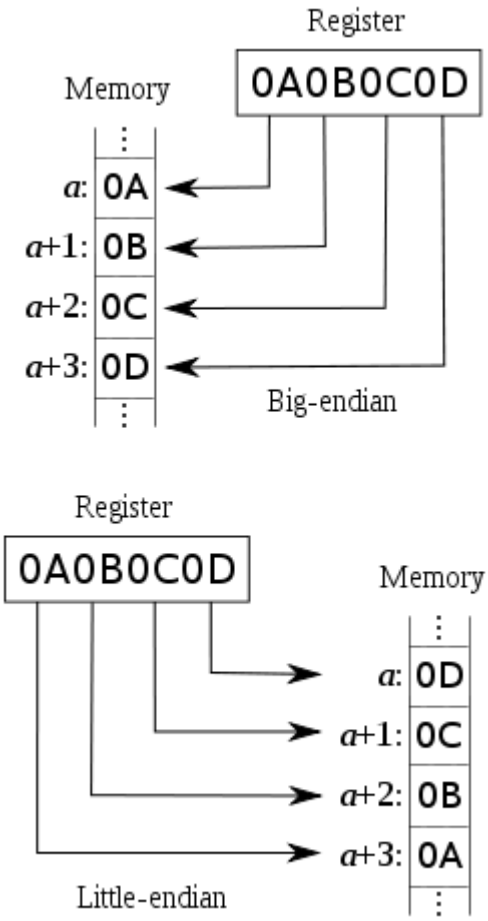
Kuvio 10. MDA-tilan MCW:n komennot ja niiden 16-bittiset arvot (Moog Motion System Division 2000a, 3.8)

Tässä työssä liikealustan ohjaamiseen käytettiin MDA-ohjaustilaa. Ohjaustila käyttää liikealustan omaa MDA-kirjastoa liikkeiden tuottamiseen. Liiketietoa voidaan lähettää liikealustalle MDA-datapakettina, kun MDA-ohjaustila on valittuna ja kun liikealusta on ENGAGED-tilassa. Tällöin datapaketin tulee olla taulukon 4 muotoinen. Komentosanan arvona tulee olla jokaisessa lähetetyssä paketissa NEWMDA, kun halutaan lähettää uutta liiketietoa alustalle.

Taulukko 4. Lähetettävän MDA-datapaketin rakenne (Moog Motion System Division 2000a, 3.6)

#	Data	Kuvaus	Yksikkö	Tyyppi
0	MCW	Motion Command Word	-	32 bit unsigned long
1	a_roll	Roll acceleration	rad/s^2	32 bit float
2	a_pitch	Pitch acceleration	rad/s^2	32 bit float
3	a_z	Acceleration vertical	m/s^2	32 bit float
4	a_x	Acceleration longitudinal	m/s^2	32 bit float
5	a_yaw	Yaw acceleration	rad/s^2	32 bit float
6	a_y	Acceleration lateral	m/s^2	32 bit float
7	v_roll	Roll rate	rad/s	32 bit float
8	v_pitch	Pitch rate	rad/s	32 bit float
9	v_yaw	Yaw rate	rad/s	32 bit float
10	roll	Roll angle	rad	32 bit float
11	pitch	Pitch angle	rad	32 bit float
12	yaw	Yaw angle	rad	32 bit float
13	buffet_roll	buffets roll	rad	32 bit float
14	buffet_pitch	buffets pitch	rad	32 bit float
15	buffet_z	buffets vertical	m	32 bit float
16	buffet_x	buffets longitudinal	m	32 bit float
17	buffet_yaw	buffets yaw	rad	32 bit float
18	buffet_y	buffets lateral	m	32 bit float
19	v_vehicle	vehicle speed	m/s	32 bit float
20		Spare	-	32 bit unsigned long
21		Spare	-	32 bit unsigned long

Kaikkien liikealustalle lähtevien datapakettien tavujen on oltava Big Endian -muodossa. Big Endian -muoto tarkoittaa sitä, että eniten merkitsevä bitti on ensimmäisenä. Liikealusta kääntää itse järjestyksen päinvastaiseksi eli järjestykseen, jossa vähiten merkitsevä bitti on ensimmäisenä (Moog Motion System Division 2000a, 3.1). Tätä järjestystä kutsutaan Little Endian -järjestykseksi (Kuvio 11.)



Kuvio 11. Big Endian- ja Little Endian -tavujärjestyksien muunnos (Mukaillen Wikipedia 2014)

3.2.5 Datan vastaanotto liikealustalta

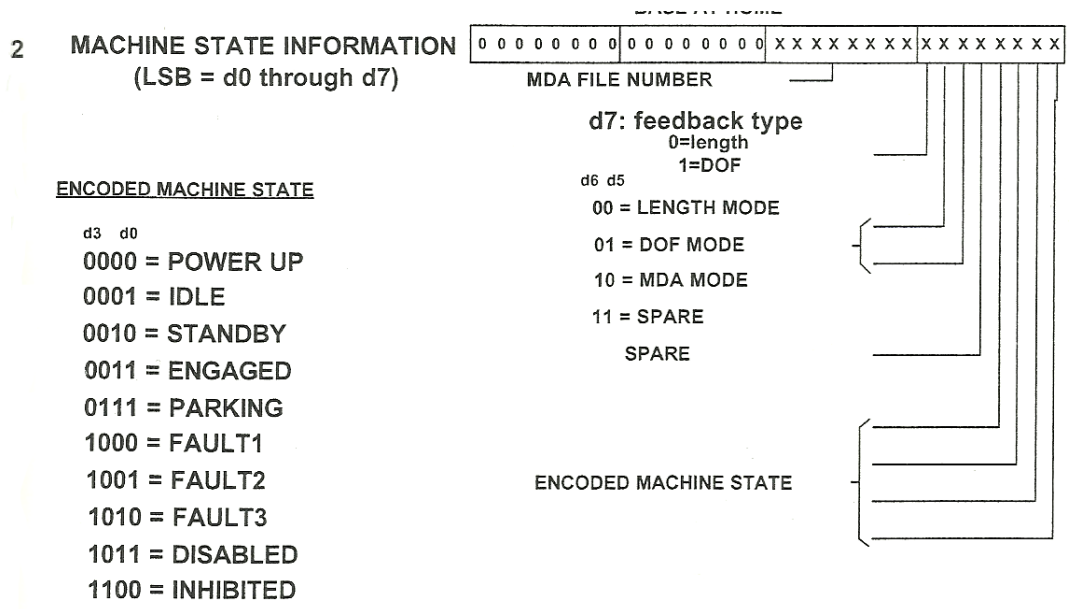
Kun liikealusta vastaanottaa validin datapaketin, liikealusta lähettää vastauksen lähettäjälle. Vastauksena liikealusta lähettää datapaketin, joka sisältää liikealustan tilatiedot. Vastauspaketit voivat olla joko LENGTH- tai DOF-tyyppisiä. LENGTH- ja DOF-paketit eroavat toisistaan siten, että LENGTH-paketti kertoo liikealustan asentotiedot sylinterin pituuksien mukaan, kun taas DOF-paketti vapausasteiden mukaan. (Moog Motion System Division 2000a, 3.13-3.18.)

Tässä työssä vastauspakettina käytettiin LENGTH-tyyppiä. Taulukossa 5 on kuvattu liikealustalta vastauksena tulevan LENGTH-datapaketin rakenne. Sylinterien pituuksien arvot ilmoitetaan metreinä.

Taulukko 5. Vastauksena tulevan LENGTH-datapaketin rakenne (Moog Motion System Division 2000a, 3.13)

#	**Data**	**Kuvaus**	**Yksikkö**	**Tyyppi**
0	fault	Latched Faul Data	-	32 bit unsigned long
1	io_info	Discrete I/O information	-	32 bit unsigned long
2	status	Machine State Information	-	32 bit unsigned long
3	length_a	Actuator Length A feedback	m	32 bit float
4	length_b	Actuator Length B feedback	m	32 bit float
5	length_c	Actuator Length C feedback	m	32 bit float
6	length_d	Actuator Length D feedback	m	32 bit float
7	length_e	Actuator Length E feedback	m	32 bit float
8	length_f	Actuator Length F feedback	m	32 bit float
9	-	Spare	-	32 bit tbd

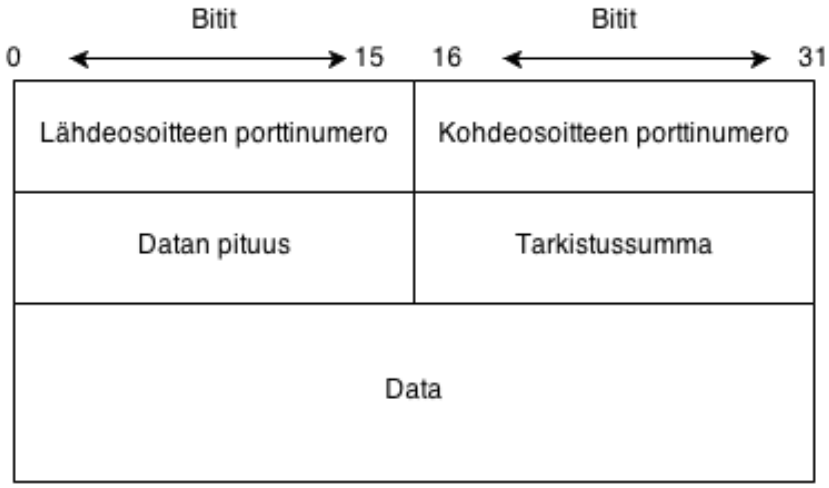
Liikealusta lähettää vastauspaketit Big Endian -muodossa (Moog Motion System Division 2000a, 3.1). Kuviossa 12 on ote LENGTH-vastauspaketin sisällöstä. Kuviossa nähdään liikealustan tilojen, toimintatilan ja vastaustyypin bittiarvot.



Kuvio 12. Ote LENGTH-vastauspaketin sisällöstä (Moog Motion System Division 2000a, 3.14)

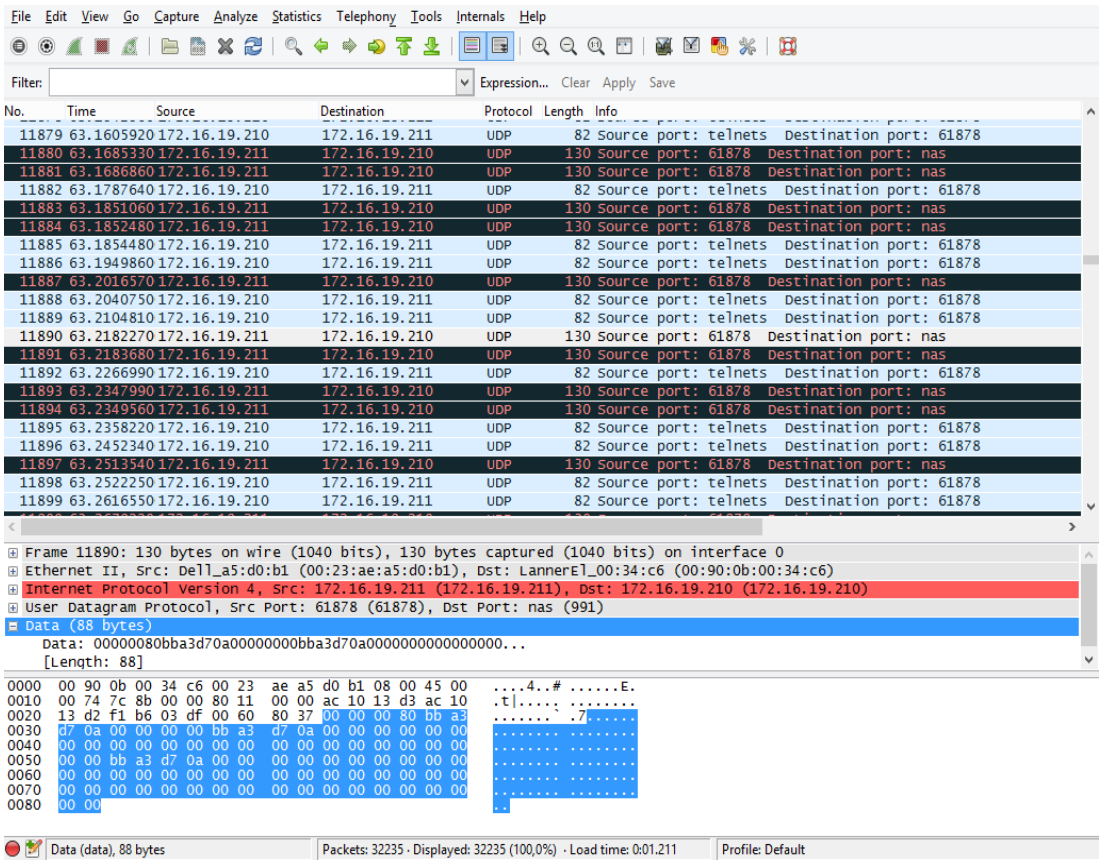
3.3 UDP-protokolla

UDP tulee sanoista User Datagram Protocol. UDP on yhteydetön tiedonsiirtoprotokolla. Yhteydettömässä tiedonsiirrossa lähetettävien pakettien perille pääsystä ei tule varmistusta lähettäjälle. Toisin sanoen dataa lähetetään tietämättä sitä, saapuvatko datapaketit perille ja onko datapaketeilla edes vastaanottajaa. Kuviossa 13 on esitetty UDP-paketin perusmuoto. Paketti sisältää tiedot lähde- ja kohdeosoitteen porttinumerosta, datan pituudesta, tarkistussummasta ja itse datasta. Porttinumeroiden avulla osoitetaan se sovellus, jonka kanssa halutaan keskustella. Tarkistussumma on vapaaehtoinen ja se voidaan sivuttaa asettamalle tarkistussumman kentän bitit nolliksi. Tarkistussumman avulla pystytään varmistamaan, että vastaanotettu paketti vastaa lähetettyä pakettia. (Internetix 2014.)



Kuvio 13. UDP-paketin perusmuoto (Mukaillen H3C 2014)

WireShark on verkkopakettien analysoimiseen tarkoitettu ohjelmisto (WireShark 2014). Ohjelmiston avulla nähdään saapuneet ja lähetetyt verkkopaketit sekä niiden sisältö (Kuvio 14). Tässä työssä WireShark oli erittäin hyödyllinen apuväline UDP-pakettien lähettämisen ja vastaanottamisen tarkasteluun. Myös bittimuunnosten tarkastelu onnistuu ohjelman avulla, sillä pakettien sisäinen data on myös luettavissa.



Kuvio 14. Kuvankaappaus lähetetystä MDA-datapaketista WireShark ohjelman kautta

3.4 Ajo-ohjain ja istuin

Liikealustalle ajajan istuimeksi hankittiin Playseat Evolution ajopelituoli. Ohjaimeksi hankittiin Logitechin G27 ajo-ohjain (Kuvio 15). Ajotuolin istuma-asentoa sekä ratin ja polkimien etäisyyttä voidaan säätää ajajalle sopivaksi.

Kuvio 15. Playseat Evolution-istuin ja Logitech G27 liikealustalla

Logitech G27 -ohjausjärjestelmä sisältää tärinäpalautteella varustetun kilpaohjauspyörän, kaasu-, jarru-, ja kytkinpolkimet sekä vaihdekepin (Kuvio 16). PC-koneiden lisäksi G27 on yhteensopiva Playstation 3 ja Playstation 2 -pelikonsoleiden kanssa. Tärinäpalautemekanismin avulla jäljitellään kilpa-ajotilannetta eli luodaan ajamisen tunnetta. Ohjauspyörä tarjoaa 900 asteen ohjauksen. Ohjauspyörä kääntyy 2,5 kierrosta ääriasennosta toiseen, kuten oikeassa kilpa-autossa. (Logitech 2014.) Hallintasovelluksen kautta ohjauspyörän kääntymisastetta ja muita asetuksia voidaan säätää mieluisiksi.

Kuvio 16. Logitech G27 liikealustalla

3.5 Oculus Rift -virtuaalilasit

Oculus Rift -virtuaalilasit on kehittänyt Oculus VR-niminen yritys (Oculus VR 2014a) (Kuvio 17). Virtuaalilasit mahdollistavat läsnäolon tunteen lisäämisen virtuaalimaailmassa. Virtuaalilasien avulla henkilö pystyy päätä kääntämällä tarkastelemaan virtuaaliympäristöä 360 asteisesti eli kuten oikeassa elämässäkin (Oculus VR 2014c). Virtuaalimaailmassa liikkuminen tuottaa läsnäolon tunteen ympäristössä.

Kuvio 17. Työssä käytetyt Oculus Rift -virtuaalilasit

Oculus Rift -virtuaalilasit tarjoavat noin 110 asteen näkökentän venyttäen virtuaalimaailmaa ääreisnäköä pidemmälle. Nykyinen Oculus Rift -kehittäjän pakkaus painaa 369 grammaa, joten paino ei ole normaaleja laskettelulaseja enempää. Oculus Rift -virtuaalilasit tukevat tällä hetkellä Windows, Mac OS X ja Linux-käyttöjärjestelmiä. (Oculus VR 2014c.)

Oculus Rift:n sovelluskehitystyökalun avulla virtuaalilasit voidaan liittää Unreal Development Kit -, Unreal Engine 4 - ja Unity 3D -pelimoottoreihin (Oculus VR 2014c). Oculus Rift:n liittäminen Unity 3D -pelimoottoriin vaatii Unity 3D:ltä pro-lisenssin. Oculus Rift:n kotisivuilta on saatavilla Unity 3D:lle valmis integrointipaketti, jonka avulla liittäminen onnistuu helposti. Oculus Rift -lasit tuottavat molemmille silmille oman kuvan kuvion 18 mukaisesti, tuottaen näin käyttäjälle kolmiulotteisen näkymän.

Kuvio 18. Oculus Rift -lasien tuottama kuva

Maaliskuussa 2013 Oculus VR ilmoitti julkaisevansa uuden version laseista kehittäjiä varten. Kehittäjäversio on nimeltään Development Kit 2. Näytön tarkkuus uudessa versiossa on 960*1080 pikseliä silmää kohden, joten yhdessä näytön tarkkuus on 1920*1080 pikseliä, joka on yhtä suuri kuin Full HD -tarkkuus. Uuden version kerrotaan myös vähentävän liikesumeutta ja tärinää. Development Kit 2 on ennakkotilattavissa Oculus Rift:n kotisivuilta noin 250 eurolla kirjoitushetkellä 3/2014. (Oculus VR 2014b.)

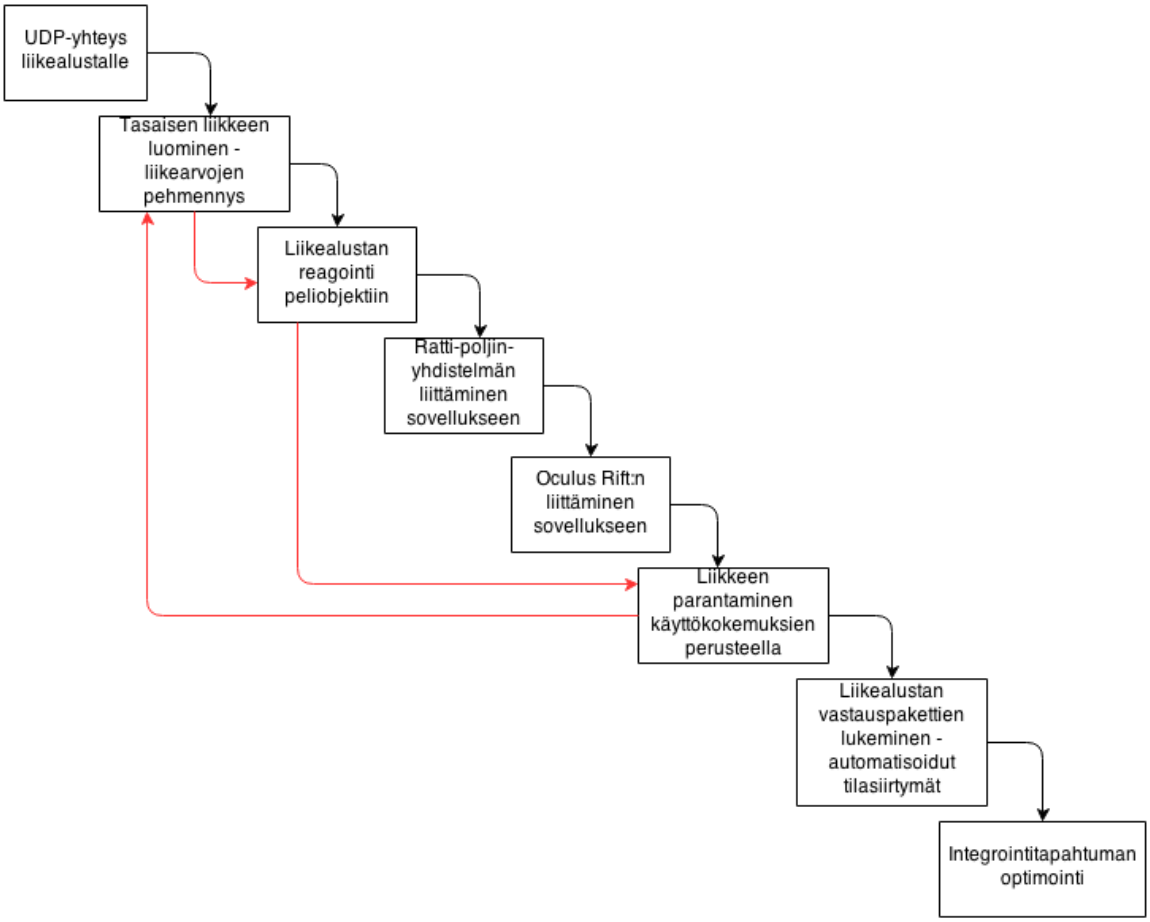
4 SOVELLUKSEN SUUNNITTELU

Työn tavoitteena oli Unity 3D -sovellus, joka ohjaa liikealustaa. Sovelluksen tuli liikuttaa liikealustaa pelimaailman auton asentojen ja liikkeiden mukaisesti. Pelimaailman auton liikkeet voivat kuitenkin olla liian äkkinäisiä ja jyrkkiä verrattuna oikean auton liikkeisiin. Sen vuoksi liikealustan tuli reagoida pelimaailman auton liikkeisiin, niin ettei immersiota heikentäviä äkkinäisiä liikkeitä tapahdu. Kun esimerkiksi pelimaailmassa auto ajaa kovaa rampin yli, sen asennot muuttuvat nopeasti ja jyrkästi. Liian nopeat ja jyrkät liikkeet eivät sovi liikealustan liikkeeksi, koska liike voi olla liian nopeatempoista ja vaarallista liikealustan päällä oleville. Kommunikointi liikealustan kanssa tuli toteuttaa UDP-yhteydellä. Lisäksi tavoitteena oli Oculus Rift -virtuaalilasien liittäminen projektiin tuomaan ajokokemukseen lisää immersiota. Lopputuloksen tuli olla helposti integroitavissa uusiin Unity 3D -projekteihin.

Sovelluksen toteuttaminen aloitettiin joulukuussa 2013, ja sovellus valmistui huhtikuussa 2014. Työn haasteena alussa oli se, ettei minkäänlaisia esimerkkiohjelmia ollut C#-kielellä saatavilla. Liikealustalle oli aikaisemmin tehty esimerkkiohjelma, jossa UDP-kommunikointi toteutettiin C++-ohjelmointikielellä. Esimerkkisovelluksen avulla liikealustaan saatiin yhteys, mutta sovellus ei mahdollistanut liikealustan liikuttamista. Ohjelman avulla saatiin kuitenkin käsitystä kommunikointisovelluksen toteutusmahdollisuuksista C#-kielellä. UDP-kommunikoinnin toteutuksessa hyödynnettiin myös esimerkkisovelluksen määrittelydokumenttia. Sovelluksen kehittäminen vaati myös liikealustan manuaalien huolellista tutkimista, jotta saatiin tarkempi käsitys laitteen toiminnasta.

Ohjelmointikielen ja kehitysalustan eroista johtuen esimerkkisovelluksen pohjalta ei voitu kuitenkaan tehdä tarkkoja suunnitelmia toteutettavasta sovelluksesta. Suunnittelu ja toteutus etenivätkin suurelta osin vaiheittain vesiputousmallin mukaisesti. Tämä toteutustapa oli luonnollisin valinta, sillä aikaisempaa kokemusta liikealustan ohjelmoinnista ei ollut. Lopussa ajosimulaatiota kehitettiin iteratiivisesti testikäyttäjien avulla. Tässä vaiheessa kerättiin käyttökokemuksia liikealustan liikkeestä. Käyttökokemuksien avulla saatiin kokonaiskuva liikkeen puutteista, joiden pohjalta sovellusta kehitettiin edel-

leen. Kuviossa 19 on kuvattu työvaiheet. Punainen nuoli kuvaa iterointivaiheita. Iterointivaiheissa kehitettiin liikealustan liikettä niin kauan, kunnes liike havaittiin mieluisaksi.

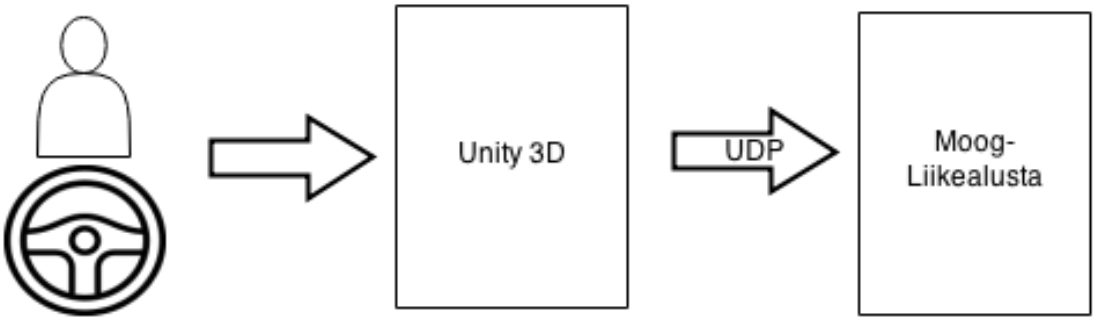


Kuvio 19. Työvaiheet ja niiden välinen siirtymä

Ensimmäinen vaihe oli UDP-yhteyden luominen liikealustalle. Vaiheen pohjalla käytettiin esimerkkisovelluksen kautta saatua tietoa kommunikoinnin toteutusmahdollisuuksista. Samassa yhteydessä toteutettiin bittijärjestyksen muunnos, jotta liikealusta tulkitsee lähetettävän liikedatan oikein. Kun yhteys oli saatu toimimaan, liikealustalle lähetettäville liikearvoille tehtiin pehmennys äkkinäisten arvojen muutoksien tapahtuessa. Seuraava vaihe oli toteuttaa liikealusta liikkumaan ohjattavan peliobjektin mukaisesti. Tämän jälkeen kokonaisuuteen liitettiin ratti-poljin-yhdistelmä ja Oculus Rift -lasit. Liikealustan liikettä kehitettiin mieluisaksi keräämällä kehitysideoita testaajilta. Kehitysideoiden perusteella sovellusta kehitettiin iteratiivisesti eli kehittämisen jälkeen testattiin liikettä aina uudelleen. Tässä vaiheessa säädettiin liikearvojen pehmennystä ja muutettiin liikealustan reagoimista peliobjektiin. Vaihetta tois-

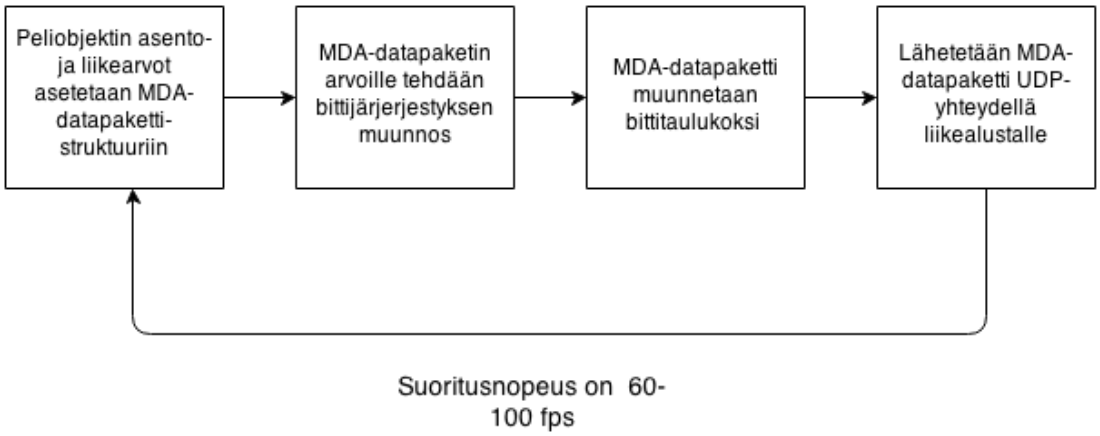
tettiin, kunnes liikealustan liikkeen katsottiin olevan miellyttävää. Sovellukseen toteutettiin myös liikealustalta tulevien vastauspakettien lukeminen. Vastauspakettien lukemisen avulla voitiin suorittaa automatisoituja tilasiirtymiä, kuten virhetilasta palautumisen. Kun toteutuksesta haluttiin vielä helposti integroitava, niin lopuksi liitettiin tarvittavat komponentit toiseen projektiin, jotta integrointitapahtumaa voitiin optimoida.

Toteutettavan ohjelmiston tuli toimia yksinkertaisuudessaan kuvion 20 mukaisella tavalla. Käyttäjä ohjaa Unity 3D -peliobjektia ratin ja polkimien avulla ja Unity 3D lähettää peliobjektin mukaiset asento- ja liiketiedot UDP-yhteydellä liikealustalle. Unity 3D-pelimoottorin peliobjektin asentotiedot saadaan selville Transform-komponentin rotaatio-arvoista. Peliobjektin kiihtyvyyden arvot puolestaan saadaan tietoon Rigidbody-komponentista. Näiden seikkojen vuoksi ohjelmisto katsottiin parhaimmaksi toteuttaa niin, että liikealustan liiketiedot luetaan suoraan peliobjektista.



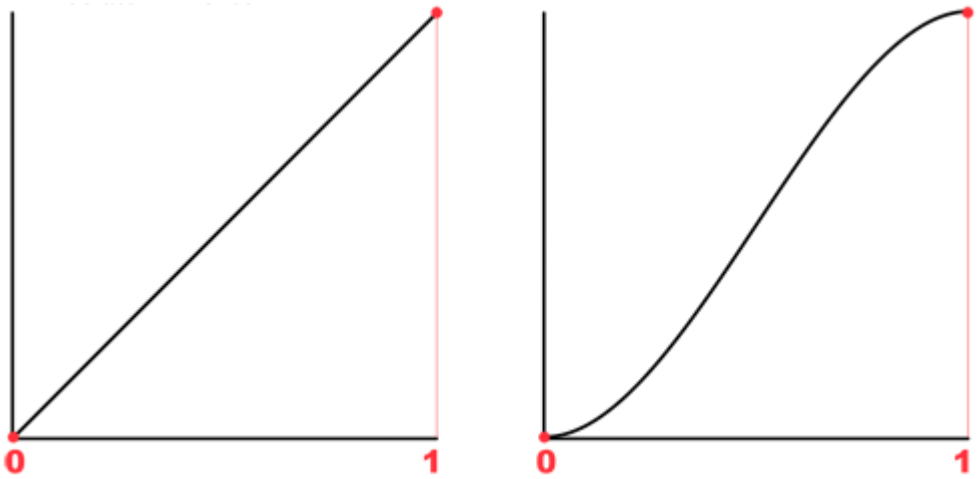
Kuvio 20. Hahmotelma sovelluksen yksinkertaistetusta toiminnasta

Kuviossa 21 on kuvattu suunnittelukaavio liikealustan asentojen päivittämisestä. Peliobjektin asento- ja liiketiedot on asetettava MDA-datapaketin sisältöä vastaavaan struktuuriin. Arvoille tulee tehdä bittijärjestyksen muunnos. Liikealusta tulkitsee vastaanotetut paketit Big Endian -muodossa, joten lähettäminen vaatii bittijärjestyksen muunnoksen. MDA-datapaketin lähettämistä varten MDA-datapaketti tulee muuntaa bittitaulukoksi. Datapaketti lähetetään liikealustalle määriteltyyn IP-osoitteeseen ja porttinumeroon. Lähetyksen on tapahduttava vähintään 60 kertaa sekunnissa, jotta yhteys liikealustaan pysyy aktiivisena.



Kuvio 21. Kaavio asento- ja liiketietojen päivittämisestä liikealustalle

Mikäli peliobjektin liike- ja asentotietoja lähetetään liikealustalle sellaisenaan, liikealustan liike ei ole sulavaa, vaan siitä muodostuu liian äkkinäistä. Tämän vuoksi peliobjektin rotaatio- ja liikearvoja tulee muokata ennen liikealustalle lähettämistä. Muokkaaminen katsottiin parhaimmaksi toteuttaa niin, että äkkinäisten muutosten tapahtuessa peliobjektin asennossa, liikealustalle lähetettävää arvon muutosta pehmennetään Unity 3D:n *SmoothDamp()* -funktion avulla.



Kuvio 22. SmoothDamp() -funktion toiminta (mukaillen Scratchapixel 2014)

Kuvion 22 avulla voidaan kuvata *SmoothDamp()* -funktion toiminta; lineaarinen käyrä pehmennetään kaarevaksi itse määritellyn ajan kuluessa. Pitemmässä mittakaavassa toiminta näkyy niin, että kaikki kulmat ovat kaarevia. Funktion avulla liikealustan arvot saadaan muuttumaan tasaisesti ja sulavasti, jotta liike tuntuisi miellyttävämmältä.

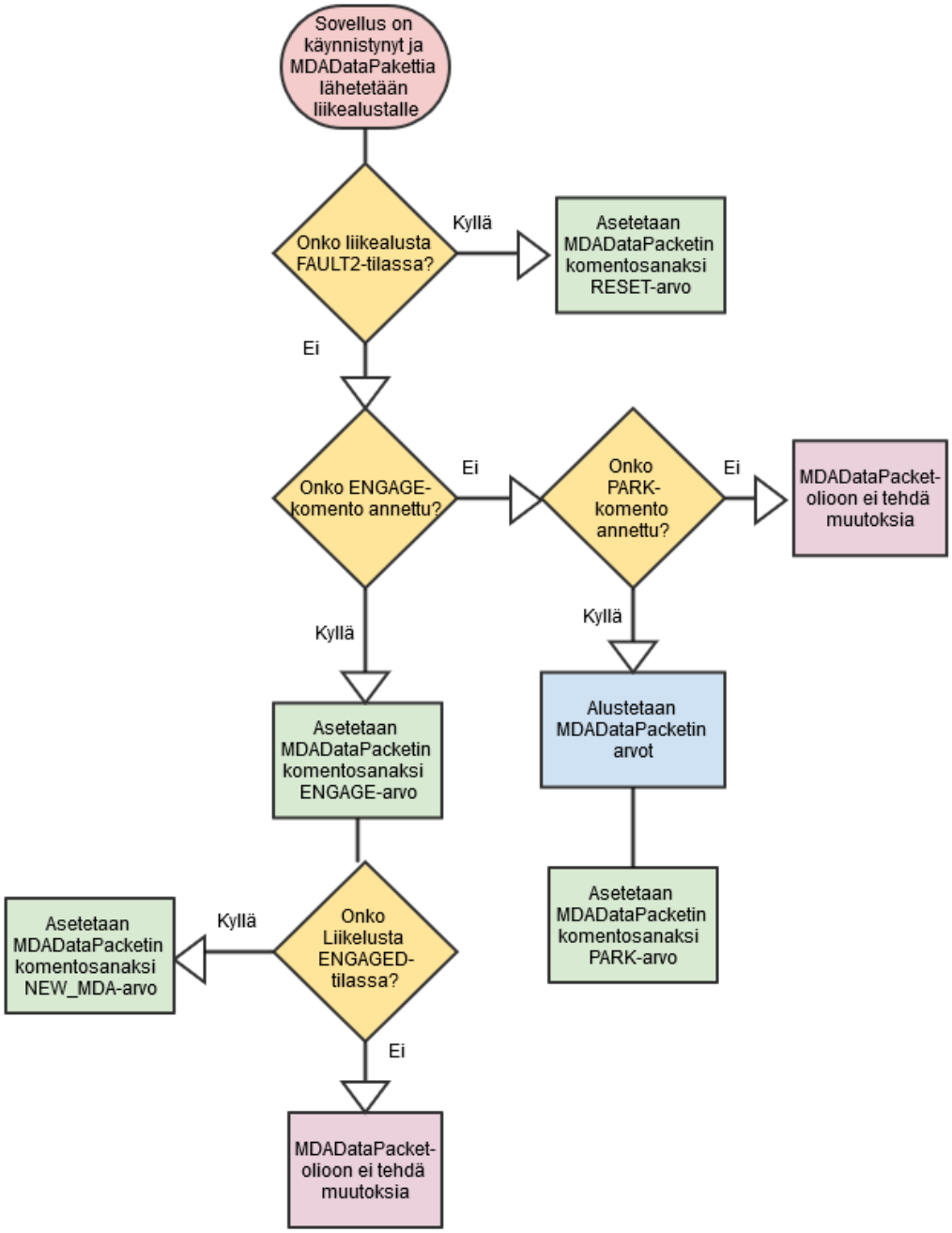
Liikealustalle valittiin ajopelituoli ja ohjauslaitteet. Ajopelituoliksi valikoitui Playseat:n Evolution -istuin ja ohjauslaitteeksi Logitechin G27. Logitech G27 katsottiin vastaavan tarpeet realistisen ohjauksen kannalta. G27 sisältää myös vaihteiston, mikä antaa lisävaihtoehtoja myöhempiin liikealustatoteutuksiin.

5 SOVELLUKSEN TOTEUTUS

5.1 Sovelluksen toimintaperiaate

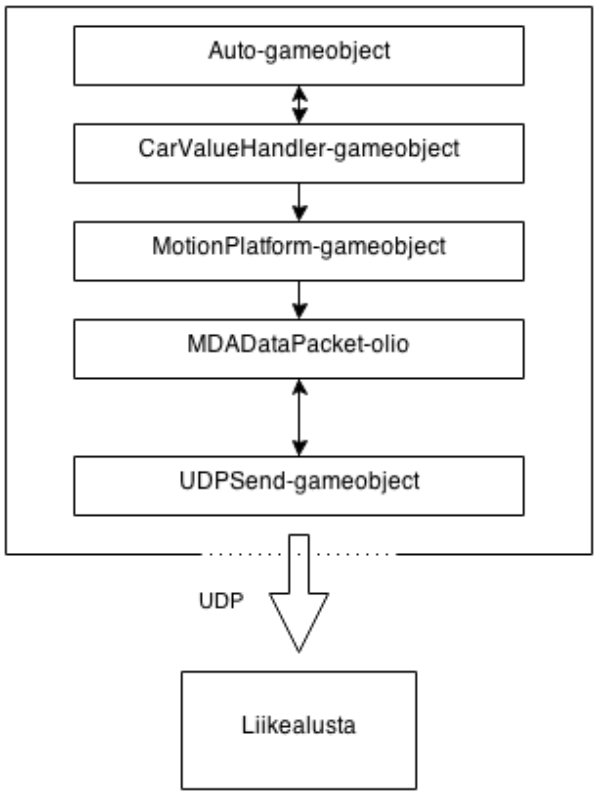
Toteutetussa ohjelmistossa liikealustalle lähetetään datapaketteja suoraan Unity 3D -pelimoottorin kautta. Kun sovellus käynnistetään luodaan *Start()*-funktion kautta tarvittavat alustukset. Käynnistyksen yhteydessä luodaan ilmentymät, kuten MDADataPacket-, UDPClient- ja IPEndPoint-oliot. Kun sovellus on käynnistynyt, MDADataPacket -olio muunnetaan bittitaulukoksi Marhal-kirjaston avulla. Bittitaulukkoa lähetetään taukoamatta *FixedUpdate()* -funktion kautta liikealustalle. Liikealusta tunnistaa yhteydenoton ja sen tila muuttuu POWER UP -tilasta IDLE-tilaan.

Menu-skriptiin on toteutettu näppäinkomentojen tarkkailu. *Menu*-skriptin *Update()*-funktiossa tarkistetaan, onko käyttäjä painanut näppäinkomentoa liikealustan tilan vaihtamiseksi. Näppäimistöön ja Logitechin G27:n näppäimiin on määritetty ENGAGE- ja PARK-tilojen komennot. ENGAGE-komennolla liikealusta nousee aloitusasentoon. Aloitusasennossa liikealustalta tulee vastauspaketin mukana tieto siitä, että liikealusta on ENGAGED-tilassa. Tämän jälkeen asetetaan liikealustalle lähetettävän MDA-datapaketin komentosanan arvoksi NEWMDA, jolloin liikealusta alkaa seuraamaan peliobjektin asento- ja liikedataa. PARK-komennolla liikealusta laskeutuu takaisin alas ja siirtyy IDLE-tilaan. Liikealustan tilan ollessa virhetilassa FAULT2, annetaan komentosanan arvoksi RESET, jolloin liikealusta palautuu takaisin IDLE-tilaan. Liikealustan tilat vaihtuvat kuviossa 23 kuvatulla tavalla.



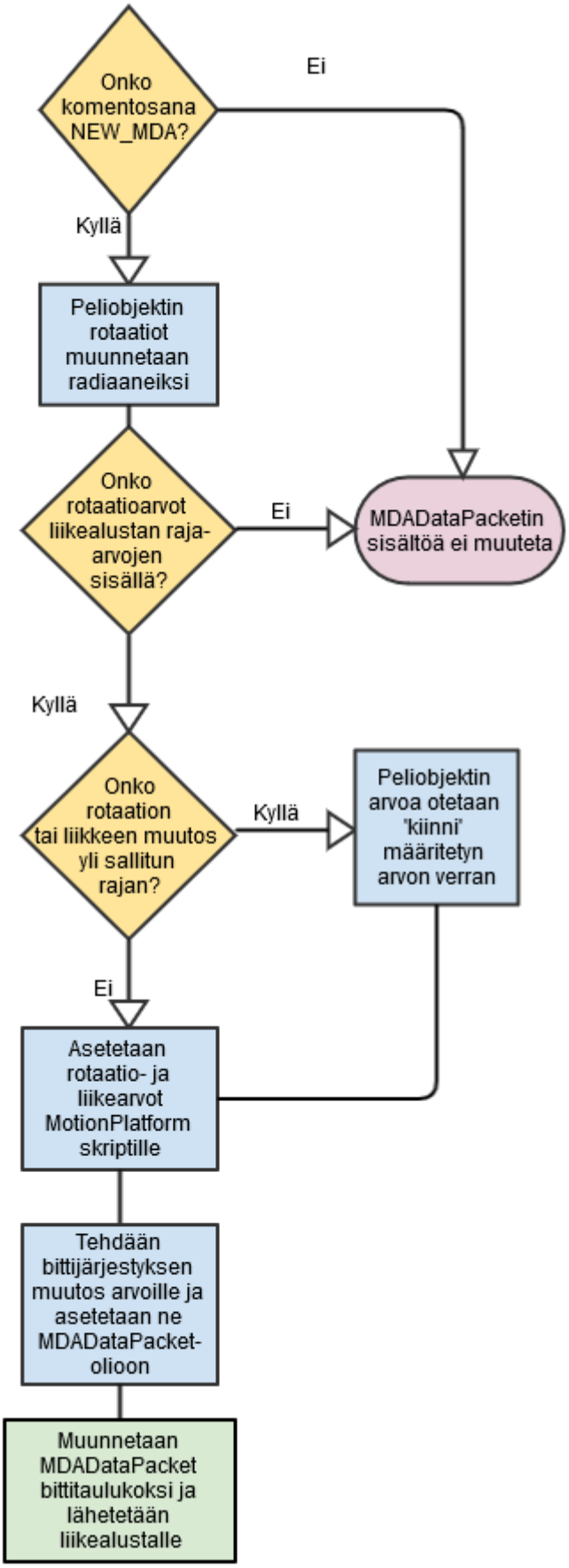
Kuvio 23. Vuokaavio sovelluksen tilojen välisestä siirtymisestä

Kun MDA-datapaketin komentosanaksi on asetettu NEWMDA, alkaa uuden liikedatan päivittäminen liikealustalle. Kuviossa 24 nähdään liikedatan päivitysreitti liikealustalle NEWMDA-tilassa. Kuvion peliobjekteista *CarValueHandler, MotionPlatform* ja *UDPSend* sisältävät vain yhden komponentin, joka on C#-skripti.



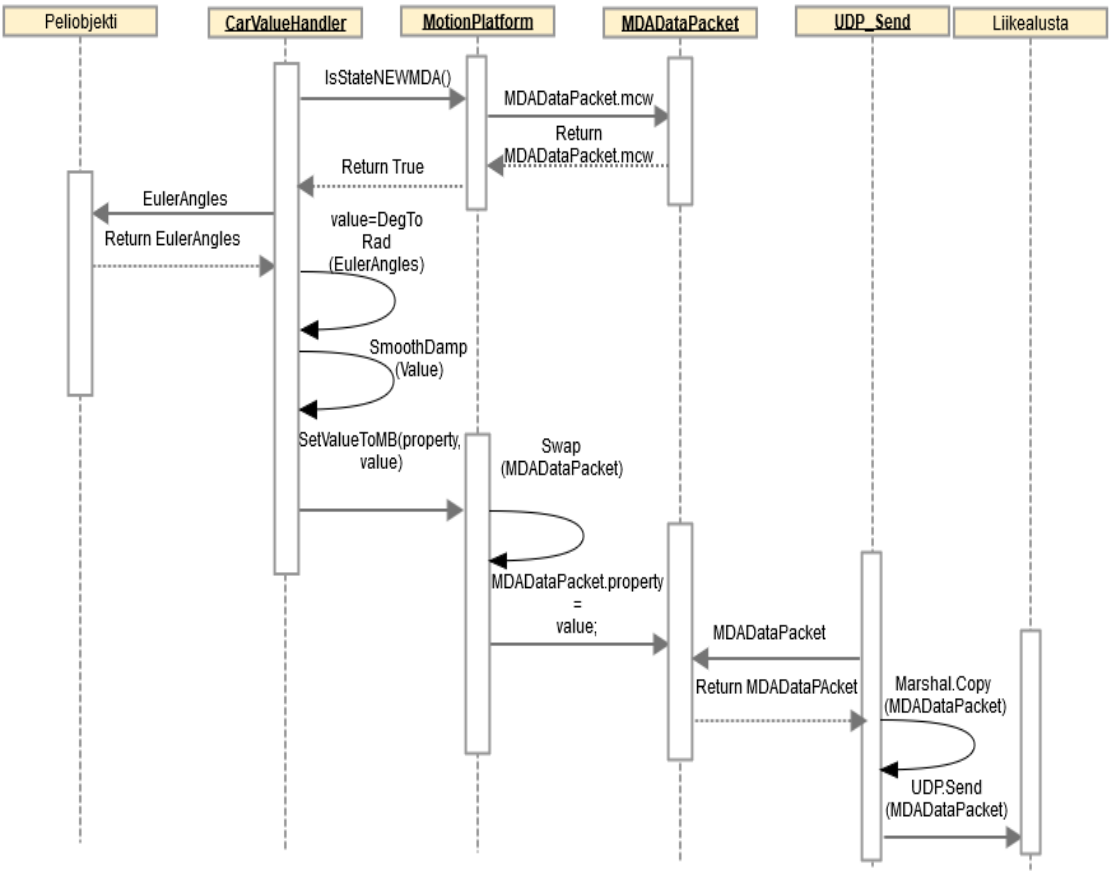
Kuvio 24. Kaavio liikedatan päivitysreitistä liikealustalle NEWMDA-komennolla

CarValuehandler tarkistaa, onko liikealustan komentosanaksi asetettu *NEWMDA*-komento. Komento tarvitaan, jotta liikealustalle voidaan lähettää uutta liiketietoa. *CarValuehandler* sisältää autopeliobjektin asentojen asteiden muunnoksen radiaaneiksi sekä liikearvojen pehmentämisen, mikäli peliobjektin asennon muutos on äkillinen. *CarValuehandler* tarkastaa, onko peliobjektin arvot liikealustan raja-arvojen sisällä ennen kuin se lähettää arvot *MotionPlatform*-peliobjektille. *MotionPlatform* hoitaa bittijärjestyksen muunnokset, joiden jälkeen muunnetut arvot asetetaan *MDADataPacket*:n sisällä olevalle *MDADataPacket*-tietueelle. *UDPSend* puolestaan muuntaa MDA-paketin bittitaulukoksi *Marshal*-nimisen C#-kirjaston avulla ja lähettää muunnettua taulukkoa kokoajan UDP-yhteydellä liikealustalle. Liikealusta sisältää sisäänrakennetut MDA-algoritmit, joiden avulla vastaanotettua dataa suoritetaan. Unity 3D -peliobjektin liikkeen ja asentojen päivittäminen liikealustalle on kuvattu vuokaaviona kuviossa 25. Liike- ja asentotietoja päivitetään liikealustalle jokaisen kehyksen välein.



Kuvio 25. Vuokaavio liikedatan päivittämisestä liikealustalle yhden kehyksen aikana

Kuviossa 26 on kuvattu komponenttien välinen keskustelu sekvenssikaaviona komentosanan ollessa NEWMDA. Komponenteista *CarValueHandler*, *MotionPlatform*, *MDADataPacket* ja *UDP_Send* ovat C#-skriptejä, jotka tekevät tarvittavat muunnokset peliobjektista saaduille asento- ja liiketiedoille, jotta ne voidaan lähettää liikealustalle.



Kuvio 26. Sekvenssikaavio komponenttien välisestä kommunikoinnista komentosanan ollessa NEWMDA

5.2 UDP-yhteys

Liikealustan ohjaustapana käytettiin MDA-tilaa, koska se mahdollistaa kehittyneemmän ohjaamisen verrattuna DOF- ja LENGTH-tiloihin. Ohjaamista varten toteutettiin MDAPacket-struktuuri, jonka ominaisuudet vastaavat MDA-paketin vaatimia ominaisuuksia (Liite 1). Unsigned Long -tyyppiset muuttujat ovat vaihdettu C#-kieltä vastaaviksi uint-tietotyyppisiksi muuttujiksi.

MDA-pakettia lähetetään UDP-yhteydellä liikealustalle Unity 3D:n *FixedUpdate()* -funktion kautta. Funktion suoritusnopeudeksi säädettiin Unity 3D:n editorin kautta noin 66 fps, jotta yhteys liikealustaan pysyy aktiivisena. UDP-yhteyttä varten ohjelma luo uuden UdpClient-olion. Itse datan lähetys tapahtuu *sendData()* -funktiossa, joka näkyy esimerkkikoodissa 1.

```
public void sendData(MDAPacket alusta)
{

    int size = Marshal.SizeOf(alusta);
    byte[] MDAdatab = new byte [size];
    IntPtr ptr = Marshal.AllocHGlobal(size);
    Marshal.StructureToPtr(alusta, ptr, true);
    Marshal.Copy(ptr, MDAdatab , 0, size);
    Marshal.FreeHGlobal(ptr);
    client.Send(MDAdatab, size, remoteEndPoint);


}
```

Esimerkkikoodi 1. Toteutettu sendData() -funktio

Funktiolle *sendData()* annetaan parametrina MDAPacket-struktuurista luotu ilmentymä, jolle tässä tapauksessa on annettu nimi ”*alusta”*. Funktio käyttää C#:n Marshall-luokkaa muuntamaan tiedot bittitaulukoksi. Tämän jälkeen *Send()*-funktio lähettää tiedot liikealustalle. Funktion ensimmäinen parametri sisältää muunnetun bittitaulukon, funktion toinen parametri sisältää bittitaulukon pituuden ja funktion kolmas parametri remoteEndPoint sisältää tiedot vastaanottajan IP-osoitteesta ja portista.

Alusta-nimisen olion asetetuille arvoille tehdään bittijärjestyksen muunnos, jotta liikealusta käsittelee vastaanotetun paketin tiedot oikein (Esimerkkikoodi 2). *MCW*:n eli komentosanan tietotyyppi on uint. Komentosanan bittijärjestyksen muunnosta varten ohjelmassa toteutettiin *SwapBytes()* -funktio, joka muuttaa ensimmäisen alkion viimeiseksi, toisen alkion toiseksi viimeiseksi ja niin edelleen.

```
public void SetRESET()
{
    alusta.Setmcw = SwapBytes(RESET);
    temp.nulltempvalues ();
    NullEverything();
}
public void SetENGAGED()
{
    temp.mcw_temp = ENGAGE;
    alusta.Setmcw = SwapBytes(ENGAGE);

}
public void SetValueToMB(string property, float radian)
{

    switch(property)
    {
    case "Set_aroll":
        temp.a_roll_temp = radian;
        alusta.Set_aroll= swap(temp.a_roll_temp);
        break;

    case "SetPitchAcceleration":
        temp.a_pitch_temp = radian;
        alusta.SetPitchAcceleration =swap(temp.a_pitch_temp);
        break;
    case "Seta_z":
        temp.a_z_temp = radian;
        alusta.Seta_z= swap(temp.a_z_temp);
        break;
```

Esimerkkikoodi 2. Ote MotionPlatform -skriptistä

Lisäksi toteutettiin float-arvojen bittijärjestyksen muunnokseen *swap()*-funktio, joka käytännössä tekee saman kuin *SwapBytes()*, mutta float tietotyyppejä varten. Kutsuttava *swap()*-funktio on *unsafe*-tyyppinen, jota Unity 3D ei oletuksena tue. Tämän vuoksi Assets-kansioon luotiin smcs.rsp-tiedosto, joka sallii *unsafe*-tyyppisten funktioiden suorittamisen Unity 3D:llä. Tiedosto sisältää pelkästään tekstin *"-unsafe"*. Liikealustan arvojen testaamiseen toteutettiin yksinkertainen käyttöliittymä, jonka avulla jokaista arvoa voi muuttaa painonappien avulla.

5.3 Tasaisen liikkeen tuottaminen

Tasaisen liikkeen luomiseksi ohjelmisto käyttää Unity 3D:n *SmoothDamp()* -funktiota. Kyseisen funktion avulla määritetään muutettavalle arvolle uusi arvo sekä aika, jonka kuluessa se vähitellen saavutetaan. Tämän ansiosta liikealustan liike on sulavaa, eikä yhtä äkkinäistä kuin pelimaailman auton liike.

Esimerkkikoodissa 3 näkyy *SmoothDamp()* -funktion käyttö ohjelmistossa. Funktiota kutsutaan *FixedUpdate()*:n kautta. Koodissa tarkastellaan edellisen kehyksen arvoa nykyisen kehyksen arvoon ja määritellään, kuinka suuri niiden välinen ero tulee olla, jotta *SmoothDamp()* -funktiota käytetään. Tällä saadaan selville, kuinka suuri pelimaailman auton kulman muutos vaaditaan, jotta arvoa aletaan pehmentämään.

```
if(pitch<(maxPitch)&& pitch>(minPitch))
{
    if( Mathf.Abs(pitch- pitch_prev)>0.005f)
    {

        Debug.Log ("pitch ero "+Mathf.Abs(pitch- pitch_prev).ToString());

        testout=testout+ Time.deltaTime;
        if(pitch_prev<pitch)
        {
            pitch_prev=Mathf.SmoothDamp(pitch_prev,pitch_prev+0.0025f,ref smoothvelocity,Time.deltaTime);
            Machine.SetValueToMB("SetPitchAngle",pitch_prev/3);
        }
        else if(pitch_prev>pitch)
        {
            pitch_prev=Mathf.SmoothDamp(pitch_prev,pitch_prev-0.0025f,ref smoothvelocity,Time.deltaTime);
            Machine.SetValueToMB("SetPitchAngle",pitch_prev/3);

        }
```

Esimerkkikoodi 3. Ote SmoothDamp() -funktion käytöstä

SmoothDamp() -funktiolla on neljä parametria, joista ensimmäinen kertoo aloitusarvon, toinen parametri kertoo saavutettavan kohdearvon ja kolmas parametri sisältää viittauksen nopeuteen. Funktio muokkaa tätä arvoa jokaisella kerralla, kun sitä kutsutaan (Unity 2014b). Neljäs parametri puolestaan kertoo missä ajassa funktio suoritetaan.

Neljänneksi parametriksi sopivin vaihtoehto oli asettaa *Time.deltatime*. Tämä kertoo ajan, joka kului viimeisen kehyksen suorituksesta. Kun tämä suoritetaan *FixedUpdate()* -funktion sisältä, saadaan suoritusajaksi jokaisella kerralla sama arvo. *FixedUpdate()* -funktion suoritusnopeutta voidaan muuttaa editorin kautta. Kohdearvoksi on valittu arvo, joka tuntuu mieleiseltä yhden kehyksen aikana tapahtuvasta liikealustan kulman muutoksesta. Toisin sanoen, kun pelimaailman auton asento muuttuu nopeasti, niin liikealusta ottaa kiinni jokaisen kehyksen aikana pelimaailman auton asentoa.

5.4 Liikealustan liike peliobjektista

Liikealustan liike tapahtuu Unity 3D:n auto-objektin mukaisesti. Pitch-, Roll- ja Yaw-arvot luetaan suoraan Transform-komponentin Rotation-arvoista (Esimerkkikoodi 4). Rotation-arvot kertovat peliobjektin asennon asteina pelimaailmassa. Pitch-akselia varten tehtiin myös tila, jossa arvo luetaan auton kiihtyvyydestä ja tila, jossa huomioidaan sekä peliobjektin kiihtyvyys että asento. Kiihtyvyys tuli huomioida, jotta voidaan tuottaa jarruttamisen ja kiihdyttämisen tunne ajajalle. Asennon kautta puolestaan liikealusta reagoi töyssyihin ja kuoppiin. Näiden huomioiden vuoksi oletustilaksi asetettiin molempien yhdistetty tila. Tilojen vaihtaminen onnistuu kesken ajon ratin tai näppäimistön painikkeiden avulla, jotta voidaan tarvittaessa testata mikä on miellyttävin liike Pitch-akselille.

Liikealustan tasokulmien yksikkötyyppinä käytetään radiaaneja, jonka vuoksi Unity 3D:n peliobjektin asteet muutetaan radiaaneiksi. Unity 3D:n ominaisuutena on se, että ajon aikana negatiiviset kulmat muunnetaan positiivisiksi. Esimerkiksi -10:n asteen kulma muunnetaan 350 asteeksi. Koodissa muunnetaan kulmat negatiivisiksi, mikäli arvo on suurempi kuin 180 astetta, jotta muunnos radiaaneiksi pitää paikkansa.

```
void Update () {

    if(Machine.IsNEWMDA() == true)
    {
        if(RulerCar.transform.eulerAngles.z>180)
        {
            roll=Mathf.Deg2Rad *(RulerCar.transform.eulerAngles.z-360f);
            Debug.Log("Roll eli z: "+roll);
            if(roll<(maxRoll)&& roll>(minRoll))
            {
                Machine.SetValueToMB("SetRoll",roll/3);
            }
        }
        else
        {
            roll=Mathf.Deg2Rad * RulerCar.transform.eulerAngles.z;
            Debug.Log("Roll eli z: "+roll);
            if(roll<(maxRoll)&& roll>(minRoll))
            {
                Machine.SetValueToMB("SetRoll",roll/3);
            }
        }
```

Esimerkkikoodi 4. Ote CarValueHandler-skriptin Roll-arvon lukemisesta liikealustalle

Lisäksi ohjelmassa tarkistetaan, että peliobjektin arvo on liikealustan vapausasteen raja-arvojen sisällä. Peliobjektin arvot asetetaan *alusta*-oliolle *Update()*-funktion kautta. *Alusta*-oliota eli MDA-datapakettia, lähetetään edelleen koko ajan liikealustalle *FixedUpdate()* -funktion kautta.

Ohjauksessa tulee ottaa huomioon kuljettajaan kohdistuvat voimat eri tilanteissa, jotta liikealustan liike tuntuisi realistisemmalta. Mutkiin tultaessa kuljettajaan kohdistuu voima, joka vetää kuljettajaa päinvastaiseen suuntaan kuin mihin ollaan kääntymässä. Liikealustan sivuttaisliikkeen mahdollistava vapausaste on kuvattu muuttujalla *lateral*. Esimerkkikoodissa 5 nähdään, miten sivuttaisen voiman tuottaminen liikealustalle peliobjektista on toteutettu.

```
Vector3 localAngularVelocity = RulerCar.transform.InverseTransformDirection(RulerCar.rigidbody.angularVelocity);
lateral=localAngularVelocity.y;
if(lateral<=(maxLateral)&& lateral>=(minLateral))
{
    if(Mathf.Abs(lateral-lateral_prev)>0.2f)
    {
        if(lateral_prev<lateral)
        {
            lateral_prev=Mathf.SmoothDamp(lateral_prev,lateral_prev+0.2f,ref smoothvelocity, Time.deltaTime);
            Machine.SetValueToMB("Seta_y",lateral);
        }
        if(lateral_prev>lateral)
        {
            lateral_prev=Mathf.SmoothDamp(lateral_prev,lateral_prev-0.2f,ref smoothvelocity, Time.deltaTime);
            Machine.SetValueToMB("Seta_y",lateral);
        }
    }
    else
    {
        Machine.SetValueToMB("Seta_y",lateral);
        lateral_prev=lateral;
    }
}
```

Esimerkkikoodi 5. Ote CarValueHandler-skriptin sivuttaisen liikkeen tuottamisesta liikealustalle

Rigidbody-fysiikkakomponentin avulla saadaan selville kulmanopeus *angularVelocity*-nimisen vektorin kautta. Tämän avulla saadaan sivuttaisliike selville vektorin y-alkiosta, kunhan peliobjektin rotaatio on asetettu niin, että sen etenemissuunta on pelimaailman z-akselin suuntainen pelin käännösvaiheessa. Peliobjektin etenemissuunta on Unity 3D:ssä kuvattu scenessä näkyvän peliobjektin sinisellä akselilla.

Transform-luokan *InverseTransformDirection()* -funktion avulla saadaan selville peliobjektin paikallinen kulmanopeus. Peliobjektin paikallinen kulmanopeus tarvitaan, jotta kulmanopeus päivittyy suhteessa auton asentoon eikä pelimaailman x-, y- ja z-koordinaatistoon. Kulmanopeuden selvityksen jälkeen tarkistetaan, onko sivuttaisliikkeen muutos edellisen kehyksen arvoon verrattuna niin suuri, että arvon muutosta pehmennetään *SmoothDamp()*:n avulla. Mikäli näin ei ole, niin arvo lähetetään suoraan liikealustalle.

Auton lähtiessä liikkeelle kohdistuu kuljettajaan voima, joka työntää kuljettajaa taaksepäin. Jarruttaessa kuljettajaan kohdistuu päinvastainen voima. Esimerkkikoodissa 6 muuttuja *surge* kuvastaa liikealustan edestakaisen liikkeen. Muuttujalle annetaan peliobjektin kiihtyvyyden arvo, joka on suhteutettu peliobjektin nopeuden arvoon. Näin ollen kovemmassa vauhdissa liikettä on enemmän kuin hitaammassa. Pelimaailman auton kiihdyttäessä liikealusta liikkuu taaksepäin. Peliobjektin jarruttaessa tai törmätessä liikealusta liikkuu eteenpäin. Pitch-akselille peliobjektin kiihtyvyyden ja nopeuden suhteutettu arvo asetetaan samalla tavalla kuin esimerkkikoodissa 6 on kuvattu. Peliobjektin nopeus saadaan selville Rigidbody-komponentin *Velocity*-vektorin kautta. Kiihtyvyys puolestaan saadaan selville vertaamalla nopeuden muutosta edellisen kehyksen nopeuteen.

```
nopeusvektori=       RulerCar.transform.InverseTransformDirection(RulerCar.rigidbody.velocity);
kiihtyvyysvektori = (nopeusvektori-kiiht_edel)/Time.fixedDeltaTime;

kiiht_edel=nopeusvektori;
nopeus=(nopeusvektori.z)*accmultiplier;
kiiht = (kiihtyvyysvektori.z*nopeus)*-1f;
if(surge<(maxSurge)&& surge>(minSurge))
{
    if( Mathf.Abs(surge- surge_prev)>0.12f)
    {
        if(surge_prev<surge)
        {

            surge_prev=Mathf.SmoothDamp(surge_prev, surge_prev+0.12f, ref smoothvelocity,Time.deltaTime);
            Machine.SetValueToMB("Seta_x",surge_prev);

        }
        else if(surge_prev>surge)
        {
            surge_prev=Mathf.SmoothDamp(surge_prev,surge_prev-0.12f, ref smoothvelocity,Time.deltaTime);
            Machine.SetValueToMB("Seta_x",surge_prev);
        }
```

Esimerkkikoodi 6. Ote peliobjektin kiihtyvyyden ja nopeuden vaikutuksesta liikealustaan

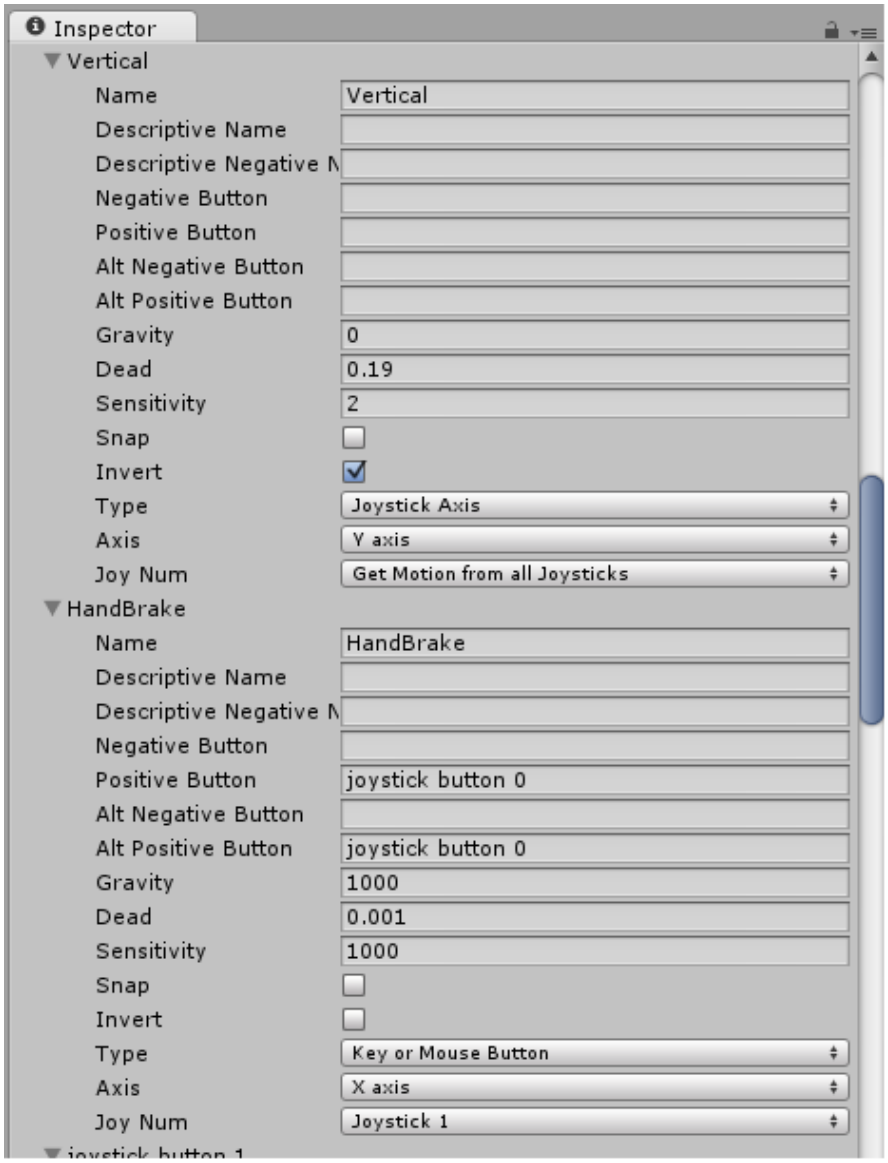
Ohjelmisto tuottaa liikealustalle pieniä ylösalaisia liikkeitä, jotta ajajalle välittyy tunne auton liikkeestä. Ylösalaisien liikkeiden toteuttamiseen käytettiin *PingPong()* -funktiota (Esimerkkikoodi 7). Toisaalta, mikäli Unity 3D -peliobjektiin on itsessään mallinnettu jousitusfysiikka, niin ylösalainen liike saadaan suoraan peliobjektista ilman *PingPong()* -funktion käyttöä. Tämän vuoksi sen toiminta toteutettiin niin, että sen suorittaminen sovelluksessa voidaan valita, mikäli niin halutaan. *PingPong()* -funktio lisää ja vähentää edestakaisesti arvoa ensimmäisen parametrina annetun ajan mukaan. Esimerkkikoodi 7:n tapauksessa kyseinen funktio vaihtaa arvoa nollan ja toisen parametrin arvon välillä.

```
if(RulerCar.rigidbody.velocity.magnitude>3f)
{

    heave=Mathf.PingPong(Time.time/10,0.1f);
    Machine.SetValueToMB("Seta_z",heave);


}
```

Esimerkkikoodi 7. Ote PingPong() -funktion käytöstä

5.5 Ratin ja polkimien lisäys

Ratti-poljin-yhdistelmäksi valittiin Logitechin G27-malli. Unity 3D:ssä eri peliohjaamien lisääminen on suhteellisen helppoa. Unity 3D:n editorin kautta peliohjaimien akseleiden ja nappien toimintaa voi hallinnoida Input Manager -asetuksista (Kuvio 27).



Kuvio 27. Kuvankaappaus Input Manager -asetuksista

Auton ohjaamiseen liittyvät peliohjaimen akseleiden ja näppäinten komennot on toteutettu Input Manager -asetuksissa. Koodissa näitä komentoja kutsutaan asetuksissa määritetyn nimen mukaan. Esimerkkikoodi 8 näyttää miten komentoa kutsutaan.

```
if(Input.GetButton("HandBrake"))
{
    if(!handbrake)
    {
        handbrake = true;
        handbrakeTime = Time.time;

    }
}
```

Esimerkkikoodi 8. Ote peliohjaimen HandBrake-näppäinkomennosta

5.6 Oculus Rift:n liittäminen

Oculus VR:n kotisivuilta ladattavan Unity 3D -integrointipaketin mukana tulee tarvittavat prefab-tiedostot, joilla Oculus Rift:n liittäminen pelimoottoriin onnistuu. Tässä työssä scenen sisälle lisättiin Oculus Rift:n kamera-peliobjekti, joka asetettiin halutulle paikalle auton sisälle sekä seuraamaan auto-peliobjektin suuntaa. Kuviossa 28 nähdään, kuinka editorin game-osiossa Oculus Rift -kamera näyttäytyy.

Kuvio 28. Kuvankaappaus Oculus Rift -kameran näkymästä Unity 3D -pelimoottorilla

5.7 Käyttökokemukset

Liikealustan liikettä testattiin Unity 3D:n esimerkkiprojektilla käyttökokemuksia keräämällä. Testauksia toteutettiin, jotta saatiin parempi kokonaiskuva liikkeen realistisuudesta. Liikealustasovellusta kehitettiin palautteen myötä. Ensimmäisten testikertojen jälkeen todettiin liikkeen olevan liian minimaalista, joten liikeratoja suurennettiin. Samalla huomattiin liikealustan nytkähtävän komentosanan vaihtuessa NEWMDA-komennoksi, kun pelimaailman auto on kaltevalla pinnalla ja kun liikealustalle on kertaalleen annettu PARK-komento. Tämä korjattiin alkuarvojen nollaamisella jokaisen PARK-komennon yhteydessä.

Testikerran yhteydessä huomattiin myös, ettei liikealusta reagoi riittävästi törmäyksien tapahduttua. Pelimaailman auton kiihdyttäessä toivottiin myös lisää kiihtyvyyden tunnetta. Törmäyksien tunnetta lisättiin säätämällä liikealustan liikettä nopeammaksi. Kiihtyvyyden tunnetta lisättiin puolestaan

toteuttamalla uusi tila, jossa liikealustan Pitch-akselin arvo luetaan peliobjektin kiihtyvyyden arvosta suhteessa nopeuteen. Kiihtyvyys laskettiin vertaamalla nopeuden muutosta edellisen kehyksen arvoon. Uuden tilan avulla saatiin myös jarruttamisen tunnetta liikkeeseen.

pLAB:ssa toteutettiin testauskerta Rovaniemen Pelilabran testikäyttäjäryhmän kanssa, kun liikealustan liikkeitä oli saatu parannettua haluttuun suuntaan. Testikerran yhteydessä kerättiin käyttökokemuksia ja parannusehdotuksia lomakkeen avulla (Liite 3). Testikerralla testattiin muun liikkeen yhteydessä liikealustan Pitch-akselin kahta eri liiketilaa, joista ensimmäisessä arvo luetaan peliobjektin asennosta ja toisessa kiihtyvyyden suhteesta nopeuteen. Testikokemusten palautteiden perusteella ei voitu täysin päätellä, kumpi tapa Pitch-akselin arvon päivittämiselle olisi parempi, vaikkakin asennon mukaan toteutetusta tilasta puuttui jarruttamisen tunne. Tästä johtuen katsottiin parhaimmaksi yhdistää tilat, niin että Pitch-akselin arvo luetaan sekä peliobjektin asennosta että kiihtyvyydestä. Testikerran palautteen johdosta myös ratin kääntöastetta suurennettiin, jottei liike ole käännyttäessä liian äkkinäistä.

Lopulta testikokemusten perusteella yhdistetty tila tuntui mieluisammalta ja tuotti parhaimman ajokokemuksen. Tilan avulla liikealusta reagoi, sekä ajopintaan että kiihdytyksiin ja jarrutuksiin. Kehitysideoita liikealustan liikkeeseen ei enää testauksen yhteydessä ilmennyt. Liikkeen testaus ja hienosäätö on pakollista kun lopputulos liitetään johonkin uuteen Unity 3D -projektiin, koska uuden peliobjektin toteutettu fysiikkamallinnus eroaa varmasti tämän työn kehittämisessä käytetyn Unity 3D -projektin auton fysiikkamallinnuksesta.

5.8 Liikealustan datan vastaanotto

Liikealusta lähettää vastauksen aina vastaanotettuaan validin datapaketin. Vastauspakettien avulla saadaan selville muun muassa liikealustan tila ja asentotiedot. Sovelluksessa oli tarve saada selville liikealustan tila, jotta voidaan tehdä automatisoituja tilasiirtymiä. Virhetiloista FAULT2-tila mahdollistaa RESET-komennolla virheestä palautumisen. Liikealustan siirtyessä ENGAGED-tilaan, voidaan puolestaan automaattisesti alkaa lähettämään liikedataa NEWMDA-komentosanalla.

Oletuksena liikealusta lähettää vastauspaketit porttinumeroon 992, joten tiedon lähettämisen yhteydessä luotu *UDPClient* -olio asetettiin kuuntelemaan kyseistä porttia. Esimerkkikoodissa 9 nähdään *ReceiveCallback()* -funktio. Funktio on asynkroninen CallBack-funktio eli tässä tapauksessa se suoritetaan, kun *UDPClient*-olio on vastaanottanut dataa liikealustalta. Liikealustalta tullut data asetetaan bittitaulukkoon *EndReceive()* -funktiokutsun avulla. *BeginReceive()* -funktiokutsulla aktivoidaan uudestaan datan asynkroninen vastaanotto. Bittitaulukko, johon vastaanotettu data asetettiin, muunnetaan biteistä *ResponsePacket* -nimiseen tietueeseen esimerkkikoodissa näkyvän *fromBytes()* -funktion avulla. *ResponsePacket* -tietueen rakenne vastaa aiemmin taulukossa 5 kuvattua LENGTH-vastauspaketin rakennetta.

```
public void ReceiveCallback(IAsyncResult ar)
{
    UdpClient u = (UdpClient)((UdpState)(ar.AsyncState)).u;
    IPEndPoint e = (IPEndPoint)((UdpState)(ar.AsyncState)).e;
    Byte[] receiveBytes = u.EndReceive(ar, ref e);
    R_Packet = fromBytes (receiveBytes);
    messageReceived = true;
    u.BeginReceive(new AsyncCallback(ReceiveCallback), s);
}

ResponsePacket fromBytes(byte[] arr)
{
    ResponsePacket str = new ResponsePacket();
    int size = Marshal.SizeOf(str);
    IntPtr ptr = Marshal.AllocHGlobal(size);
    Marshal.Copy(arr, 0, ptr, size);
    str = (ResponsePacket)Marshal.PtrToStructure(ptr, str.GetType());
    Marshal.FreeHGlobal(ptr);
    m_response.CheckState (R_Packet);
    return str;
}
```

Esimerkkikoodi 9. Ote UDP-vastaanoton toteuttamisesta ja tiedon lukemisesta bittitaulukkoon

Response-skriptissä tarkistetaan liikealustan tila ja tehdään tarvittavat automatisoidut tilasiirtymät (Esimerkkikoodi 10). Ensin *ResponsePacket*-tietueesta luodun olion *status*-muuttujalle tehdään bittijärjestyksen muunnos Little Endian -muotoon. Muunnoksen jälkeen *status*-muuttujan arvolle suoritetaan bittimaskaus &-operaattorin avulla. Maskaus suoritetaan, jotta saadaan selville mitä tilaa liikealustalta tullut arvo vastaa. Liikealustan ollessa FAULT2-virhetilassa suoritetaan RESET-käsky, jonka avulla liikealusta pa-

lautuu IDLE-tilaan. Liikealustan ollessa ENGAGED-tilassa asetetaan lähetettävän paketin komentosanan arvoksi NEWMDA, jotta liikealusta alkaa heti ottamaan uutta liikedataa vastaan.

```
// CHECKING MOTIONPLATFORM BASE STATUS
public void CheckState(ResponsePacket resp)
{
    uint status= Machine.SwapBytes(  resp.status);
    uint maskvalue = ((status) & 0x000F);
    if (BaseStatusValue != maskvalue)
    {
        switch (maskvalue)
        {
            case Motionplatform.FAULT2:

                BaseStatusValue = maskvalue;
                BaseStatus = "FAULT2";
                Debug.Log (BaseStatus);
                Machine.SetRESET ();
                break;
            case Motionplatform.ENGAGED:

                BaseStatusValue = maskvalue;
                BaseStatus = "ENGAGED";
                Debug.Log (BaseStatus);
                Machine.NEWMDA ();
                break;
            case Motionplatform.IDLE:
```

Esimerkkikoodi 10. Ote CheckState() -funktiosta Response-skriptissä

Response-skriptissä toteutettava bittimaskaus AND-operaattorilla selvittää, missä tilassa liikealusta on. AND-operaattorin merkkinä käytetään ”&”-merkkiä. Ennen maskausta bittijonon järjestys muunnetaan Little Endian -muotoon. Liikealustalta tuleva tilainformaatio voidaan kuvata bittijonona kuviossa 29 esitetyllä tavalla. AND-operaattorin avulla saadaan selville liikealustan tila. Koodissa asetettiin verrattavaksi heksadesimaaliarvoksi 0x000F, jotta pelkästään tilan kertovat neljä viimeistä bittiä huomioidaan. Operaattori vertaa liikealustalta tulleen bittijonon ja verrattavan jonon bittejä keskenään. Molempien bittien ollessa 1 asetetaan myös tulokseksi 1, muuten tulokseksi asetetaan 0 (IBM 2014). Tämän jälkeen koodissa verrataan, mitä tilaa tulosjono vastaa, kuten esimerkkikoodissa 10 nähdään.

Tilat

0000 = POWER UP
0001 = IDLE
0010 = STANDBY
0011 = ENGAGED
0111 = PARKING
1000 = FAULT1
1001 = FAULT2
1011 = DISABLED
1100 = INHIBITED

Bittijono	1001	0100	1000	0001
&	0000	0000	0000	1111
Tulos	0000	0000	0000	0001

IDLE-tila

Kuvio 29. Esimerkki liikealustan tilatietojen bittimaskauksesta AND-operaattorilla

6 LIIKEALUSTASOVELLUKSEN KÄYTTÄMINEN

6.1 Sovelluksen hallinta

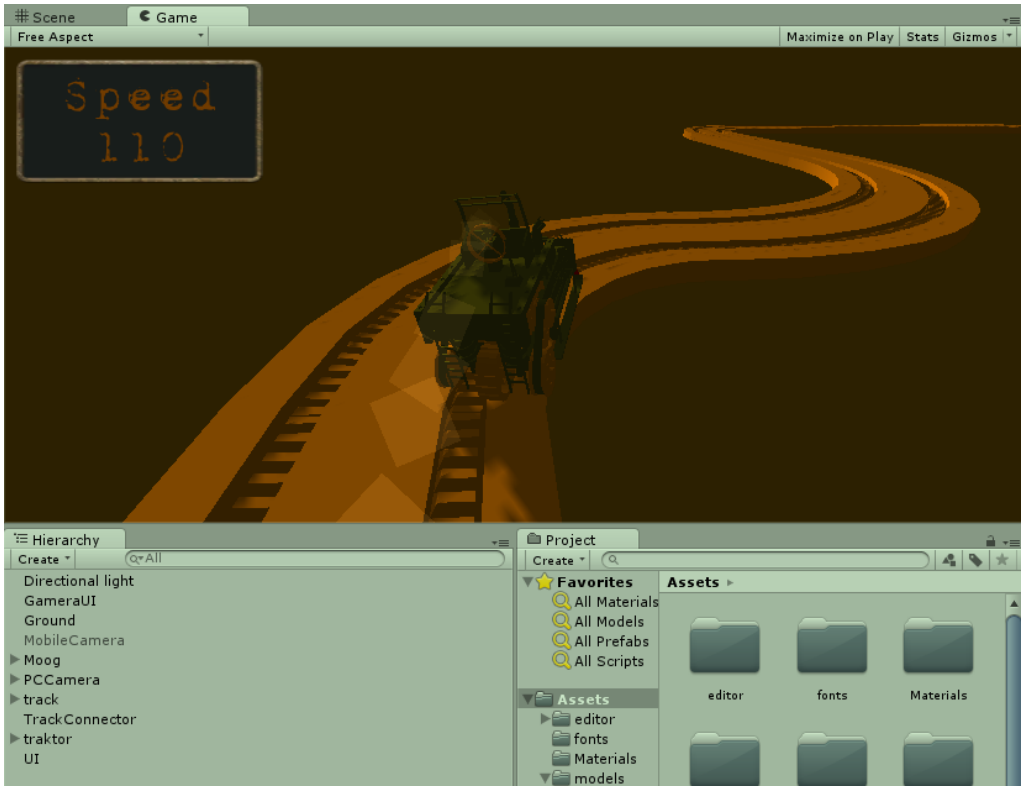
Ennen liikealustan käynnistämistä voimavirtakytkimestä, tulee liikealustan akut kytkeä päälle. Käynnistämisen jälkeen liikealustan tilaa voidaan seurata statusnäytöltä. Liikealustan käynnistyessä POWERUP-tilaan, voidaan käynnistää myös Unity 3D -sovellus. Sovelluksen kysyessä resoluutiota ja grafiikan laatua, tulee resoluutioksi valita Oculus Rift:n resoluutio eli 1024*800 ja grafiikan laaduksi *Fastest*. Käynnistyessään sovellus ottaa automaattisesti yhteyden liikealustaan ja liikealustan tila muuttuu IDLE-tilaan.

Liikealustan ENGAGE- ja PARK-komentoja voidaan hallinnoida näppäimistön tai Logitechin G27:n näppäinten kautta. Näppäimistön painikkeista ENGAGE-komennoksi on määritelty F2 ja PARK-komennoksi on määritelty F9. Liikealusta nousee aloitusasentoon ENGAGE-näppäinkomennon saatuaan. Aloitusasentoon nousu kestää n. 5 sekuntia. Kun ENGAGE-tila on aktiivinen, voidaan aloittaa ajaminen. Ohjelmaan toteutettiin kolme eri ajotilaa, jotka kertovat miten liikealustan Pitch-akseli luetaan peliobjektista. Näiden kolmen tilan avulla voidaan määrittää liikealustan Pitch-akseli seuraamaan peliobjektin asentoa, kiihtyvyyttä tai niitä molempia samanaikaisesti. Oletuksena seurataan sekä peliobjektin asentoa että kiihtyvyyttä, mutta tiloja voidaan muuttaa rattiin määritellyistä näppäimistä.

PARK-komennolla voidaan ajaminen lopettaa, jolloin liikealusta laskeutuu alas ja siirtyy takaisin IDLE-tilaan. Liikealusta on palautunut FAULT2-virhetilasta, mikäli liikealusta siirtyy kesken kaiken IDLE-tilaan ilman näppäinkomentoja. Virhetilaan joutuminen johtuu useimmiten yhteyden aikakatkaisusta.

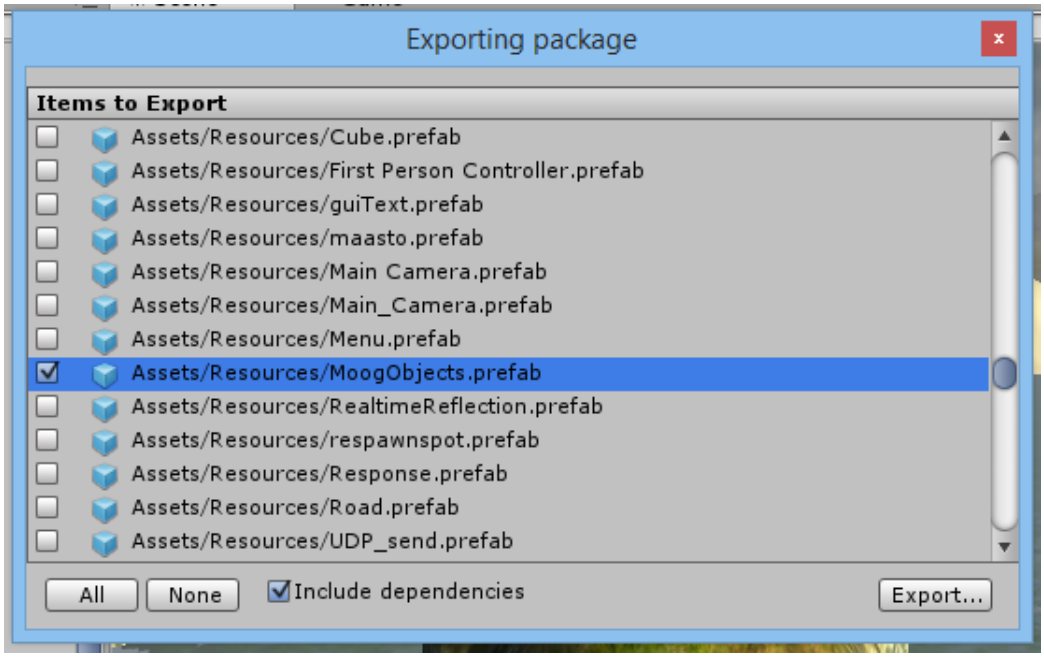
6.2 Toteutuksen integrointi

Integrointia testattiin liittämällä liikealustan liikuttamiseen tarvittavat peliobjektit uuteen Unity 3D -projektiin (Kuvio 30). Yhdistämällä liikealustakommunikointiin tarvittavat komponentit ja peliobjektit uuteen projektiin saatiin selvitettyä yksinkertainen integrointitapa. Toteutuksen integroinnin avulla pystyttiin myös optimoimaan ohjelmointikoodia.



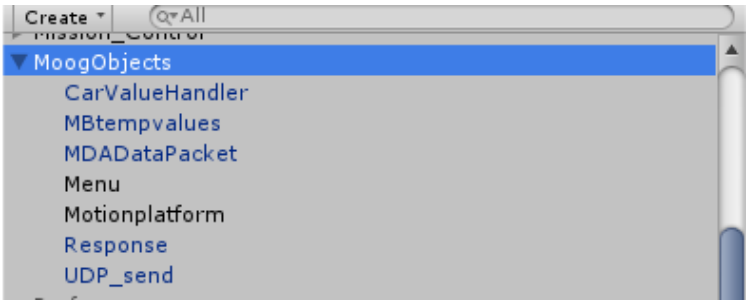
Kuvio 30. Kuvankaappaus toteutuksen liittämisestä uuteen projektiin

Unity 3D -pelimoottori mahdollistaa *unitypackage*-tiedostojen tekemisen, jotka ovat pakettitiedostoja. Pakettien avulla tarvittavien resurssitiedostojen siirtäminen uuteen Unity 3D -projektiin on helppoa. Paketin luominen tapahtuu kuvion 31 mukaisesti.



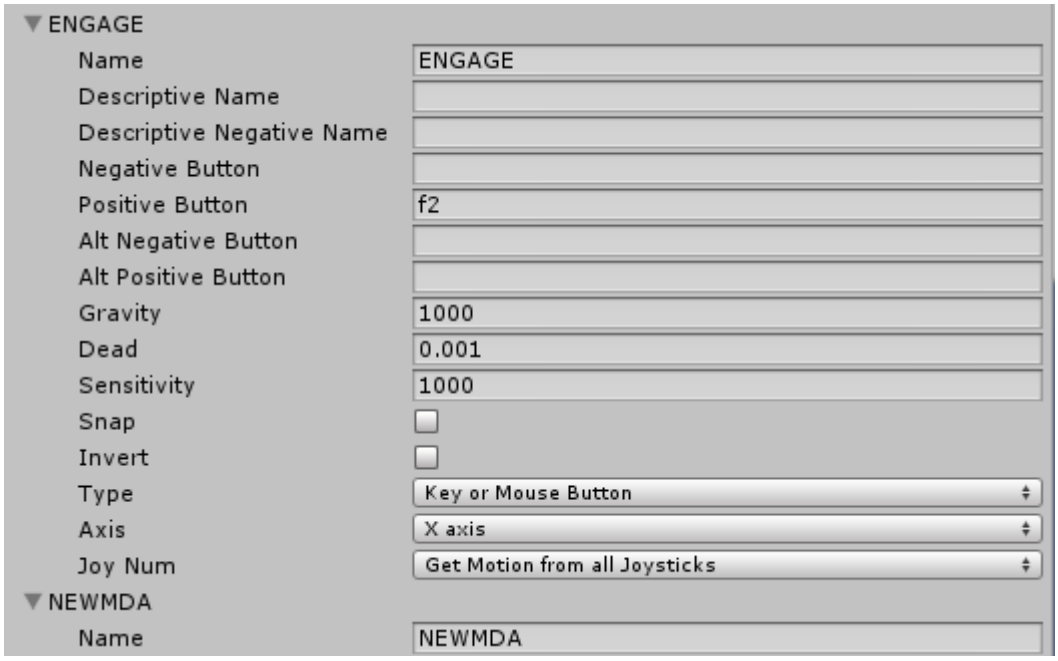
Kuvio 31. Kuvankaappaus unitypackage-tiedoston luomisesta

Testauksen avulla saatiin karsittua turhia koodinpätkiä pois ja yksinkertaistettua integrointitapahtumaa uuteen projektiin. Liikealustan liikuttamiseen vaadittava Unitypaketti vaatii yhteensä seitsemän skriptiä sekä smcs.rsp-tiedoston, joka vaaditaan bittijärjestysmuunnoksista vastaavan *unsafe*-tyyppisen funktion toimimiseen pelimoottorilla. Vaaditut skriptit liitetään skriptejä vastaaviin peliobjekteihin, jotka saadaan *unitypackage*-tiedostossa olevan yhden prefab-tiedoston kautta. Kuviossa 32 näkyvät vaadittavat peliobjektit, jotka ovat yhdistettynä yhden *MoogObjects*-nimisen objektin alle.



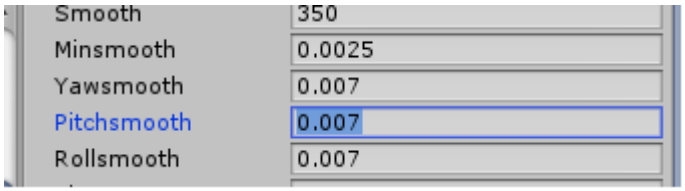
Kuvio 32. Kuvankaappaus integrointivaiheesta

Input Manager -asetuksiin lisättiin liikealustan hallitsemiseen vaadittavat näppäinkomennot, kuten ENGAGE JA PARK (Kuvio 33). Komennot voidaan määrittää halutuiksi näppäimiksi. G27:n ohjauspyörän kääntymisastetta voidaan hallinnoida Logitechin Gaming Software -ohjelman kautta. Kääntymisaste vaikuttaa siihen, kuinka herkkä peliobjektin kääntyminen on ohjauspyörää kääntäessä. Peliobjekti, jonka asentojen mukaisesti liikealustaa ohjataan, vaatii *HeadCar* -nimisen tagin. Tagin avulla tunnistetaan se peliobjekti, jota seurataan. Tagin lisääminen onnistuu inspectorin kautta.



Kuvio 33. Kuvankaappaus Input Manager -asetuksista integrointivaiheessa

CarValueHandler-skriptissä olevien julkisten muuttujien avulla liikettä voidaan hienosäätää uuden peliobjektin mukaiseksi (Kuvio 34). Julkisten muuttujien etuna on se, ettei koodiin tarvitse puuttua vaan niitä voidaan muokata suoraan editorin kautta myös ajon aikana. Muuttujien avulla hallitaan liikkeen tasaisuutta. Nämä muuttujat määräävät sen, millä arvoilla liikkeen pehmennys aloitetaan ja mikä on se kehyskohtainen arvo, jolla otetaan pelimaailman objektin asentoa kiinni, jotta liike on mieluista.



Kuvio 34. Kuvankaappaus liikkeen nopeuden julkisista muuttujista

7 YHTEENVETO

Työlle asetetut tavoitteet saavutettiin hyvin. Toteutetun ohjelmiston avulla liikealustaa voidaan ohjata suoraan Unity 3D -peliobjektin liikkeen ja asennon mukaisesti. Liikealusta reagoi esimerkiksi tiellä oleviin kuoppiin ja pintoihin kuten peliobjektikin. Lisäksi liikealusta reagoi peliobjektin kiihdytyksiin ja jarrutuksiin. Peliobjektin ohjaaminen tapahtuu ratin ja polkimien avulla. Liikealustan liikkeestä saatiin sulavaa, eikä äkkinäisiä liikkeitä tapahdu. Oculus Rift -virtuaalilasit mahdollistavat kokonaisvaltaisen ajosimulaatiokokemuksen.

Ohjelmisto toteutettiin siten, että se on myöhemmin mahdollisimman helppo liittää mihin tahansa Unity 3D -projektiin ja peliobjektiin. Liittämisen tuli olla helppoa, jotta ohjelmaa voidaan käyttää uusiin käyttötarkoituksiin. Siirto uuteen projektiin onnistuu helposti vain muutamien toimintojen suorittamisella. *CarValueHandler*-skriptin julkisten muuttujien avulla liikealustan liikettä voidaan hienosäätää, jotta liike on miellyttävää uuden projektin peliobjektille. Tätä hienosäätöä tarvitaan integroitaessa toteutus uuteen Unity 3D -projektiin, koska loppujen lopuksi liike riippuu aina peliobjektin omasta fysiikkamallinnuksesta. Miellyttävä lopputulos voidaan saavuttaa testaamalla liikealustan liikettä sekä testaamalla, millä arvoilla immersio on mieleisintä.

Toteutetun sovelluksen pohjalta aloitetaan ambulanssisimulaattorin kehittäminen. Ambulanssisimulaattorin yksityiskohtien sisältö on tällä hetkellä vielä suunnitteluvaiheessa. Simulaattoria varten tehdään oma virtuaaliympäristö, jossa hoitotilanteita voidaan harjoitella. Ajettavaksi peliobjektiksi mallinnetaan ambulanssi, jonka fysiikkamallinnus toteutetaan vastaamaan mahdollisimman hyvin reaalimaailman ambulanssia.

Henkilökohtaisesti työn tekeminen oli erittäin mielenkiintoista ja mukavaa. Unity 3D -pelimoottorin ominaisuudet ja soveltamismahdollisuudet ovat osoittautuneet laajemmiksi, kuin pelimoottoriin tutustuessa ajattelin. Oculus Rift:n liittäminen toteutukseen toi lisää mielenkiintoa. Virtuaalilasien voimakas läsnäolon tunne oli positiivinen yllätys. Liikealusta puolestaan tuottaa käyttäjälle liikkeen tunteen, joka uupuu virtuaalilasien käyttökokemuksesta. Lasien ja liikealustan yhteiskäytön avulla simulaatiokokemus paranee huomattavasti.

Oculus Rift -virtuaalilasien HD-version ilmestyttyä niillä voidaan parantaa käyttökokemusta varsinkin läsnäolon tunnetta. Oculus Rift:n lisäksi markkinoille on tulossa myös muiden valmistajien 3D-virtuaalilaseja, jotka voivat myös tarjota paremman käyttökokemuksen. Ohjaustuntuman parantamiseen liittyen Logitechin G27-ohjauspyörä mahdollistaa tärinäpalautusmekanismin käytön peleissä. Mekanismin avulla pystytään jäljittelemään muun muassa pidon häviämistä ja epätasaisella pinnalla ajamista. Tärinäpalautusmekanismi ei kuitenkaan suoraan toimi Unity 3D -pelimoottorilla eikä valmista liitännäistä ole tällä hetkellä saatavilla Unity 3D:n uusimmille versioille. Unity 3D -liitännäisen toteuttaminen parantaisi ajamisen tunnetta. Logitechin G27-ohjain sisältää vaihdekepin, joten myös manuaalivaihteiston mallintaminen simulaattorin autopeliobjektiin on mahdollista tulevaisuudessa. Itse ohjelmistoa voidaan kokonaisuudessaan soveltaa täysin uusiin liikealustan simulaatiokäyttötarkoituksiin.

LÄHTEET

H3C 2014. QoS Introduction. Osoitteessa http://www.h3c.com/portal/Products___Solutions/Technology/QoS/Technology_Introduction/200701/195599_57_0.htm. 12.3.2014.

HyperPhysics 2014. Centripetal Force. Osoitteessa http://hyperphysics.phy-astr.gsu.edu/hbase/corf.html. 28.2.2014.

IBM 2014. Bitwise AND operator &. Osoitteessa https://publib.boulder.ibm.com/infocenter/comphelp/v8v101/index.jsp?topic=%2Fcom.ibm.xlcpp8a.doc%2Flanguage%2Fref%2Fbitande.htm. 2.4.2014.

Internetix 2014. TCP- ja UDP-protokollat. Osoitteessa http://oppimateriaalit.internetix.fi/fi/avoimet/6tekniikkatalous/verkko/tcp_ja_udp_protokollat. 12.3.2014.

Logitech 2014. G27 Racing Wheel. Osoitteessa http://gaming.logitech.com/fi-fi/product/g27-racing-wheel. 5.3.2014.

McMahan, A 2003. Immersion, Engagement and Presence. Saatavilla http://people.ict.usc.edu/~morie/SupplementalReadings/ch3-McMahanrev.pdf. 24.1.2014.

Moog 2014. Moog. Osoitteessa: http://www.moog.com/. 4.3.2014.

Moog Motion Systems Division. 2000a. MOOG 6DOF2000E Motion system Interface definition Manual.

- 2000b. MOOG 6DOF2000E Motion system User's Manual.

Moog Systems Group 2013. Series 6DOF2000E -Esite. Osoitteessa http://www.hidrapa.com.br/Documentos/Moog/6DOF2000E.pdf. 21.12.2013

Oculus VR 2014a. About Oculus VR. Osoitteessa http://www.oculusvr.com/company/. 31.1.2014.

- 2014b. Announcing the Oculus Rift Development Kit 2. Osoitteessa http://www.oculusvr.com/blog/announcing-the-oculus-rift-development-kit-2-dk2/. 21.3.2014.

- 2014c. Next-Gen Virtual Reality. Osoitteessa http://www.oculusvr.com/rift/. 31.1.2014.

Piiroinen, M. 2008. Opinnäytetyö. Ajosimulaation kehitys liikealustalle.

pLAB 2013. pLAB – Ohjelmistotekniikan laboratorio. Osoitteessa http://plab.ramk.fi/fi/etusivu/. 14.11.2013.

RAMK 2013. ENVI - Hyvinvointialojen virtuaalikeskus. Osoitteessa http://www.ramk.fi/fi/Opiskelijalle/Oppimisymparistoja/ENVI---Hyvinvointialojen-virtuaalikeskus. 14.11.2013.

Scratchapixel 2014. Linear smoothstep curve. Osoitteessa http://www.scratchapixel.com/assets/Uploads/Noise%20Part%201/linear-smoothstep-curve.png. 22.1.2014.

Unity 3D 2013a. Company. Osoitteessa http://unity3d.com/company/public-relations/. 16.11.2013.

- 2013b. Constant Force. Osoitteessa http://docs.unity3d.com/Documentation/Components/class-ConstantForce.html. 29.12.2013.

- 2013c. Integrated editor. Osoitteessa http://unity3d.com/unity/workflow/integrated-editor. 29.12.2013.

- 2013d. License Comparison. Osoitteessa http://unity3d.com/unity/licenses#. 29.12.2013.

- 2013e. Multiplatform. Osoitteessa http://unity3d.com/unity/multiplatform/. 16.11.2013.

- 2013f. Store. Osoitteessa https://store.unity3d.com. 29.12.2013.

- 2013g. Unity Scripting. Osoitteessa http://unity3d.com/unity/workflow/scripting/. 16.11.2013.

- 2013h. Unity Physics. Osoitteessa http://unity3d.com/unity/quality/physics. 29.12.2013.

2013i. Using DirectX 11 in Unity 4. Osoitteessa http://docs.unity3d.com/Documentation/Manual/DirectX11.html. 29.12.2013.

Unity 3D 2014a. Creating Scenes. Osoitteessa http://docs.unity3d.com/Documentation/Manual/CreatingScenes.html. 5.1.2014.

- 2014b. Mathf.SmoothDamp. Osoitteessa http://docs.unity3d.com/Documentation/ScriptReference/Mathf.SmoothDamp.html. 23.1.2014.

- 2014c. Mesh Filter. Osoitteessa http://docs.unity3d.com/Documentation/Components/class-MeshFilter.html. 5.1.2014.

- 2014d. MonoBehaviour. Osoitteessa http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html. 5.1.2014.

- 2014e. PhysicMaterial. Osoitteessa http://docs.unity3d.com/Documentation/ScriptReference/PhysicMaterial.html. 22.3.2014.

University of Central Florida 2013. What Is M&S? Osoitteessa: http://www.ist.ucf.edu/background.htm. 5.2.2013.

Wikipedia 2014. Endiannes. Osoitteessa http://en.wikipedia.org/wiki/Endianness. 17.1.2014.

WireShark 2014. User's Guide. Osoitteessa http://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html#ChIntroWhatIs. 17.1.2014.

LIITTEET

MDA-struktuuri Liite 1
MotionPlatform.cs Liite 2
Käyttökokemus -lomake Liite 3

MDA-struktuuri Liite 1

```
public struct MDAPacket
{
            //##ModelId=3F02B9B201E5
            public uint mcw;

            //##ModelId=3F02B9B201F4
            public float a_roll;

            //##ModelId=3F02B9B201F5
            public float a_pitch;

            //##ModelId=3F02B9B20203
            public float a_z;

            //##ModelId=3F02B9B20204
            public float a_x;

            //##ModelId=3F02B9B20213
            public float a_yaw;

            //##ModelId=3F02B9B20214
            public float a_y;

            //##ModelId=3F02B9B20222
            public float v_roll;

            //##ModelId=3F02B9B20223
            public float v_pitch;

            //##ModelId=3F02B9B20232
            public float v_yaw;

            //##ModelId=3F02B9B20233
            public float roll;

            //##ModelId=3F02B9B20242
            public float pitch;

            //##ModelId=3F02B9B20251
            public float yaw;

            //##ModelId=3F02B9B20252
            public float buffet_roll;

            //##ModelId=3F02B9B20261
            public float buffet_pitch;

            //##ModelId=3F02B9B20262
            public float buffet_z;

            //##ModelId=3F02B9B20271
            public float buffet_x;

            //##ModelId=3F02B9B20272
            public float buffet_yaw;

            //##ModelId=3F02B9B20280
            public float buffet_y;

            //##ModelId=3F02B9B20281
            public float v_vehicle;

            //##ModelId=3F02B9B20290
            public uint spare1;

            //##ModelId=3F02B9B20291
            public uint spare2;
}
```

MotionPlatform.cs Liite 2

```
// Skripti hoitaa bittijärjestyksen muunnokset ja asettaa arvot MDADatapacket-struct:iin
using System.Collections;
using System.ComponentModel;
using System.Globalization;
using System.Reflection;
using System;
public class Motionplatform : MonoBehaviour {
public const uint ESTOP_CMD  =     0xE6;

/******* LIIKEALUSTAKOMENNOT  ******/

public const uint DISABLE                    =   0xDC;
public const uint PARK                       =   0xD2;
public const uint LOW_LIMIT_ENABLE           =   0xC8;
public const uint LOW_LIMIT_DISABLE          =   0xBE;
public const uint ENGAGE                     =   0xB4;
public const uint START                      =   0xAF;
public const uint LENGTH_MODE                =   0xAC;
public const uint DOF_MODE                   =   0xAA;
public const uint RESET                      =   0xA0;
public const uint INHIBIT                    =    0x96;
public const uint NEW_POSITION               =    0x82;
public const uint MDA_MODE                   =   0x8c;
public const uint NEW_MDA                    =   0x80;

/* Liikealustan vastauspaketin tila-arvot */

public const uint POWER_UP                   =0x0000;
public const uint IDLE                       =0x0001;
public const uint STANDBY                    =0x0002;
public const uint ENGAGED                    =0x0003;
public const uint PARKING                    =0x0007;
public const uint FAULT1                     =0x0008;
public const uint FAULT2                     =0x0009;
public const uint FAULT3                     =0x000A;
public const uint DISABLED                   =0x000B;
public const uint INHIBITED                  =0x000C;
public const uint PARKED                     =0x000D;

public MDAPacket alusta;
public menu m_menu;
public MBClientDefinition MBDatadef;
public MDADataPacket MDAData;
public MBtempvalues temp;
public UDP_send client;
public bool runmoog=false;

// Haetaan tarvittavat komponentit ja alustetaan MDADatapacketin arvot
void Start ()
{
   client = GameObject.Find ("UDP_send").GetComponent<UDP_send> ();
   m_menu = GameObject.Find ("Menu").GetComponent<menu>();
   MDAData = GameObject.Find ("MDADataPacket")
   .GetComponent<MDADataPacket> ();
   temp = GameObject.Find ("MBtempvalues").GetComponent<MBtempvalues> ();
   NullEverything ();
}

// Nollataan MDA-paketin arvot
```

```
public void NullEverything()
{
   alusta.pitch = 0.0f;
   alusta.mcw = NEW_MDA;
   alusta.a_roll = 0.0f;
   alusta.a_pitch = 0.0f;
   alusta.a_z = 0.0f;
   alusta.a_x = 0.0f;
   alusta.a_yaw = 0.0f;
   alusta.a_y = 0.0f;
   alusta.v_roll =0.0f;
   alusta.v_pitch = 0.0f;
   alusta.v_yaw = 0.0f;
   alusta.roll = 0.0f;
   alusta.yaw = 0.0f;
   alusta.buffet_roll =0.0f;
   alusta.buffet_pitch = 0.0f;
   alusta.buffet_z = 0.0f;
   alusta.buffet_x = 0.0f;
   alusta.buffet_yaw = 0.0f;
   alusta.buffet_y = 0.0f;
   alusta.v_vehicle = 0.0f;
   alusta.spare1 = 0;
   alusta.spare2 = 0;
   temp.nulltempvalues ();
}

// Kertoo onko liikealustan komentosanaksi asetettu ENGAGE
public bool IsStateEngaged()
{

   if(alusta.mcw == SwapBytes(ENGAGE))
   {
      return true;
   }
   else
   {
      return false;
   }

}

// Kertoo onko liikealustan komentosanaksi asetettu NEWMDA
public bool IsStateNEWMDA()
{

   if(alusta.mcw == SwapBytes(NEW_MDA))
   {
      return true;
   }
   else
   {
      return false;
   }

}

// Bittijärjestyksen muunnos funktiot
public uint SwapBytes(uint x)
{
   return ((x & 0x000000ff) << 24) +
    ((x & 0x0000ff00) << 8) +
    ((x & 0x00ff0000) >> 8) +
```

```
    ((x & 0xff000000) >> 24);
}
public static uint ToUInt32(byte[] value, int index)
{
   return (uint)(
   value[0 + index] << 0 |
   value[1 + index] << 8 |
   value[2 + index] << 16 |
   value[3 + index] << 24);
}

public static unsafe float ToSingle(byte[] value, int index)
{
   uint i = ToUInt32(value, index);
   return *(((float*)&i));
}

public static float swap(float input)
{
   var floatArray1 = new float[] { input};
   byte[] floats = new byte[floatArray1.Length * 4];
   Buffer.BlockCopy(floatArray1, 0, floats, 0, floats.Length);
   byte[] tmpOut = new byte[4];
   tmpOut[0] = floats[3];
   tmpOut[1] = floats[2];
   tmpOut[2] = floats[1];
   tmpOut[3] = floats[0];
   return ToSingle(tmpOut,0);
}
// Bittijärjestyksen muunnos funktiot päättyvät

// Wait()-funktio asettaa mcw:ksi PARK-arvon
IEnumerator wait()
 {
   alusta.Setmcw = SwapBytes(PARK);
   yield return new WaitForSeconds(5);
   client.cancelInvokes();
}

// MCW:n asettamiseen tarkoitetut funktiot
public void NEWMDA()
{
   alusta.Setmcw = SwapBytes (NEW_MDA);
}

public void SetMDA_MODE()
{
   alusta.Setmcw = SwapBytes(MDA_MODE);;
}

public void SetENGAGED()
{
   alusta.Setmcw = SwapBytes(ENGAGE);
}

public void SetPARK()
{
   NullEverything();
   temp.nulltempvalues ();
   StartCoroutine(wait());
   runmoog=false;
}
```

```
public void SetSTART()
{
   alusta.Setmcw = SwapBytes(START);
}

public void SetRESET()
{
   alusta.Setmcw = SwapBytes(RESET);
   temp.nulltempvalues ();
   NullEverything();
}
// MCW:n asettamiseen tarkoitetut funktiot päättyivät


// Asettaa MDApacket-oliolle bittimuunnetut arvot
public void SetValueToMB(string property, float radian)
{
   switch(property)
   {
      case "Set_aroll":
      temp.a_roll_temp = radian;
      alusta.Set_aroll= swap(temp.a_roll_temp);
      break;

      case "SetPitchAcceleration":
      temp.a_pitch_temp = radian;
      alusta.SetPitchAcceleration =swap(temp.a_pitch_temp);
      break;

      case "Seta_z":
      temp.a_z_temp = radian;
      alusta.Seta_z= swap(temp.a_z_temp);
      break;

      case "Seta_x":
      temp.a_x_temp = radian;
      alusta.Seta_x= swap(temp.a_x_temp);
      break;

      case "Seta_y":
      temp.a_y_temp = radian;
      alusta.Seta_y= swap(temp.a_y_temp);
      break;

      case "Seta_yaw":
      temp.a_yaw_temp = radian;
      alusta.Seta_yaw= swap(temp.a_yaw_temp);
      break;

      case "Seta_v_roll":
      temp.v_roll_temp = radian;
      alusta.Seta_v_roll= swap(temp.v_roll_temp);
      break;

      case "Seta_v_pitch":
      temp.v_pitch_temp = radian;
      alusta.Seta_v_pitch= swap(temp.v_pitch_temp);
      break;

      case "Seta_v_yaw":
      temp.v_yaw_temp = radian;
      alusta.Seta_v_yaw= swap(temp.v_yaw_temp);
      break;
```

```
      case "SetRoll":
      temp.roll_temp = radian;
      alusta.SetRoll= swap(temp.roll_temp);
      break;

      case "SetPitchAngle":
      temp.pitch_temp = radian;
      alusta.SetPitchAngle= swap(temp.pitch_temp);
      break;

      case "SetYaw":
      temp.yaw_temp = radian;
      alusta.SetYaw= swap(temp.yaw_temp);
      break;

      case "Setbuffet_roll":
      temp.buffet_roll_temp= radian;
      alusta.Setbuffet_roll= swap(temp.buffet_roll_temp);
      break;

      case "Setbuffet_pitch":
      temp.buffet_pitch_temp= radian;
      alusta.Setbuffet_pitch= swap(temp.buffet_pitch_temp);
      break;

      case "Setbuffet_z":
      temp.buffet_z_temp= radian;
      alusta.Setbuffet_z= swap(temp.buffet_z_temp);
      break;

      case "Setbuffet_y":
      temp.buffet_y_temp= radian;
      alusta.Setbuffet_y= swap(temp.buffet_y_temp);
      break;

      case "Setbuffet_x":
      temp.buffet_x_temp= radian;
      alusta.Setbuffet_x= swap(temp.buffet_x_temp);
      break;

      case "Setbuffet_yaw":
      temp.buffet_yaw_temp= radian;
      alusta.Setbuffet_yaw= swap(temp.buffet_yaw_temp);
      break;

      case "Setv_vehicle":
      temp.v_vehicle_temp= radian;
      alusta.Setv_vehicle= swap(temp.v_vehicle_temp);
      break;

      case "Setspare1":
      break;

      case "Setspare2":
      break;
    }

  }
  }
```

Liikealustatestaus Liite 3

Missä on parannettavaa? Kerro huomiot liikkeiden puutteista.

Tasaisella ajaminen:

Kääntyminen:

Sivuttaisliike:

Kiihtyvyyden tunne:

Törmäys:

Ilmalennot:

Ala-ja ylämäet:

Liikkeen nopeus/hitaus:

Auton ohjattavuus:

Muu: