Automaation tietotekniikka labra 2

Oppimistavoitteet

- 3D maailman koordinaatiston havainnollistaminen ja visuaalinen hahmottaminen yleisesti ja erityisesti robotiikan tarpeisiin
- yksinkertaisen karteesisen robotin ohjelmointi tyhjästä 3D ympäristössä
- robotin reitinsuunnittelu (APP – Assembly Path Planning) yksinkertaisen esimerkin kautta

Tehtävänanto

Kommentoi Main.javasta se osuus joka luo lego instanssit ja liittää ne rootNodeen.

Java vs Python
Javassa kommentoidaan rivi laittamalla sen alkuun `//` Useampi rivi kommentoidaan laitamalla alkuun `/*` ja loppuun `*/`

Tässä harjoituksessa tehdään kokoonpanoasema joku koostuu työtasosta ja sen päällä olevasta robotista. Tehdään aluksi työtaso. Mutta ensin pitää päättää että missä kohtaa meidän 3D maailmaa on tuotantosolun lattia. Lisätään Main luokkaan seuraava kenttä

```
public static float floorHeight = -15;
```

Static tarkoittaa että sen arvo on sama kaikille luokan olioille. Näin ollen sen käyttö jostain muusta luokasta käsin ei vaadi `<olion nimi>.<attribuuttin nimi>` syntaksia vaan `<luokan nimi>.<attribuutin nimi>`

Opetuksellinen lähestymistapa
Kun uusia Java tekniikoita tulee ensimmäisen kerran, annetaan aika tarkat keittokirjaohjeet. Jatkossa kun niitä pitää soveltaa uusiin tehtäviin, niin ohjeet vaan kertoo mitä pitää tehdä. Esim seuraava osuus ei toista harjoituksen osan 1 yksityiskohtaisia ohjeita. PC-luokan viikoittaisissa harjoituksissa opettaja on käytettävissäsi, eli jos aiemman soveltaminen on hankalaa tai jos koet että ohjeistus on jossain kohdin suppeaa, kysy pois niin katsotaan yhdessä!

Luo sitten AssemblyStation luokka samalla tavalla kuin loit Lego luokan. Tee sille seuraavanlainen konstruktori:

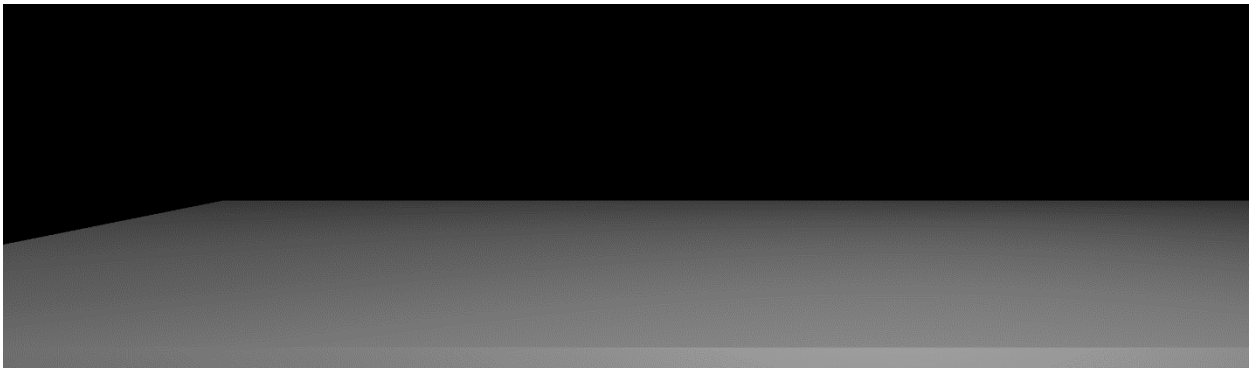
```
public AssemblyStation(AssetManager assetManager, Node rootNode, float xOffset, float zOffset) {

}
```

Konstruktorin parametrien xOffset ja zOffset avulla voidaan vaikuttaa työtason paikkaan 3D maailmassa. Luo ensin samanlainen laatikko kuin mitä teit Legolle ja anna väriksi `ColorRGBA.LightGray`. Laita boxin kooksi (20, yExtent, 10). Määrittele muuttuja yExtent ja anna sille arvoksi 6. Muuttujan käyttö on tarpeen, jotta voit myöhemmin laskea tason pinnan korkeuden `= floorHeight + 2 * yExtent`. Kun olet luonut geometrian ja liittänyt boxin siihen niin vaihda sen paikka:

```
geom.setLocalTranslation(xOffset, Main.floorHeight + yExtent, zOffset);
```

(y-koordinaatti kertoo boxin keskiosan sijainnin, joten jotta boxin alaosa saataisiin lattian tasolle niin pitää lisätä yExtent, mikä on alareunan ja keskipisteen välinen etäisyys y-suunnassa.) Main luokassa luo instanssi AssemblyStationista siten että xOffset=5 ja zOffset-11. Kun käynnistät sovelluksen niin pitäisi tulla seuraava näkymä:



Seuraavaksi luodaan karteesinen robotti, eli luo RobotArm luokka. Laita sille Node tyyppinen attribuutti node ja liitä se rootNodeen.

Karteesinen robotti soveltuu ’pick-and-place’ tehtäviin, joissa kappaleisiin pääsee kiinni yläkautta ja joissa ei ole tarpeen tehdä kiertoliikkeitä. Esimerkiksi monet pakkaus ja elektroniikkakokoonpano tehtävät ovat tällaisia. Tässä harjoituksessa ei huomioida sellaisia reaalimaailman asioita kuten kiihtyvyyksiä tai säätöjä, eli robotin eri osia ajetaan vakionopeudella, kunnes haluttu paikka on saavutettu. Robotissa on seuraavat osat:

- **masto**: pystysuora liikkumaton tanko, joka on kiinni työtasossa
- **käsivarret xArm, yArm, zArm** joista kukin on kyseisen akselin suuntainen suora neliskanttinen tanko. zArm on kiinni mastossa, xArm zArmissa ja yArm xArmissa.
- **työkalu (tooltip)** joka on kiinni yArm alaosassa. Tässä harjoituksessa työkalu on imukuppi, mutta se on havainnollistettu boxilla, joka on vähän kapeampi kuin robotin muut osat.

Luo kaikki allamainitut osa käyttäen box luokkaa ja liitä ne geometria olioihin. Käytä materiaalin värinä ColorRGBA.Orange.

Kaikki yksiköt ovat ”world unit”. Jos on tarpeen käyttää fyysisen maailman yksiköitä niin käyttäjä voi päättää skaalauksen. Yleinen ratkaisu on world unit = 1m.

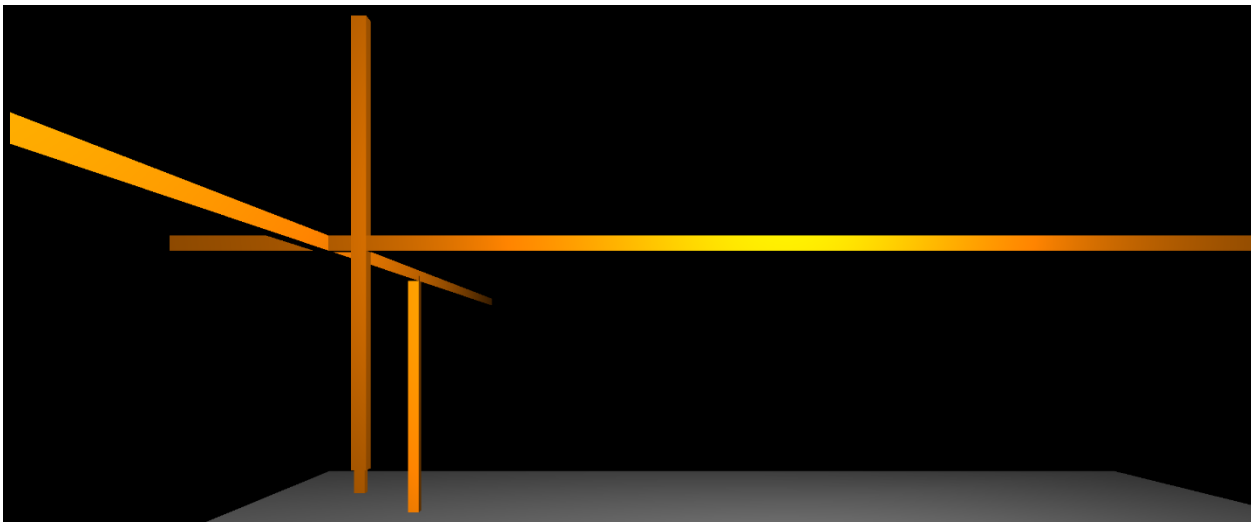
	Mitat			Paikka (Local translation)		
	X	Y	Z	X	Y	Z
mast	0.2	6	0.2	-8	0	-10
zArm	0.2	0.2	20	-8	6	-8
xArm	18	0.2	0.2	6	6	0
yArm	0.2	6	0.2	-7	6	0
tooltip	0.14	0.4	0.14	Kts alla		

Laita kaikkien osien geometria oliohin taulukon mukaiset paikat `setLocalTranslation()` metodilla, poikkeuksena tooltip. Luo sille oma `Node` tooltipNode ja se lapseksi robotin noodiin. Sitten liitä tooltipin geometria tooltipNodeen. Tämä siksi että robotin kyydissä oleva kappale voidaan jatkossa liittää tähän noodiin, jolloin se liikkuu automaattisesti robotin tooltipin mukana. Aiemmin luotu robotin noodi ei tule liikkumaan minnekään, kun robotti työskentelee.

Lopuksi halutaan laittaa tooltipin noodin localTranslation sen alapintaan. Jos laitetaan tooltipNode yArmin koordinaatteihin, sen keskipiste on yArmin keskipisteessä, jolloin tooltip ei näy mikäli käynnistäisit ohjelman tässä vaiheessa. Taulukosta käy ilmi, että etäisyys yArm keskipisteestä alareunaan on 6, joten nodeTooltip pitää siirtää alas y-akselilla tämän verran. Nyt tooltip keskipiste on yArm alapinnan tasolla, joten pitää vielä siirtää alas 0.4 jotta tooltip yläpinta on yArm alapinnassa kiinni.

HUOM: käytä tooltip **noodin** `setLocalTranslation()`. Jos siirrät noodin sijasta geometriaa niin saat kyllä alla olevan kuvan mutta myöhemmin tulee ongelmia kun liitetään legoja tooltipin noodiin.

Kun tooltipNodea siirrettiin, niin tooltipin geometria liikkui automaattisesti sen mukana. Kun käynnistät ohjelman ja liikut vähän taakse ja ylös, pitäisi tulla seuraavan kaltainen näkymä:



Teoria

Kokoonpanorobotin tehtävä koostuu sekvenssistä, joka määrittelee missä järjestyksessä osat käsitellään. Tämän sekvenssin suunnittelu tunnetaan lyhenteellä ASP (Assembly Sequence Planning). Yksittäisen kappaleen käsittelyä varten pitää suunnitella reitti, jotta törmäyksiä ei tapahdu ja jotta

kappale voidaan nostaa varastopaikasta ja liittää kokoonpanoon oikeassa kulmassa (meidän robotin tapauksessa pystysuoraan). Reitinsuunnittelu tunnetaan lyhenteellä APP (Assembly Path Planning). Yleinen tapa suunnitella reitti on määritellä välietappeja (waypoints), joiden kautta robotin tulee liikkua, siten että viimeinen piste on paikka minne halutaan mennä.

Reittiä varten luo Trajectory luokka. Laitetaan välietappien 3D koordinaatit ArrayList tietorakenteeseen.

Java vs Python
Pythonissa on List tietorakenne, minne voi laittaa kaikentyyppistä sisältöä. Javassa samankaltainen tietorakenne on ArrayList<> mutta kulmasulkujen väliin pitää laittaa tietotyyppi, jolloin listaan saa laittaa vain tämän tyyppistä sisältöä. Trajectory luokan tapauksessa tyyppi on Vector3f. Tietorakenteen käyttö edellyttää: `import java.util.ArrayList;`

Trajectory luokan sisältö on seuraavanlainen:

```
ArrayList<Vector3f> points;
int index; // 'points' listan indeksi
int size; // kuinka monta waypointtia 'points' listassa on

// alustaa yllämainitut points ja index muuttujat
public Trajectory() {
}

// lisää pisteen listan hännille
public void addPoint(Vector3f v) {
}

// nollaa indeksin ja asettaa size muuttujalle oikean arvon
public void initTrajectory() {
}

// palauttaa indexin kohdalla olevan pisteen tai null jos ei enää pisteitä
public Vector3f nextPoint() {
}
```

Kirjoita metodien koodi ja hyödynnä ArrayList dokumentaatiota:

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Malliratkaisussamme koodirivien määrä on 1-5 metodia kohden.

APP tehdään AssemblyStation luokassa, joten robotissa pitää olla toiminnallisuus, jolla se kykenee ajamaan annettuun välietappiin. 3D Java toimii kuten PLC ja monet muut automaatiosovellukset siinä, että Main luokassa on metodi, jonka sisältö suoritetaan syklisesti monta kertaa sekunnissa. jMonkeyn tapauksessa kyseinen metodi on `Main.simpleUpdate()` ja sen suoritussykli on niin nopea kuin mitä koneessa on vääntöä.

Sitten päivitetään RobotArm luokka. Lisää seuraavat attribuutit:

```
private Vector3f targetLocation; // välietappi
float step = 0.1f; // etäisyys akselia kohden mikä liikutaan yhden syklin aikana
```

ja seuraavat metodit (täydennä ??? kohdat).

HUOM: koodi olettaa että käytit joitain muuttujan nimiä kuten 'zArmGeom', jolloin tulee virheilmoituksia. Luultavasti käytit muita nimiä, joten pitää päätellä mikä muuttuja on kysymyksessä ja päivittää koodi.

```
// target on välietappi johon kuuluu ajaa
public void initMove(Vector3f target) {
    targetLocation = target;
}


// palauttaa tooltipin alapinnan keskipisteen koordinaatit maailma-koordinaateissa
// käytä Geometry luokan getWorldTranslation()
public Vector3f getToolTipLocation() {
    ???
}


// moves towards target location and returns false when it reached the location
public boolean move() {
    Vector3f location = getToolTipLocation();

    // lasketaan etäisyys määränpäähän maailma-koordinaateissa
    float xDistance = targetLocation.getX() - location.getX();
    float zDistance = ???
    float yDistance = ???

    // booleanit ilmaisee että onko kyseisen akselin suuntainen liike valmis
    boolean xReady = false;
    boolean yReady = false;
    boolean zReady = false;

    float x; // x-akselin suuntainen liike tämän syklin aikana
    float y; // y-akselin suuntainen liike tämän syklin aikana
    float z; // z-akselin suuntainen liike tämän syklin aikana

    // siirrytään stepin verran oikeaan suuntaan jos matkaa on yli stepin verran
    // muuten siirrytään targetLocationin x koordinaattiit
    if (xDistance > step ) {
        x = step;
    } else if ((-1 * xDistance) > step) {
        x = -1 * step;
    } else {
        xReady = true;
        x = xDistance;
    }

    if (zDistance > step ) {
        ???
    }

    if (yDistance > step ) {
         ???
    }
```

```
    // siirretään mastossa kiinni oleva zArm, joka liikkuu siis z-suuntaan
    // 0.5f siitä syystä että robotti ulottuu paremmin (xArm liikuu zArmia pitkin)
    Vector3f v = new Vector3f(0, 0, 0.5f*z);
    zArmGeom.setLocalTranslation(zArmGeom.getLocalTranslation().add(v));

    // xArm on zArmin varassa minkä lisäksi se liikkuu sitä pitkin, joten nyt
    // käytetä 0.5f kerrointa kuten äsken
    Vector3f v1 = new Vector3f(0, 0, z);
    xArmGeom.setLocalTranslation(xArmGeom.getLocalTranslation().add(v1));

    // yArm liikkuu xArm Pitkin x suuntaan ja tekee myös y-suuntaisen liikkeen,
    // minkä lisäksi zArmin liike siirtää myös yArmia
    Vector3f v2 = new Vector3f(x, y, z);
    yArmGeom.setLocalTranslation(yArmGeom.getLocalTranslation().add(v2));

    // nodetoolTip paikaksi on määritelty yArm alapinta, mutta nodetoolTipin parent
    // noodi ei liiku, joten nodetoolTip pitää siirtää kuten yArm
    // samalla liikkuu nodetoolTippiin liitetty tooltipin geometria
    nodetoolTip.setLocalTranslation(nodetoolTip.getLocalTranslation().add(???));

    if((yReady && xReady) && zReady) {
        return false; //i.e. not moving anymore
    } else {
        return true;
    }
}
```

Sitten AssemblyStationiin:

```
float maxHeight = 4; // max korkeus reitin välietapeille
boolean moving = false; // true jos matkalla seuraavaan välietappiin
Trajectory trajectory;

// tehdään APP eli reitinsuunnittelu destination koordinaatteihin
public void initTestMove(Vector3f destination) {
    trajectory = new Trajectory();

    // eka välietappi suoraan ylös max korkeuteen
    Vector3f v1 = assemblyArm.getToolTipLocation();
    v1.setY(maxHeight);
    trajectory.addPoint(v1);

    // toka välietappi max korkeuteen destination ylle
    ???

    trajectory.addPoint(destination);
    trajectory.initTrajectory();
}
```

```
// käskyttää robottia ajamaan reitin, joka on määritelty trajectory-attribuuttiin
// palauttaa false jos saavutettiin trajectory viimeinen (väli)etappi, eli
// initTestMove() saama destination. Muuten palauttaa true.
// tätä tulee kutsua syklisesti kunnes se palauttaa false
public boolean move() {
    if (moving) {
        moving = assemblyArm.move();
        return true;
    } else {
        // tänne tullaan jos edellinen välietappi saavutettiin
        Vector3f nextPoint = trajectory.nextPoint();
        if (nextPoint == null) {
            ???
        } else {
            // debug printit tulee konsoliin näkyviin kun suljet ohjelman
            System.out.println(nextPoint.toString());
            // annetaan robotille seuraava välietappi ja alustetaan moving seuraavaa
            // move() kutsua silmälläpitäen
            ???
        }
    }
}
```

Lopuksi jonnekin pitää panna tuotantosolun tasoinen ohjauslogiikka. Koska tässä harjoituksessa logiikka on lyhyt ja tuotantosolussa on vähänlaisesti tuotantoresursseja, laitetaan tämä Main luokkaan. Lisää aluksi `simpleInitApp()` loppuun:

```
station.initTestMove(new Vector3f(0,0,-5));
```

Voit kokeilla eri koordinaateilla sitten kun tämä toimii. Lopuksi laitetaan enää syklinenkutsu AssemblyStationin moveen:

```
public void simpleUpdate(float tpf) {
    station.move();
}
```

Kun ajat ohjelman, pitäisi näkyä että robotti lähtee suoraan ylös, liikkuu etuviistoon oikealle ja menee suoraan alas niin että toolTip on ruudun keskiosassa mikäli et liikuttanut hiirtä tai liikuttanut kameran paikkaa ’qwasdz’ näppäimillä. Konsoliin tulee seuraavat debug printit, jotka ovat siis :

```
(-7.0, 4.0, 0.0)
(0.0, 4.0, -5.0)
(0.0, 0.0, -5.0)
```

Lopuksi muuta RobotArm step arvoksi 1.0 ja aja ohjelma – robotti singahtaa paikalle. Jatkossa kun testaat ohjelmaa niin kannattaa käyttää pieniä arvoja, jos haluat nähdä tarkkaan mitä robotti tekee ja isoja arvoja jos haluat että testi etenee nopeasti. Voit myös muuttaa stepin arvoa koodissasi kesken ajon, jos haluat esim aloittaa nopealla vauhdilla ja sitten hidastaa, kun tullaan siihen koodiin mitä haluat debugata.

Reflektointi

Motivointi
Jos teet mallilukujärjestyksen mukaan automaatiopääainetta, niin robotiikkakurssi on edessä. Kurssin opettajan mukaan monella opiskelijalla on ongelmia hahmottaa eri 3D koordinaatistoja. Siellä asiat pitää pystyä omaksumaan varsin matemaattisen kuivasta esityksestä. Tällä kurssilla annetaan pehmeää laskua aiheeseen käytännön esimerkkien ja 3D visualisoinnin kautta.

Reflektoidaan dokumentin alussa mainittuihin tavoitteisiin liittyviä asioita.

- Selitä milloin ja miksi käytettiin maailma (world) koordinaatteja vs paikallisia (local) koordinaatteja
- missä luokassa tehtiin APP ja mitä kohtaa sovelluksesta pitäisi muuttaa, jos olisi tiedossa joku este, johon robotti ei saa törmätä?
- jos meillä olisi karteesisen robotin sijasta käsivarsirobotti, jossa ohjataan niveliä, niin minkälaisia muutoksia tulisi APPhen ja robotin ohjaukseen?